Quantifying the Effects of Recent Protocol Improvements to Standards-Track TCP*

(Extended Version)

Michele C. Weigle, Kevin Jeffay, and F. Donelson Smith
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA
{mcweigle,jeffay,smithfd}@cs.unc.edu

ABSTRACT

We assess the state of Internet congestion control and error recovery through a controlled study that considers the integration of standards-track TCP error recovery and both TCP and router-based congestion control. The goal is to examine and quantify the benefits of deploying standards-track technologies for contemporary models of Internet traffic as a function of the level of offered network load. We limit our study to the dominant and most stressful class of Internet traffic: bursty HTTP flows. Contrary to expectations and published prior work, we find that for HTTP flows (1) there is no clear benefit in using TCP SACK over TCP Reno, (2) unless congestion is a serious concern (*i.e.*, unless average link utilization is approximately 80% or higher), there is little benefit to using RED queue management, (3) above 80% link utilization there is potential benefit to using Adaptive RED with ECN marking, however, complex performance trade-offs exist and the results are dependent on parameter settings.

I. INTRODUCTION

Improvements to TCP's error recovery and congestion control/avoidance mechanisms have been a mainstay of contemporary networking research. Representative innovations in error control include the use of fast transmissions (TCP Reno), fast retransmission in the face of multiple losses (TCP New Reno), and selective acknowledgements (TCP SACK). Representative innovations in congestion control include the congestion avoidance and the additive-increase, multiplicative decrease algorithm (TCP Tahoe), fast recovery (TCP Reno), early congestion detection in routers (RED), and explicit congestion notification (ECN).

While simulation studies have shown performance improvements with the addition of each new piece of networking technology, the evaluations have often been simplistic and have largely considered each improvement in isolation. In this work we assess the state of congestion control and error recovery proposed for the Internet through a controlled study that considers the integration of standards-track TCP error recovery and TCP/router-based congestion control. The goal is to examine and quantify the benefits of deploying these technologies for contemporary models of Internet traffic as a function of the level of offered network load.

Although numerous modifications to TCP have been proposed, we limit our consideration to proposals that have either been formally standardized or are being proposed for standardization as these are the most likely to be widely deployed on the Internet. We report the results of an extensive study into the impact of using TCP Reno versus TCP SACK in networks employing drop-tail queue management versus random early detection queue management (specifically, adaptive, gentle RED) in routers. We assess the performance of combinations of error recovery and router queue management through traditional network-centric measures of performance such as link utilization and loss-rates, as well as through user-centric measures of performance such as response time distributions for Web request-response transactions.

The results are somewhat surprising. Considering both network and end-user-centric measures of performance, for bursty HTTP traffic sources:

- There is no clear benefit in using SACK over Reno, especially when the complexity of implementing SACK is considered. This result holds independent of load and pairing with queue management algorithm.
- Adaptive RED (ARED) with ECN marking performs better than ARED with packet dropping and the value of marking increases as the offered load increases. ECN also offers more significant gains in performance when the target delay parameter is small (5 ms).
- However, unless congestion is a serious concern (*i.e.*, for average link utilizations of 80% or higher with bursty sources), there is little benefit to using RED queue management in routers.
- In congested networks, a fundamental trade-off exists between optimizing response time performance of short responses versus long responses. If one favors short responses, ARED with ECN marking and a small target delay (5 ms) performs better than drop-tail FIFO independent of the level of congestion (at the expense of long responses). This conclusion should be tempered with the caution that, like RED, ARED performance is also sensitive to parameter

* This work supported in parts by grants from the National Science Foundation (grants ITR-0082870, CCR-0208924, EIA-0303590, and ANI-0323648), Cisco Systems Inc., and the IBM Corporation.

This is an extended version of a paper by the same name published in: Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Orlando, FL, October 2003, pages 226-229.

settings. At all loads there is little difference between the performance of drop-tail FIFO and ARED with ECN and a larger target delay (60 ms). If one favors long responses, drop-tail FIFO performs better than ARED with ECN marking. In addition, as load increases FIFO also results in higher link utilization.

In total, we conclude that for user-centric measures of performance, router-based congestion control (active queue management), especially when integrated with end-system protocols (explicit congestion notification) has a greater impact on performance than protocol improvements for error recovery. However, for lightly to moderately loaded networks (*e.g.*, 50% average utilization) neither queue management nor protocol improvements significantly impact performance.

The following sections provide background on the specific error recovery and congestion control improvements we are considering as well as a summary of past evaluations of each. Section 5 explains our experimental methodology and Section 6 presents a summary of our main results.

II. Background — Congestion Control and Avoidance

A. TCP Reno

TCP Reno is by far the most widely deployed TCP implementation. Reno congestion control consists of two major phases: *slow-start* and *congestion avoidance*. In addition, congestion control is integrated with related *fast retransmit* and *fast recovery* error recovery mechanisms.

Slow-start restricts the rate of packets entering the network to the same rate that acknowledgments (ACKs) return from the receiver. The receiver sends cumulative ACKs which acknowledge the receipt of all bytes up to the sequence number carried in the ACK (*i.e.,* the receiver sends the sequence number of the next packet it expects to receive). The congestion window, *cwnd*, controls the rate at which data is transmitted and loosely represents the amount of unacknowledged data in the network. For every ACK received during slow-start, *cwnd* is incremented by one segment.

The congestion avoidance phase conservatively probes for additional bandwidth. The slow-start threshold, *ssthresh*, is the threshold for moving from slow-start to congestion avoidance. When *cwnd* > *ssthresh*, slow-start ends and congestion avoidance begins. Once in congestion avoidance, *cwnd* is increased by one segment every round-trip time, or as commonly implemented, by 1/*cwnd* of a segment for each ACK received.

When packet loss occurs, *ssthresh* is set to 1/2 the current value of *cwnd*, and as explained below, the value of *cwnd* is set depending on how the loss was detected. In the simplest case, (a loss that is detected by the expiration of a timer) the connection returns to slow-start after a loss.

B. Random Early Detection

Internet routers today employ traditional FIFO queuing (called "drop-tail" queue management). Random Early Detection (RED) is a router-based congestion control mechanism that seeks to reduce the long-term average queue length in routers [3]. A RED router monitors queue length statistics and probabilistically drops arriving packets even though space exists to enqueue the packet. Such "early drops" are performed as a means of signaling TCP flows (and others that respond to packet loss as an indicator of congestion) that congestion is incipient in the router. Flows should reduce their transmission rate in response to loss (as outlined above) and thus prevent router queues from overflowing.

Under RED, routers compute a running weighted average queue length that is used to determine when to send congestion notifications back to end-systems. Congestion notification is referred to as "marking" a packet. For standard TCP end-systems, a RED router drops marked packets to signal congestion through packet loss. If the TCP end-system understands packet-marking, a RED router marks and then forwards the marked packet.

RED's marking algorithm depends on the average queue size and two thresholds, min_{th} and max_{th}. When the average queue size is below min_{th}, RED acts like a drop-tail router and forwards all packets with no modifications. When the average queue size is between min_{th} and max_{th}, RED marks incoming packets with a certain probability. When the average queue size is greater than max_{th}, all incoming packets are dropped. The more packets a flow sends, the higher the probability that its packets will be marked. In this way, RED spreads out congestion notifications proportionally to the amount of space in the queue that a flow occupies.

The RED thresholds min_{th} and max_{th}, the maximum drop probability max_p, and the weight given to new queue size measurements w_q, play a large role in how the queue is managed. Recommendations [4] on setting these RED parameters specify that max_{th} should be set to three times min_{th}, w_q should be set to 0.002, or 1/512, and max_p should be 10%.

C. Adaptive RED

Adaptive RED (ARED) [5] is a modification to RED which addresses the difficulty of setting appropriate RED parameters. ARED adapts the value of max_p so that the average queue size is halfway between min_{th} and max_{th}. The maximum drop probability, max_p is kept between 1-50% and is adapted gradually. ARED includes another modification to RED, called "gentle RED" [7]. In gentle RED, when the average queue size is between max_{th} and $2 \times max_{th}$, the drop probability is varied linearly from max_p to 1, instead of being set to 1 as soon as the average is greater than max_{th}. When the average queue size is between max_{th} and $2 \times max_{th}$, selected packets are no longer marked, but always dropped.

ARED's developers provide guidelines for the automatic setting of min_{th}, max_{th}, and w_q. Setting min_{th} results in a trade-off between throughput and delay. Larger queues increase throughput, but at the cost of higher delays. The rule of thumb suggested by the authors is that the average queuing delay should only be a fraction of the round-trip time (RTT). If the target average queuing delay is $target_{delay}$ and C is the link capacity in packets per second, then min_{th} should be set to $target_{delay} \times C/2$. The guideline for setting max_{th} is that it should be $3 \times min_{th}$, resulting in a target average queue size of $2 \times min_{th}$. The weighting factor w_q controls how fast new meas-

urements of the queue affect the average queue size and should be smaller for higher speed links. This is because a given number of packet arrivals on a fast link represents a smaller fraction of the RTT than for a slower link. It is suggested that w_q be set as a function of the link bandwidth, specifically, $1 - exp(-1/C)$.

In addition to ARED, many extensions to RED have been proposed and been studied. We use ARED in our study because it includes the "gentle" mode, which is now the recommended method of using RED, and because we believe ARED is the most likely RED variant to be standardized and deployed.

D. Explicit Congestion Notification

Explicit Congestion Notification (ECN) [8, 9] is an optimization of active queue management that allows routers to notify end systems when congestion is present in the network. When an ECN-capable router detects that its average queue length has reached a threshold, it marks packets by setting the CE ("congestion experienced") bit in the packets' TCP headers. (The decision of which packets to mark depends on the queue length monitoring algorithm in the router.) When an ECN-capable receiver sees a packet with its CE bit set, an ACK with its ECN-Echo bit set is returned to the sender. Upon receiving an ACK with the ECN-Echo bit set, the sender reacts in the same way as it would react to a packet loss (*i.e.*, by halving the congestion window). Ramakrishnan and Floyd [9] recommend that since an ECN notification is not an indication of packet loss, the congestion window should only be decreased once per RTT, unless packet loss does occur. A TCP sender implementing ECN thus receives two notifications of congestion, ECN and packet loss. This allows senders to be more adaptive to changing network conditions.

ECN is recommended for use in routers that monitor their average queue lengths over time (*e.g.*, routers running RED), rather than those that can only measure instantaneous queue lengths. This allows for short bursts of packets without triggering congestion notifications.

III. Background — Error Recovery

A. Error Recovery in TCP Reno

TCP Reno provides two methods of detecting packet loss: the expiration of a timer and the receipt of three duplicate acknowledgements. Whenever a new packet is sent, the retransmission timer (RTO) is set. If the RTO expires before the packet is acknowledged, the packet is assumed to be lost. When the RTO expires, the packet is retransmitted, *ssthresh* is set to 1/2 *cwnd*, *cwnd* is set to 1 segment, and the connection re-enters slow-start.

Fast retransmit specifies that a packet can be assumed lost if three duplicate ACKs are received. This allows TCP Reno to avoid a lengthy timeout during which no data is transferred. When packet loss is detected via three duplicate ACKs, fast recovery is entered. In fast recovery, *ssthresh* is set to 1/2 *cwnd*, and *cwnd* is set to *ssthresh* + 3. For each additional duplicate ACK received, *cwnd* is incremented by 1 segment. New segments can be sent as long as *cwnd* allows. When the first ACK arrives for the retransmitted packet, *cwnd* is set back to *ssthresh*. Once the lost packet has been acknowledged, TCP leaves fast recovery and returns to congestion avoidance.

Fast recovery also provides a transition from slow-start to congestion avoidance. If a sender is in slow-start and detects packet loss through three duplicate ACKs, after the loss has been recovered, congestion avoidance is entered. The only other way to enter congestion avoidance is if *cwnd* > *ssthresh*. In many cases, though, the initial value of *ssthresh* is set to a very large value, so packet loss is often the only trigger to enter congestion avoidance.

TCP Reno can only recover from one packet loss during fast retransmit and fast recovery. Additional packet losses in the same window may require that the RTO expire before being retransmitted. The exception is when *cwnd* is greater than 10 segments. In this case, Reno could recover from two packet losses by entering fast recovery twice in succession. This causes *cwnd* to effectively be reduced by 75% in two RTTs [10].

B. Selective Acknowledgments

A recent addition to the standard TCP implementation is the selective acknowledgment option (SACK) [2, 10]. The SACK option contains up to four (or three, if RFC 1323 timestamps are used) SACK blocks, which specify contiguous blocks of received data. Each SACK block consists of two sequence numbers which delimit the range of data the receiver holds. A receiver can add the SACK option to ACKs it sends back to a SACK-enabled sender. In the case of multiple losses within a window, the sender can infer which packets have been lost and should be retransmitted using the information in the SACK blocks.

A SACK-enabled sender can retransmit multiple lost packets in one RTT. The SACK recovery algorithm only operates once fast recovery has been entered via the receipt of three duplicate ACKs. Whenever an ACK with new information is received, the sender adds to a list of packets (called the *scoreboard*) that have been acknowledged. These packets have sequence numbers past the current value of the highest cumulative ACK. At the beginning of fast recovery, the sender estimates the amount of unacknowledged data "in the pipe" based on the highest packet sent, the highest ACK received, and the number of packets in the *scoreboard*. This estimate is saved in the variable *pipe*. Each time a packet is sent the value of *pipe* is incremented. The *pipe* counter is decremented whenever a duplicate ACK arrives with a SACK block indicating that new data was received. When *pipe* is less than *cwnd*, the sender can either send retransmissions or transmit new data. When the sender is allowed to send data, it first looks at the *scoreboard* and sends any packets needed to fill gaps at the receiver. If there are no such packets, then the sender can transmit new data. The sender leaves fast recovery when all of the data that was unacknowledged at the beginning of fast recovery has been acknowledged.

Allowing up to three SACK blocks per SACK option ensures that each SACK block is transmitted in at least three ACKs, providing some amount of robustness in the face of packet loss.

IV. Related Work

A. Evaluations of TCP SACK

A simulation study [10] comparing the use of TCP Reno to TCP SACK concluded that there were benefits to using SACK. One of the most notable improvements arises from decoupling the determination of the time to retransmit from the determination of which packets to retransmit. In simulations with up to four lost packets in a window, SACK avoided costly retransmission timeouts and performed significantly better than Reno. Additional studies [11] showed that the presence of SACK flows does not detrimentally affect Reno flows.

Mathis *et al*. [1] looked at the large-scale behavior of several TCP congestion control algorithms, including SACK-enhanced protocols. The authors ran simulations in NS-1 using two flows. They found that with large buffers on drop-tail queues, TCP SACK flows kept large persistent queues because they avoided timeouts, which serve to drain the queue. This behavior was not seen with RED queues, which randomize packet drops.

Yan and Xu [13] compared TCP SACK with TCP Reno via simulations where bi-directional background traffic was generated according to the *tcplib* model. Different ratios of background to foreground traffic and different protocols for the background traffic were used. In the foreground were bulk transfers of 512 KB. In each case, SACK had the highest throughput, but retransmitted almost as much as Reno did. The authors speculated that due to the high aggregation and short flows in web traffic, SACK may not have the same gains as in a long-lived traffic mix. Experiments were also run through a 17-hop path over the Internet again using 512 KB transfers. In this case SACK still saw higher throughput and fewer retransmitted bytes than Reno.

Bruyeron *et al*., ran experiments over a testbed with single file transfers using a 16 KB receiver buffer size to compare the performance of TCP Reno and TCP SACK [14]. Packet losses were generated artificially using several degrees of packet loss probability. With isolated drops, SACK performed slightly better than Reno with a moderate drop probability (3%). With a low loss probability (1%), Reno was able to recover from most losses with a single fast retransmit (only one loss in a window), so SACK did not offer improvement. With a high loss probability (9%), SACK suffered many of the retransmission timeouts that Reno did. This is because SACK must wait for a RTO to expire if a retransmitted packet is lost or if several ACKs are lost. The authors also performed experiments on the testbed with bursty losses. The burst size was set to three packets (*i.e.,* each time a loss event was scheduled, three packets would be lost). Since there was only one flow, a burst of three packet losses would always affect that flow, meaning that most of the time, there would be multiple losses in a window. Not surprisingly, SACK performed better than Reno with bursty losses. With high loss probability (3%), SACK performed no better than Reno.

Bruyeron *et al*. also ran experiments over a cross-country Internet path were also run. A Reno FTP session was carried out for 2 minutes, followed by 2 minutes of a SACK FTP transfer. Experiments were run at different times during the day to correspond to heavy congestion, average congestion, and light congestion. In all instances, SACK got higher throughput than Reno. In times of heavy and average congestion, SACK saw about 30% higher throughput. The authors found that the average loss burst size was between 20 to 30 packets.

Bolliger *et al*. [15] looked at the performance of several congestion control protocols over various Internet links. Using data gathered from running the network probe daemon, they found that TCP SACK could have avoided about 12% of the timeouts experienced by TCP Reno. The authors note that Reno and SACK see the same performance with no packet loss and with high packet loss (9% for single drops and 3% for burst drops). In cases of medium levels of packet loss (less than 10%), the SACK saw 49-89% higher throughput than Reno.

Balakrishnan *et al*. [12] analyzed traces from a busy web server (1996 Atlanta Olympics web server). They report that almost 50% of all packet losses were detected via the expiration of the RTO and conclude that using SACK would have only avoided 4% of the timeouts.

B. Evaluations of RED

Christiansen *et al*. [6] ran experiments with one-way web traffic using TCP Reno over a RED router. Using HTTP response times as the main performance metric, they found that RED offered no improvement over drop-tail. Tuning RED parameters was done through extensive experimentation, and the best parameters for web traffic were not intuitive. In fact, the recommended values performed worse for web traffic than drop-tail, especially at very high loads.

May *et al*. [21] ran experiments with web-like traffic over commercial routers running RED. They found that RED does not offer a large advantage over drop-tail, RED's performance was hindered by small buffers, and tuning RED parameters was difficult.

Zhang and Qiu [22] compared RED, RED with ECN, and drop-tail with various traffic mixes, using both Reno and SACK. They ran 200-second simulations with both one-way and two-way traffic scenarios. The main metric of performance used was average goodput for groups of flows (web or FTP). Short-lived transfers saw improved goodput with RED as opposed to drop-tail when competing against long-lived flows. The long-lived flows had large congestion windows, so the probability of seeing more than one drop in a window increased. When SACK was used as the base protocol, the long-lived transfers saw an increase in goodput, because SACK was avoiding many of the timeouts that were experienced with Reno. The benefit of SACK for long-lived flows diminished when using RED because RED smoothed the loss pattern, reducing the number of drops in a single window. Whenever FTP flows saw increased goodput, the goodput of web flows necessarily diminished. Conversely, whenever FTP flows saw decreased goodput, web flows saw increased goodput. With web traffic in both directions, the authors found that when the congestion level in the reverse path was high, using RED results in higher goodput than drop-tail. When the congestion in

the reverse path was low and the congestion level in the forward path was high, there was no meaningful difference between RED and drop-tail. When the congestion levels in each direction were low, drop-tail saw slightly higher goodput than RED.

C. Evaluations of ECN

In RFC 2884 [28], Ahmed and Salim report on experiments run to evaluate the performance of ECN. They ran experiments with competing ECN and non-ECN flows against one-way background traffic. The RED parameters and the amount of background traffic were selected so that the average queue size remained between min_{th} and max_{th} for most of the experiment. The authors observed that when the average queue size was above max_{th}, RED acted like a full drop-tail queue. The authors tested both bulk transfers and short web-like transfers. In the bulk transfer experiments, the ECN flows saw very little retransmissions hence the ECN flows saw better goodput than the non-ECN flows. The authors also note that at high congestion levels, the benefits of ECN over non-ECN may be greater when the base protocol is Reno rather than SACK, since Reno may suffer timeouts when multiple packets are dropped in a single window. With the transactional transfers (short, web-like transactions), the advantage of ECN increased as the level of congestion increased. Also, they observed that as object sizes increased there was more of a chance for Reno's fast retransmit mechanism to recover quickly from packet loss, and so, ECN's advantage diminished.

Pentikousis *et al*. [24] performed studies comparing the performance of a set of ECN clients to a set of non-ECN clients. In these experiments, ECN flows did not compete against non-ECN flows. The authors found that ECN did not see higher goodput than non-ECN or drop-tail. Overall, they found that ECN reduced the number of packet drops, but did not necessarily improve goodput.

