A Performance Study of *Bistro*, a Scalable *Upload* Architecture[1]

[Appeared in ACM SIGMETRICS Performance Evaluation Review, March 2002]

William C. Cheng[2], Cheng-Fu Chou[3], Leana Golubchik[4], and Samir Khuller[3]

Abstract

Hot spots are a major obstacle to achieving scalability in the Internet. We have observed that the existence of hot spots in *upload* applications (whose examples include submission of income tax forms and conference paper submission) is largely due to approaching deadlines. The hot spot is exacerbated by the long transfer times. To address this problem, we proposed *Bistro*, a framework for building scalable wide-area *upload* applications, where we employ intermediaries, termed *bistro*s, for improving the efficiency and scalability of uploads. Consequently, appropriate assignment of clients to *bistro*s has a significant effect on the performance of upload applications and thus constitutes an important research problem. Therefore, in this paper we focus on the assignment of clients to *bistro*s problem and present a performance study which demonstrates the potential performance gains of the *Bistro* framework.

1 Introduction

Hot spots are a major obstacle to achieving scalability in the Internet. At the application layer, hot spots are usually caused by either (a) high demand for some data or (b) high demand for a certain service, which is typically the result of a *real-life event* involving availability of new data or approaching deadlines. At the application layer, hot spot problems have traditionally been dealt with using some combination of (1) increasing capacity; (2) spreading the load over time, space, or both; and (3) changing the workload.

We note that the classes of solutions stated above have been studied mostly in the context of applications using the following types of communication (a) one-to-many (data travels primarily from a server to multiple clients, e.g., web download, software distribution, and video-on-demand); (b) many-to-many (data travels between multiple clients, through either a centralized or a distributed server, e.g., chat rooms and video conferencing); and (c) one-to-one (data travels between two clients, e.g., e-mail and e-talk). However, to the best of our knowledge there is no existing work, except ours [3, 7], on making applications using *many-to-one* communication scalable and efficient (existing solutions, such as web based submissions, simply use many independent one-to-one transfers). This corresponds to an important class of applications, whose examples include the various *upload* applications such as submission of income tax forms, conference paper submission, proposal submission through the NSF FastLane system, homework and project submissions in distance education [18], voting in digital democracy applications [19], voting in interactive television [15], and many more.

To address the problem of scalable uploads[1], we proposed *Bistro*, a framework for building scalable wide-area *upload* applications. In the *Bistro* framework we employ the use of intermediaries, termed *bistro*s, for improving the efficiency and scalability of uploads. Consequently, appropriate assignment of clients to *bistro*s has a significant effect on the performance of upload applications and thus constitutes an important research problem. Therefore, in this paper we focus on the assignment of clients to *bistro*s problem and present a performance study which demonstrates the potential performance gains of the *Bistro* framework and gives insight into the general upload problem.

1.1 Our Framework for Upload Applications

We observe that the existence of hot spots in uploads is largely due to approaching deadlines. The hot spot is exacerbated by the long transfer times. We also observe that what is actually required is an assurance that specific data was submitted before a specific time, and that the transfer of the data needs to be done in a timely fashion, but does *not* have to occur by that deadline (since the data is often not consumed by the server right away).

Thus, our approach is to break the upload problem into pieces. Specifically, we break up our original deadline-driven upload problem into: (a) a real-time *timestamp* subproblem, where we ensure that the data is timestamped and that the data cannot be subsequently tampered with; (b) a low latency *commit* subproblem (which is the focus of this paper), where the data goes "somewhere" and the user is ensured that the data is safely and securely "on its way" to the server; and (c) a timely *data transfer* subproblem, which can be carefully planned (and coordinated with other uploads) and must go to the original destination. (Note that, this is somewhat analogous to sending a certified letter through a postal service.) This means that we have taken a traditionally *synchronized client-push* solution and replaced it with a *non-synchronized* solution that uses some combination of *client-push* and *server-pull* approaches. Consequently, we eliminate the hot spots by spreading most of the demand on the server over time; this is accomplished by making the actual

[1]This work is supported in part by the NSF grant EIA-0091474.

[2]This work was partly done while the author was with the University of Maryland.

[3]Department of Computer Science and UMIACS, University of Maryland at College Park.

[4]Computer Science Department, University of Southern California, Los Angeles, CA 90089, leana@cs.usc.edu. This work was partly done while the author was with the University of Maryland.

[1]We refer the reader to [3] for a discussion of why uploads require a solution different from downloads.

data transfers "independent" of the deadline.

Given the breakup of the original upload problem into subproblems (i.e., *timestamp*, *commit*, and *data transfer*), the original data transfer is now done using two data transfers (1) from a client to one or possibly more hosts on the Internet, termed *bistro*s, and then (2) from one or more *bistro*s to the server (termed the "destination *bistro*"). This flow of data is illustrated in Figure 1(b). Although Figure 1 only depicts a single upload event, it is understood that the *bistro*s may be shared by many simultaneous upload activities/applications, each with different deadlines, characteristics, and requirements. Coordination of *multiple* simultaneous upload applications is outside the scope of this paper.

Note that, the timestamp has to be produced before the deadline; the commit has to be performed with low latency, and the data transfer from a *bistro* to the server has to be done in a timely manner. The exact constraints on all these operations are a function of the requirements of the particular upload application.

Moreover, the client-to-*bistro* data transfer also produces the timestamp and the commit, i.e., the data is timestamped so the server has a guarantee that it cannot be tampered with after the deadline, and the client receives a commit, i.e., a "receipt" that guarantees that the data will be delivered to the server and that its integrity and privacy will be preserved. (We present one solution to the timestamp problem in [7].)

