A Measurement Study of the Interplay Between Application Level Restart and Transport Protocol

Philipp Reinecke[1], Aad van Moorsel[2], and Katinka Wolter[1]

[1] Humboldt-Universität zu Berlin,
Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
`{preineck,wolter}@informatik.hu-berlin.de`
[2] aadvanmoorsel.com
Goßlerstraße 11, 12161 Berlin, Germany
`conferences@aadvanmoorsel.com`

Abstract. Restart is an application-level mechanism to speed up the completion of tasks that are subject to failures or unpredictable delays. In this paper we investigate if restart can be beneficial for Internet applications. For that reason we conduct and analyze a measurement study for restart applied to HTTP GET over TCP. Since application-level restart and TCP time-out mechanisms may interfere, we discuss in detail the relation between restart and transport protocol. The analysis shows that restart may especially be beneficial in the TCP set-up phase, in essence tuning TCP time-out values for the application at hand. In addition, we discuss the design of and experimentation with a proxy-based restart tool that includes a statistical oracle module to automatically adapt and optimize the restart time.

1 Introduction

Internet applications require effective ways to deal with unpredictable and highly fluctuating response times. Recently, restart has been proposed as a technique that may achieve that goal. Restart simply implies that an application retries an attempt if no result returns before some time-out. The applications researched range from Internet agent executions [4,9] and web crawlers [9] to randomized database queries [13] and randomized algorithms [1,8]. In this paper we investigate whether restart may work for straightforward HTTP GET, and how restart mechanisms interact with and relate to transport protocols (particularly TCP).

It is especially important to understand the inter-workings of application level restart with underlying transport protocols. An an example, consider a web browser's reload button, which most people routinely use to retry downloads that (seem to) halt. In essence, pushing the reload button is itself a restart mechanism, since it terminates the previous download and starts a new one [11]. Moreover, as one can read in detail in [7], pushing the reload button corresponds to 'overruling' a time-out value within the TCP protocol (namely the Retransmission Timeout (RTO)). Since the TCP protocol is core to the well-functioning of the Internet, interfering with TCP may be a dangerous play. In the worst case, if restart is used wide-spread by Internet users, overall performance may deteriorate, as discussed in [9] and in a broader context in for instance [2,6].

This paper provides a measurement study to investigate at what stages of HTTP downloads one could apply restart. Restart decreases completion time of a job if the completion time after restart is less than the remaining completion time without restart. To find out whether this is the case, we first examine if consecutive attempts are independent and identically distributed (i.i.d.). If attempts are i.i.d, it is likely that they better respond to restarts than if they are positively correlated. In addition, if attempts are i.i.d., we can compute the restart time that minimizes the expected completion time. If the optimal restart time is finite, we conclude that restart reduces the completion time of the job.

From the analysis in this paper we conclude that successive downloads to different URLs are i.i.d, and that downloads to the same URL are i.i.d. in a majority of the cases. The fact that successive completion times are largely independent is a somewhat surprising and very Internet-specific fact, but supported by other measurement studies [7]. It also turns out that during all phases of HTTP GET restart improves completion time. However, if we consider retries to the same URL, restart is primarily beneficial because attempts fail with a certain probability. If we separate out failures, the completion time distribution of TCP connection set-up benefits from restart in only a small minority of the cases. We therefore conclude that (when applied to a single HTTP GET application) restart is especially useful to avoid very long connection set-up times that arise because the RTO time-out value is not optimal for HTTP GET.

We note that the above conclusions apply to the case that only one application executes restart. All other active Internet applications are assumed to not change their behavior. We refer to [9] for a simulation study of the case that multiple or all applications apply restart. Note also that the restart approach is a black-box approach, as is our measurement study. We have no data about the particulars of network or server status, and thus cannot draw conclusions regarding the relation between completion times and server or network load.

Applying restart would be greatly helped by a software module that can be used for various applications, and flexibly adapts the restart time based on the application at hand and the performance measured. To that end, we designed an on-line oracle with algorithms that dynamically adapt and optimize the restart time. The results we outlined above have all been obtained using this restart tool.

The work in this paper is similar in purpose to the experiments in [14]. We subscribe to the conclusion from [14] that restart can substantially improve connection set-up, especially for the tail of its completion time distribution. We add to that a to-the-point discussion of the inter-working between TCP and restart, computation of optimal restart times for our specific experiments, and the design of an adaptive restart software tool.

2 Experiment Design

In our experiment we gather a large set of data reflecting performance of an application that uses HTTP GET, the most obvious example being the web browser. For each individual web page, a three-stage process is to be considered: (1) IP address resolving, (2) connection establishment and downloading of data comprising the page, and (3) page rendering by the browser. In our experiment we focus on step (2), arguing that the first step usually happens only once for each host, and the time taken by the third can in most cases be neglected.