D. Summary

There have been many studies that look at SACK and RED (with and without ECN) individually. Some of these studies focus on long-lived traffic that is not representative of the traffic on the Internet of today that is dominated by comparatively short web transfers. When web-like traffic is used, measures such as overall average goodput for a group of flows are used. We prefer response time as a better indicator of the service received by a user of the Web. Other studies use artificial packet dropping mechanisms, sometimes to simulate wireless transmission errors, sometimes to study the performance for a specific loss rate. Studies also have used only one-way traffic, which ignores phenomena such as ACK compression and ACK loss. Some studies use very limited RTT distributions, which may lead to synchronization of end-systems' transmissions. Other simulation studies have been performed using NS-2's one-way TCP model. Given the prevalence of short transfers, one-way TCP is not a realistic simulation vehicle as it lacks TCP's connection setup[1] and teardown phases (which can take longer than the actual data transfer). In addition one-way TCP does not allow variable packet sizes and bi-directional data traffic in the same connection.

We extend previous studies of SACK, RED, and ECN to analyze the interactions between the mechanisms and their effects on realistic two-way web traffic. Our traffic model (and RTT model) is based on the analysis of recent Internet traffic traces, combining both short-lived and long-lived flows. As a result, we get different results from previous studies.

V. METHODOLOGY

A. Experimental Setup

We ran simulations in NS-2[2] [16] with varying levels of two-way HTTP 1.0 traffic. These two-way traffic loads provide roughly equal levels of congestion on both the "forward" path and "reverse" path in our network. The following pairings of error recovery and queue management techniques were tested: Reno with drop-tail FIFO queuing in routers, Reno with ARED using packet drops, ECN-enabled Reno with ARED using ECN packet marking, SACK with drop-tail FIFO, SACK with ARED using packet drops, and ECN-enabled SACK with ARED using packet marking. Table 1 presents a summary of the error-recovery and queue management pairings that were run.

The network we simulate consists of two clouds of web servers and clients positioned at each end of a 10 Mbps bottleneck link (Figure 1). There is a 10 Mbps bottleneck link between the two routers, a 20 Mbps link between each PackMime cloud and its corresponding aggregation node, and a 100 Mbps link between each aggregation node and the nearest router. The 20 Mbps limit is so that transient spikes are limited to 40 Mbps and will not overload the 100 Mbps link between the aggregation node and the first router. The data we present herein comes from measurements of the traffic on the "forward" path in the network.

The aggregation nodes in our simulations are NS-2 nodes that we developed called DelayBox to delay packets in the simulation. DelayBox is an NS analog to *dummynet* [20],

Table 1: Experiments

TCP	Queuing method	Queue Length (1,250 B packets)	ARED Delay $delay_{target}$ (min_{th}, max_{th})
Reno	Drop-Tail	111 pckts (1.5 BDP) 148 pckts (2 BDP)	
Reno	ARED	370 pckts (5 BDP)	5 ms (5,15) 60 ms (30, 90)
Reno	ARED + ECN	370 pckts (5 BDP)	5 ms (5,15) 60 ms (30, 90)
SACK	Drop-Tail	111 pckts (1.5 BDP) 148 pckts (2 BDP)	
SACK	ARED	370 pckts (5 BDP)	5 ms (5,15) 60 ms (30, 90)
SACK	ARED + ECN	370 pckts (5 BDP)	5 ms (5,15) 60 ms (30, 90)

[1] Although one-way TCP does not include specific mechanisms for connection setup, it does include an option to simulate the SYN packet and a 6 second timeout when the first data packet is lost.

[2] In order to use SACK and ECN in Full-TCP, we had to modify the ns-2.1b9 source code. See *http://www.cs.unc.edu/~mcweigle/ns/* for details.

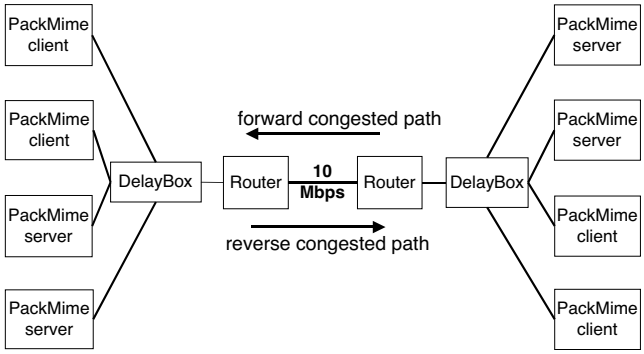


Figure 1: Simulated network environment.

which is used in network testbeds to delay packets. With DelayBox, packets from a TCP connection can be delayed before being passed on to the next node. This allows each TCP connection to experience a different minimum delay (and hence a different round-trip time), based on random sampling from a delay distribution. In our experiments DelayBox uses an empirical delay distribution from the PackMime model. This results in RTTs ranging from 1 ms to 3.5 seconds. The median RTT is 54 ms, the mean is 74 ms, and the 90th percentile is 117 ms. RTTs are assigned independently of request or response size and represent only propagation delay and do not include queuing delays.

The mean packet size for the HTTP traffic (excluding pure ACKs, but including headers) is 1,250 bytes.[3] This includes the HTTP responses for the forward path and the HTTP requests for the reverse path. For a target bottleneck bandwidth of 10 Mbps, we compute the bandwidth-delay product (BDP) to be 74 1,250-byte packets. In all cases, we set the maximum send window for each TCP connection to the BDP.

The simulation parameters used for TCP and Adaptive RED are listed in Tables 2-3.

B. HTTP Traffic Generation

The HTTP traffic we generate comes from the PackMime model [17] developed at Bell Labs. This model is based on the analysis of HTTP connections in a trace of a 100 Mbps Ethernet link connecting an enterprise network of approximately 3,000 hosts to the Internet [18, 19]. The fundamental parameter of PackMime is the TCP/HTTP connection initiation rate (a parameter of the distribution of connection interarrival times). The model also includes distributions of the size of HTTP requests, and the size of HTTP responses.

The request size distribution and the response size distribution are heavy-tailed (Figure 2). There are a large number of small request sizes and a few very large request sizes. Almost 90% of the requests are under 1 KB and fit in a single packet. The largest request is almost 1 MB. 60% of the responses fit into one packet and 90% of the responses fit into 10 packets, yet the largest response size is over 100 MB. Using this distribution, we will have many short-lived transfers, but also some very long-lived flows.

[3] We assume an Ethernet MSS of 1,500 bytes and use a maximum TCP data size of 1,420 bytes, counting for 40 bytes of base TCP/IP header and 40 bytes maximum of TCP options.

In each experiment we examine the behavior of traffic that consists of over 250,000 flows, with a total simulated time of 40 minutes. The distribution of response sizes has an infinite variance and hence simulations take a very long time to reach steady-state. Running the simulation for only a few minutes would take into account a small portion of the rich behavior of the traffic model. We ran our simulation for as long as the available hardware and software environments would support to capture a significant amount of this behavior.

We implemented PackMime traffic generation in NS-2 using Full-TCP, which includes bi-directional TCP connection flows, connection setup, connection teardown, and variable packet sizes. In our implementation, one PackMime "node" represents a cloud of HTTP clients or servers. The traffic load is driven by the user-supplied connection rate parameter, which is the number of new connections starting per second. The connection rate corresponding to each desired link loading was determined by a calibration procedure described below. New connections begin at their appointed time, whether or not any previous connection has completed.

Previous studies, including [25, 26, 27, 28], have shown that the size of files transferred over HTTP is heavy-tailed. With heavy-tailed file sizes, there are a large number of small files and a non-negligible number of extremely large files. These studies have also shown that the arrival pattern of web traffic is self-similar, which exhibits burstiness over several time scales. In other words, as the aggregation interval increases, the rate of packet arrivals are not smoothed out.

Figures 3-4 demonstrate the burstiness of the traffic in the PackMime model. We calculated the number bytes and packets arriving at the bottleneck link per interval over several timescales. The goal is to show that as the timescale increases, the traffic remains bursty. We plot 150 points at different timescales starting at 500 seconds into the simulation. Figure 3

Table 2: TCP Parameters

TCP parameter	Value
Initial window size	2 segments
Timestamp option	false
TCP tick	10 ms
BSD 4.2 bug fix	true
min RTO	1 second
Initial RTO	6 seconds
RFC 2988 RTT calculation	true
delayed ACK	false
SACK block size	8 bytes
max SACK blocks	3

Table 3: Adaptive RED Parameters

ARED parameter	Value
Packet mode	true
alpha	0.01
beta	0.9
Interval for adaptations	500 ms
max max_p	0.5
min max_p	0.01

shows the number of bytes arriving at the router for 150 intervals. Over all aggregation intervals, from 10 ms to 10 seconds, the traffic is bursty. The same observations can be made for the number of packets arriving at the router (Figure 4).

C. Queue Management

1) Drop Tail Settings: Christiansen *et al.* [6] recommend a maximum queue size between 1.25 x BDP and 2 x BDP for reasonable response times for drop-tail queues. The maximum queue buffer sizes tested in our drop-tail experiments were 1.5 x BDP and 2 x BDP.

2) Adaptive RED Settings: We ran sets of ARED experiments using the default ARED settings in NS-2 (target delay = 5 ms) and with parameters similar to those suggested by Christiansen (min_{th} = 30 and max_{th} = 90) giving a target delay of 60 milliseconds. Note that in both cases, min_{th} and max_{th} are computed automatically in NS based on the mean packet size, target delay, and link speed. The maximum router queue length was set to 5 x BDP. This ensured that there would be no tail drops, in order to isolate the effects of ARED.

D. Levels of Offered Load and Data Collection

The levels of offered load used in our experiments are expressed as a percentage of the capacity of a 10 Mbps link. We initially ran our network at 100 Mbps and determined the PackMime connection rates (essentially the HTTP request rates) that will result in average link utilizations (in both forward and reverse directions) of 5, 6, 7, 8, 8.5, 9, 9.5, 10, and 10.5 Mbps. The connection rate that results in an average utilization of 8% of the (clearly uncongested) 100 Mbps link will be used to generate an offered load on the 10 Mbps link of 8 Mbps or 80% of 10 Mbps link. Note that this "80% load" (*i.e.*, the connection rate that results in 8 Mbps of traffic on the 100 Mbps link) will not actually result in 8 Mbps of traffic on the 10 Mbps link. The bursty HTTP sources will cause congestion on the 10 Mbps link and the actual utilization of the link will be a function of the protocol and router queue management scheme used. (And the link utilization achieved by a given level of offered load is a metric for comparing protocol/queue management combinations. See Figure 5.)

Given the bursty nature of our HTTP traffic sources, we used a 120-second "warm-up" interval before collecting any data. After the warm-up interval, we allow the simulation to proceed until approximately 250,000 HTTP request-response pairs have been transmitted. We also require that at least one 10 MB response has started a transfer before 1,000 seconds after the warm-up period. This ensures that we will have some very long transfers in the network along with the typical short-lived web transfers.

E. Performance Metrics

In each experiment, we measured the HTTP response times (time from sending HTTP request to receiving entire HTTP response), link utilization, throughput (number of bytes entering the bottleneck), average loss rate, average percentage of flows that experience loss, and average queue size. These summary statistics are given for each experiment in Figures 5-6.

Table 4: Summary of labels and abbreviations.

Abbreviation	Description
DT-111q	Drop-Tail with 111 packet queue (1.5 BDP)
DT-148q	Drop-Tail with 148 packet queue (2 BDP)
ARED-5ms	Adaptive RED with 5 ms target delay
ARED-60ms	Adaptive RED with 60 ms target delay
ARED+ECN-5ms	Adaptive RED with ECN & 5 ms target delay
ARED+ECN-60ms	Adaptive RED with ECN & 60 ms target delay

HTTP response time is our main metric of performance. We report the CDFs of response times for responses that complete in 1,500 ms or less. When discussing the CDFs, we discuss the percentage of flows that complete in a given amount of time. It is not the case that only small responses complete quickly and only large responses take a long time to complete. For example, between a 500 KB response that has a RTT of 1 ms and a 1 KB response that has a RTT of 1 second, the 500 KB response will likely complete before the smaller response.

VI. Results

We first present the results for different queue management algorithms when paired with TCP Reno end-systems. Next, results for queue management algorithms paired with TCP SACK end-systems are presented and compared to the Reno results. Finally, the two best scenarios are compared. In the following response time CDF plots, the response times obtained in a calibration experiment with an uncongested 100 Mbps link are included for a baseline reference as this represents the best possible performance. Table 4 lists the labels we use to identify experiments.

For completeness, we include the response time CDFs for all of the experiments we performed (Figures 7-21) at all load levels.[4] However, for space considerations here we discuss only the results for offered loads of 80-105%. This is motivated by the observation that while generally HTTP response time performance decreases as load increases, for loads less than 80%, there is no difference in link utilization (Figures 5-6) for any of the protocol/queue management combinations we consider. Moreover the differences in response times are more significant in the 80-10% load range.

A. Reno + DropTail

Figure 9 shows the CDFs of response times for Reno-DT-111q and Reno-DT-148q. There is little performance difference between the two queue sizes, though there is a crossover point in the response times. The crossover is described here only for illustration purposes, since the difference is minimal. At 80% load, the crossover is at coordinate (700 ms, 80%). This means that for both DT-111q and DT-148q, 80% of the HTTP responses completed in 700 ms or less. For a given response time less than 700 ms, DT-111q produces a slightly higher percentage of responses that complete in that time or less than does DT-148q. For a given response time greater

[4] Note that our plots are best viewed in color.

than 700 ms, DT-148q yields a slightly higher percentage of responses that complete in that time or less than DT-111q does.

B. Reno + Adaptive RED

Response time CDFs for Reno-DT-111q, Reno-DT-148q, Reno-ARED-5ms, and Reno-ARED-60ms are shown in Figure 10. At almost all loads both of the FIFO drop-tail queues perform no worse than ARED-60ms. There is a distinct crossover point between Reno-ARED-5ms and Reno-ARED-60ms (and Reno-DT-148q and Reno-DT-111q) at 400 ms. This points to a tradeoff between improving response times for some flows and causing worse response times for others. For responses that complete in less than 400 ms, ARED-5ms offers better performance. For responses that complete in over 400 ms, ARED-60ms, Reno-DT-111q, or Reno-DT-148q are preferable. As the load increases, the crossover remains near a response time of 400 ms, but the percentage of completed responses in that time or less decreases. Also, as load increases, the performance of ARED-5ms for longer responses is poor.

ARED-5ms keeps a much shorter average queue than ARED-60ms (Figure 5), but at the expense of longer response times for many responses. With ARED-5ms, many flows experience packet drops. Many of these connections are very short-lived, often consisting of a single packet (60% of responses consist of only one packet and 80% consist of five or fewer packets) and when they experience packet loss, they are forced to suffer a retransmission timeout, increasing their HTTP response times. For ARED-5ms, the CDF of response times levels off after the crossover point and does not increase much until after 1 second, which is the minimum RTO in our simulations. This indicates that a significant portion of the flows suffered timeouts.

As congestion increased toward severe levels, the response time benefits from the FIFO queue became substantially greater. At 105% load about 60% of responses (those taking longer than 300-400 milliseconds to complete) have better response times with the FIFO queue while only 40% of responses are better with ARED-5ms. For this same 60% of responses, ARED-60ms is also superior to ARED-5ms.

ARED should give better performance than the original RED design without the "gentle" option. At high loads, a significant number of packets arrive at the queue when the average queue size is greater than max_{th}. Without the "gentle" option, these packets would all be dropped, rather than being dropped probabilistically. To see the full differences between ARED including the gentle option and the original RED design we compared the results between the two designs at loads of 80%, 90%, and 100% (Figure 11). In these comparisons, two configurations of the original RED minimum and maximum thresholds were used: (5,15) and (30,90) which correspond roughly to the 5ms and 60ms target queue lengths used for ARED. For the original RED experiments, we used the recommended settings of w_q = 1/512 and max_p = 10%. For the ARED experiments, w_q was set automatically based on link speed to 0.001, and max_p was adapted between 1-50%. Figure 11 shows that the adaptation of max_p and the linear increase in drop probability from max_p to 1.0 in ARED is an improvement over the original RED design.

C. Reno + Adaptive RED + ECN

The full value of ARED is realized only when it is coupled with a more effective means of indicating congestion to the TCP endpoints than the implicit signal of a packet drop. ECN is the congestion signaling mechanism intended to be paired with ARED. Figure 12 presents the response time CDFs for Reno-ARED-5ms, Reno-ARED-60ms, Reno-ARED+ECN-5ms, and Reno-ARED+ECN-60ms. Up to 90% load, ARED+ECN-5ms delivers superior or equivalent performance for all response times. Further, ECN has a more significant benefit when the ARED target queue is small. As before, there is a tradeoff where the 5 ms target delay setting performs better before the crossover point and the 60 ms setting performs slightly better after the crossover point. The tradeoff when ECN is paired with ARED is much less significant. ARED+ECN-5ms does not see as much drop-off in performance after the crossover point. For the severely congested case, ECN provides even more advantages, especially when coupled with a small target delay.

At loads of 80% and higher, ARED+ECN-5ms produces lower link utilization than ARED+ECN-60ms (Figure 5). This tradeoff between response time and link utilization is expected. ARED+ECN-5ms keeps the average queue size small so that packets see low delay as they pass through the router. Flows that experience no packet loss should see very low queuing delays, and therefore, low response times. On the other hand, larger flows may receive ECN notifications and reduce their sending rates so that the queue drains more often. As expected, drop-tail results in the highest link utilization and the lowest drop rates.

Finally, we ran a set of experiments with one-way traffic to see how well ECN and ARED would perform in a less complex environment. By one-way traffic, we mean that there was HTTP response traffic flowing in only one direction on the link between routers. Thus the reverse-path carrying ACKs and HTTP request packet was very lightly loaded. This means that ACKs are much less likely to be lost or "compressed" at the router queue and that (1) TCP senders will receive a smoother flow of ACKs to "clock" their output segments and (2) ECN congestion signals will be more timely. In some sense, this is a best case scenario for ARED with ECN. The results for this case are given in Figure 14. With two-way traffic, performance is significantly reduced over the one-way case especially for the severe congestion at 105% offered load. These results clearly illustrate the importance of considering the effects of congestion in both directions of flow between TCP endpoints.

D. SACK

The experiments described above were repeated by pairing SACK with the different queue management mechanisms in place of Reno. Figures 18-20 show a comparison between Reno and SACK based on response time CDFs when paired with drop-tail FIFO, ARED, and ARED+ECN. Overall, SACK provides no better performance than Reno. When

paired with ARED+ECN, SACK and Reno are essentially identical independent of load. When paired with drop-tail FIFO, Reno appears to provide somewhat superior response times especially when congestion is severe.

We expected to see improved response times with SACK over Reno with drop-tail queues. SACK should prevent some of the timeouts that Reno would have to experience before recovering from multiple drops in a window. Why is there not a large improvement with SACK over Reno with drop-tail? Recall that with HTTP most TCP connections never send more than a few segments and for loss events in these connections, SACK never comes into play. For the relatively small number of connections where the SACK algorithm is invoked, the improvement in response time by avoiding a timeout is modest relative to the overall length of the responses that experience losses recoverable with SACK.

E. Drop-Tail vs. ARED+ECN

Here, we compare the performance of the two "best" error recovery and queue management combinations. Figure 21 presents the response time CDFs for Reno-DT-148q and Reno-ARED+ECN-5ms. With these scenarios, the fundamental tradeoff between improving response times for some responses and making them worse for other responses is clear. Further, the extent of the tradeoff is quite dependent on the level of congestion. At 80% load, Reno-ARED+ECN-5ms offers better response-time performance for nearly 75% of responses but marginally worse for the rest. At levels of severe congestion the improvements in response times for Reno-ARED+ECN-5ms apply to around 50% of responses while the response times of the other 50% are degraded significantly, Reno-DT-148q and Reno-ARED+ECN-5ms are on the opposite ends of the queue-management spectrum, yet, they each offer better HTTP response times for different portions of the total set of responses. Further complicating the tradeoff is the result that Reno-DT-148q gives higher link utilization along with a high average queue size, while Reno-ARED+ECN-5ms gives low average queue sizes, but also lower link utilization (Figure 5).