1.2 Advantages of the Bistro Upload Framework

Given the current state of upload applications, i.e., everyone uploads directly to the final destination server (refer to Figure 1(a)), a specific upload flow, from some client to the destination server, can experience the following potential bottlenecks (or hot spots): (a) poor connectivity of the client, (b) congestion somewhere between the client and the server, or (c) overload on the server, or a combination of these bottlenecks. Given these bottlenecks, traditional solutions (or a combination of these solutions), such as get a bigger server, buy a bigger pipe, and co-locate the server(s) at the ISP(s), exhibit shortcomings including the lack of flexibility and lack of scalability. We note that an important characteristic of the *Bistro* framework is the notion of *resource sharing*. It is fairly clear that, for instance, buying a bigger cluster for a "one time event" (which may not be "big enough" for the next similar event) is not the most desirable or flexible solution to the upload problem. The ability to share an infrastructure, such as an infrastructure of proxies or *bistro*s, between a variety of wide-area applications has clear advantages over the traditional solutions described above. Some advantages of such an approach are that: (a) it is more *dynamic* and therefore more *adaptive* to system and network conditions; (b) it provides for more resource sharing opportunities and thus can result in a more cost effective solution to wide-area upload problems; and (c) it *does not rely* on the existence of a private infrastructure (such as the co-location approach described above), but it *does not preclude* it either.

Our intent for deploying the *Bistro* platform is *not* to rely on adding resources (such as hosts) to the Internet[2]. Rather, we envision that people will want to install *Bistro* on their hosts on the *public* Internet and contribute their resources to the overall *Bistro* infrastructure because it will improve their performance as well. In turn, the existing *bistro*s will discover the new installations and integrate them into a *Bistro* infrastructure.

In summary, we believe that the *Bistro* framework is a more *flexible* solution that *takes advantage of whatever resources are available in the system and the Internet to the "best" extent possible.*

1.3 Our Contributions

In this paper we focus on the low latency commit subproblem, which, for the purposes of this work, can be reduced to (a) assignment of clients to *bistro*s participating in a particular upload event and possibly (b) placement, i.e., choosing which *bistro*s should participate in a particular upload event, if such a choice is possible. Both problems are NP-complete [3]. Given that the assignment problem is sufficiently difficult, in this work we, for the most part, focus on a quantitative study of the *assignment* problem, and only briefly investigate potential benefits of better placement in Section 3.4.3. We illustrate that both have a significant effect on the system's performance.

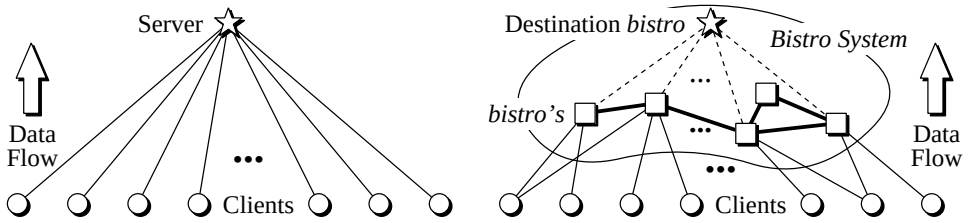
Thus, the main contribution of this paper is that it is the first performance study of a scalable and efficient solution to the deadline-driven upload problem. Other contributions of this work are as follows; we: (a) present a quantitative performance study of the commit subproblem; (b) develop an approximation to a lower bound on the commit subproblem (for comparison purposes); and (c) give insight into the general upload problem and characterize potential performance gains of the *Bistro* framework.

2 Related Work

In general, work related to wide-area data transfers falls into multiple categories and requires solutions in areas such as communication networks, performance evaluation, algorithms for load balancing, scheduling and facility location, and security. Moreover, relief of hot spots in the Internet through the use of data and service replication (e.g., proxy servers) has been studied extensively in the context of download applications, for instance, as in [1, 14, 12, 13]. To the best of our knowledge *none* address the problem of making wide-area *upload* applications scalable and efficient. We note that some work exists on efficient design of multipoint-to-point aggregation mechanisms at the IP layer, e.g., [2]. However, the solutions suggested there in the context of IP routers and active networks will not carry over to the upload problem presented here, i.e., it is not designed to reduce the load at or near the server, and it requires the use of the active network framework which is not currently deployed over the public Internet.

As is evident from Section 1.1 our basic approach involves

[2]Note that deployment over private networks can also be done but is relatively straightforward from the deployment point of view.



(a) upload without the *Bistro System* (b) upload with the *Bistro System*

Figure 1: Depiction of a *single* upload event with and without *Bistro*.

partial replication of services (i.e., *bistro*s). Replication is a relatively traditional solution to scalability problems in Internet-related services, and hence the problems we consider in this work, e.g., assignment of clients to servers, arise in other applications as well. Specifically, there is recent interest in addressing server selection problems for services replicated across wide-area networks which can be distinguished based on: (1) the selection method applied (static, statistical or dynamic); and (2) where the method is applied (server-side, client-side, or router). We note that (a) server-side selection is more appropriate for a cluster of servers while in our case the *bistro*s are distributed across the Internet, and (b) selection at routers requires network-based assistance, while *Bistro* is an application level framework. Thus we do not consider either any further in this paper.

Client-side server selection methods, as in [4, 13, 16, 17], are more suitable in environments with heterogeneous servers spread over wide-area networks. Furthermore, client-side selection methods are more scalable. However, the difficulty then is in trading off load balancing clients across servers vs. locating a server which is "closer" (or local) to a client. This tradeoff is important, especially in the context of upload applications, and is studied here, as described in Section 3.

Static as well as statistical server selection methods are less useful in environments where network load characteristics are highly variable, and hence, dynamic methods (e.g., ping-based) are more useful in these cases [4, 8].