Within step (2), we concentrate on three important aspects:

1. TCP connection set-up time (hereafter referred to as CST).
2. Download time of a single object in a web page, e.g. a picture (object download time, ODT).
3. Time from beginning to end of a page download (total download time, TDT).

Note that the total download time includes connection set-up as well as a number of object downloads. Each object download may include a TCP connection set-up. If HTTP/1.1 is being used, downloads from the same IP address can use the same TCP connection, thus alleviating the need for multiple TCP connection set-up phases.

Our experiments consisted of two stages, where the point of the first one was an investigation into the nature of time data encountered downloading web pages, while the second aimed at examining a possible improvement doing restarts.

2.1 First stage

We carried out the experiments of the first stage in two phases, first downloading a large number of pages and afterwards concentrating on a few interesting examples.

Host List Construction To obtain a large number of samples for the first step we built a list of web page URLs to be downloaded. We repeatedly fed entries of a word list into the Google search engine (http://www.google.com), requesting to be shown the first 100 hits in the reply. From this page, all linked URLs were extracted and sorted in lexicographic order. We then isolated the host name components and removed duplicate entries. As far as possible, we also checked by informal means that the selection of hosts is not limited to any geographical area or part of the Internet. Altogether we hope to have created a more or less random set of URLs with which we conducted our experiment.

We used three machines running Linux as clients. Two of these were connected to the Internet by 768/128 kbit ADSL, the third one by 100Mbit Ethernet. Iterating through the list of hosts a client downloads each server's index page exactly once. This yields three sets of data, with characteristics as in Table 1. Note that the number of objects listed under ODT is larger than the number of TCP connections listed under CST, since HTTP/1.1 allows that TCP connections can be reused for multiple objects from the same IP address.

Table 1. Constructed data sets of phase 1.

Data set	# of CST $\neq 0$	# of ODT samples	# of TDT samples
I	234848	799265	56117
II	231793	794324	55397
III	232525	795986	55558

As the second phase we then selected 309 URLs to be examined further. These URLs were chosen based on the findings of the first run; we selected those that (1) consisted of a minimum of 50 objects and (2) provided several connection setups (i.e., probably did not use HTTP/1.1). These criteria were based on our decision to concentrate on CSTs. Hence, to draw reliable conclusions, we needed a large number of CST samples. URLs from this list were then repeatedly downloaded in batches of 25, each URL 10 times in a row, and the whole batch 10 times as well. Entries that failed as well as those that had already provided at least 1000 CSTs were removed from subsequent iterations in order to speed up the experiment. This phase was run on one Linux system using ADSL.

2.2 Second Stage

In the second stage we compared performance of connections through our proxy with and without doing restarts at connection setup. Here we used the same method as described in the second step above, but employing the proxy in-between. This stage was conducted in two ways, (1) using a list of 17 URLs that exhibited a relatively large number of CSTs above 3 seconds and (2) using 25 URLs without CSTs in this range, but with a high standard deviation of samples, both based on observations in the second phase of the first stage.

2.3 Tools Used

For all our experiments we relied on a modified version of the GNU *wget* utility. We modified the program's usual output to display time stamps consisting of seconds and microseconds (as given by the *gettimeofday()* C library function), at the start and end of (1) connection set-up, (2) download of a page component (possibly including more than one connection set-up), and (3) download of a whole page. To obtain such detailed data, we ran *wget* with the *'-p'* option, thereby downloading all elements necessary to render the page, i.e., requesting the HTML code as well as any objects referred to.

3 Analysis for Multiple URLs

For some Internet applications restart implies connecting with a different URL at each attempt. This can for instance be the case with web crawlers [9] or certain agent applications that find quotes from e-commerce web sites. For such

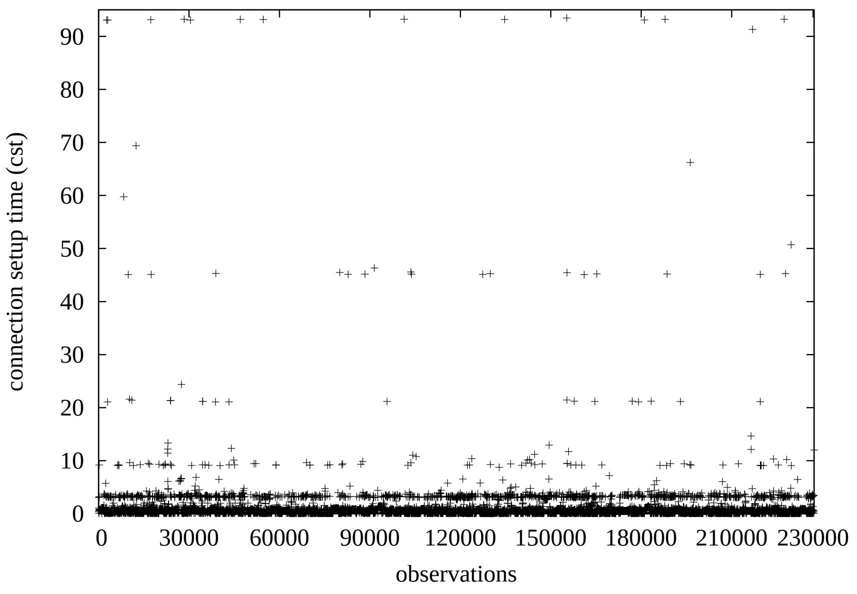

Fig. 1. All sampled connection set-up times.