F. Performance for Offered Loads less than 80%

For load levels lower than 80%, there is an advantage for ARED+ECN-5ms over DT-148q for shorter responses. The same tradeoff is present for 50-80% load as with loads over 80% and hence is a fundamental tradeoff. ARED+ECN-5ms has lower average queue sizes and the corresponding better response time performance for shorter responses, with similar link utilization as DT-148q. DT-148q performs better for responses that take longer than 600 ms to complete.

Much below 80% offered load, SACK and Reno have identical performance and ARED+ECN performs only marginally better than drop-tail. At these loads, there is no difference in link utilization between any of the queue management techniques. There is also very little packet loss (no average loss over 3%). For loads under 80%, given the complexity of implementing RED, there is no compelling reason to use ARED+ECN over drop-tail. Our complete results are summarized in Figures 7-8.

VII. CONCLUSIONS

Our results provide an evaluation of the state-of-the-art in TCP error recovery and congestion control in the context of Internet traffic composed of "mice" and "elephants." We used bursty HTTP traffic sources generating a traffic mix with a majority of flows sending few segments (< 5), a small number sending many segments (> 50), and a number in the [5, 50] segment range.

Using NS simulations, we evaluated how well various pairings of TCP-Reno, TCP-SACK, drop-tail FIFO, Adaptive RED (with both packet marking and dropping), and ECN perform in the context of HTTP traffic. Our primary metric of performance was the response time to deliver each HTTP object requested along with secondary metrics of link utilization, router queue size and packet loss percentage. Our main conclusions based on HTTP traffic sources and these metrics are:

- There is no clear benefit in using SACK over Reno, especially when the complexity of implementing SACK is considered. This result holds independent of load and pairing with queue management algorithm.
- As expected, ARED with ECN marking performs better than ARED with packet dropping and the value of ECN marking increases as the offered load increases. ECN also offers more significant gains in performance when the target delay is small (5 ms).
- Unless congestion is a serious concern (*i.e.*, for average link utilizations of 80% or higher with bursty sources), there is little benefit to using RED queue management in routers.
- ARED with ECN marking and a small target delay (5 ms) performs better than drop-tail FIFO with 2xBDP queue size at offered loads having moderate levels of congestion (80% load). This finding should be tempered with the caution that, like RED, ARED is also sensitive to parameter settings.
- At loads that cause severe congestion, there is a complex performance trade-off between drop-tail FIFO with 2xBDP queue size and ARED with ECN at a small target delay. ARED can improve the response time of about half the responses but worsens the other half. Link utilization is significantly better with FIFO.
- At all loads there is little difference between the performance of drop-tail FIFO with 2xBDP queue size and ARED with ECN marking and a larger target delay (60 ms).

ACKNOWLEDGMENTS

We thank Jin Cao, Bill Cleveland, Dong Lin, and Don Sun from Bell Labs for help in implementing their PackMime model in NS-2.

REFERENCES

[1] Matthew Mathis, Jeffrey Semke, and Jamsid Mahdavi, "The macroscopic behavior of the TCP congestion avoidance algorithm," *Computer Communication Review,* vol. 27, no.3, July 1997.

[2] M. Mathis, J. Mahdivi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," RFC 2018, October 1996.

[3] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[4] Sally Floyd, "RED: Discussions of setting parameters," email, available at http://www.icir.org/floyd/REDparameters.txt, November 1997.

[5] S. Floyd, R. Gummadi, & S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," (unpublished) http://www.icir.org/floyd/papers/adaptiveRed.pdf, 2001.

[6] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith, "Tuning RED for web traffic," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 249–264, June 2001.

[7] S. Floyd, "Recommendation on using the gentle variant of RED," note, available at http://www.icir.org/floyd/red/gentle.html, March 2000.

[8] Sally Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10—23, October 1994.

[9] K.K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification to IP," RFC 2481, Experimental, January 1999.

[10] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, & SACK TCP," *ACM Computer Communications Review*, July 1996.

[11] S. Floyd, "Issues of TCP with SACK," Tech. Rep., LBL, March 1996.

[12] Hari Balakrishnan, Venkata Padmanabhan, Srini Seshan, Mark Stemm, and Randy H. Katz, "TCP behavior of a busy internet server: Analysis and improvements," in *Proceedings of IEEE INFOCOM*, 1998.

[13] Elliot Limin Yan and Ya Xu, "Empirical analyses of SACK TCP Reno and modified TCP Vegas," unpublished, available at http://citeseer.nj.nec.com/246505.html, 1997.

[14] R. Bruyeron, B. Hemon, and L. Zhang, "Experimentations with TCP selective acknowledgment," *ACM Computer Communication Review*, vol. 28, no. 2, April 1998.

[15] J. Bolliger, U. Hengartner, and T. Gross, "The effectiveness of end-to-end congestion control mechanisms," TR 313, ETH Zurich, Feb. 1999.

[16] S. McCanne and S. Floyd, "NS network simulator," http://www.isi.edu/nsnam/ns/, 1995.

[17] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "PackMime: an Internet traffic generator," National Institute of Statistical Sciences Affiliates Workshop on Modeling and Analysis of Network Data, March , 2001.

[18] William S. Cleveland, Dong Lin, and Don X. Sun, "IP packet generation: statistical models for TCP start times based on connection-rate superposition," in *Proceedings of ACM SIGMETRICS,* 2000.

[19] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "On the nonstationarity of Internet traffic," in *Proceedings of ACM SIGMETRICS*, 2001.

[20] Luigi Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31—41, January 1997.

[21] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," in *Proceedings of IWQoS'99*, London, March 1999.

[22] Yin Zhang and Lili Qiu, "Understanding the end-to-end performance impact of RED in a heterogeneous environment," Cornell CS Technical Report 2000-1802, July 2000.

[23] Jamal Hadi Salim and Uvaiz Ahmed, "Performance evaluation of explicit congestion notification in IP networks", RFC 2884, July 2000.

[24] K. Pentikousis, H. Badr, and B. Kharmah, "On the performance gains of TCP with ECN," in *Proc. of the 2nd European Conference on Universal Multiservice Networks (ECUMN 2002)*, Colmar, France, April 2002.

[25] B. Mah, "An empirical model of HTTP network traffic," in *Proceedings of INFOCOM '97*, April 1997.

[26] P. Barford and M. E. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of Performance '98/ACM SIGMETRICS '98*, pp. 151-160, 1998.

[27] K. Park, G. Kim, and M. Crovella, "On the relationship between file sizes, transport protocols, and self-similar network traffic," in *Proceedings of ICNP*, 1996.

[28] W. Willinger and V. Paxson, "Where mathematics meets the internet," *Notices of the American Mathematical Society*, vol. 45, no. 8, pp. 961—970, Sept. 1998.

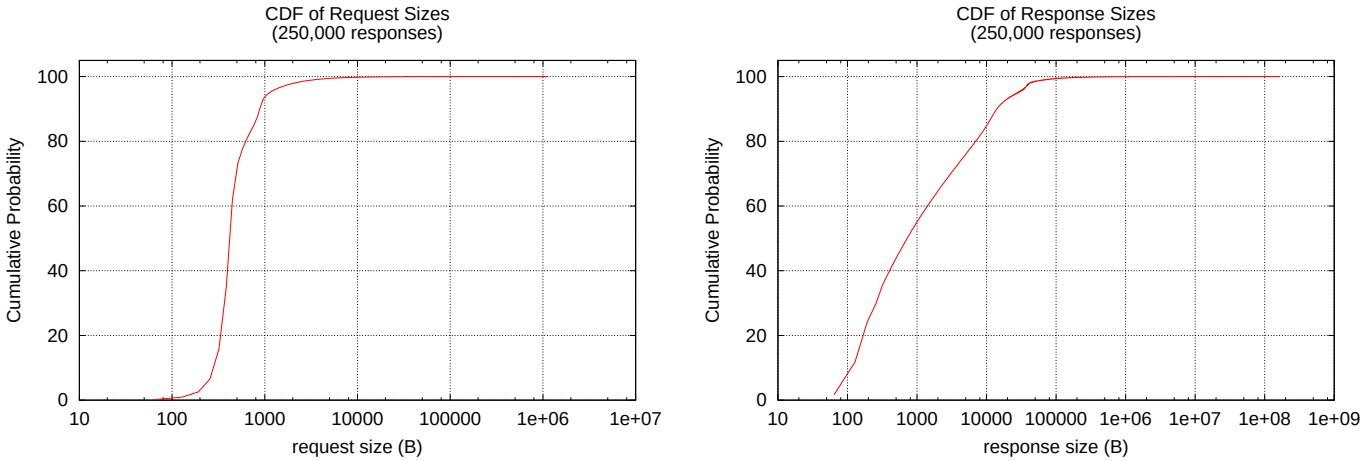


Figure 2: Distribution of HTTP request and response sizes for a typical experiment using PackMime

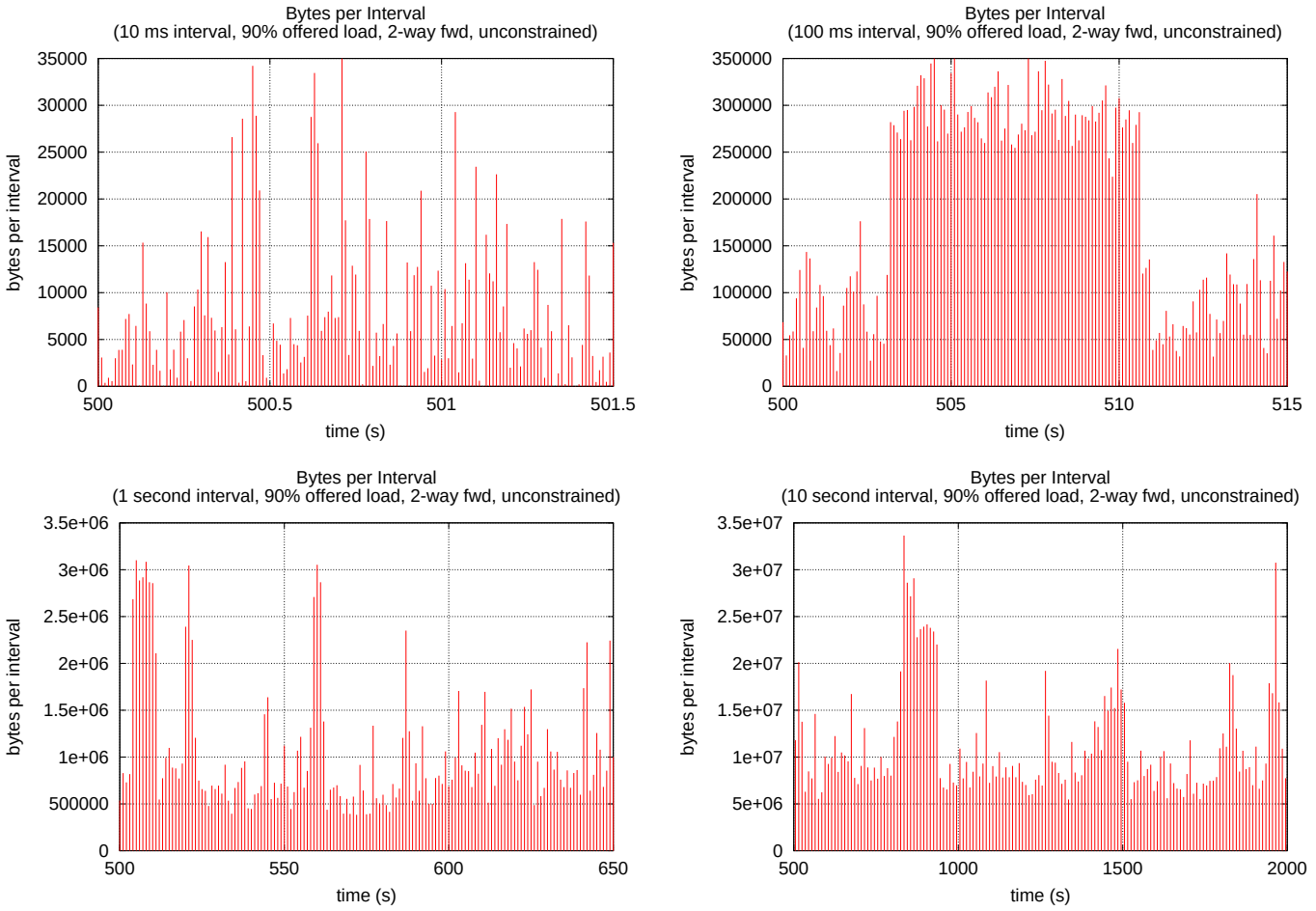


Figure 3: Illustration of the bursty nature of the HTTP traffic: bytes arriving at the router in various intervals

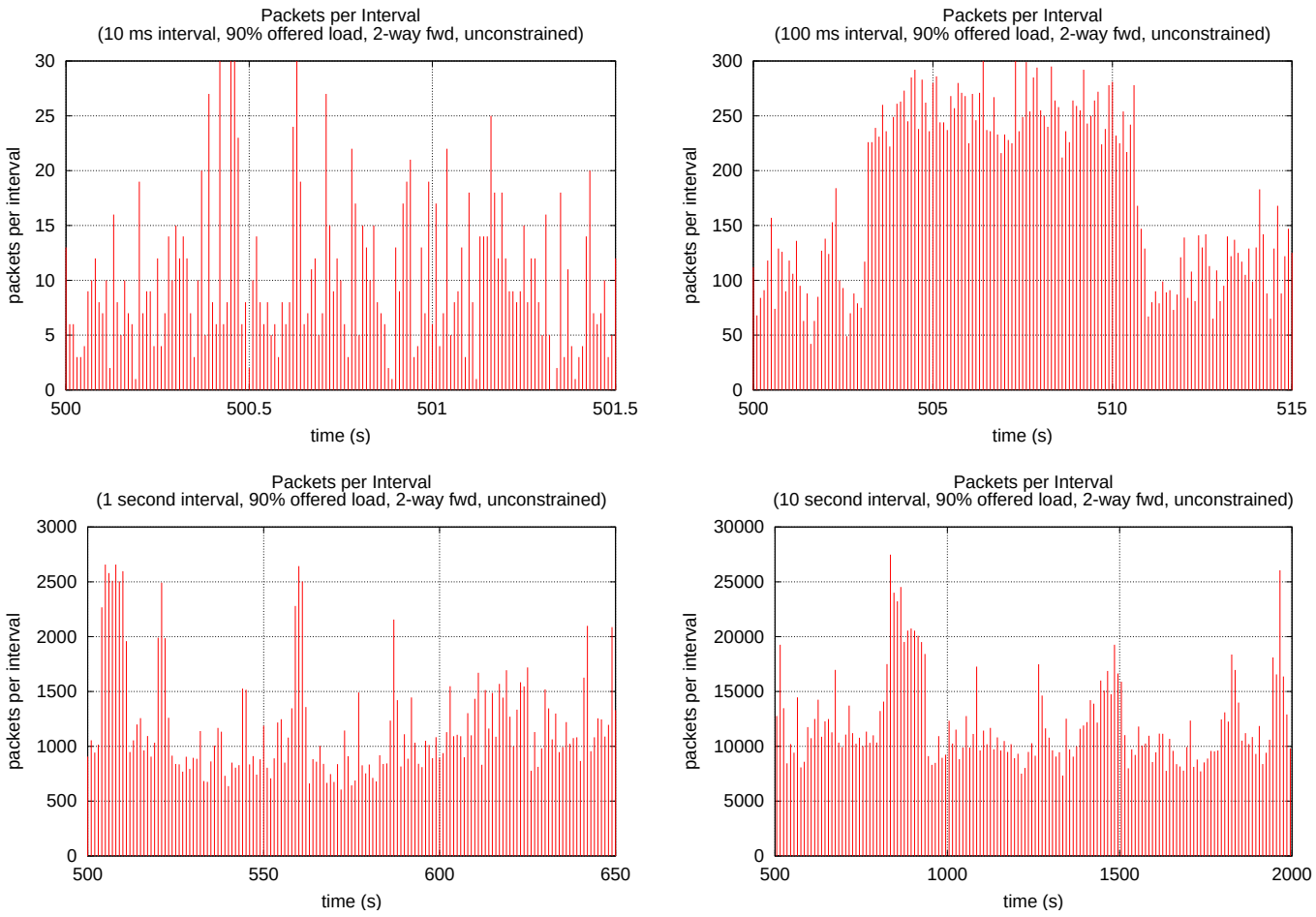


Figure 4: Illustration of the bursty nature of the HTTP traffic: packets arriving at the router in various intervals

