Implement lower timer granularity for retransmission of TCP

How a timer wheel algorithm can reduce overhead

Venkat Venkatsubra
G Shantala

Reduce the overhead of per-tick processing with a timer wheel algorithm that implements the retransmission timer. The AIX® Transmission Control Protocol (TCP) has seven timers (per-connection) and uses global timer functions with two granularities to implement the timers. In this article, learn how to get lower granularity with your retransmission timer by using the AIX TCP fast timer, and discover other advantages of lower timer granularity.

Introduction

AIX® Transmission Control Protocol (TCP) maintains seven timers for each connection:

- Connection establishment
- Retransmission
- Delayed acknowledgment (ACK)
- Persist
- Keepalive
- FIN_WAIT_2
- TIME_WAIT

To implement these per-connection timers, TCP uses global timer functions that offer two granularities:

- `tcp_fasttimo`: It's invoked every 200ms and implements the fast timer.
- `tcp_slowtimo`: It's invoked every 500ms and implements the slow timer.

With TCP's retransmission timer implementation, the retransmission timeout is stored in units of ticks in TCP's control block, where one tick = 500ms. When TCP's slow timer expires every 500ms, `tcp_slowtimo` gets called. This routine walks through the table of TCP control blocks and, for each connection, it decrements the timeout, specified in ticks, by one. When the number of
ticks reaches zero for a timer, it calls the timeout handler routine to handle the corresponding timeout for that connection. This implementation imposes a lower limit of 500ms for a TCP timer.

As you know, TCP dynamically calculates timeout based on the round-trip time measured by TCP. Currently, TCP retransmission timeout takes on a minimum value of three seconds. However, with high-speed networks, such as Gigabit Ethernet and 10 Gigabit Ethernet, the round-trip time (hence retransmission timeout) is expected to be much lower. You lose 120MB per second of throughput for each second you are not transmitting. A better method is needed to deal with high-speed and low-latency networks.

One way to achieve lower granularity with a retransmission timer is to use TCP’s fast timer (200ms), which could default to 50ms instead of 200ms. The drawback with this approach is handling the overhead of scanning all the protocol control blocks (PCBs) a higher rate. A PCB is an internal structure that holds the control information for a connection. This article explains an alternative method for implementing TCP’s retransmission timer based on a timer wheel algorithm.

**Timer wheel algorithm**

A timing wheel has $N$ number of slots. A slot represents a time unit such as $s_i$ (slot interval). A cursor in the timing wheel moves one location every time unit, just like a second hand on a clock. Whenever a cursor moves to a slot, for example, $c_s$ (current slot), it implies that the list of timers in that slot, if any, expire at that instant or when the cursor reaches the same slot in the subsequent cycles.

When a new timer with a timer interval, such as $t_i$ (time interval), is to be added to this wheel, the slot for the new timer, $t_s$ (timer slot), is calculated, as follows:

$$t_s = \left( c_s + \frac{t_i}{s_i} \right) \mod N$$

Assume that the maximum timer interval value any timer takes on does not exceed an upper limit ($t_{\text{max}}$). If $N$ is large enough to accommodate $t_{\text{max}}$ within a rotation from the current cursor position, then when the cursor moves to a specific slot, all timers in that slot expire at the same instant (no subsequent cycles). This saves you from traversing the list to check which of the timers expire now and which of them expire in the subsequent cycles.

For example, in the timing wheel shown in **Figure 1** below, there are 8 slots, 0 through 7. You see timer entries pivoted to slots 1, 3, 5, 4, and 6. Currently, the cursor is at slot 1. The three timers pivoted to this slot expire at this instant or when the cursor reaches the same slot in the subsequent cycle.