Therefore, in the remainder of this paper, as in [8], we focus on dynamic client-side selection methods. However, important differences here include (1) workload characteristics of the *deadline-driven upload* applications we consider as contrasted with the workload characteristics of download applications considered in [8] and (2) the existence of the *data transfer* to the final destination step in our *Bistro* framework. Specifically, the workload differences, as compared to the study in [8], include (a) many simultaneous data transfers where the temporal proximity of the transfers is due to the approaching deadline (hence the hot spot); and (b) relatively large file sizes, e.g., as in IRS tax form submissions [11] or conference paper submissions. The consideration of the data transfer step also makes the server selection problem for *uploads* different from *downloads*, i.e., the commit and the data transfer subproblems (as outlined above) are, in general, not independent. However, investigation of explicit dependence between the two steps is subject of future research, i.e., in this paper we study the "commit" problem in isolation.

Lastly, much work has been done on facility location problems, e.g., [5], where the objective is to open a subset of facilities from a given set F, and to then minimize the sum of distances from a given set of customers to their closest facility, subject to a variety of constraints. This problem is NP-complete and several good polynomial time approximation algorithms are available for it. However, these solutions do not directly address the problem we are interested in, where the main differences are (1) there is an underlying network and different clients are competing for resources in the network; and (2) we cannot choose a route to the server at the application layer (we can only choose a server and the network chooses the route for us, at least in the Internet), which makes it hard to apply network flow based methods for assignment of clients to servers. These new constraints lead to very interesting optimization problems, as stated in [3].

3 A Performance Study of "Commit"

In this section we focus on possible solutions to the "commit" problem, and the corresponding performance characteristics.

3.1 Extreme Cases

We first describe the following extreme cases of the *Bistro* framework (in order to illustrate the range of performance considerations):

1. the final destination of the data transfer is the only *bistro* (this is essentially the current state of things); or
2. "all" hosts are *bistro*s, i.e., each client serves as its own *bistro*, with essentially zero transfer time — it is not clear whether this is "practical", but regardless, the commit problem is trivial in that case, and hence we do not consider this extreme any further here;
3. each "organization" with an upload client has its own "local" *bistro*, where granularity of organization is the same as for news (NNTP) servers, DNS servers, and so on. This local *bistro* can be behind organizational firewalls and may not be accessible from the outside world. In this case, a submission still has to travel to (be *pushed*) outside of the organization into the public Internet so that the *data transfer* part of the upload process can take over (i.e., the data transfer to the final

destination, in the *Bistro* framework, is a server-pull). Therefore, the *commit* problem still exists. As far as a user is concerned, the submission is completed as soon as the transfer to a local *bistro* is complete. However, the *Bistro* system will only consider the "commit" step to be completed when the submitted data has traveled to a public *bistro*[3]. Thus, in this case, the "commit" step is broken down into two steps, where the first step is almost as trivial as the one in extreme case 2 above. In the performance study below, we only consider public *bistro*s.

3.2 The Middle Ground

It is often difficult to deploy a server infrastructure over the public Internet. In order for the *Bistro* system to succeed, we must demonstrate that a system with a limited number of *bistro*s can provide benefits to the users of this system. In this section, we provide quantitative evidence, through a simulation study, that even a system with a limited number of *bistro*s can provide such benefits.

We note that, in the *Bistro* framework there are potentially two basic problems within the "commit" step:

1. *assignment problem*: this is the situation where the locations of *bistro*s are fixed, and the difficulty is in assigning clients to the *bistro*s, i.e., deciding which client should upload to which *bistro*; and
2. *placement (plus assignment) problem*: this is the situation where the locations of *bistro*s are flexible, and hence the difficulty is in both, deciding where to place the *bistro*s (i.e., which nodes to choose to act as *bistro*s for a particular upload event) and how to assign clients to them.

In [3] we defined and characterized these problems more formally (and determined that both are NP-complete). The assignment problem alone is hard under general network topologies; of course, the addition of flexible placement of *bistro*s makes the problem more difficult. Thus, in the remainder of the paper we consider heuristic algorithms and focus on the assignment problem, for the most part. We do, however, investigate the potential benefits of better placement in Section 3.4.3. To this end, we briefly motivate the need to consider placement, i.e., the situation where given N potential *bistro* sites, we only consider, for a particular upload event, the use of M of those *bistro*s, where $M < N$. This motivation is as follows: (1) as our performance evaluation results indicate below, M does not have to be very large to obtain much of the benefits of the *Bistro* architecture; (2) if we take the the data transfer from the *bistro*s to the final destination into consideration, then intuitively coordination of transfers from a large set of *bistro*s is not necessarily better than from a smaller set; and (3) our long-term goal is to allow multiple simultaneous upload applications to use the same *Bistro* infrastructure, in which case, due to contention, it may be beneficial to allow each application to use a subset of available *bistro*s.

[3]If the *Bistro* system is successfully deployed, we expect *bistro* installations to be as common as news and mail server installations.

3.3 Assignment Policies

Given that the assignment problem is NP-complete, we first consider the following simple heuristic policies for the assignment problem given a fixed set of M *bistro*s (some of these policies have been considered in some form in previous works in the context of *download* applications):

1. *random*: each client is assigned to a random *bistro* for the upload;
2. *ping5-mean* (or simply *ping-m*): each client pings each *bistro* five times [4], and chooses the *bistro* with the minimum average ping time for the upload;
3. *ping5-var* (or simply *ping-v*): each client pings each *bistro* five times, and chooses the *bistro* with the minimum variance in ping time for the upload.