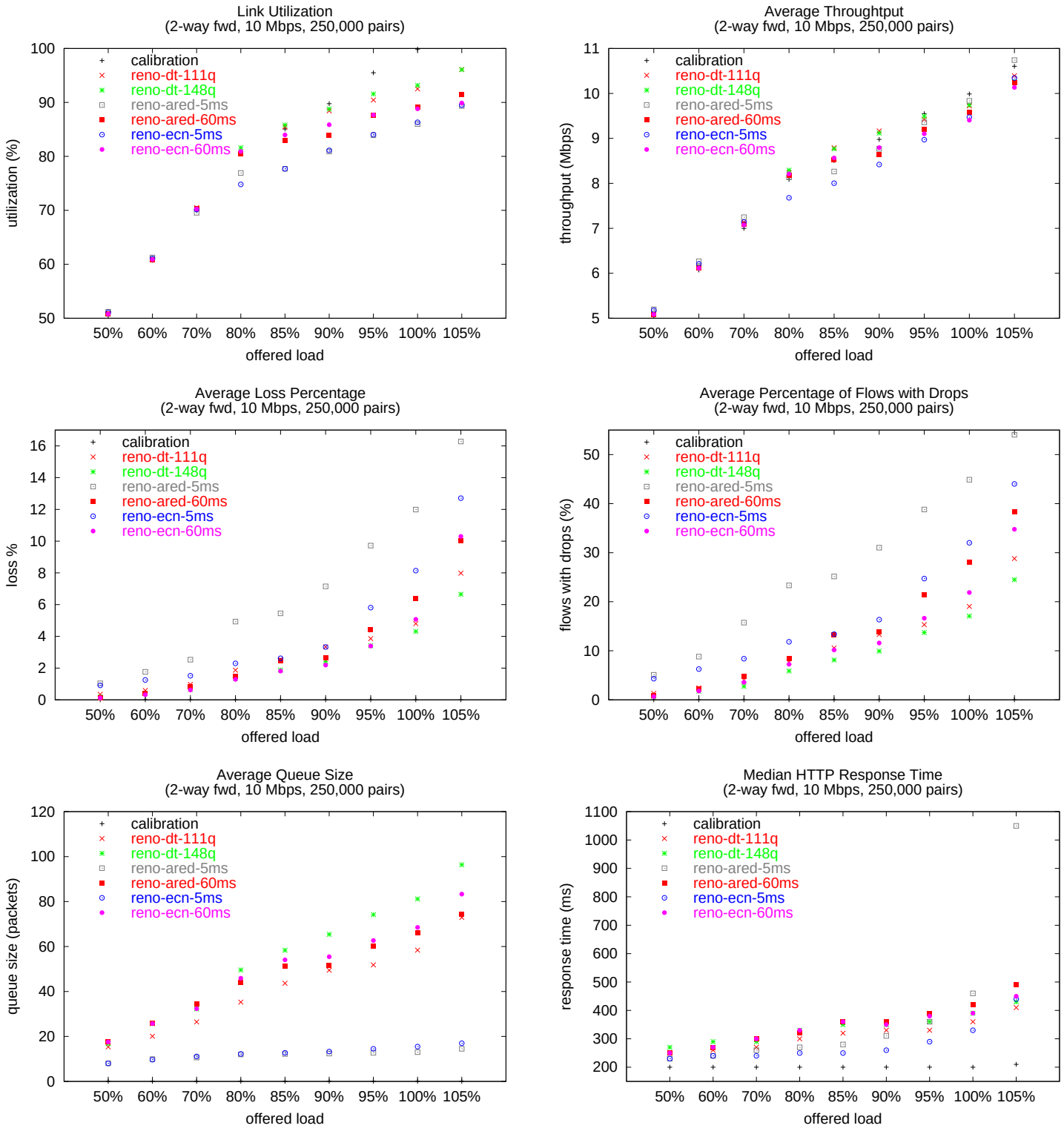


Figure 5: Summary statistics for Reno measured at the congested 10 Mbps link

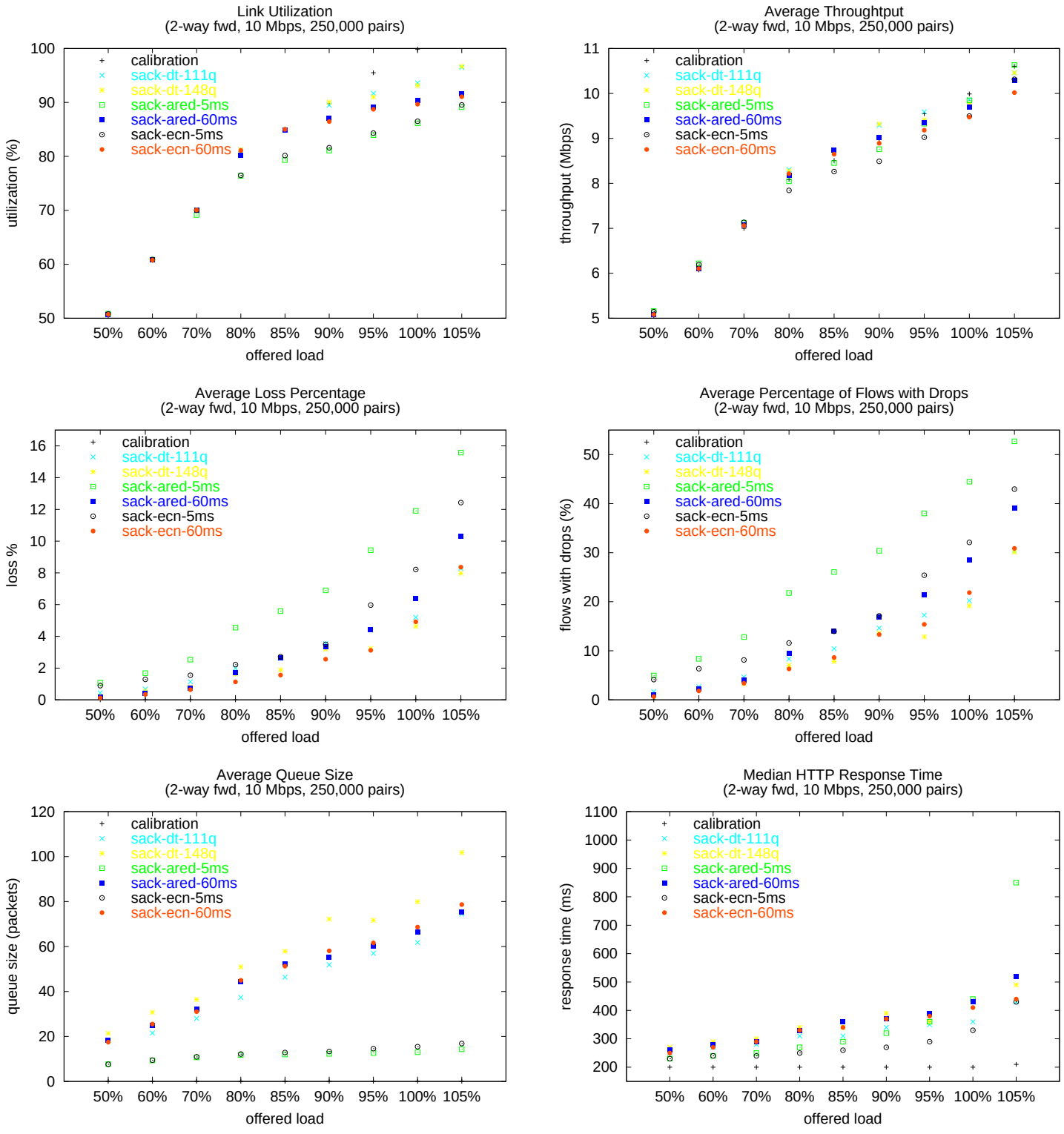


Figure 6: Summary statistics for SACK measured at the congested 10 Mbps link

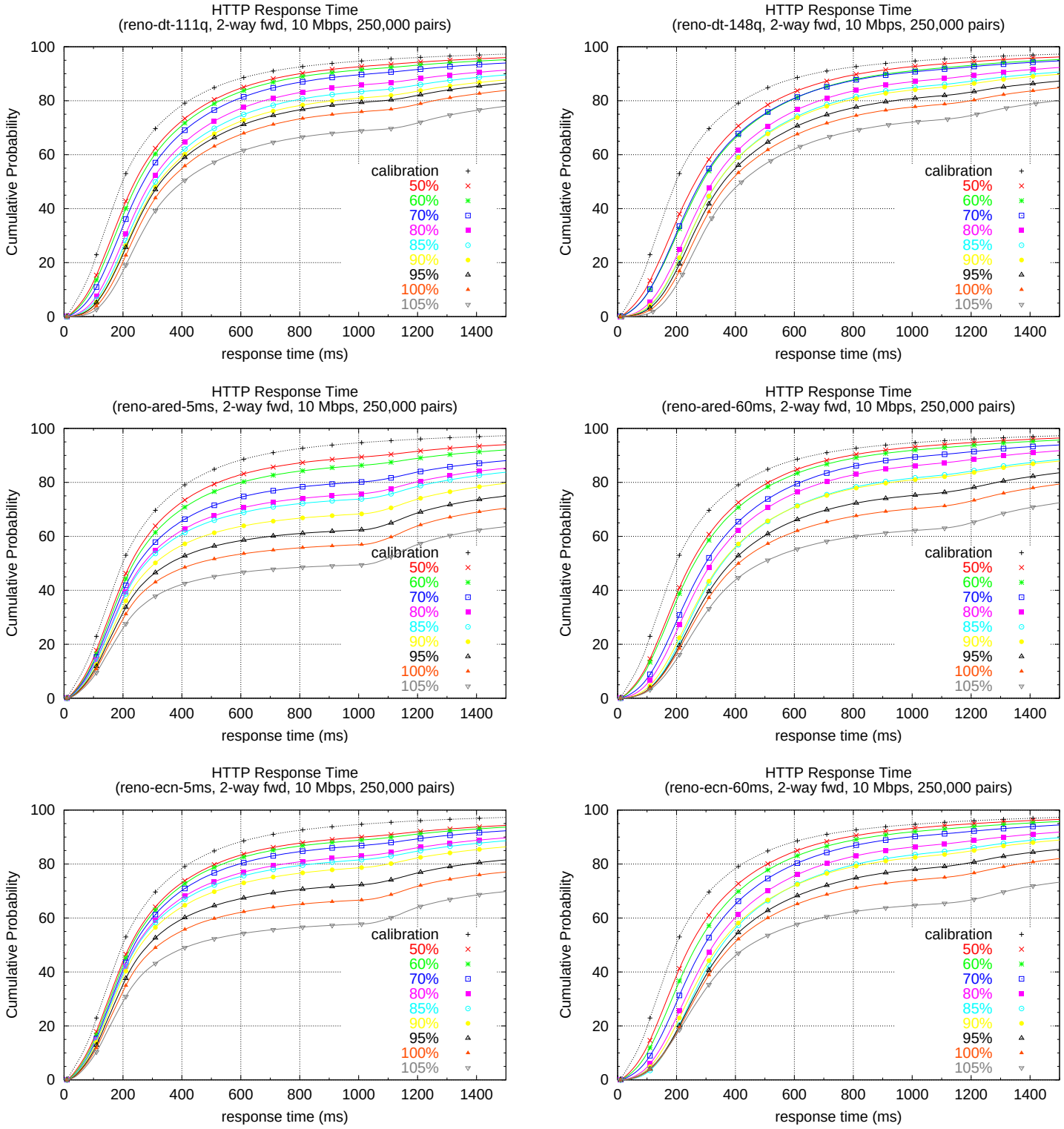


Figure 7: Distribution of HTTP response times for Reno as load increases with varying queue management schemes: Drop-Tail with 111-packet buffer (dt-111q), Drop-Tail with 148-packet buffer (dt-148q), Adaptive RED with 5 ms target delay (ared-5ms), Adaptive RED with 60 ms target delay (ared-60ms), Adaptive RED + ECN with 5 ms target delay (ecn-5ms), Adaptive RED + ECN with 60 ms target delay (ecn-60ms)

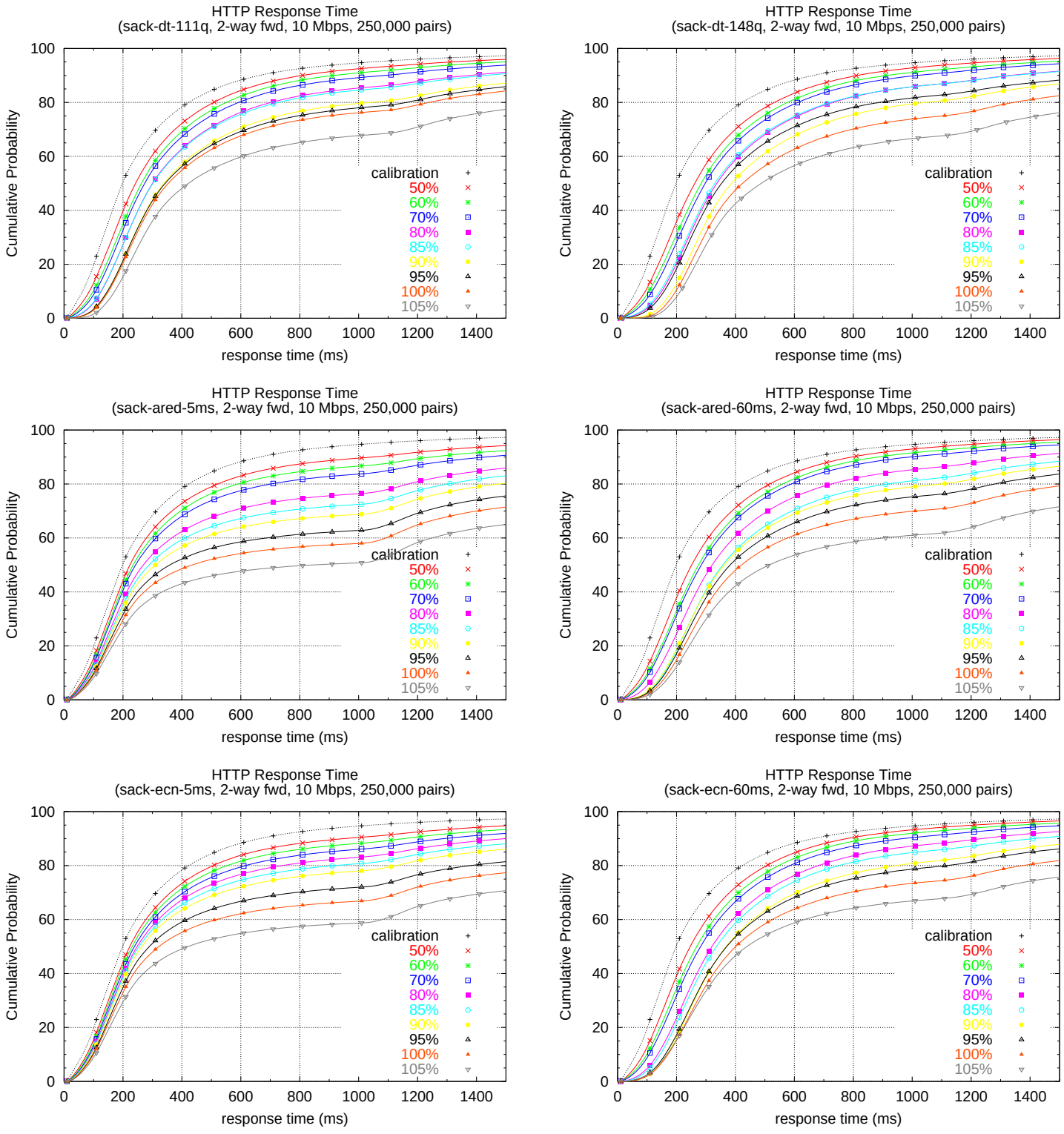


Figure 8: Distribution of HTTP response times for SACK as load increases with varying queue management schemes

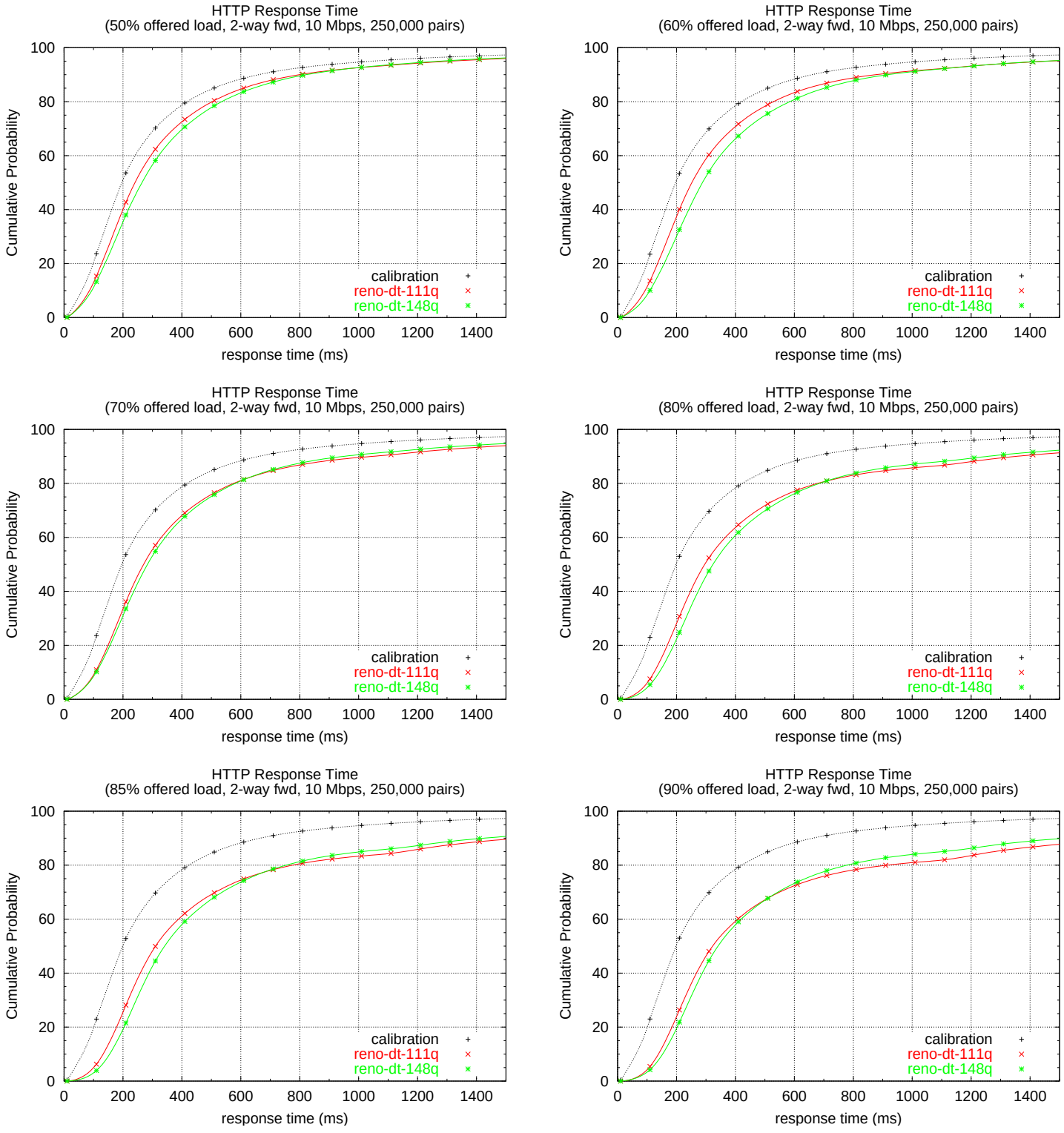
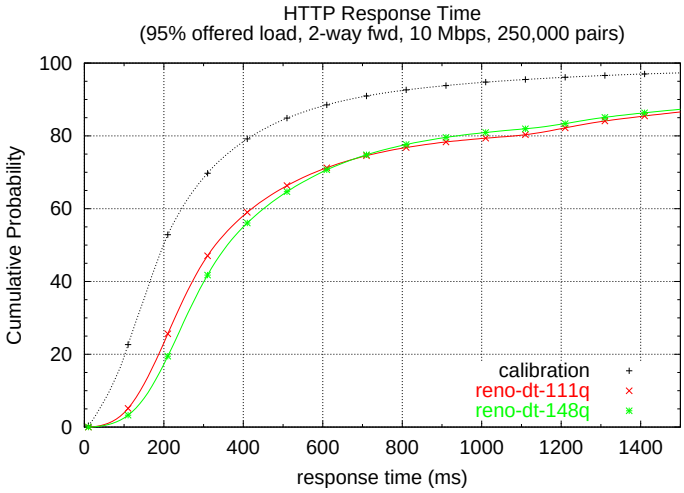
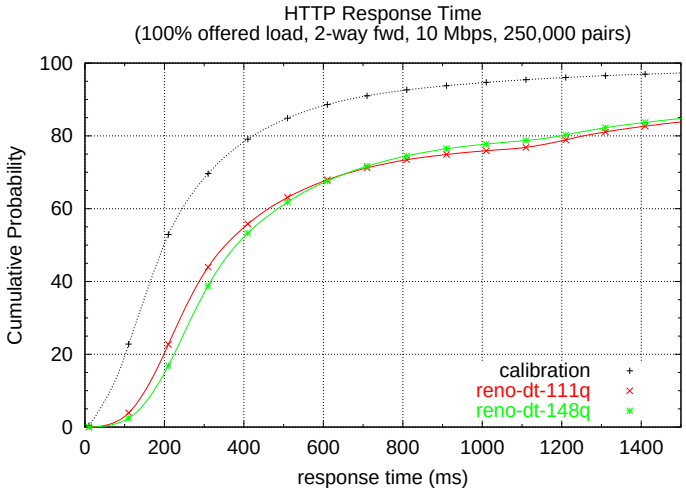


Figure 9: Distribution of HTTP response times for Reno Drop-Tail: Drop-Tail with 111-packet buffer (dt-111q), Drop-Tail with 148-packet buffer (dt-148q)