applications, one needs to analyze data for varying URLs. We focus mainly on data set II of Table 1, but the results for the other data sets are similar. Fig. 1 shows the time for connection set-up of the close to 240 thousand samples of data set II.[1] The plot nicely visualizes the role of the retransmission time-out in TCP connection set-up. The RTO is used to trigger a retransmission of a connection attempt if no acknowledgment has been received in time. In Fig. 1 one recognizes the RTO values through the 'stripes' formed a little above RTO values: first after 3 seconds, upon the second trial after 9 seconds, then 21, 45 and 93 seconds, exactly according to specification [7].

Importantly, almost all connection set-up times are less than 1.0 seconds. That is, either the connection set-up succeeds quickly, or one waits for the RTO timer to trigger a new attempt. Very rarely (roughly 1 in 5000 attempts) does a connection get established after one second, but before the next RTO time-out. In addition, out of the roughly 230 000 observations, about 1200 'fail' (i.e. they take longer than 3.0), thus leading to retries. Hence, the CST distribution is with probability 0.995 distributed as given by the samples, and with probability 0.005 it fails. If we compute the optimal restart time for this data set (see Appendix A and [11]), we obtain that restarts should be done every 0.43 seconds.

[1] For practical reasons we plot in several figures throughout this paper only a subset of the thousands of samples. In none of these cases this changes the visual experience when interpreting the figures.

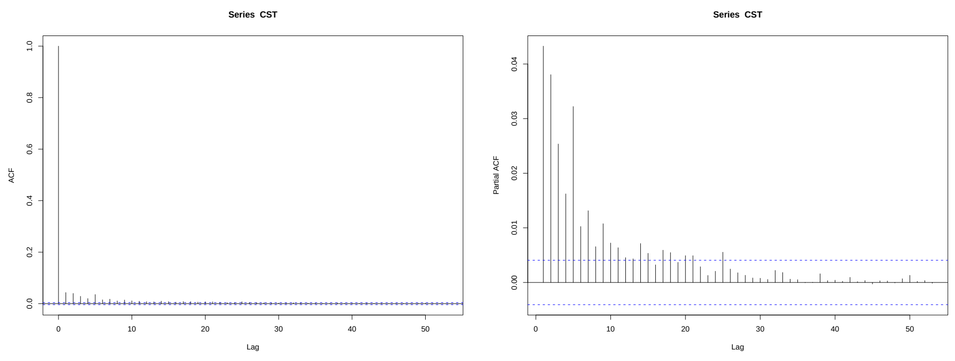

Fig. 2. Lag versus autocorrelation and partial autocorrelation function for CSTs to multiple URLs.

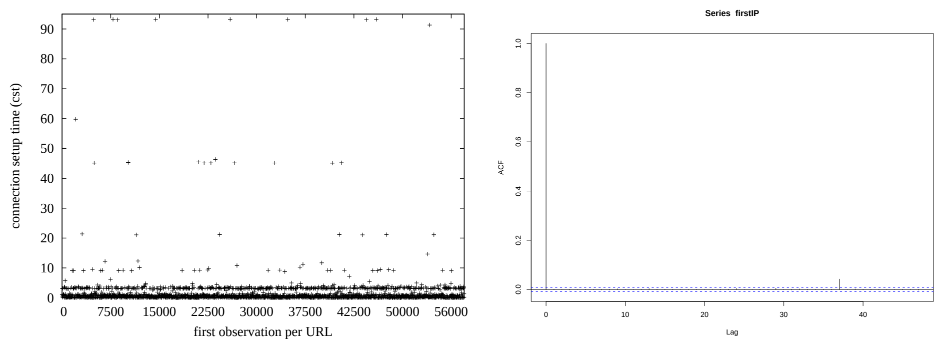

Fig. 3. Connection set-up times for the first connection per URL download, and its autocorrelation function over the lag.

Such computation, however, assumes independence of consecutive tries, a fact we investigate now using the statistical package R [12].

We compute the autocorrelation function and, since it provides better visual evidence, the partial autocorrelation function (see Appendix B for statistical background information), as shown in Fig. 2. Both figures show that the series exhibits correlation, since the values of the autocorrelation function are not close to zero for lower lags.