In addition to these simple heuristic policies we also propose the following new policy, termed *pingk-3std* (or simply *ping-3std*). This policy is motivated by trying to reach an appropriate compromise between simply finding the "best" responding *bistro* (with respect to a particular client) and load balancing the clients between the *bistro*s, i.e., we are trying to avoid a situation where many clients simultaneously choose to upload to the same *bistro* and hence "interfere" with each other and experience bad performance even though the probes "predicted" good performance. The need for this compromise will become more evident as we present the quantitative results of our study below.

The *pingk-3std* policy works as follows. Each client pings each *bistro* k times,

$$k = \lfloor 5 * \frac{log_2 M^*}{log_2 M} + log_{10}(\frac{file\ size}{100K}) \rfloor$$

where M^* is chosen to be large enough such that $k \geq 5$ and M is the number of *bistro*s from which a client is choosing, i.e., we use 5 pings as our baseline [4]. The motivation for varying the number of pings according to the number of *bistro*s and the file size is that we use the pings not only to probe the performance from a client to a *bistro*, but also, in some sense, to "signal" the other clients which may be uploading their data at the same time. The intuition here is that the lower M is, i.e., the fewer the number of *bistro*s from which client j is choosing, the higher is the probability that client j will upload to a particular *bistro* i ($1 \leq i \leq M$) and hence the higher is the probability that other clients interested in *bistro* i will experience "interference" from client j on *bistro* i. Consequently, client j "signals" its interest in *bistro* i with higher "intensity", i.e., it uses more probes thereby increasing the probability that these probes will "interfere" with probes of other clients interested in *bistro* i and thus the mutual conflict will be "detected" with higher probability. Likewise, the larger the file size the higher is the probability of such "interference" between clients on *bistro* i, which similarly motivates the increase in number of probes based on file size.

The remainder of the policy proceeds as follows. For a particular client, we compute T_i, the average response time of the k probing packets for each *bistro* i, as well as T_{avg} and T_{std}, the average response time and its standard deviation, respectively, over all M *bistro*s. If there is no loss of probing packets, the client then uploads its data to the *bistro* with the minimum T_i. Otherwise, the client separates the *bistro*s into two sets, based on T_i's. The "near-by" set A includes each *bistro* i, where $T_i \leq T_{avg} - 3 * T_{std}$. The "remote" set B contains the remaining *bistro*s. If A is not empty, we choose one *bistro* from that set at random. Otherwise, we choose one *bistro* from set B at random.

In other words, we use packet loss to "guess" whether or not the part of the network of interest to us is congested. If not, we choose the "best" responding *bistro*. Otherwise, we only choose the "best" responding *bistro* if it is significantly better than the others (i.e., we look for particularly good outliers). When no outliers are detected, we opt for load balancing (hence the random approach). The intuition here is that when the part of the network of interest to us is under low to moderate congestion, then load balancing is not as important and we opt for the "best" responding *bistro* we can find without regard for other clients. In contrast, under higher levels of congestion (indicated by packet loss) we only opt for the "best" responding *bistro* if it presents significant potential for improvement; otherwise, we give preference to load balancing.

Of course, in some sense these policies are fairly "basic" and one could construct numerous hybrids and variations of these policies by combining them in a number of different ways. However, we will limit our performance study to these variants in order to illustrate our points.

3.4 Performance Study

We now illustrate performance characteristics of the assignment policies as well as the upload problem in general. (We also briefly consider potential benefits of better placement in Section 3.4.3.)

3.4.1 Simulation Set-up & Performance Metrics: We first describe our simulation setup[4]. We use ns2 [9] for all simulation results reported below. We note that ns2 does not provide detailed simulation of hosts/servers, and hence the following performance study is of bottlenecks in the network (i.e., not on the server) only. In conjunction with ns2, we use the GT-ITM topology generator [10] to generate a transit-stub type graph for our network topology. Specifically, we use the GT-ITM topology generator to create a transit-stub graph with 152 nodes. The number of transit domains is 2, where each transit domain has, on the average, 4 transit nodes with there being an edge between each pair of nodes with probability of 0.6. Each node in a transit domain has 3 stub domains connected to it; there are no additional transit-stub edges and no additional stub-stub edges. Every stub domain has, on the average, 6 nodes with there being an edge between every pair of nodes with probability of 0.2. The capacity of a "transit node to transit node" edge is 1 Mbit/s. The capacity for a "transit node to stub node" edge or a "stub node to stub node" edge is 256 Kbits/s. Note that, the the size of our model is motivated by what is practical to simulate with ns2 in a reasonable amount of time.

Furthermore, except for the results in Section 3.4.4, there is a total of 96 simultaneously uploading clients in each simulation, each one with a file size which is uniformly distributed between 100 KBytes and 2 MBytes[5]. In all simulations, except for Section 3.4.4, all clients begin their uploads essentially at the same time, i.e, the interarrival time of upload requests is zero. Hence, if we were to coordinate the upload of all clients to a *single* server and make it *sequential*, then the total transfer time of all clients should be on the order of 3000 seconds and the average transfer time (i.e., from begining of a particular client's transfer to its end) should be on the order of 33 seconds. As we will illustrate below, the simulation results indicate that the total transfer time for *simultaneous* transfer of all clients to a *single* server is on the order of 3000 seconds. However, the average transfer time is over 2000 seconds which clearly indicates the "interference" between clients; this is due to TCP fairly sharing the bottleneck link between all clients. This "interference" is what users actually observe in the current state of upload applications around the deadline time (e.g., submission of papers around the conference deadline as well as submission of NSF proposals through FastLane around the proposal deadline).

Although, in a real system there will be some non-zero interarrival time between client upload requests, we first consider simultaneous uploads in order to isolate the effects of assignment policies described above. In Section 3.4.4 we include interarrival time characteristics and observe the resulting effects on the average response time.