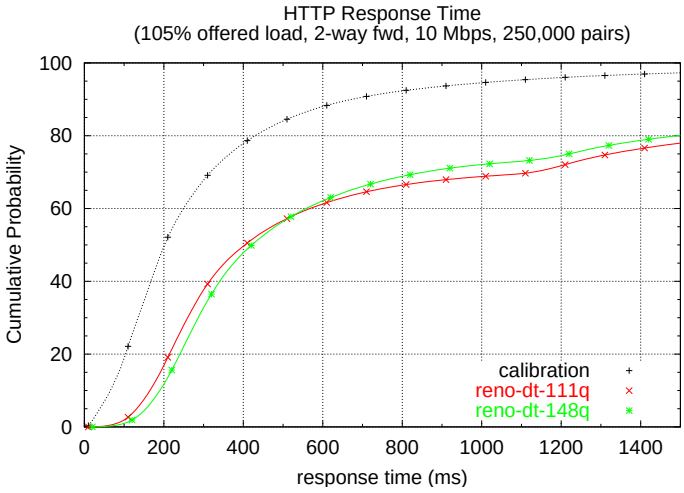


Figure 9: (continued) Distribution of HTTP response times for Reno Drop-Tail

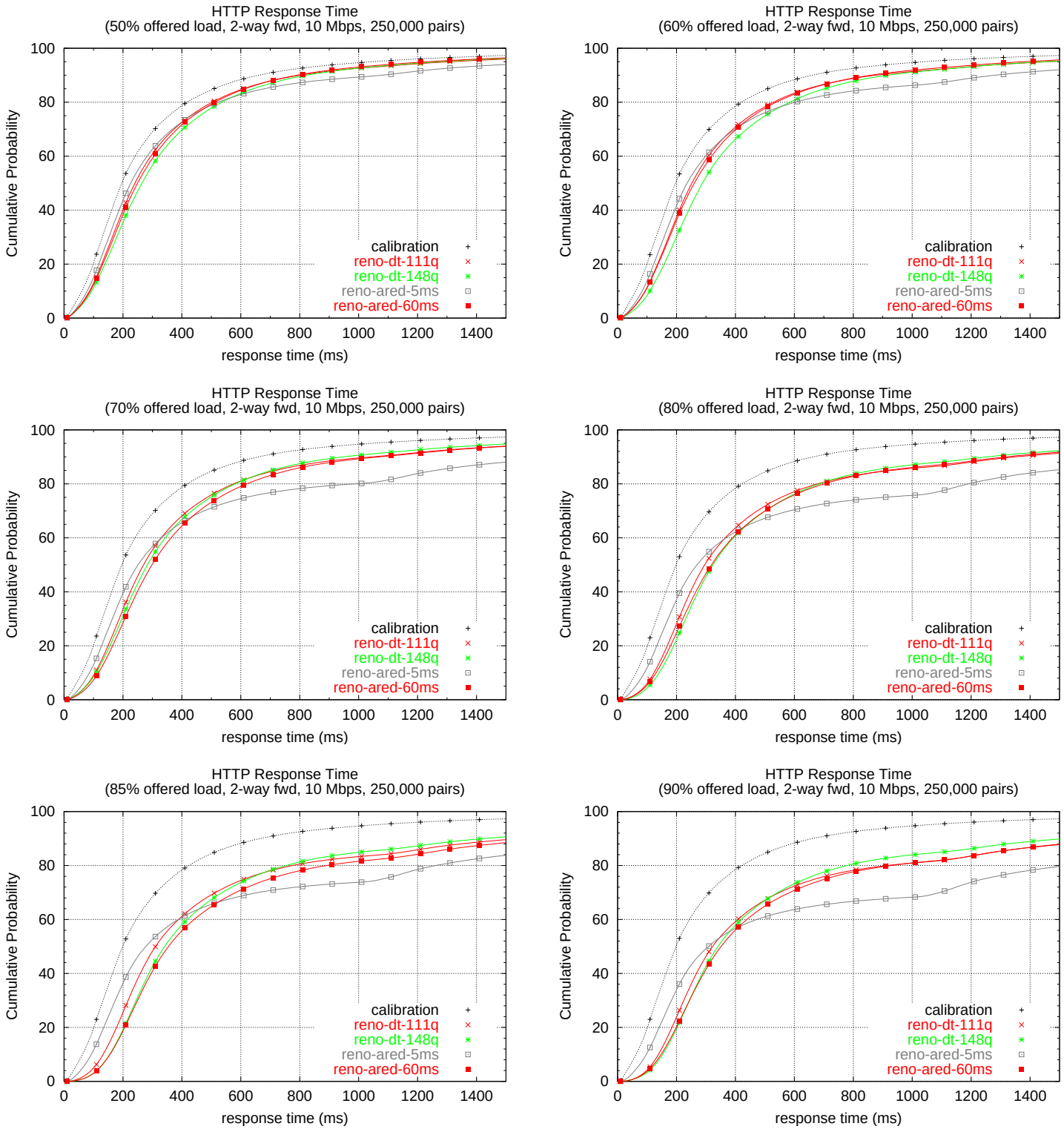
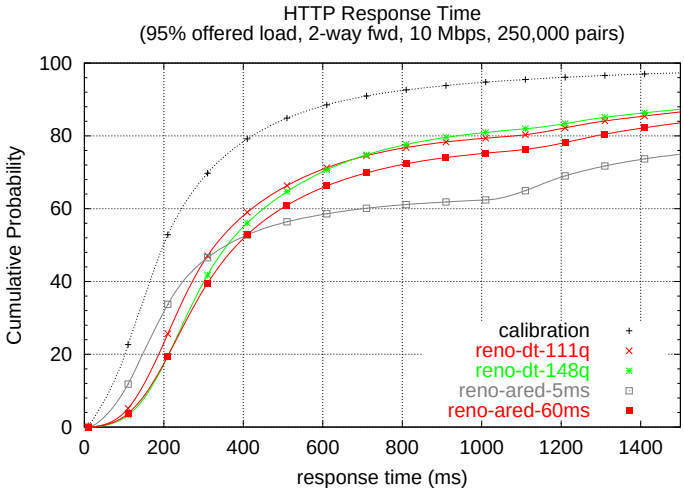
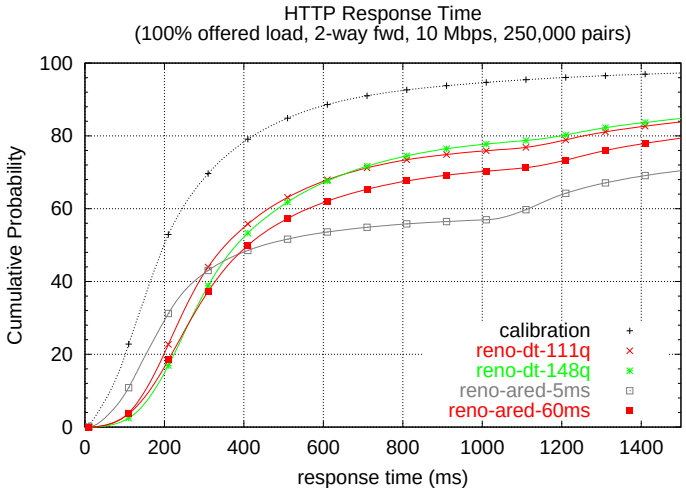


Figure 10: Distribution of HTTP response times for Reno Drop-Tail and Adaptive RED: Drop-Tail with 111-packet buffer (dt-111q), Drop-Tail with 148-packet buffer (dt-148q), Adaptive RED with 5 ms target delay (ared-5ms), Adaptive RED with 60 ms target delay (ared-60ms)





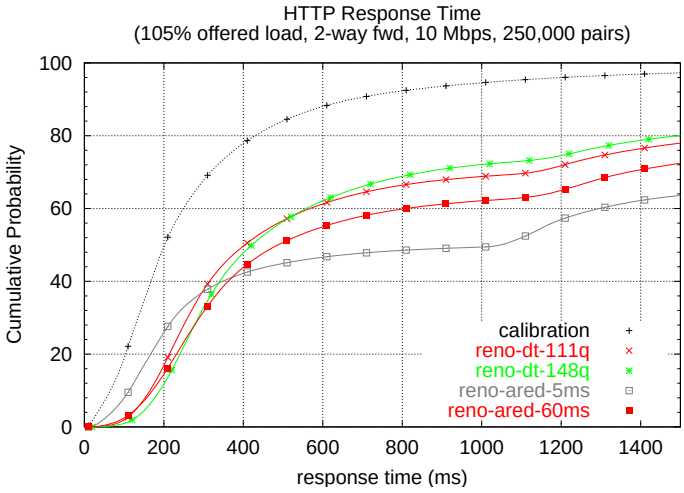


Figure 10: (continued) Distribution of HTTP response times for Reno Drop-Tail and Adaptive RED

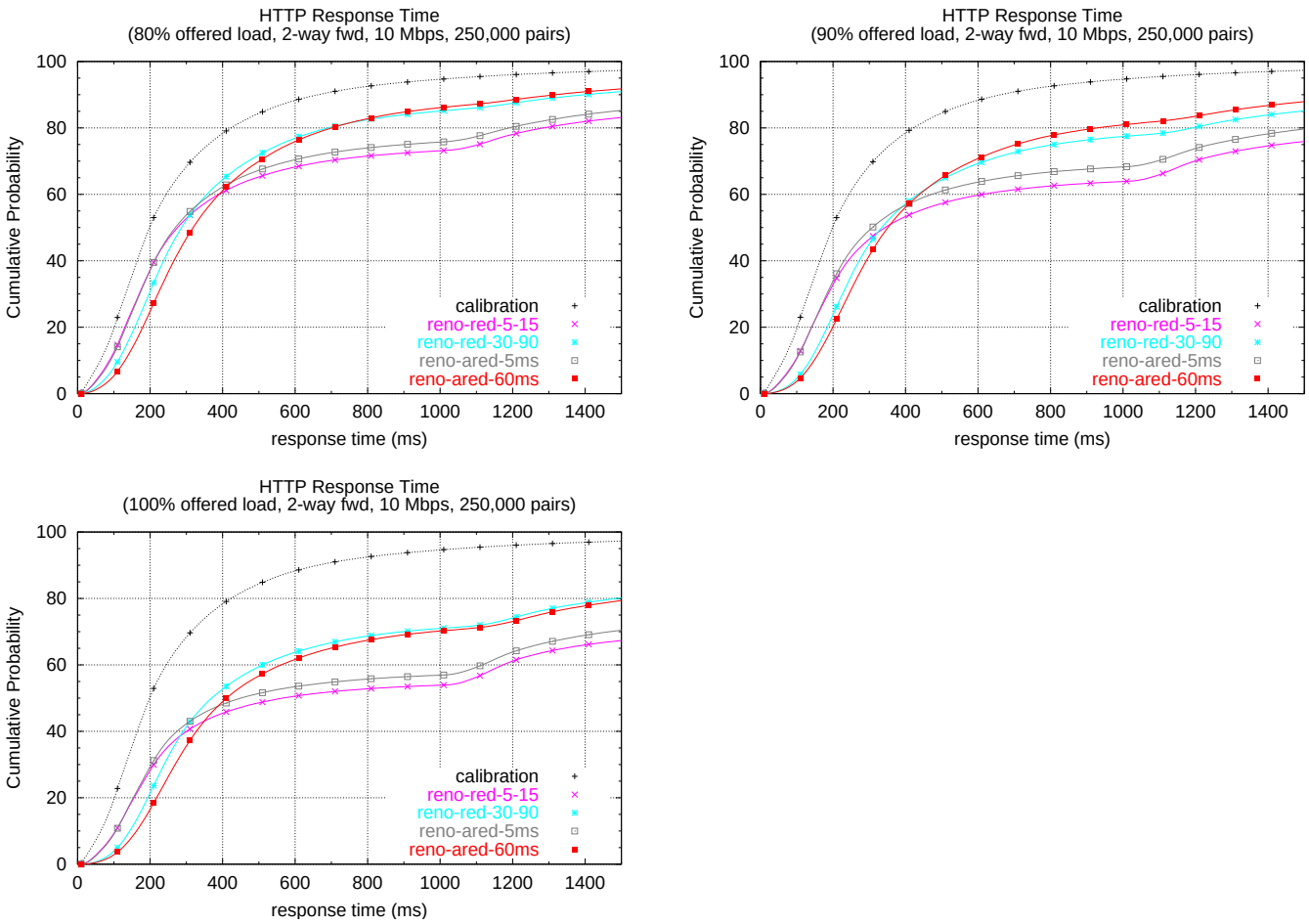


Figure 11: Distribution of HTTP response times for RED and Adaptive RED: RED with 5-packet minimum threshold and 15-packet maximum threshold (red-5-15), RED with 30-packet minimum threshold and 90-packet maximum threshold (red-30-90), Adaptive RED with 5 ms target delay (ared-5ms), Adaptive RED with 60 ms target delay (ared-60ms)

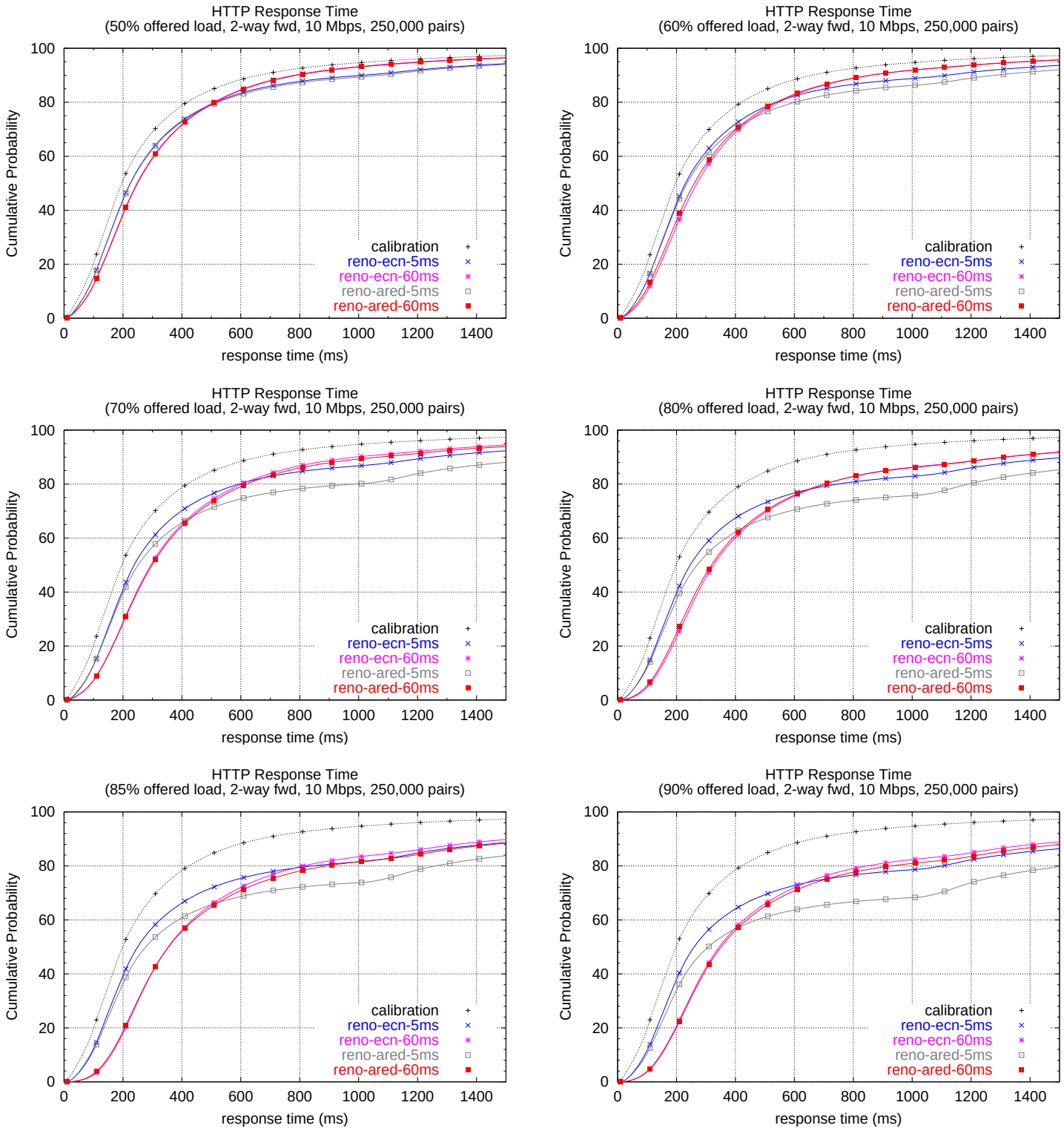
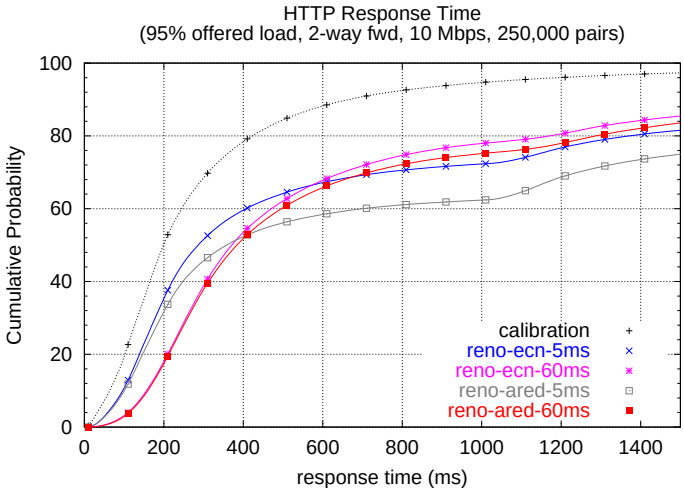
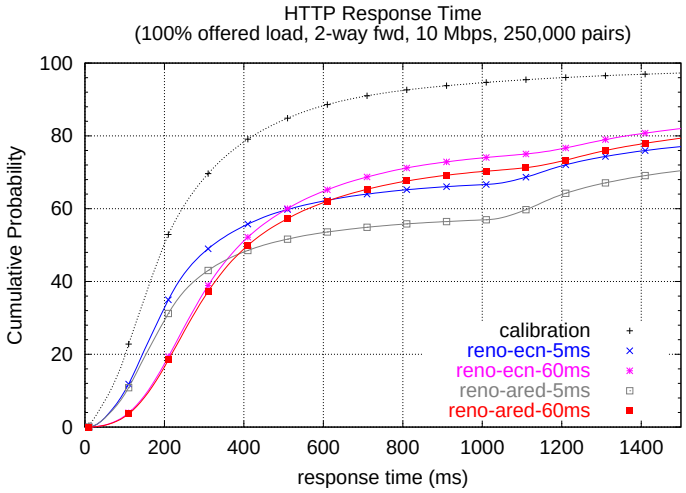


Figure 12: Distribution of HTTP response times for Reno Adaptive RED and Adaptive RED+ECN: Adaptive RED with 5 ms target delay (ared-5ms), Adaptive RED with 60 ms target delay (ared-60ms), Adaptive RED + ECN with 5 ms target delay (ecn-5ms), Adaptive RED + ECN with 60 ms target delay (ecn-60ms)





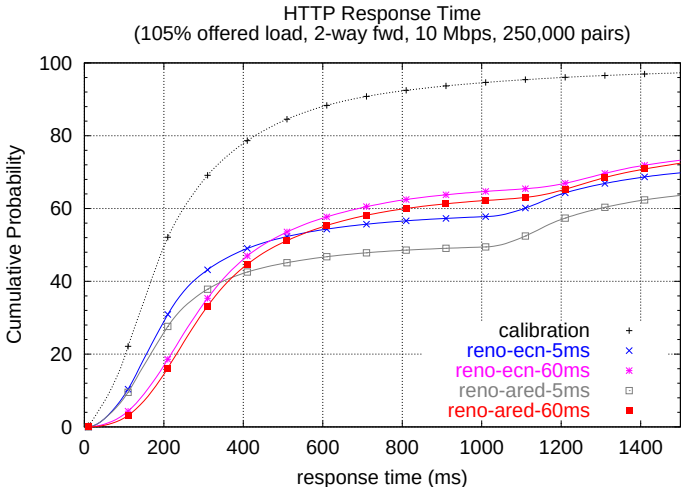


Figure 12: (continued) Distribution of HTTP response times for Reno Adaptive RED and Adaptive RED+ECN