We believe that part of the explanation of the correlation between consecutive connection set ups in data set II is the clustering effect introduced by the different URLs: URLs are accessed sequentially and connections to the same URL tend to take times similar to each other. From how it is assumed to occur one expects it to disappear if for each download in data set II only the first CST is selected.

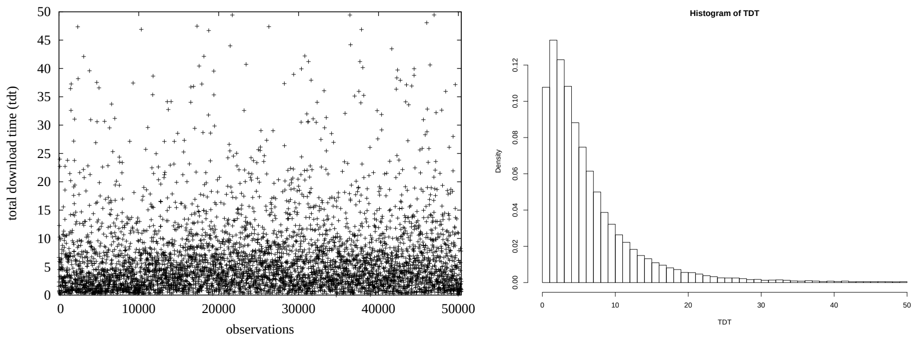

Fig. 4. Scatter plot and histogram of the total download time.

Fig. 3 shows the results (roughly 56000 samples). We see from Fig. 3 on the left side that the first set-up time to each server has similar characteristics as in the overall sampling of connection set-ups in Fig. 1. Even though the pattern in the CST is the same for all sampled CSTs as for the first connection set-up to a location, the dependence characteristics are not. The autocorrelation, shown in Fig. 3 on the right side, is very close to zero. Note the outlier at lag 37, which we cannot explain – each of the other two data sets also exhibits one such outlier. In the first data set it is for lag 2, in the third for lag 28. In general, however, we conclude that this data is uncorrelated and hence is more likely to be amenable to restart.

The connection set-up is only the first step in the download of a page. The final objective of restarting a download is to receive the whole page faster. Therefore we are interested in the total download times. In this case, the results we obtain show fewer obvious patterns. This is because the file sizes vary greatly, and also the number of objects belonging to one page is quite diverse. To give a picture of the total download times we sampled, Fig. 4 shows TDTs for our data set II. In addition, the right side of Fig. 4 shows the probability density of the data in the left plot. Note that both figures display samples with a download time of at most 50 seconds, thus omitting the 726 largest values. It is interesting to note that if one applies the expressions from Appendix A to this data, one finds that the optimal restart time is about 8 seconds.

Of course, it is not at all clear if the assumptions under which the optimal restart time can be computed apply to the data in Fig. 4, and we therefore study the correlation of consecutive downloads. From Fig. 5 we see that there is quite strong autocorrelation in the object download time, probably since many objects belong to the same index page (up to 200 objects on one page). However, the total download times of consecutive attempts seems highly uncorrelated.

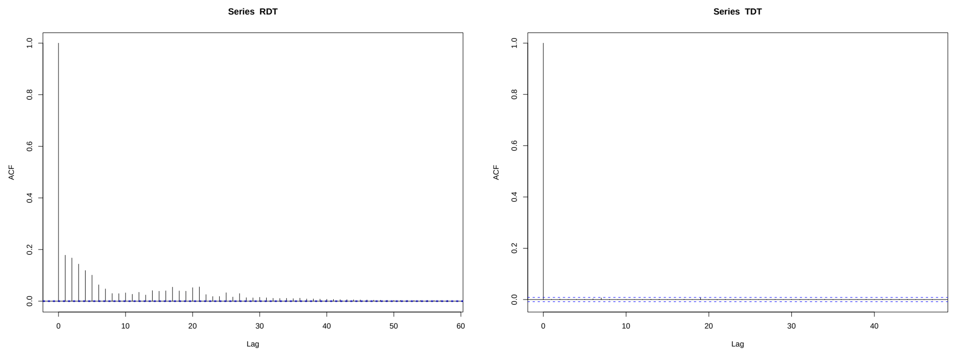

Fig. 5. Autocorrelation function for the object download time and the total download time.

4 Analysis of Individual URLs

Until now we discussed the situation of multiple URLs. Restart resulting in HTTP GET actions to rather randomly selected URLs may be useful for some Internet applications, but, obviously, for the most common, a restart would use the same URL as the previous one. This is certainly true for web browsing. Therefore, we must do the statistical analysis for consecutive requests to the *same* URL, which we did in the second phase of this stage.