In addition to the upload traffic, we generate two types of background traffic in our simulations, "low" and "high". In both cases the background traffic consists of ftp's between pairs of stub domain nodes, where each ftp corresponds to file transfer of an infinite size. The "low" background traffic is generated by randomly choosing 30% of stub domain nodes to participate in the infinite ftp's. The "high" background traffic is generated by randomly choosing 70% of stub domain nodes to participate in infinite ftp's.

Lastly, the performance metrics used in the remainder of the paper are (a) mean response (transfer) time over all clients participating in the upload and (b) in the case of all clients uploading simultaneously, total (or maximum) response (transfer) time, i.e., the time needed to complete the data transfer of all clients (which in the case of simultaneous uploads also corresponds to the transfer time of the "slowest" client since they all begin the transfer at approximately the same time). We believe that these metrics reflect the quality-of-service characteristics that would be of interest to users of upload applications. That is, not only is the average transfer time of interest but also is the maximum transfer time since it reflects the worst case behavior. All performance results include over-

[4]Most results presented here are obtained with $95 \pm 10\%$ confidence intervals; all results are within $90 \pm 10\%$ confidence intervals.

[5]These are reasonable file sizes for upload applications; e.g., a basic US IRS tax form 1040 is on the order of 100 KBytes [11] and a typical paper submitted to a typical conference is on the order of 1 to 2 Mbytes.

heads, e.g., those due to dynamic probing.

3.4.2 Performance Results & Discussion: In Figures 2 and 3 we illustrate the performance gains obtained from the *Bistro* architecture, using the policies in Section 3.3 for assignment of clients to *bistro*s, as a function of the number of *bistro*s. Figures 2(a,b) and 3(a,b) correspond to "low" background traffic and Figures 2(c,d) and 3(c,d) correspond to "high" background traffic. In these figures we fix the placement policy, and consider the effects of the assignment policy only. The placement policy is fixed at "random" placement in Figure 3, i.e., we emulate the situation where we do not have control over the placement of *bistro*s. It is fixed at "heuristic" placement in Figure 2. (*Note*: we describe our *heuristic* placement and assignment policy as well as the motivation for using it below. For now, we focus our attention on the performance of the assignment policies, and investigate the potential benefits of better placement policies in Section 3.4.3.)

Figure 2 is intended to illustrate the basic performance gains that can be obtained from the *Bistro* framework even with simple assignment policies (these gains are mainly due to parallelism). Figure 3 is intended to illustrate the gains we are able to obtain with a better assignment policy (namely *ping-3std*) which adapts to a variety of workload conditions (as detailed below).

In all cases depicted in Figures 2 and 3, as we increase the number of *bistro*s, we first observe a nearly linear gain in performance[6]. As expected, further increases in the number *bistro*s result in diminishing gains in performance. We note that, not surprisingly the performance gains depicted in these figures are mostly due to parallelism, as we are experimenting with an essentially symmetric network topology, i.e., no client or *bistro* is connected through an extremely low capacity link. Also note that, due to this symmetry the expected time to complete the *data transfer* to the destination *bistro* (after the "commit" step is complete) is expected to be same regardless of the assignment policy used in the "commit" step. Future work includes investigation of effects of *bistro* placement policies on system performance (both in the case of the "commit" step and the *data transfer* to the destination *bistro* step) in the context of low connectivity hosts (i.e., we expect that proper placement will result in performance gains not only due to parallelism but also due to provision of services in "better" places on the network or of bringing of services "closer" to the clients).

Furthermore, we observe that under light background loads, ping-v does better than *ping-m*. We conjecture that under light background loads, both *ping-m* and *ping-v* are fairly good at detecting the "best" responding *bistro*; however, neither aims at load balancing, and consequently may direct too many clients to the same *bistro*. However, *ping-v* is *less* accurate than *ping-m* at detecting the "best" responding server and hence exhibits better load balancing characteristics, but only as a "side-effect". Under heavy background loads *ping-v* and *ping-m* perform about the same. Under both background loads the random assignment policy appears to result in more "steady" or "predictable" performance, which is not surprising as random-based policies tend to be more "robust" and less sensitive to the dynamics of the network.

It should also be clear from Figure 3 that our new *ping-3std* policy is able to "adapt" to both high and low background loads, i.e., that it tends to do better than other policies under high background loads and does almost as well as *ping-v* under low background loads, i.e., it improves significantly on *ping-m*'s performance in the light background load case.

3.4.3 How Well Are These Policies Doing?: We now motivate the need for our heuristic placement and assignment policy mentioned above. Given the performance results in Section 3.4.2, it is clear that benefits can be obtained from the general *Bistro* framework; these are of course benefits due to simple parallelism. However, one remaining reasonable question would be "how well are we doing with these policies?" — that is, ideally we would like to construct a lower bound on the mean and total response time metrics, in order to illustrate the "goodness" or "badness" of the simple random-based and ping-based policies and to study the potential performance gains that may or may not be obtained with more sophisticated algorithms. That is, "should we look for more sophisticated policies in the hopes of obtaining better performance?".

Motivated by this question, we developed an "unrealistic" heuristic policy which serves as an estimate of a lower bound in the remainder of this performance study. (We resort to the use of an unrealistic heuristic since an actual lower bound is difficult to characterize in such a complex system.) The policy is unrealistic in the sense that it assumes complete knowledge of the network topology as well as the background traffic. It is a heuristic since, as we noted above, even the assignment problem alone is NP-complete. This heuristic policy uses minimization of the maximum (total) response (transfer) time as its objective. It computes a "good" estimate of an initial (a) placement of *bistro*s and (b) client to *bistro*s assignment, followed by a local search, used to improve on this initial assignment. The actual details of the heuristic are fairly complex, and hence we omit them due to lack of space (they can be found in [6]). We believe that, for our purposes, this heuristic policy is a reasonable replacement for a true lower bound, since it uses information that would not be typically available to a "real" policy at run time, such as complete knowledge of the network topology, the background traffic, and so on; hence, it is expected to almost always do better than a "realistic" policy.