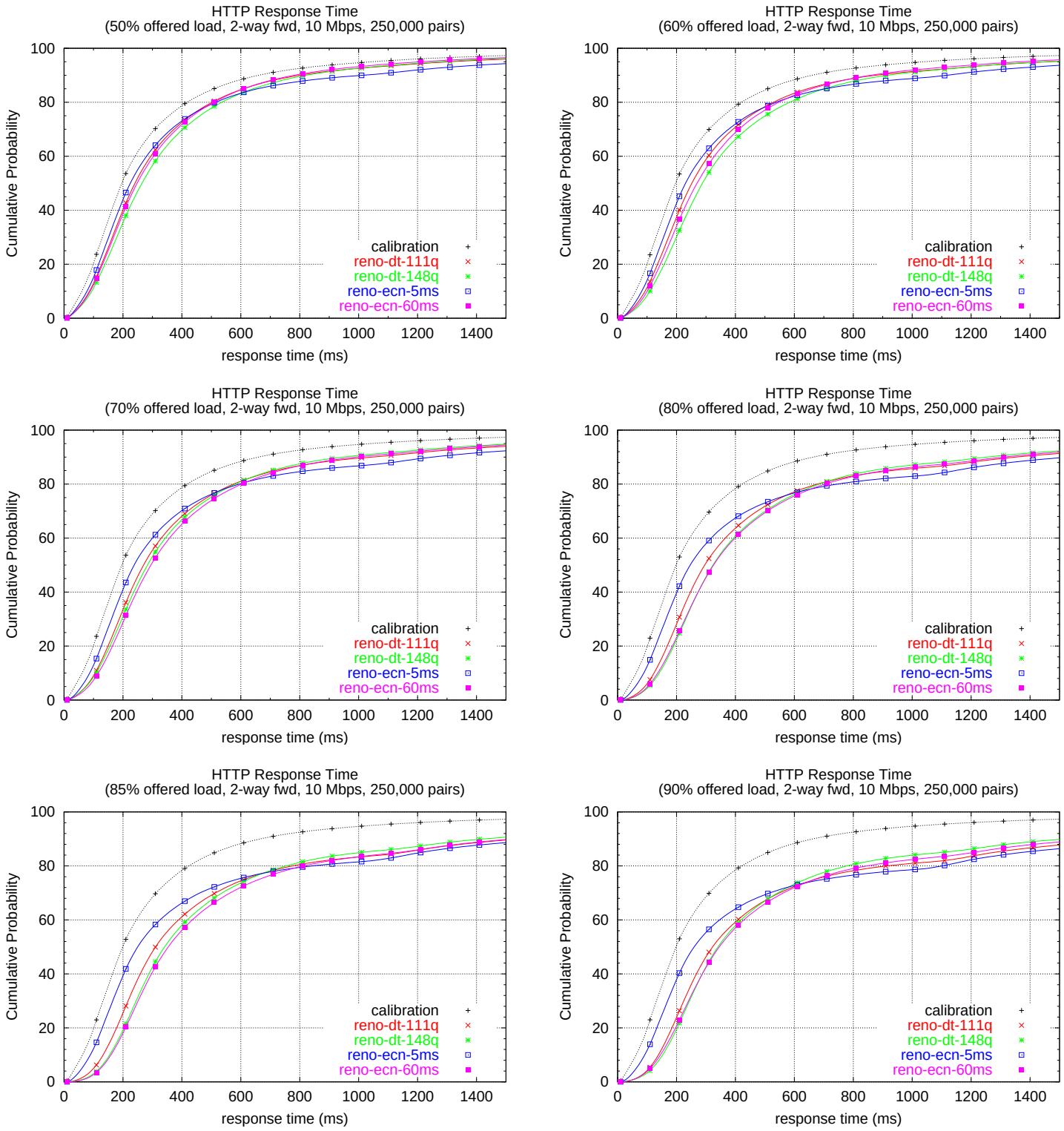
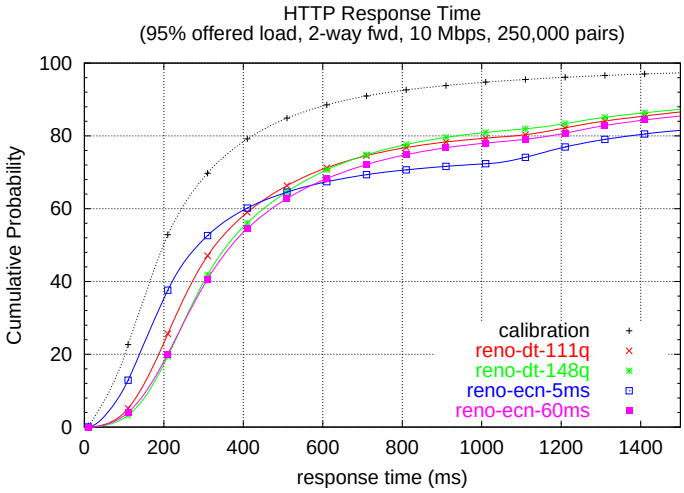
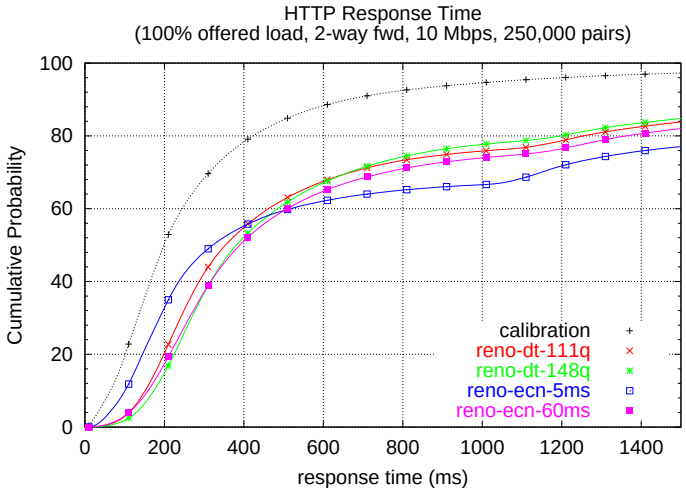


Figure 13: Distribution of HTTP response times for Reno Drop-Tail and Adaptive RED+ECN: Drop-Tail with 111-packet buffer (dt-111q), Drop-Tail with 148-packet buffer (dt-148q), Adaptive RED + ECN with 5 ms target delay (ecn-5ms), Adaptive RED + ECN with 60 ms target delay (ecn-60ms)





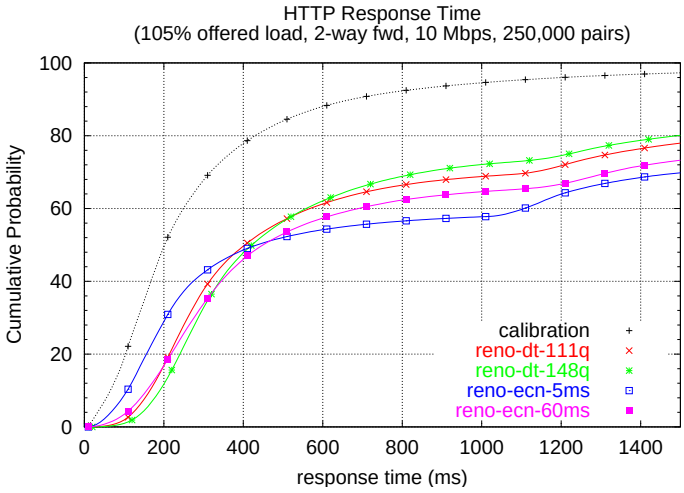


Figure 13: (continued) Distribution of HTTP response times for Reno Drop-Tail and Adaptive RED+ECN

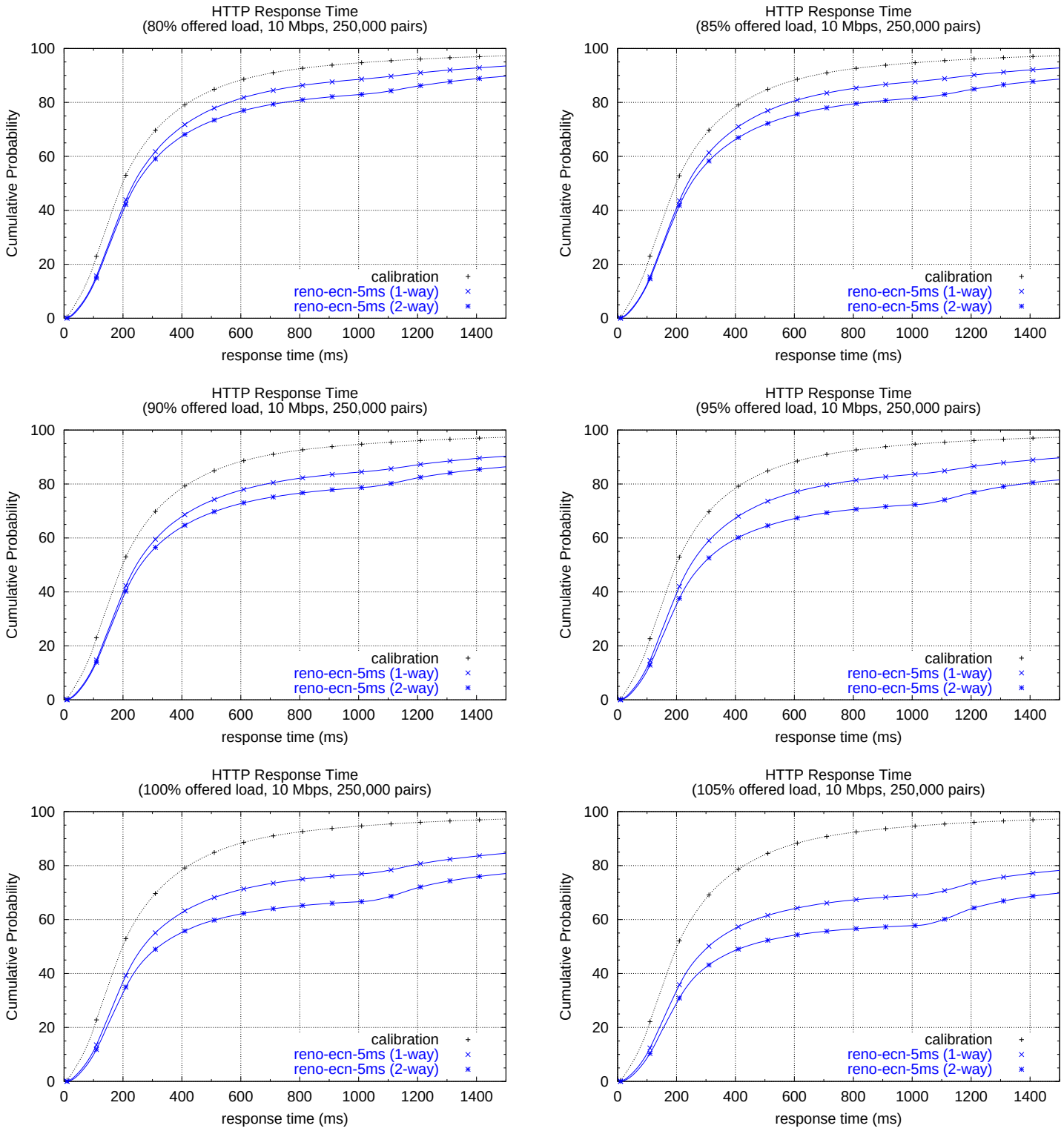


Figure 14: Distribution of HTTP response times for Reno Adaptive RED + ECN with 5 ms target delay for 1-way and 2-way traffic

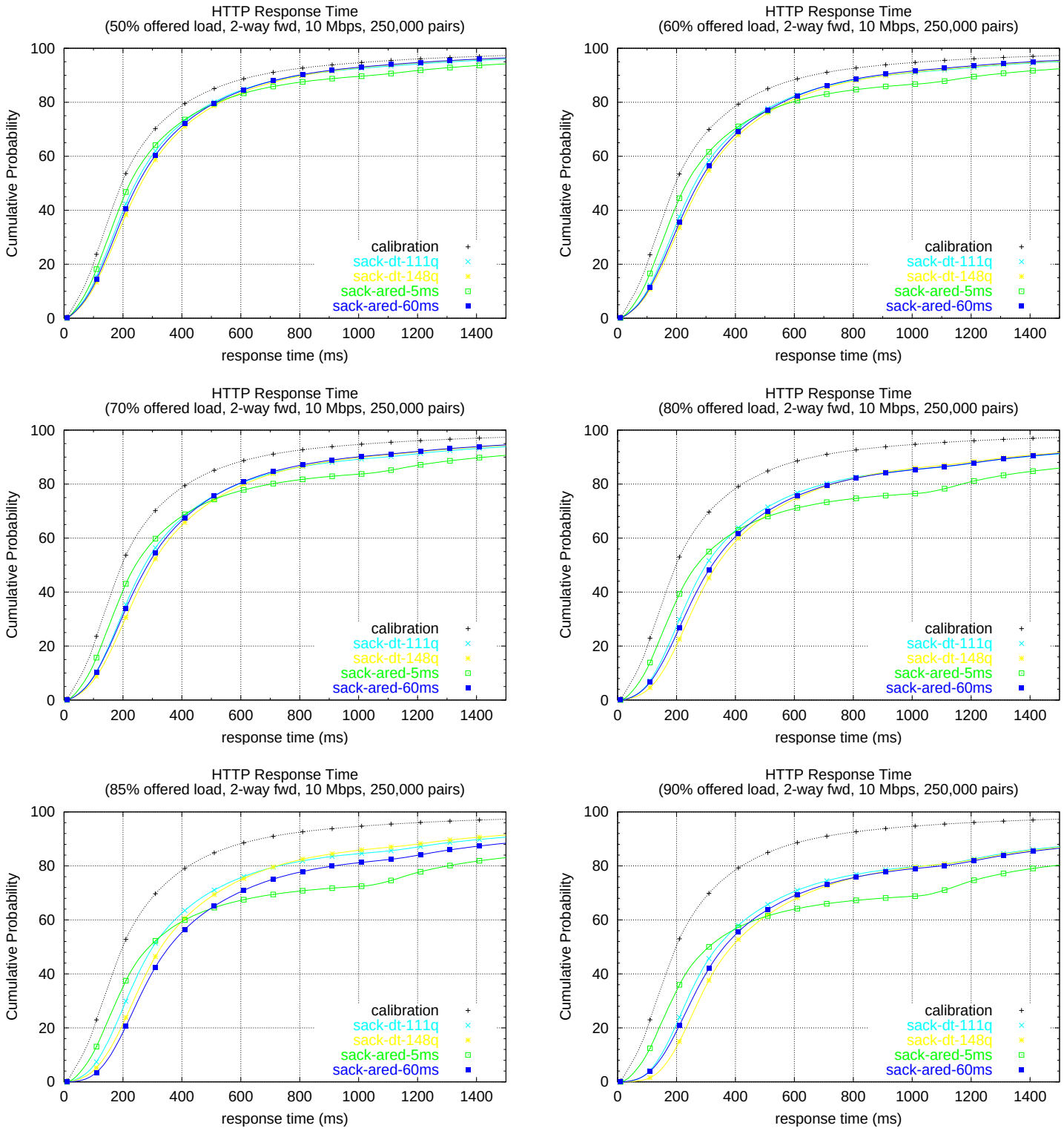
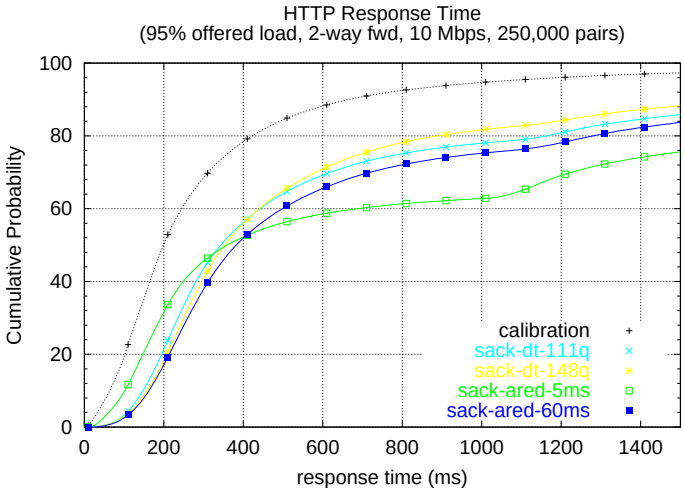
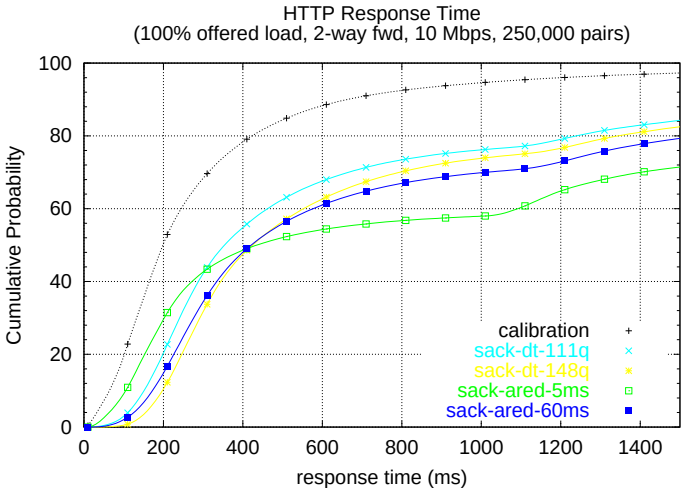


Figure 15: Distribution of HTTP response times for SACK Drop-Tail and Adaptive RED: Drop-Tail with 111-packet buffer (dt-111q), Drop-Tail with 148-packet buffer (dt-148q), Adaptive RED with 5 ms target delay (ared-5ms), Adaptive RED with 60 ms target delay (ared-60ms)





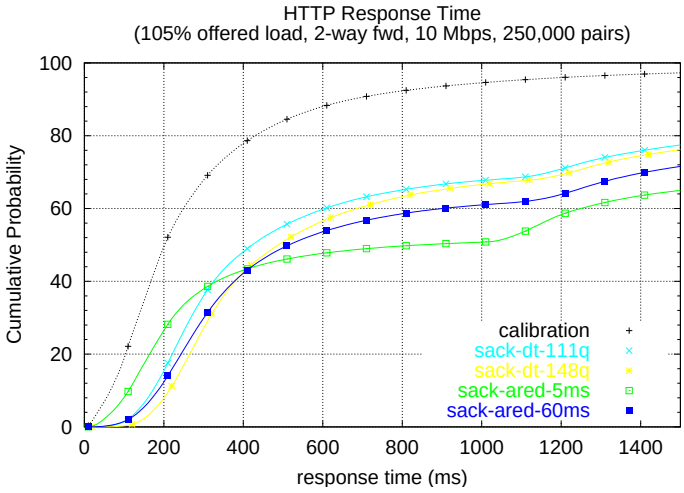


Figure 15: (continued) Distribution of HTTP response times for SACK Drop-Tail and Adaptive RED

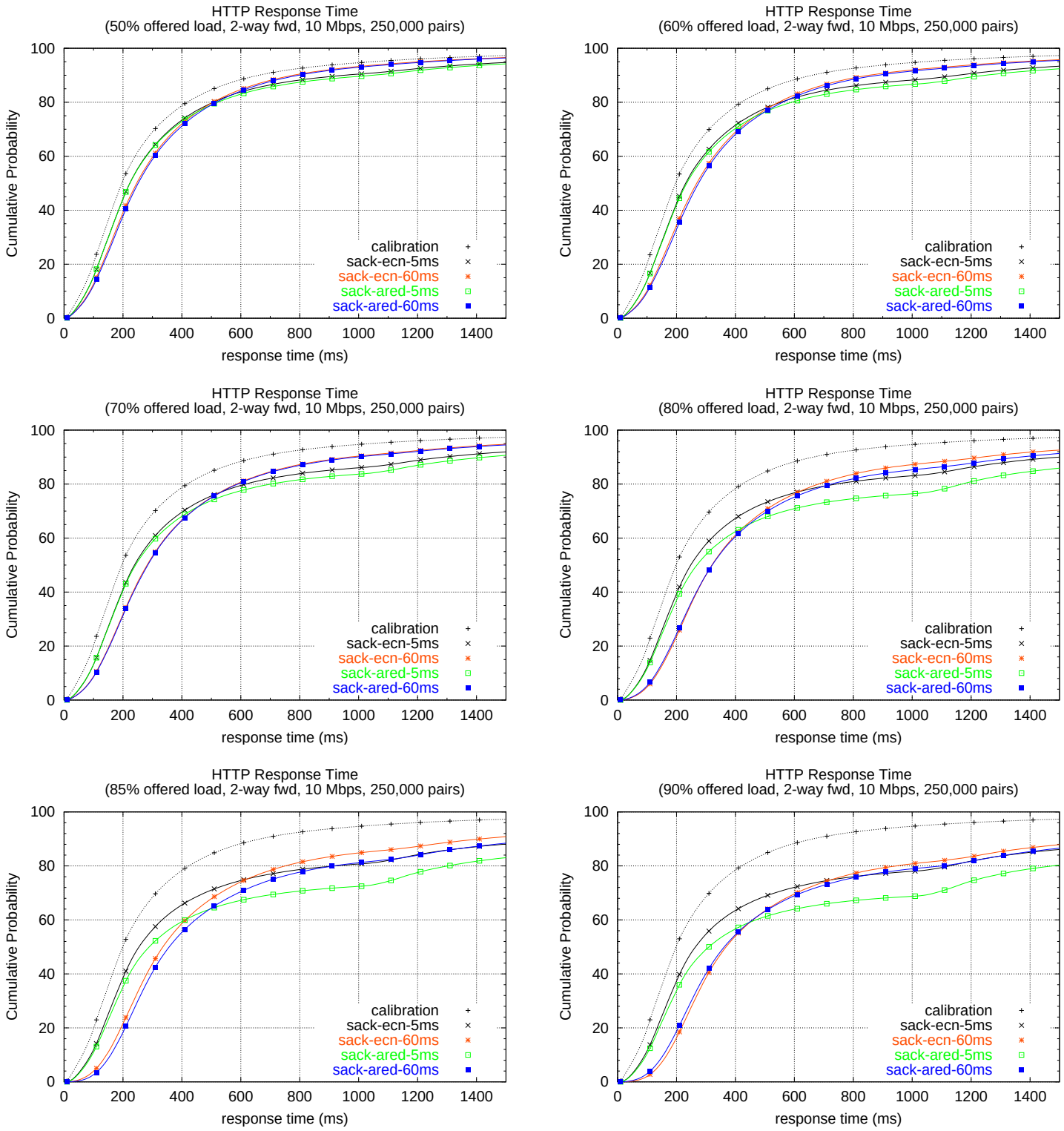
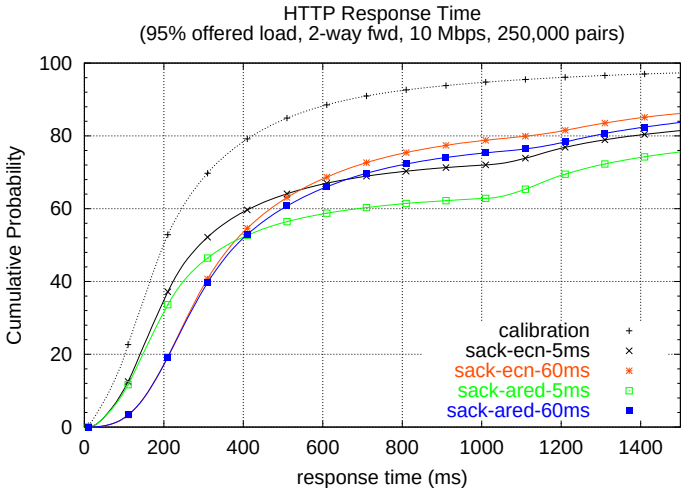
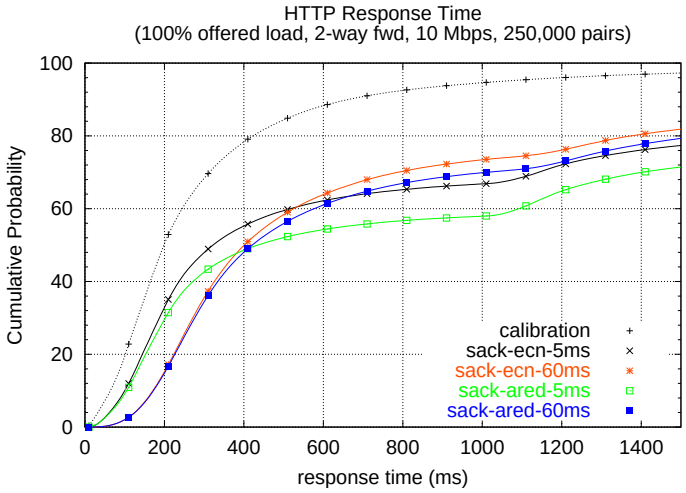