Drawing conclusions for the correlation of individual URLs is much more complicated than that for all URLs together. If we consider connection set-up time, about two-thirds of the URLs can be said to be uncorrelated. Of this some exhibit no correlation at all. The majority shows some lags with positive correlation, but we do not think this indicates serious dependence, as we discuss below for one typical example. The correlation exhibited by about one-third of the URLs can be explained in various ways. For some URLs something drastically changed during the experiment, resulting in a shift up or down of all CST values – in such cases correlation is extremely high (this accounts for about one third of correlated URLs). We also identified URLs for which connection set-up times show multi-modality. That is, CSTs are roughly equal to one of several values, possibly because of servers in a server pool with different speed. Finally, we noticed some URLs with periodicity in the results.

As an example, we discuss a typical URL, namely http://www.jp.arm.com/. We refer to this URL as 'URL 29'. Fig. 6 shows the CSTs for URL 29 as well as the autocorrelation function. Note that the confidence interval here is much larger than for earlier shown data, because the number of observations is comparatively low. We see from Fig. 6 that consecutive CSTs to the same URL are surprisingly independent.

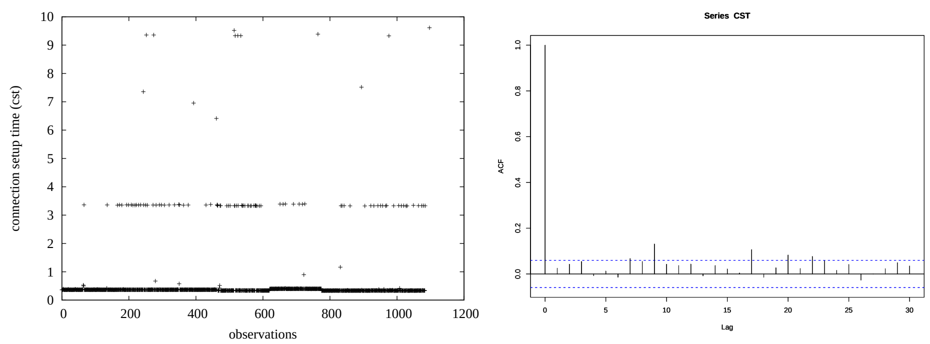

Fig. 6. Connection set-up times and autocorrelation function for URL 29.

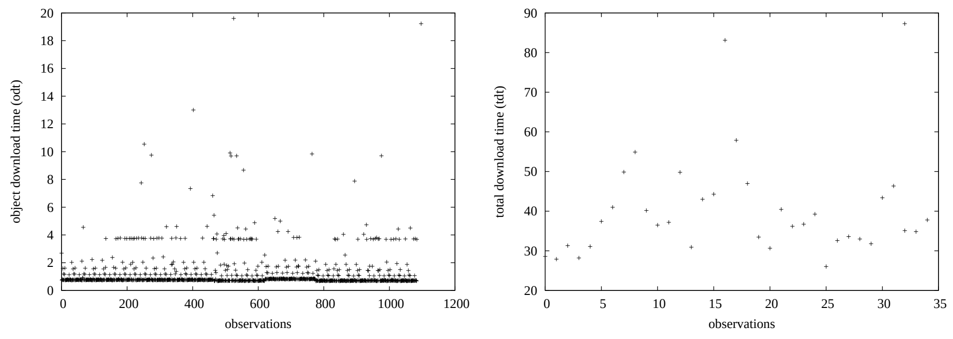

Fig. 7. Object download time and total download time for URL 29.

Fig. 7 shows the object download times and the total download times for requests for URL 29. The autocorrelation function for ODT is similar to that for CST in Fig. 6. We can compute for URL 29 what the optimal restart times are based on the data for CST, ODT and TDT. We find that for connection set up, one would restart every 0.5 seconds, for objects one would also restart every 0.5 seconds, and for total download of URL 29 one would never restart.

If we consider CSTs, it turns out that the dominant reason for which one would do restart is when connection set up fails. For all our URLs an optimal restart time with a value of less than three seconds exists, if the CST includes RTO expirations. In the examples for which the RTO timer never expires, restart pays off far less frequently. In particular, for only about 10 percent of URLs without major correlation, restart is beneficial if no RTO timers expire. We note, however, that these results are very sensitive to the tail of the completion time distribution, and are therefore hard to estimate accurately.

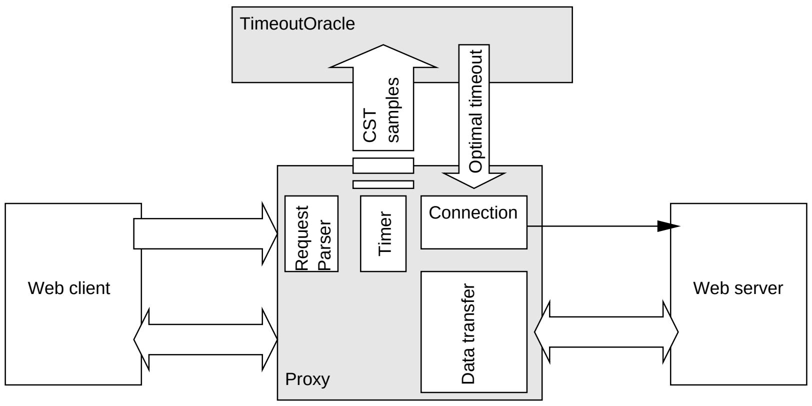

Fig. 8. Proxy design.