As is clear from Figure 4, the unrealistic heuristic policy performs significantly better than the random- and ping-based policies, indicating that there is potentially room for improving on these simple policies. However, this the topic of future work, as we do not actually know where the real lower bound is.

Moreover, we attempt to isolate the significance of the heuristic placement vs. the significance of the heuristic assignment. To this end, we consider the results in Figures 2 and 3, where we hold the placement policy fixed (at the heuristic placement

[6]Note that, the first point on the X-axis of these figures corresponds to 2 *bistro*s. The results for 1 *bistro* are given in the legends of the figures.

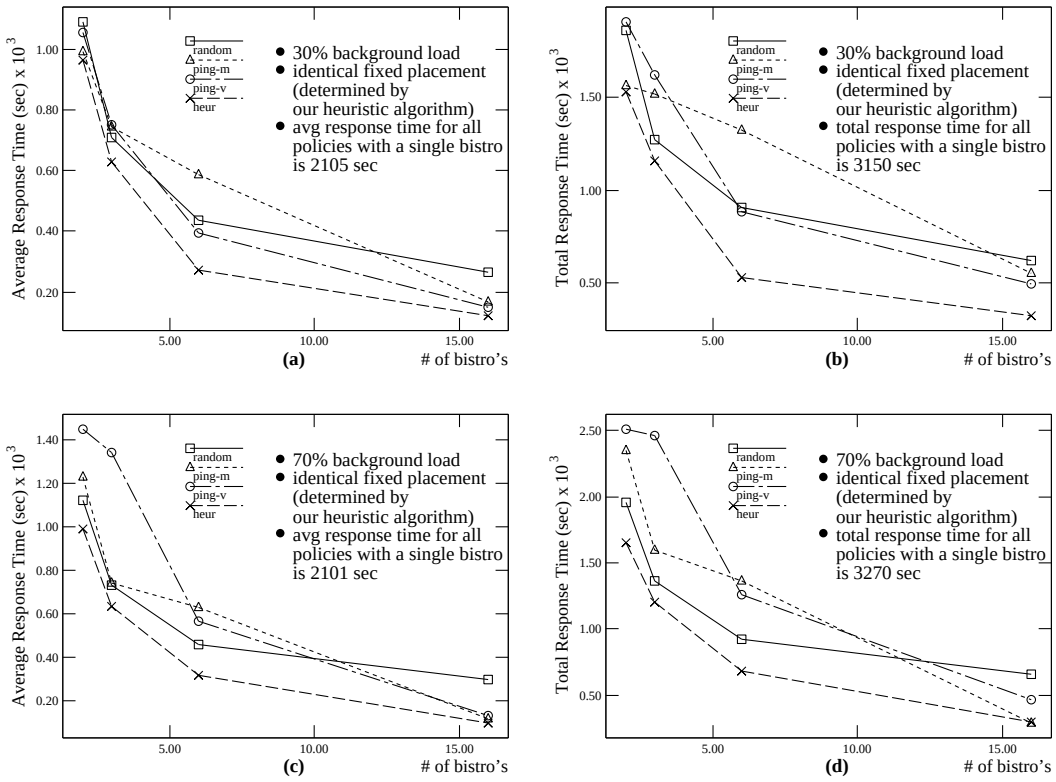


Figure 2: Comparison of schemes under low and high background loads with fixed placement.

in Figure 2 and at random placement in Figure 3). We conjecture that better placement policies can have a significant effect on performance, since the difference between random and heuristic *placement*, for a given assignment policy, can be almost a factor of 2.5 (refer to Figures 2 and 3).

3.4.4 Effects of Interarrival Times: We now study the effects of non-zero interarrival times on the mean response time experienced by the same 96 clients as in the previous sections. Figure 5 depicts these results for the cases of 6 and 16 *bistro*s available for uploading; in each case, the *bistro*s are placed randomly. As can be seen from Figure 5, some of the policies that performed well under zero inter-arrival times, such as the *random* policy, do not perform as well (relatively speaking) under higher interarrival times. This is of course due to the fact that random policies are good at load balancing; however, with higher interarrival times (i.e., lower loads) load balancing is less important. For the same reason *ping-m* tends to perform better at higher interarrival times, i.e., it is good at finding the "best" responding *bistro*, and its difficulty with respect to load balancing (as explained above) is of less importance under these workloads.

However, it should be clear from Figure 5 that our new *ping-3std* policy is able to "adapt" to a variety of workloads, i.e., it tends to "mimic" (a) policies that are good at load balancing under shorter inter-arrival times, and (b) policies that are good at finding "best" responding *bistro*s under longer interarrivals times. That is, it is robust under a variety of workload conditions.

4 Conclusions

In this paper we presented a performance study of our solution to the "commit" problem for upload applications within the *Bistro* framework; this involved assignment of clients to the *bistro*s and to a lesser extent placement of *bistro*s. We demonstrated the potential performance gains of *Bistro*. We believe that the above mentioned assignment problem, studied in this paper in the context of upload applications, is different from previous works on similar problems (often referred to as server selection problems), where one important difference is largely due to the workload properties induced by approaching deadlines, which are characteristic of many upload applications.

In summary, we would like impress upon the reader that even simple solutions to the placement and assignment problems produce significant performance gains through the use of our *Bistro* framework, i.e., due to parallelism. Hence, we believe that further studies of this framework and solution to the other open problems stated in [3] are a fruitful area of future research.

