Dynamic Connection Closing Time Selection for HTTP/1.1 Servers

Moncef Elaoud
University of Wisconsin
elaoud@ece.wisc.edu

Cormac J. Sreenan
AT&T Labs-Research
cjs@research.att.com

Parameswaran Ramanathan
University of Wisconsin
parmesh@ece.wisc.edu

Prathima Agrawal
Bellcore
pagrawal@bellcore.com

1 Introduction

Persistent HTTP (HTTP/1.1 [1]) is aimed at improving the throughput and the delay as perceived by a client while decreasing the server load. This is accomplished by allowing a TCP connection between the Web server and a client to persist beyond the lifetime of a single request-reply cycle. Specifically, multiple requests from a client can be pipelined on the same TCP connection resulting in higher throughput and smaller response time.

Since a connection can not be maintained indefinitely, a decision on when a persistent connection is to be closed needs to be made. A fixed timeout period after which a server automatically closes a connection is suggested [1,2]. A quasi-dynamic solution, where two fixed timers are used, is proposed in [2]. Nielsen et. al. [3] suggest a connection to be closed after servicing a fixed number of requests. Version 1.3.1 of the Apache Web server uses a combination of the solutions proposed in [2, 3].

None of these solutions accounts for the variability of the server load during different hours of the day and different days of the week. Adaptability is important since Web servers usually have periods of high load and periods of low load. We propose a solution that uses the inter-arrival patterns of requests at the server as well as the clients’ behavior to balance between the need to maximize the throughput as seen by the clients and the server need to reduce the likelihood of its resource depletion while servicing the largest possible number of clients. Depending on the server’s load, our solution dynamically selects the largest timer without jeopardizing the availability of resources. The timer selection is done using a simple table-driven approach. Our simulation results show that the proposed adaptive timer outperforms the fixed timer solution and a traditional Least Recently Used (LRU) solution. A full description of the treatment of this subject can be found in [4].

2 Dynamic Timer Selection

As mentioned above, the problem with the fixed timer solution is its inability to adapt to the server load. A larger timer value achieves a better client throughput. With as a large timer, TCP gets a chance to build its send window as multiple replies are sent on the same TCP connection. However, more resources are needed with the larger timer as connections are allowed to stay idle.

The proposed solution is composed of two phases: (i) an off-line phase where a function $G(T)$ for each timer value $T$ is calculated depending on clients’ behavior. The values of $G(T)$ are stored in a table indexed by $T$. (ii) a run time phase where periodically depending the server load a timer value is selected based on the precalculated table.

Timer Table Calculation: Given an aggregate distribution of inter-arrival periods between requests from a particular client at the server $F_X(\cdot)$, the quantity $G(T)$ is calculated, as shown below, for each value of the timer $T \in [T_{\text{min}}, T_{\text{max}}]$.

$$G(T) = \frac{T + \tau + E[X/X \leq T] \cdot \frac{F_X(T)}{1-F_X(T)}}{S},$$

where $T_{\text{min}}$ ($T_{\text{max}}$) is the minimum (maximum) value the timer can take, $E[X/X \leq T]$ is the conditional expected value of the inter-arrival period between requests given that the inter-arrival period is less than $T$, $S$ is the maximum number of sockets available at the server, and $\tau$ is duration of TCP’s time-out period. The values of $G(T)$ are stored in a table indexed by the value of $T$. Here we assume that the granularity of the timer is one second. Since in practice the exact distribution $F_X(\cdot)$ is not known, we need to develop a
strategy to estimate it. The client’s inter-request period generally depends on client behavior (think time) as well as the content of the page retrieved. Both of these events do not generally change over short periods of time. Hence an estimate of \( F_X(\cdot) \) can be obtained from previous server’s access logs. For example the distribution may be reestimated at the end of every day or even every week.

**Run Time Phase:** During the run time phase, the server periodically (every \( A \) seconds) estimates the load \( L \) by measuring the number of requests \( N \) that it receives during a period of time \( A \). The load \( L \) is then given by \( L = N/A \). Given a load \( L \), the server picks the largest timer value \( T \) from the table such that: \( G(T) \leq L \).

When the server receives a request from a client \( c \), it checks if client \( c \) has an open connection with the server. In the affirmative, the request is serviced on the same socket, and the socket closing time is reset to the time of the request plus the timer \( T \). However, if client \( c \) does not have an open connection with the server, a new connection is opened if resources are available. The socket closing time is set to the request arrival time plus the value \( T \) of the timer. If no resources are available, the request can either be rejected or queued to wait on resources.

### 3 Results

In the simulation we assumed that the requests that can not be serviced, due to lack of resources, are queued to wait on resources. Therefore instead of measuring connection rejection rate, we measure queuing delays encountered by clients. In addition we measure the average throughput seen by clients for each policy. We compared our proposed solution to 15 second fixed timer (used by Apache 1.3.1) as well as to an LRU scheme. We can see from the Figure 1 that the proposed solution achieves almost double the throughput obtained by the fixed timer during periods of light load (from 12:00 A.M. to noon). Our solution also achieves slightly better throughput than the LRU scheme during those same ours. All three schemes perform equally well during busy (afternoon and evening) hours. Figure 2 shows that the LRU solution accumulates very large queuing delays during busy hours. Both the fixed timer and the proposed solution perform equally well with respect to the queuing delay performance metric. Therefore, we conclude that the proposed solution achieves better client throughput and smaller queuing delays than the fixed timer and LRU schemes during most of the day. A full presentation of the results can be found in [4].

### References