**Figure 1. Timing wheel**
RTO implementation using the timer wheel algorithm

This section explains a retransmission time-out (RTO) implementation using a timer wheel algorithm on AIX. On AIX, the number of slots in the timer wheel is chosen as N=7000. The slot interval \( (s_i) \) of the timer wheel implementation is configurable using the no option with \texttt{timer\_wheel\_tick}. The range of values for \texttt{timer\_wheel\_tick} is zero to 100 ticks, where one tick = 10ms. Thus, if \texttt{timer\_wheel\_tick} is set to one, for example 10ms, the timer-period per cycle is \( N \times s_i = 70 \) seconds. Since the maximum value \( (t_{max}) \) for the AIX TCP RTO is 64 seconds, this is large enough to ensure that all retransmission timers in a slot expire at the same time and not in the subsequent cycles.

The RTO value is configurable using the no option with \texttt{tcp\_low\_rto}. The range of values for \texttt{tcp\_low\_rto} is zero to 3000 ms. If configured, the RTO value would be set as the initial retransmission timeout for all the TCP connections on the system.

Even though \texttt{tcp\_low\_rto} is configured, the algorithm takes effect for a connection only when it experiences packet drops, as shown in Figure 2 below. It is not efficient to use this algorithm otherwise. In the traditional method of retransmission timer implementation for connections that do not experience packet drops, all you do is set the number of ticks in TCP’s control block to start a retransmission timer. When the acknowledgment arrives, the retransmission timer is stopped by setting the value to zero. This method works best for start or stop cases where the retransmission timer never expires.

You can continue to use the traditional method of retransmission timer implementation until either of these conditions occurs:

- Retransmission timer expires
- Fast-retransmit phase is turned on for a connection

Figure 2. Low granularity connection

A. Trigger low RTO feature

Retransmission timer expires for a connection for the first time or the fast retransmit phase is turned on. You enable lower granularity for the retransmission timer for this connection and
 resend the packet. Since this is a retransmission, you do not time this segment for response time tracking (RTT) calculation. The retransmission timer is started by inserting the timer into an appropriate slot in the timer wheel.

**B. Receipt of acknowledgment**
Acknowledgment for the retransmitted packet is received. Cancel the retransmission timer (remove the timer from the timer wheel).

**C. Time a segment**
A segment is sent with sequence \texttt{b} and the retransmission timer is started. You time this segment for RTT calculation by storing the timestamp. The retransmission timer is started by inserting the timer into an appropriate slot in the timer wheel.

**D. Process ACK received for a timed segment**
An ACK with acknowledgment \texttt{c} greater than \texttt{b} is received. You stop the retransmission timer by removing the timer from the timer wheel. You measure new RTT as current time minus the stored timestamp when the segment with sequence \texttt{b} was sent. The measured RTT is used to compute the new retransmission timeout value.

**E. Transmit a segment with RTO calculated as per low RTO feature**
The retransmission timer is inserted into an appropriate slot in the timer wheel.

**F. Retransmission timer expiry**
Retransmission timer expires for a connection that is using the timer wheel algorithm. The cursor moves to the slot in the timer wheel where the retransmission timer was inserted previously. A timer handling routine is called to process the retransmission timer expiry.

**Summary**

With the timer wheel approach, starting and stopping a timer becomes expensive. If the timers in a slot are implemented as linked lists, then the addition or deletion of a timer to or from the list has to be synchronized using a slot chain lock.

On the other hand, using a timer wheel algorithm for implementing a retransmission timer provides the advantage of reduced overhead of per-tick processing. With a timer wheel, you have only those timers in a slot that expire when the cursor reaches the slot. You do not need to scan through all PCBs to check which of the timers are set for each connection.

The lower timer granularity implementation lets the administrator configure an RTO value as low as 10ms. This is particularly useful for TCP connections running over high-speed and low latency networks, such as Gigabit and 10 Gigabit Ethernet, that experience occasional packet drops.
Related topics

- **AIX and UNIX®**: The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- **AIX Wikis**: Discover a collaborative environment for technical information related to AIX.
- Search the AIX and UNIX library by topic:
  - System administration
  - Application development
  - Performance
  - Porting
  - Security
  - Tips
  - Tools and utilities
  - Java™ technology
  - Linux®
  - Open source
- **IBM® trial software**: Build your next development project with software for download directly from developerWorks.

© Copyright IBM Corporation 2007

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)