5 On-Line Optimization of Restart Times

To determine if restart indeed improves the set up and/or download times for HTTP GET, we need to implement restart and apply it to the URLs of our test set. Therefore, we designed and implemented a client-side HTTP-based proxy capable of dynamically adapting restart times.

5.1 Proxy Design

As depicted in Fig. 8, the restart proxy consists of the core proxy part and a general sample data collection/computation part, which we call the Timeout Oracle. When the proxy, after accepting a request from the client, connects to the appointed server, it measures the time elapsed in the set-up of this TCP connection. Every connection set-up duration is limited by a timeout. In case an attempt takes longer than the timeout, it is aborted and retried, possibly with another timeout. This may repeat up to ten times, after which the proxy gives up.

The CST samples obtained by measuring successful as well as timed-out connection attempts are fed into the Oracle, which in turn provides a new recommendation on an optimal timeout for later connection set-up attempts. It does this based on the sample-based estimators given in Appendix A, which use results from [10,11]. The optimization target is the first moment of connection set up time, and in the formulas we assume unlimited number of retries (we could make this more precise for the limit of 10 retries, but as some results in [11] indicate, one would not expect this to make an important difference). After connection set-up, the proxy simply forwards HTTP requests/replies between server and client. For the moment, we do not implement restart during data transfer, but we hope to do this in the future.

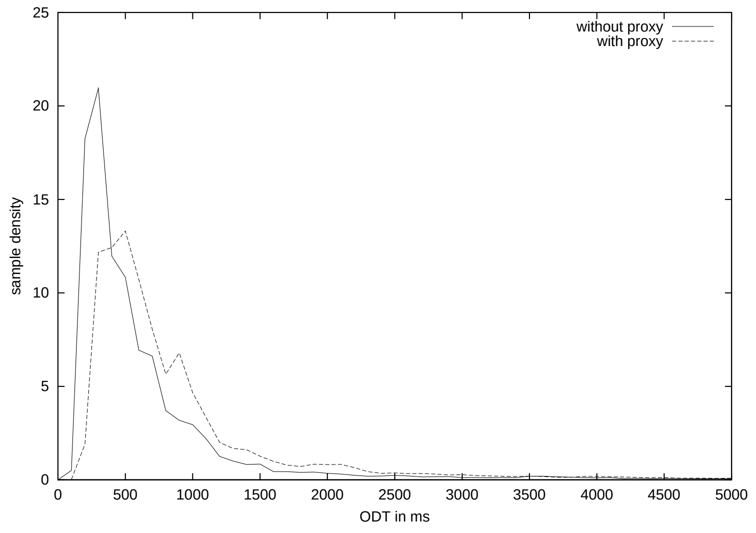

Fig. 9. Comparison of ODT samples with and without the proxy.

Parameters of the Oracle. The Timeout Oracle implements the on-line algorithm as outlined in the appendix. In principle, the Timeout Oracle can be used by any application, thus providing a general optimal timeout computation facility to applications. Its parameters are the maximal timeout (set to 24,000 ms in our experiment), the number of buckets ($H = 100$, see Appendix A) and the penalty, i.e., the time needed for a restart (somewhat arbitrarily chosen to be $c = 100$ms). When computing a new timeout value, the Oracle always uses all samples collected so far. It can be argued that this could potentially lead to slower adaptation of optimal timeout to current samples after a high number of samples have been entered. Tuning the restart time needs to be studied further but is not the topic of this paper.

We tested the usefulness of the proxy in the second stage of our experiments. Therein, we use CST, ODT and TDT as performance metrics: ODT and TDT are measured through *wget* as before, but CST samples are now taken from the proxy. In the case of restarts, all consecutive CSTs during the establishment of a TCP connection are summed to reflect the actually elapsed time.

Before discussing restart for connection set-ups, we look at the overhead the proxy introduces when downloading web pages. The proxy requires setting up an additional connection to the proxy and the proxy also uses CPU cycles to compute the optimal restart time and to handle the request. Fig. 9 gives the overhead for ODT, and the same can be demonstrated for TDT. (Note that the values on the y-axis of Fig. 9 are an artifact of the way we plotted the densities, and have no physical interpretation.) One sees that the overhead is not that

Table 2. Restart results for CST.