Figure 16: Distribution of HTTP response times for SACK Adaptive RED and Adaptive RED+ECN: Adaptive RED with 5 ms target delay (ared-5ms), Adaptive RED with 60 ms target delay (ared-60ms), Adaptive RED + ECN with 5 ms target delay (ecn-5ms), Adaptive RED + ECN with 60 ms target delay (ecn-60ms)





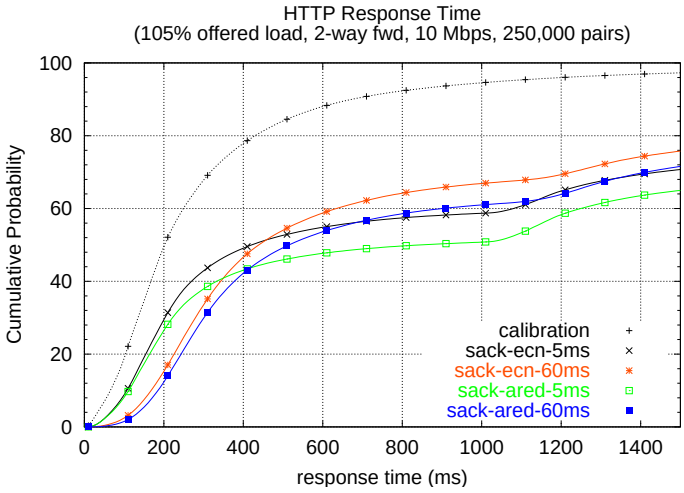


Figure 16: (continued) Distribution of HTTP response times for SACK Adaptive RED and Adaptive RED+ECN

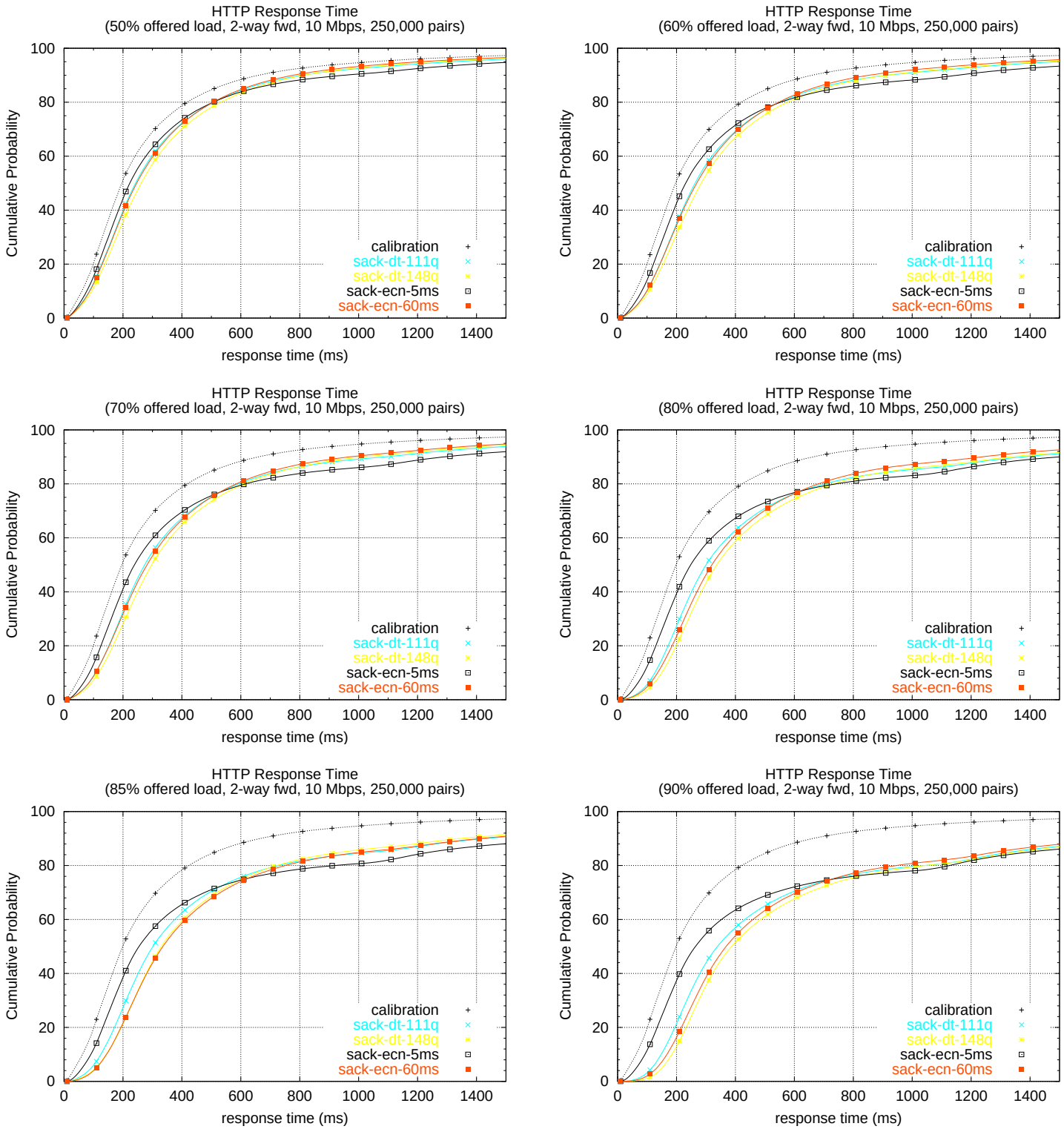
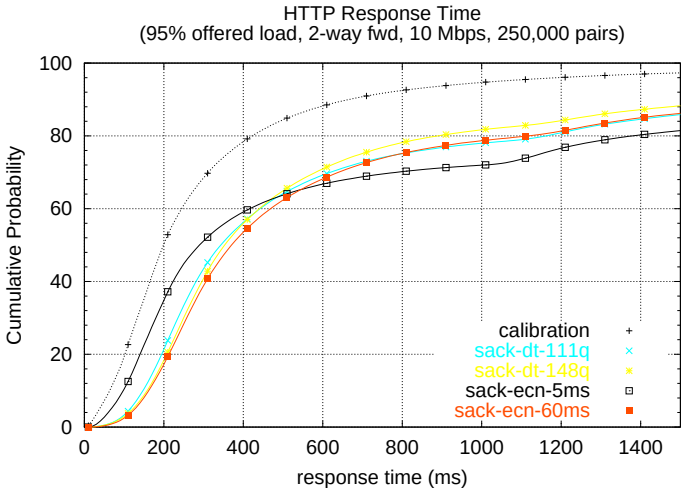
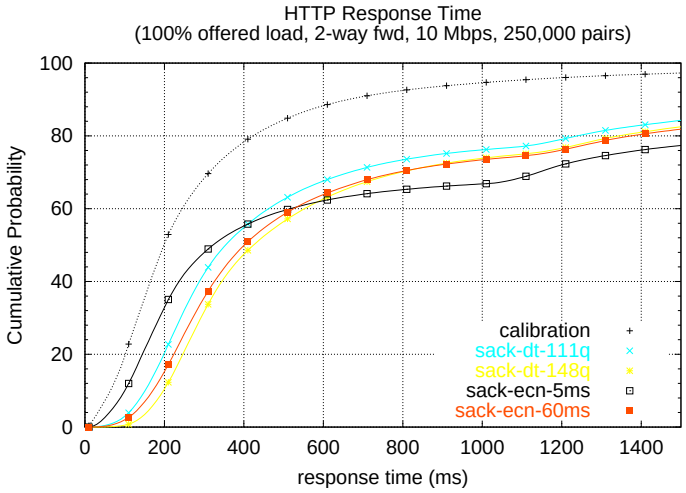


Figure 17: Distribution of HTTP response times for SACK Drop-Tail and Adaptive RED+ECN: Drop-Tail with 111-packet buffer (dt-111q), Drop-Tail with 148-packet buffer (dt-148q), Adaptive RED + ECN with 5 ms target delay (ecn-5ms), Adaptive RED + ECN with 60 ms target delay (ecn-60ms)





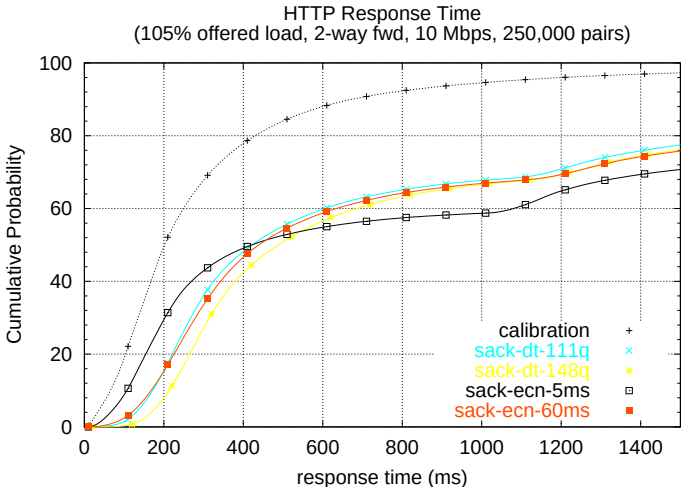


Figure 17: (continued) Distribution of HTTP response times for SACK Drop-Tail and Adaptive RED+ECN

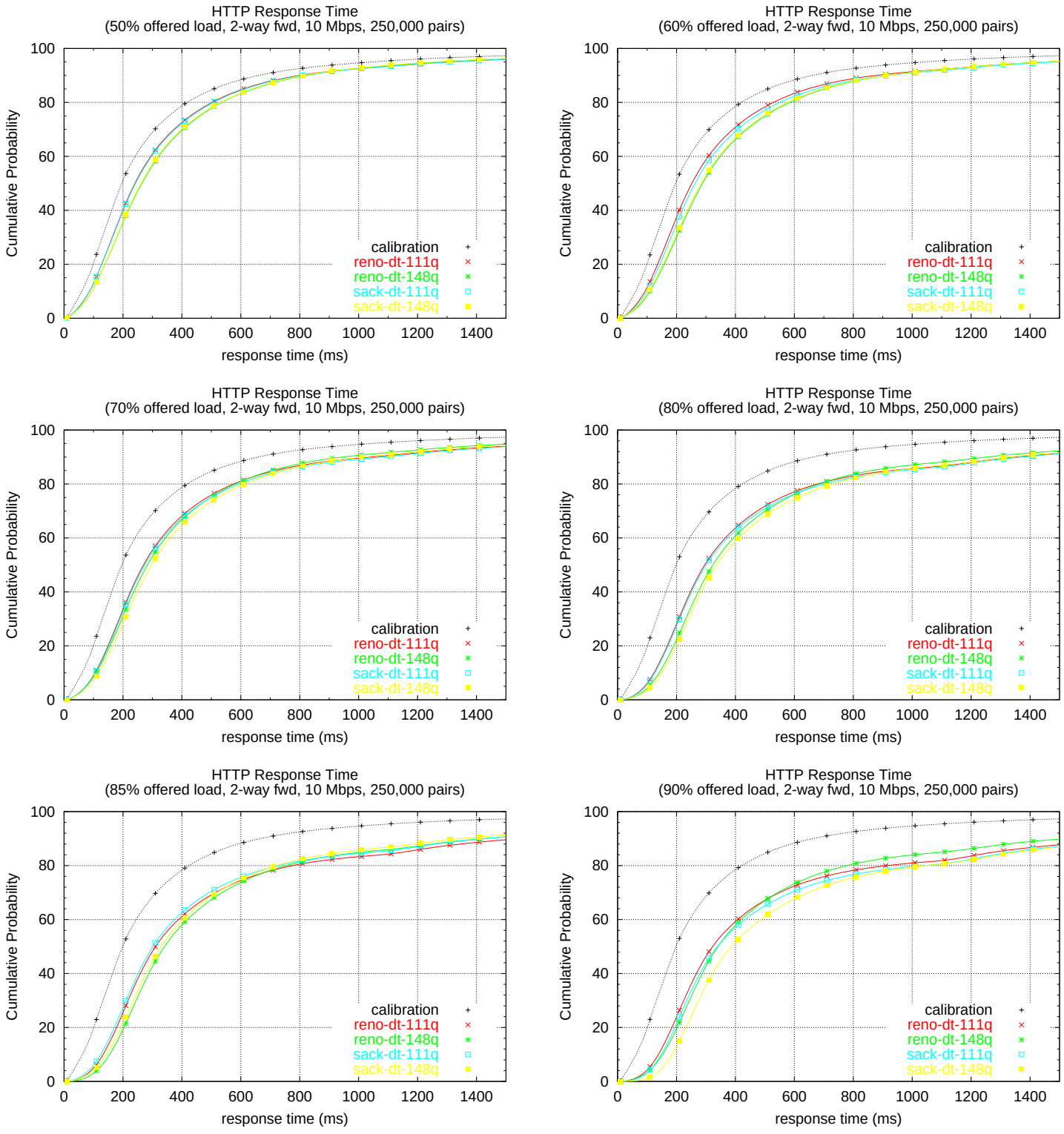
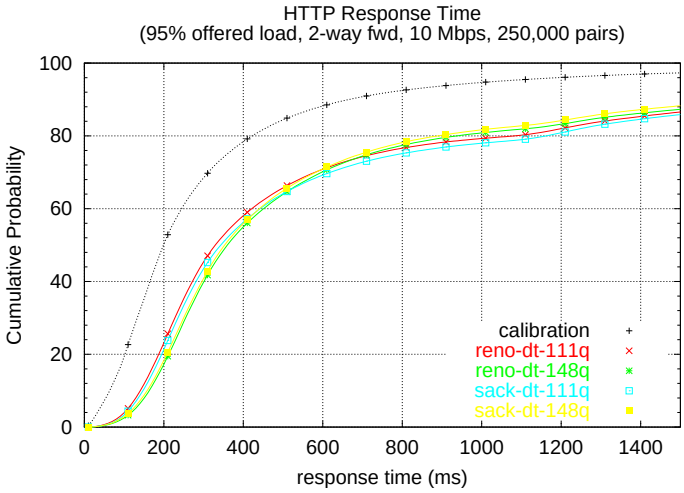
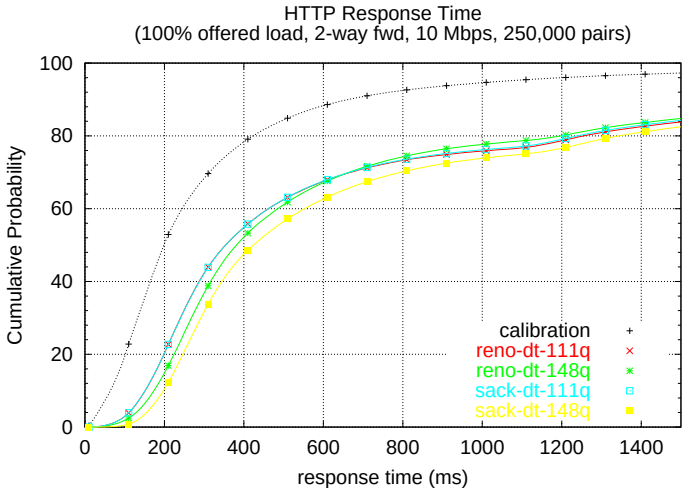


Figure 18: Distribution of HTTP response times for Drop-Tail: 111-packet buffer (dt-111q), 148-packet buffer (dt-148q)