Acknowledgements: we would like to thank Bobby Bhattacharjee for helping us with the simulation setup and for numerous useful discussions.

References

[1] I. Akamai Technologies. *Home page*. http://www.akamai.com/, 2001.

[2] B. R. Badrinath and P. Sudame. Gathercast: An ef-

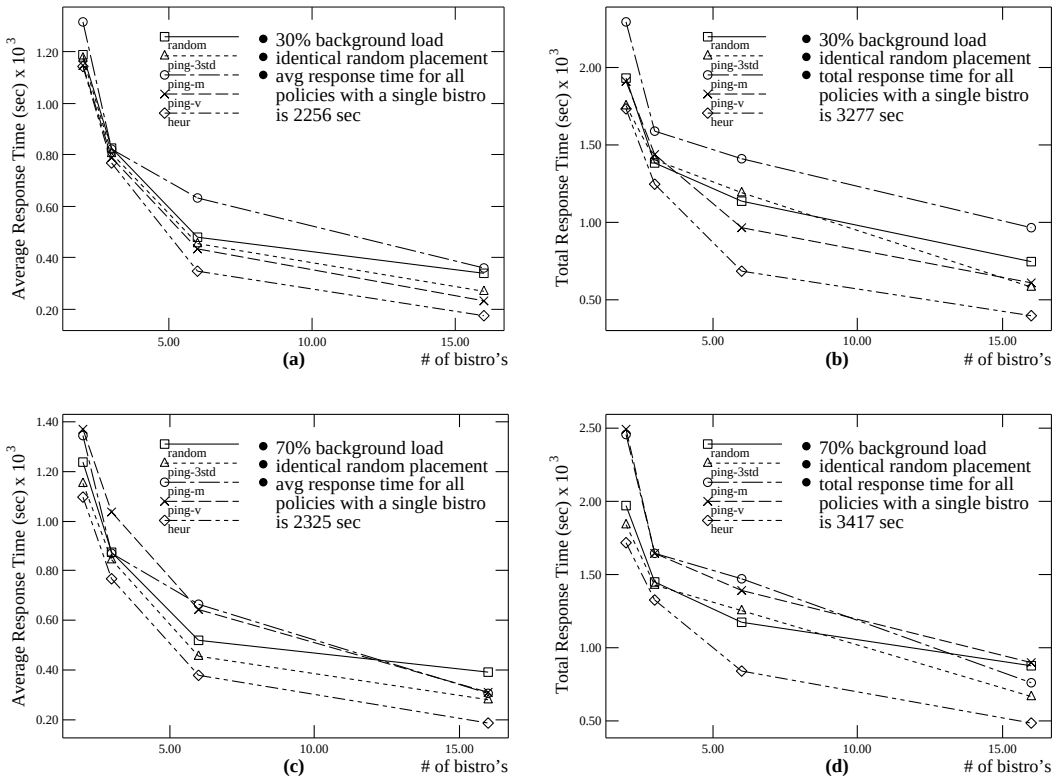


Figure 3: Comparison of ping-3std policy with other schemes under low and high background loads with fixed placement.

ficient mechanism for multi-point to point aggregation in ip networks. Technical Report DCS-TR-362, Computer Science Department, Rutgers University, July 1998.

[3] S. Bhattacharjee, W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. Bistro: a platform for building scalable wide-area upload applications. *ACM SIGMETRICS Performance Evaluation Review (also presented at the Workshop on Performance and Architecture of Web Servers (PAWS) in June 2000)*, 28(2):29–35, September 2000.

[4] R. L. Carter and M. E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. In *Proceedings of IEEE INFOCOM*, 1997.

[5] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Foundations of Computer Science Conference*, pages 378–388, October 1999.

[6] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A performance study of bistro, a scalable wide-area upload architecture. Technical Report CS-TR-4260, University of Maryland, June 2001.

[7] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A secure and scalable wide-area upload service. In *Proceedings of the 2nd International Conference on Internet Computing, Volume 2*, pages 733–739, June 2001.

[8] S. G. Dykes, K. A. Robbins, and C. L. Jeffery. An empirical evaluation of client-side server selection algorithms. In *Proceedings of IEEE INFOCOM*, 2000.

[9] http://www-mash.cs.berkeley.edu/ns/. *UCB/LBNL/VINT Network Simulator - ns (version 2)*.

[10] http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html. *Georgia Tech Internetwork Topology Generator*.

[11] IRS. *Fill-in Forms*. http://www.irs.ustreas.gov/prod/forms_pubs/fillin.html, 2001.

[12] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Relieving hot spots on the world wide web. *STOC*, 1997.

[13] K. Kong and D. Ghosal. Mitigating server-side congestion in the internet through pseudoserving. *IEEE/ACM Transactions on Networking*, 7(4):530–544, Augest 1999.

[14] Napster. *Home Page, Napster, Inc.* http://www.napster.com/, 2001.

[15] L. Press. The Internet and interactive television. *Communications of the ACM*, 36(12):19–23, 1993.

[16] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithm for replicated web servers. In *Performance evaluation Review*, pages 44–50, Dec 1998.

[17] S. Seshanm, M. Stemm, and R. Katz. SPAND: Shared passive network performance discovery. In *Proceedings of the First USENIX Symp. on Internet Technologies and Systems*, Dec 1997.

[18] P. Thomas, L. Carswell, M. Petre, B. Poniatowska, B. Price, and J. Emms. Distance education over the Internet. *Proceedings of the conference on integrating technology into computer science education*, pages 147–149, 1996.

[19] R. T. Watson, S. Akselsen, B. Evjemo, and N. Aarsaether. Teledemocracy in local government. *Communications of the ACM*, 42(12):58–63, 1999.