URL	Mean CST w/o restart	Mean CST w. restart	# > 3 sec w/o restart	# > 3 sec w. restarts	# RTOs	# Restarts
1	353	320	11	0	11	1
2	233	254	0	0	0	1
3	246	242	0	0	0	1
4	244	232	3	0	3	0
5	376	370	2	0	2	1
6	552	513	16	1	16	5
7	291	290	0	0	0	0
9	729	109	7	1	7	16
10	115	104	9	3	9	11
11	200	197	1	0	1	1
12	193	194	0	0	0	1
13	116	113	9	1	9	19
14	107	115	6	1	6	19
15	113	107	5	0	5	6
16	88	117	0	2	0	12
17	424	438	0	0	0	0
average	274	232	4.3	0.6	4.3	5.9

drastic, so we hope that our experiments with the proxy will be representative for other future implementations of restart, for instance within the web browser.

5.2 Results

Table 2 shows results for the adaptive restart proxy, for CSTs of 16 URLs (URL 8 is omitted in the table since we no longer could connect to it halfway through our experiments). For each URL we have an equal number of samples with and without restart (about 1000 samples for each URL). We see that restart gives clear advantages for the tail of the connection set-up time distribution. In the two middle columns there are many more set-up times of more than 3 seconds without restart than with restart (average of 4.3 per thousand CSTs and 0.6 per thousand CSTs, respectively). In other words, the TCP RTO timer is set too high (at 3 seconds) to be efficient, and our restart mechanism improves considerably on the results with RTO. If we consider the mean set-up time, the difference is much less clear. Interesting are the URLs without RTO time-outs (URLs 2, 3, 7, 12, 16 and 17). We see that for these URLs set-up times with and without restart are roughly equal. The most noticeable exception is URL 16, for which no RTO expires, but for which restart triggers reconnection 12 times. It seems that for this example, the restart time was set too tight, because the average CST with restart is much higher than the average CST without restart.

For URL 9, Table 2 shows that the mean CST without restart is far worse than with restart. This happens when the TCP RTO timer expires several times in a row, increasing to 9, 21, 45 or 93 seconds, subsequently. Our restart mechanism does not increase so drastically (the algorithm discussed in Appendix A does in fact lead to an increase of the restart time, but at a far slower pace than the RTO increase). Of course, we have to realize that in such cases the network might have been down temporarily, also rendering quicker restarts unsuccessful (or even harmful if overload conditions caused the network problems [9]). Such effects can not be traced back using our data sets. Note furthermore that our data comes from the case that only a single application applies restart, leaving the rest of the Internet unchanged. In [9] the authors analyze network effects if multiple agents apply restart simultaneously. Moreover, using our on-line adaptive algorithm, there is more analysis to be done to assure the stability of such a control algorithm. This is all for future work.

In conclusion we see that our experiments indicate that the average connection set-up time changes little with or without restart, but that very long TCP connection set-up times can be avoided using restart.

References

1. H. Alt, L. Guibas, K. Mehlhorn, R. Karp and A. Wigderson, A Method for Obtaining Randomized Algorithms with Small Tail Probabilities, *Algorithmica,* Vol. 16, Nr. 4/5, pp. 543–547, 1996.
2. D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms," in *Proceedings ACM SIGCOMM 2001*, San Diego, CA, USA, Aug. 2001.
3. P. Brockwell and R. Davis, *Time Series: Theory and Methods*, 2nd Edition, Springer Verlag, New York, 1991.
4. P. Chalasani, S. Jha, O. Shehory and K. Sycara, "Query Restart Strategies for Web Agents," in *Proceedings of Agents98,* AAAI Press, 1998.
5. W. Cochran, *Sampling Techniques*, John Wiley, New York, 1977.
6. S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," in *IEEE/ACM Transactions on Networking,* Vol. 7, No. 4, pp. 458–472, 1999.
7. B. Krishnamurthy and J. Rexford, *Web Protocols and Practice,* Addison Wesley, 2001.
8. M. Luby, A. Sinclair and D. Zuckerman, "Optimal Speedup of Las Vegas Algorithms," *Israel Symposium on Theory of Computing Systems,* pp. 128–133, 1993.
9. S. M. Maurer and B. A. Huberman, "Restart strategies and Internet congestion," in *Journal of Economic Dynamics and Control,* vol. 25, pp. 641–654, 2001.
10. A. van Moorsel and K. Wolter, "Optimization of Failure Detection Retry Times," in *Performability workshop*, Monticello, IL, Oct. 2003.
11. A. van Moorsel and K. Wolter, "Analysis and Algorithms for Restart," accepted for publication in *Quantitative Evaluation of Systems, QEST04*, Sep. 2004.
12. R Development Core Team, *R: A Language and Environment for Statistical Computing,* R Foundation for Statistical Computing, Vienna, Austria, http://www.r-project.org, 2003.
13. Y. Ruan, E. Horvitz and H. Kautz, "Restart Policies with Dependence among Runs: A Dynamic Programming Approach," in *Proceedings of the Eight International Conference on Principles and Practice of Constraint Programming,* Ithaca, NY, Sept. 2002.