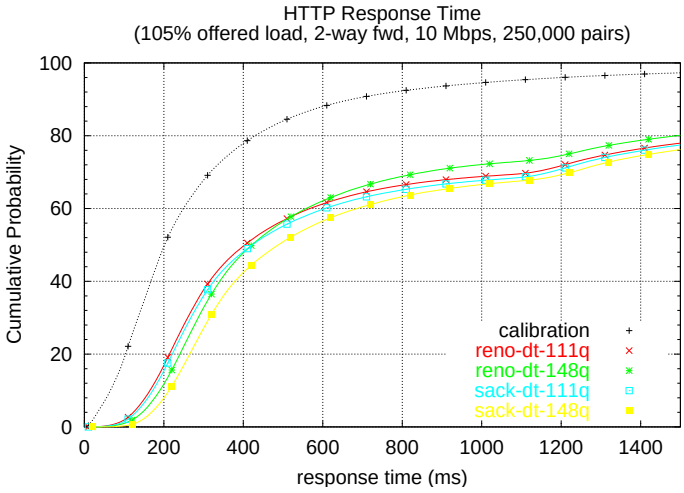


Figure 18: (continued) Distribution of HTTP response times for Drop-Tail

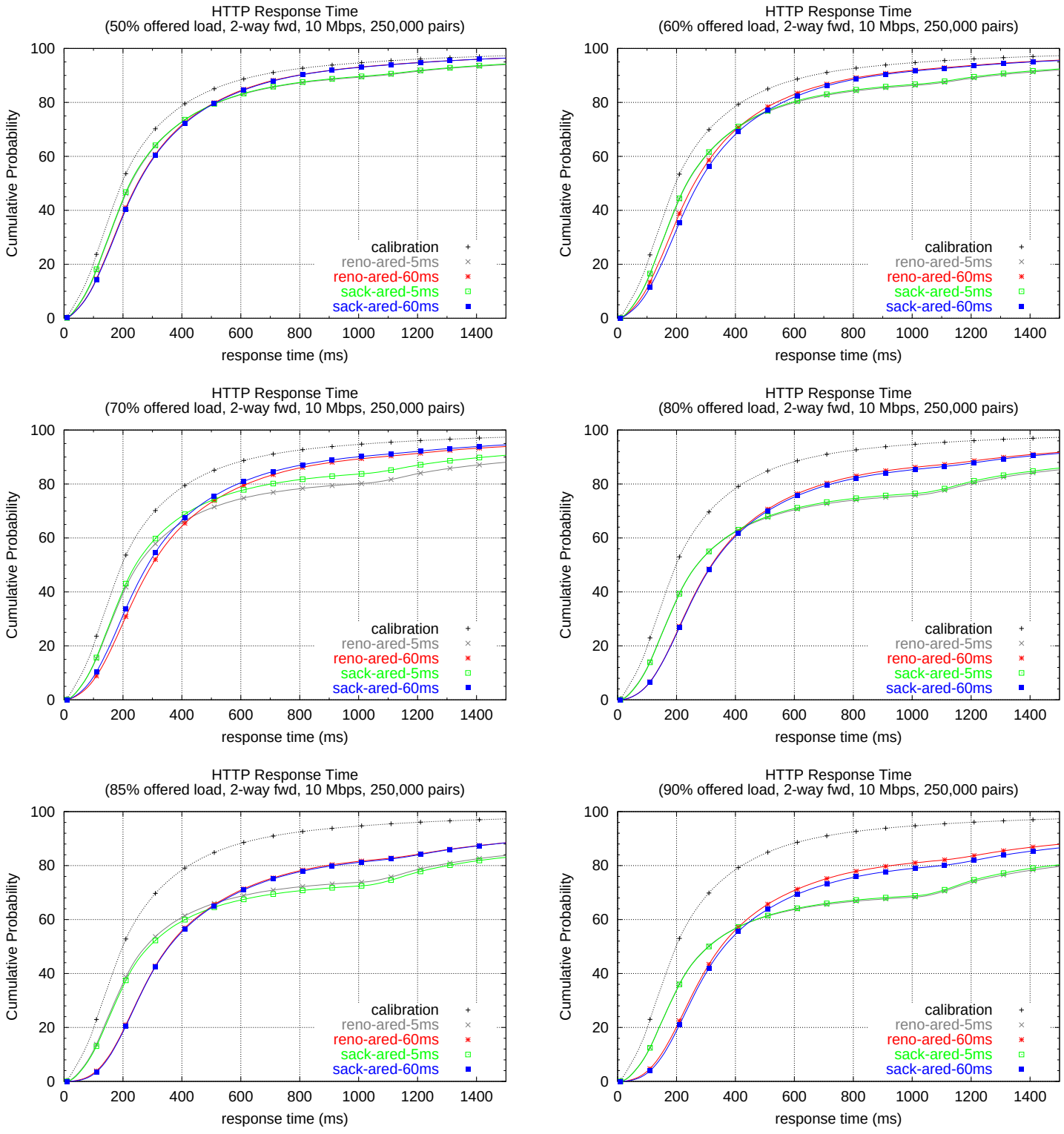
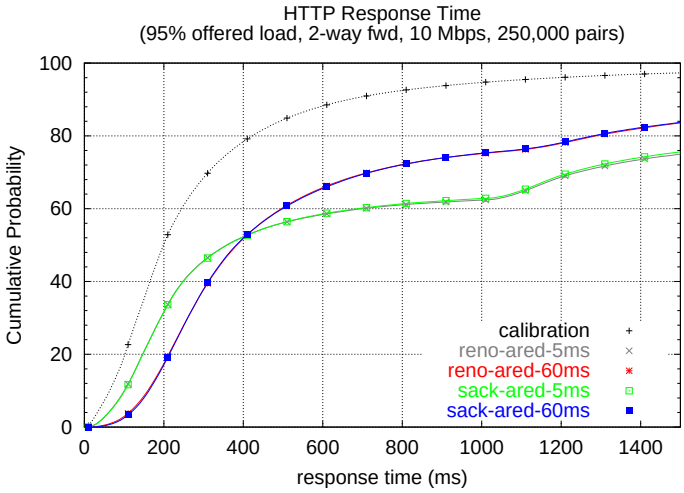
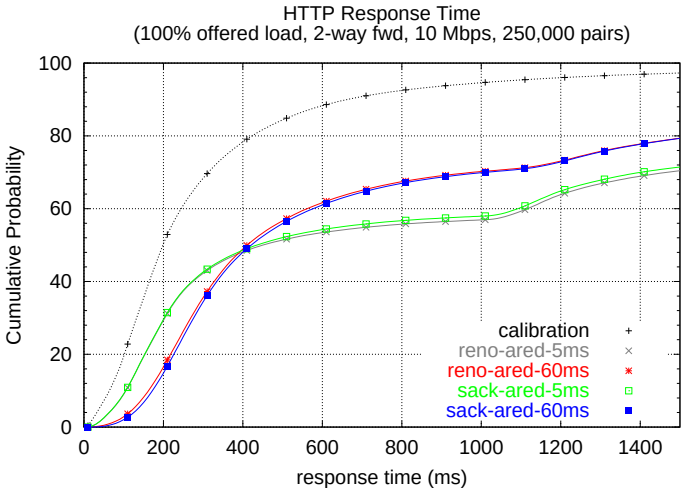


Figure 19: Distribution of HTTP response times for Adaptive RED: 5 ms target delay (ared-5ms), 60 ms target delay (ared-60ms)





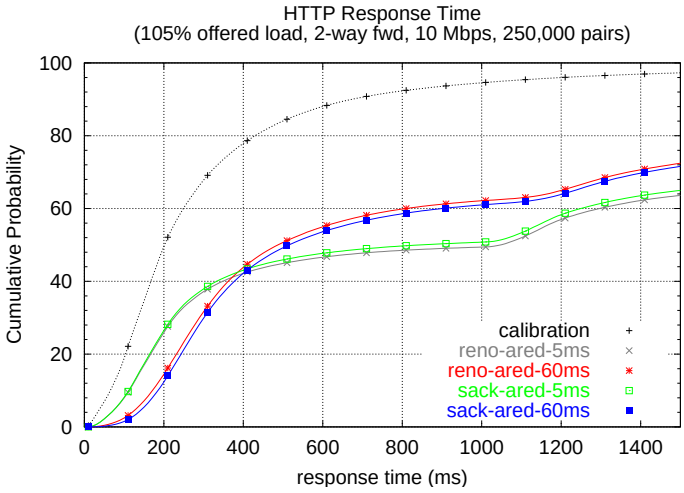


Figure 19: (continued) Distribution of HTTP response times for Adaptive RED

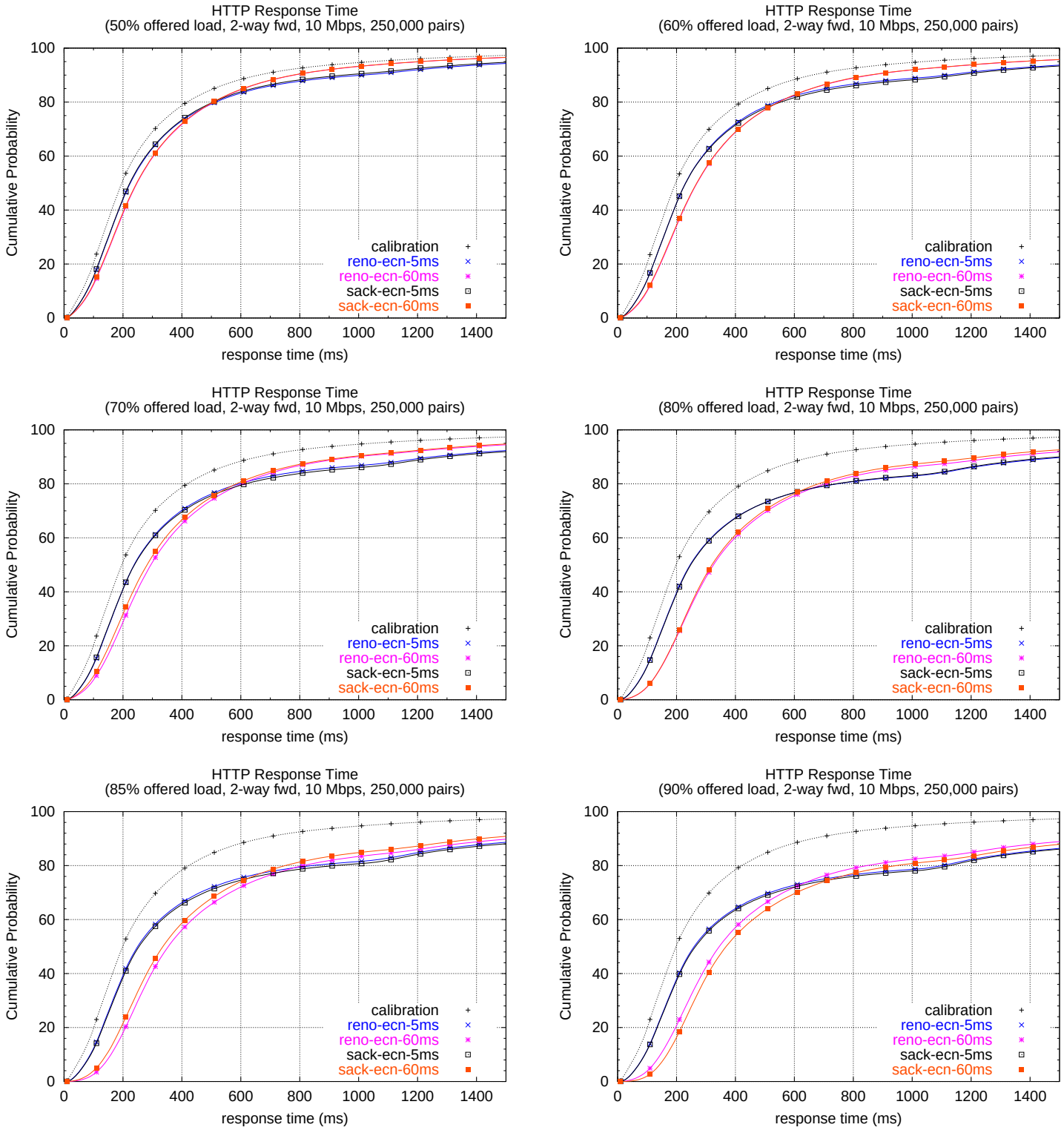
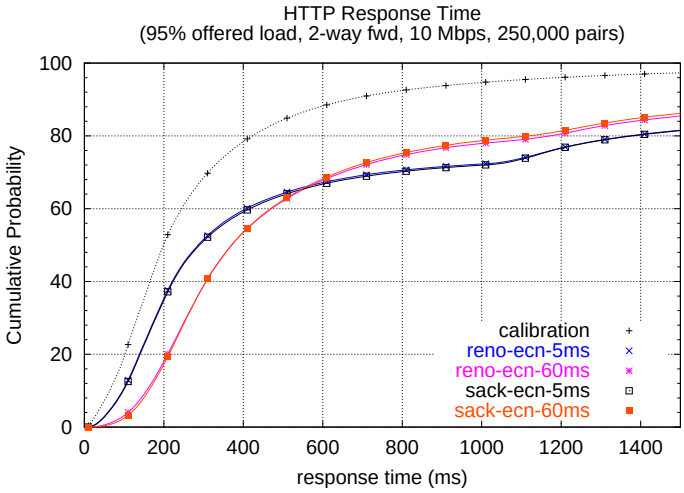
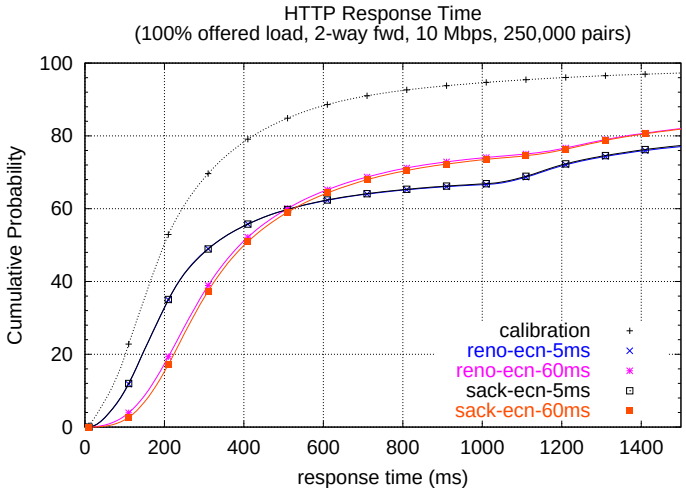


Figure 20: Distribution of HTTP response times for Adaptive RED+ECN: 5 ms target delay (ecn-5ms), 60 ms target delay (ecn-60ms)





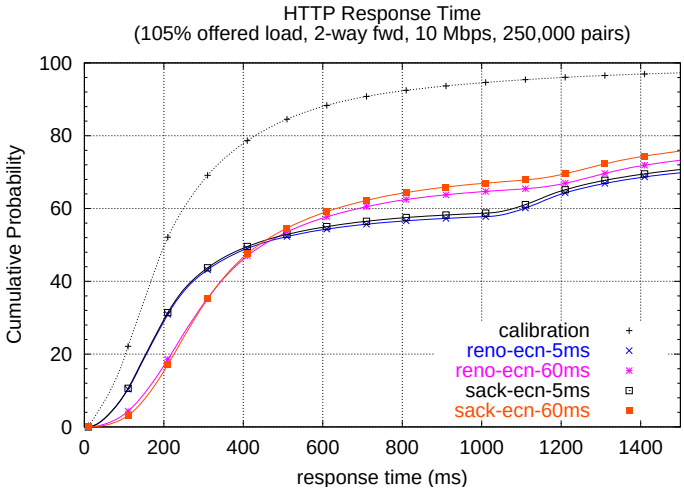


Figure 20: (continued) Distribution of HTTP response times for Adaptive RED+ECN

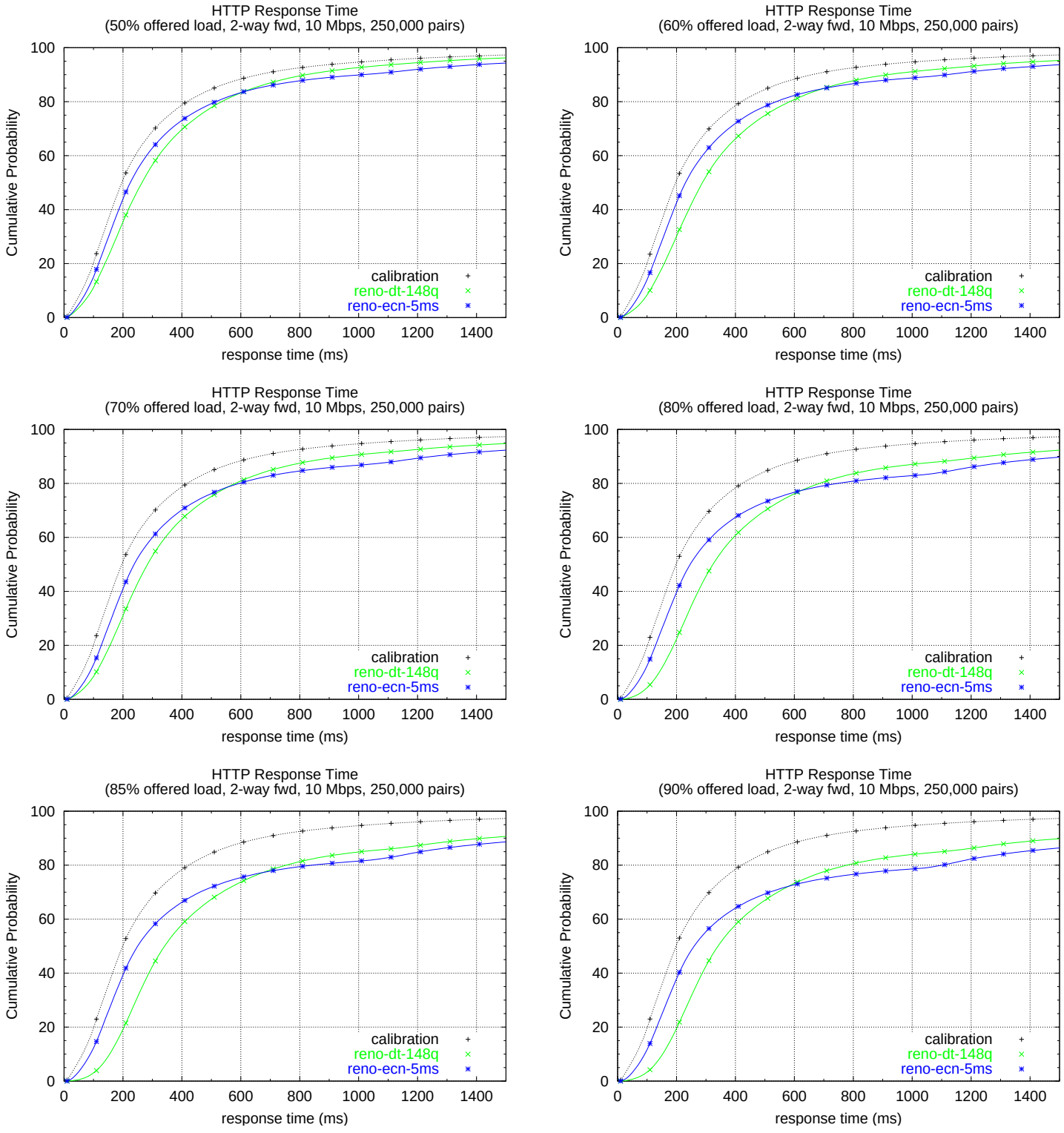
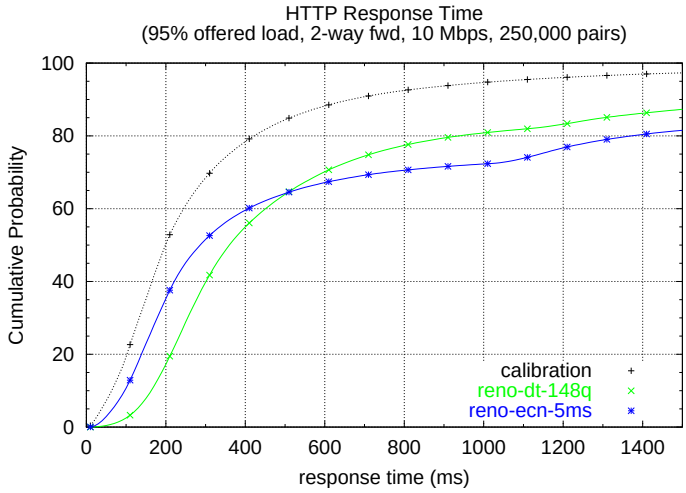
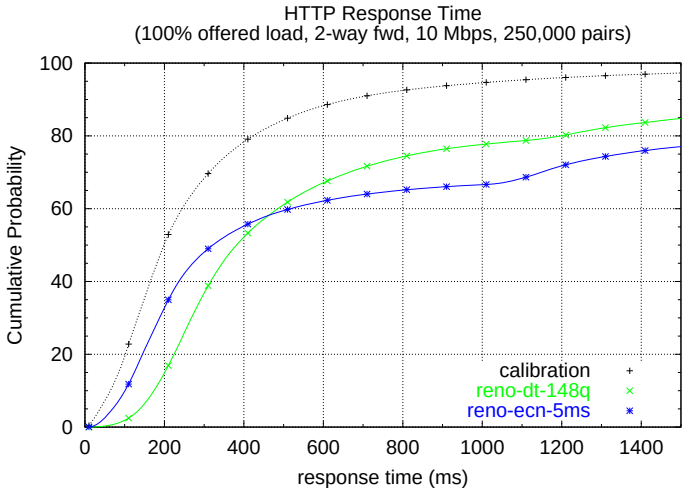


Figure 21: Distribution of HTTP response times for Reno Drop-Tail with 148-packet buffer and Adaptive RED + ECN with 5 ms target delay





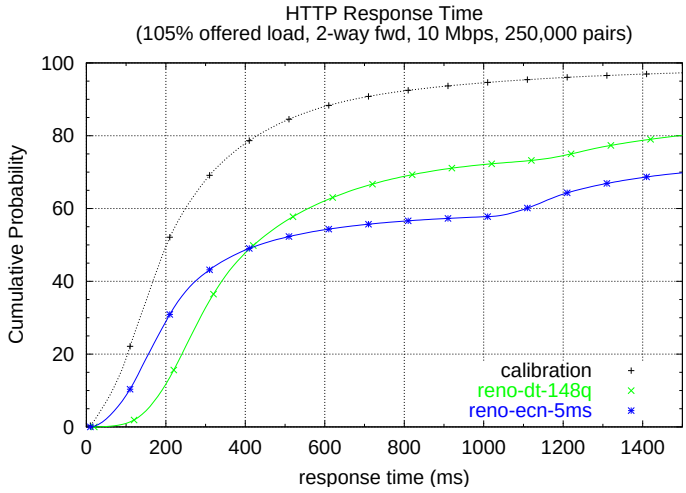


Figure 21: (continued) Distribution of HTTP response times for Reno Drop-Tail with 148-packet buffer and Adaptive RED + ECN with 5 ms target delay