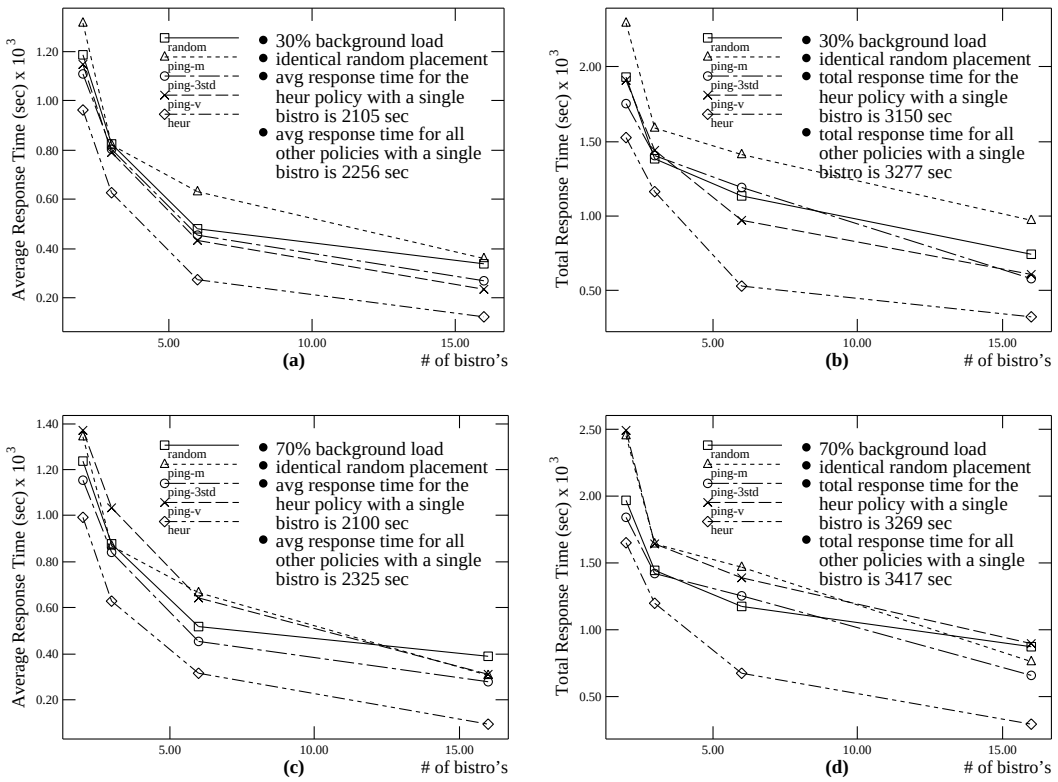


Figure 4: Comparison under low and high background loads where each scheme uses its own placement and assignment policies.

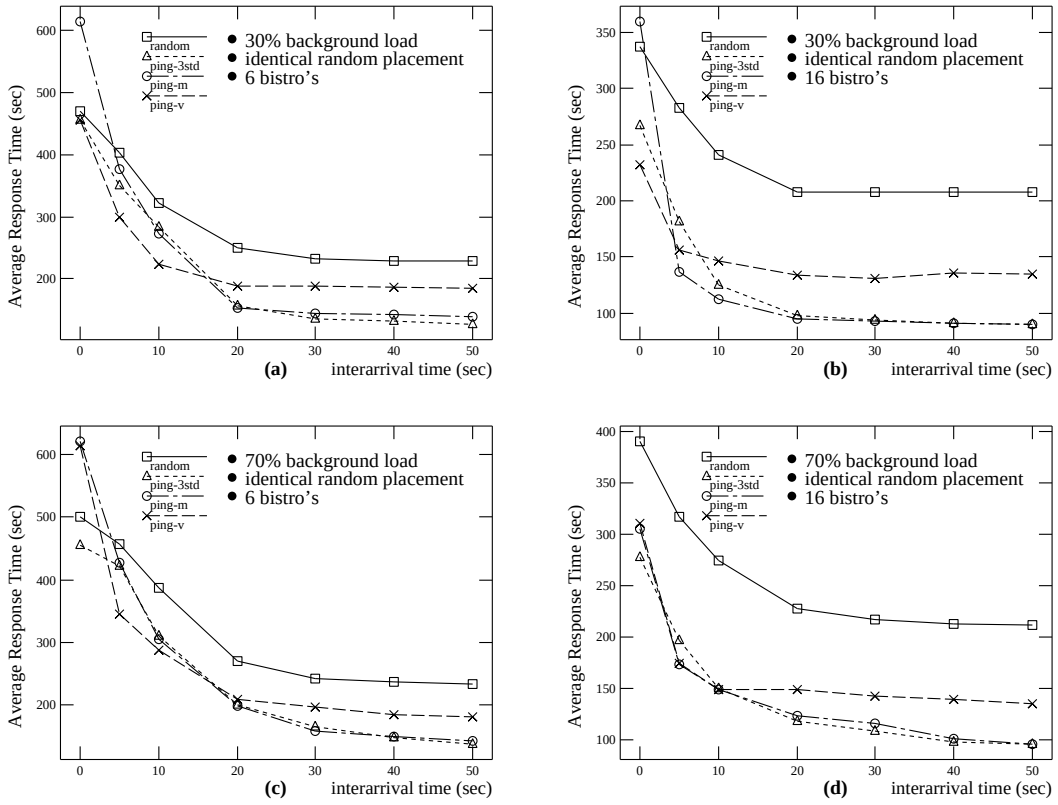


Figure 5: Comparison of schemes with varying interrival times under low and high background loads with fixed placement.