14. M. Schroeder and L. Buro, “Does the Restart Method Work? Preliminary Results on Efficiency Improvements for Interactions of Web-Agents,” in T. Wagner and O. Rana, editors, *Proceedings of the Workshop on Infrastructure for Agents, MAS, and Scalable MAS at the Conference Autonomous Agents 2001,* Springer Verlag, Montreal, Canada, 2001.

Appendix A. On-Line Determination of Restart Time

Our on-line algorithms attempt to minimize the expected time it takes to set up a TCP connection, by using restart. In terms of job completion time, as used in [11], the ‘job’ thus corresponds to connection set up. If $f(t)$ is the probability density function for the job completion time, then to minimize the expected job completion time E_τ, with restart every τ time units (for as long as a job does not complete), we have the following result [11]:

$$E_\tau = \frac{M(\tau)}{F(\tau)} + \frac{1 - F(\tau)}{F(\tau)}(\tau + c), \tag{1}$$

where F denotes the probability distribution for the time a job can take, that is,

$$F(\tau) = \int_0^\tau f(t)dt, \tag{2}$$

and M denotes the first partial moment of download times:

$$M(\tau) = \int_0^\tau t f(t)dt. \tag{3}$$

Finally, c denotes the time it takes to execute a restart.

In our scalable on-line algorithm, we assume we collect data for a system with some restart time τ set beforehand, out of our control. Based on the collected data, we adapt τ to improve the expectation of the completion time. Of course, if one continues collecting data, the amount of data eventually gets prohibitively large. We therefore keep track of results per ‘bucket,’ that is, we divide the observations over H buckets, each of size $h = \tau/H$, and only keep track of the average return time M_i and number of samples N_i within each interval ($i = 1, 2, \ldots, H$). In the i-th bucket, we thus consider the observations with values in the interval $[(i-1) \cdot h, i \cdot h)$. If we label the observations $t_{i,1} \ldots t_{i,N_i}$, M_i is estimated by:

$$\hat{M_i} = \frac{1}{N_i} \sum_{j=1}^{N_i} t_{i,j}. \tag{4}$$

We also keep count of N_τ, the total number of observations that take at least τ time units. (Note that for these observations a restarts is initiated.) For candidate retry time $\tau_i = i \cdot h$, we then obtain the following estimators for (2) and (3):

$$\hat{F}(\tau_i) = \frac{\sum_{k=1}^{i} N_k}{\sum_{k=1}^{H} N_k + N_\tau}, \tag{5}$$

$$\hat{M}(\tau_i) = \frac{\sum_{k=1}^{i} N_k \cdot \hat{M}_k}{\sum_{k=1}^{i} N_k}. \tag{6}$$

Ultimately, we thus estimate the expected diagnosis time E_{τ_i} by the asymptotically unbiased ratio estimator [5]

$$\hat{E}_{\tau_i} = \frac{\hat{M}(\tau_i)}{\hat{F}(\tau_i)} + \frac{1-\hat{F}(\tau_i)}{\hat{F}(\tau_i)} \cdot (\tau_i + c). \tag{7}$$

The optimal restart time is then obtained by selecting the value of $\tau_i = i \cdot h, i = 1, 2, \ldots, H$, which minimizes (7).

Appendix B. Used Statistics

Since our data did not suggest noteworthy trends or seasonal indications (with exception of the clustering effect discussed in Section 3), we may assume we deal with (weakly) stationary time series, see Paragraph 1.4 of [3]. Stationarity implies that for a time series $\{X_i, i \in \aleph\}$, the second moment of all X_i is finite, the mean value identical for all X_i, and the correlation between X_i and X_{i+k} is independent of i (see Definition 1.3.2 in [3] for a mathematically precise definition.)

The figures in this paper plot the 'ACF' or auto-correlation function, as well as the 'PACF' or partial auto-correlation function. These can directly be obtained using the statistical package 'R' [12], using standard sample-based estimators corresponding to the following definitions of (partial) autocorrelation. Auto-correlation $Corr(X_n, X_{n+k})$ of lag k of a stationary time series $\{X_i, i \in \aleph\}$ is defined as:

$$Corr(k) = \frac{Cov(X_n, X_{n+k})}{Cov(X_n, X_n)} = \frac{E[X_n X_{n+k}] - E[X_n]E[X_{n+k}]}{Var[X_n]}, \tag{8}$$

where, as usual, E is used to denote the expectation of a random variable, while Var denotes variance. Partial autocorrelation can be said to adjust the autocorrelation with lag k for the intervening observations (that is, for those with lesser lag). Compared to the ACF, the PACF indicates if auto-correlation can be explained by the information in the intermediate variables. For the precise definition of PACF, we refer to Definition 3.4.1 of [3]. For consecutive random variables in a time series to be independent, the autocorrelation function must be zero. For the sample-based computation, this translates into the autocorrelation function to be within a confidence interval around zero, as depicted in the various graphs in this paper.