May 2020

SCALABLE AND EFFICIENT REAL-TIME CLOCK MAINTENANCE IN A DISTRIBUTED NETWORK

Venkat Reddy Ennu

SCALABLE AND EFFICIENT REAL-TIME CLOCK MAINTENANCE IN A DISTRIBUTED NETWORK

AUTHOR:

Venkat Reddy Ennu

ABSTRACT

Techniques are described herein for maintaining real-time software clocks in a distributed network. Generally, the techniques may approximate a synchronized timestamp for a delay request sent by a slave node based on a timestamp of a sync request received at the slave node. Then, a time stamping unit associated with the slave node may update the correction factor of the timestamp to adjust the synchronized timestamp appropriately. Notably, this avoids the need to read timestamps from first in, first out buffers of time stamping units and/or from field programmable gate arrays (FGPAs) and, thus, may significantly improve the scale of 1588 sessions while preserving slave node resources.

DETAILED DESCRIPTION

Institute of Electrical and Electronics Engineers (IEEE) 1588 is a protocol designed to synchronize real-time clocks in the nodes of a distributed network. Typically, nodes are organized into a master-slave hierarchy and slave nodes are synchronized to master nodes using delay request-response or peer delay mechanisms. With delay request-response, a time stamping unit (TSU) of a slave node may store a synchronized timestamp ("T3") that is synchronized to a timestamp of a master node ("T1") based on a determined delay between a timestamp of a sync request received at the slave node ("T2") and timestamp T1. The TSUs associated with slave nodes often store T3 timestamps in first in, first out (FIFO) data structures, such as a FIFO buffer. Then, software executing on the slave node has to periodically read the FIFO data structure to match T3 timestamps with metadata from the TSU with a delayed response timestamp ("T4"). However, these periodic T3 FIFO reads limit the scale of 1588 sessions at least because FIFO data structures can be

small and FIFO reads can be slow. Among other issues, this can cause process context switches.

In some instances, a transparent clock (TC) functionality of TSUs can be leveraged to avoid reading T3 FIFO from the TSU. However, in TC mode, a central processing unit (CPU) of a slave node needs to timestamp packets and a TSU needs to update the correction factor (CF) field of a precision time protocol (PTP) packet with "time taken for packet to reach from CPU to egress of TSU." This requires the CPU to maintain its own time of day (TOD) counter in synchronization with the TSU's TOD counter (e.g., via a mechanism that performs periodic synchronizations) with software executing timers to maintain full 80 bit 1588 counters. Alternatively, a TOD counter can be maintained in a field programmable gate array (FPGA) of a slave CPU that is synchronized with the TSU's TOD counter. Then, software can read the TOD from the FGPA every time a timestamped PTP packet needs to be sent out from the slave CPU. However, both of these approaches limit the scale of a 1588 session (especially if a TSU only provides Management Data Input/Output (MDIO) access). In fact, the latter approach might reduce the scale of 1588 sessions more drastically than the former approach because the FPGA will be read to get the TOD for every 1588 event packet. The latter approach also unwantedly uses FGPA resources.

The techniques presented herein resolve these issues and increase 1588 session scale. The techniques are illustrated in Figure 1 below. Overall, the techniques approximate a T3 timestamp (i.e., a synchronized timestamp for a delay request sent by a slave node) based on a T2 timestamp (i.e., a timestamp of a sync request received at the slave node) and use CF data from a TSU to adjust the T3 timestamp appropriately. This avoids the needs for T3 FIFO and FGPA reads that limit scaling.

More specifically, and as can be seen in FIG. 1 below, when a master TSU associated with a master CPU (i.e., a master node) generates a sync packet, the packet is timestamped with time T1 and sent to a slave TSU associated with a slave CPU (i.e., a slave node). The slave CPU saves the T2 timestamp (timestamp of received sync packet) received from its TSU in its 1588 protocol stack, and continues overwriting the same value, referred to herein as variable "X," every time new a new T2 timestamp is received.

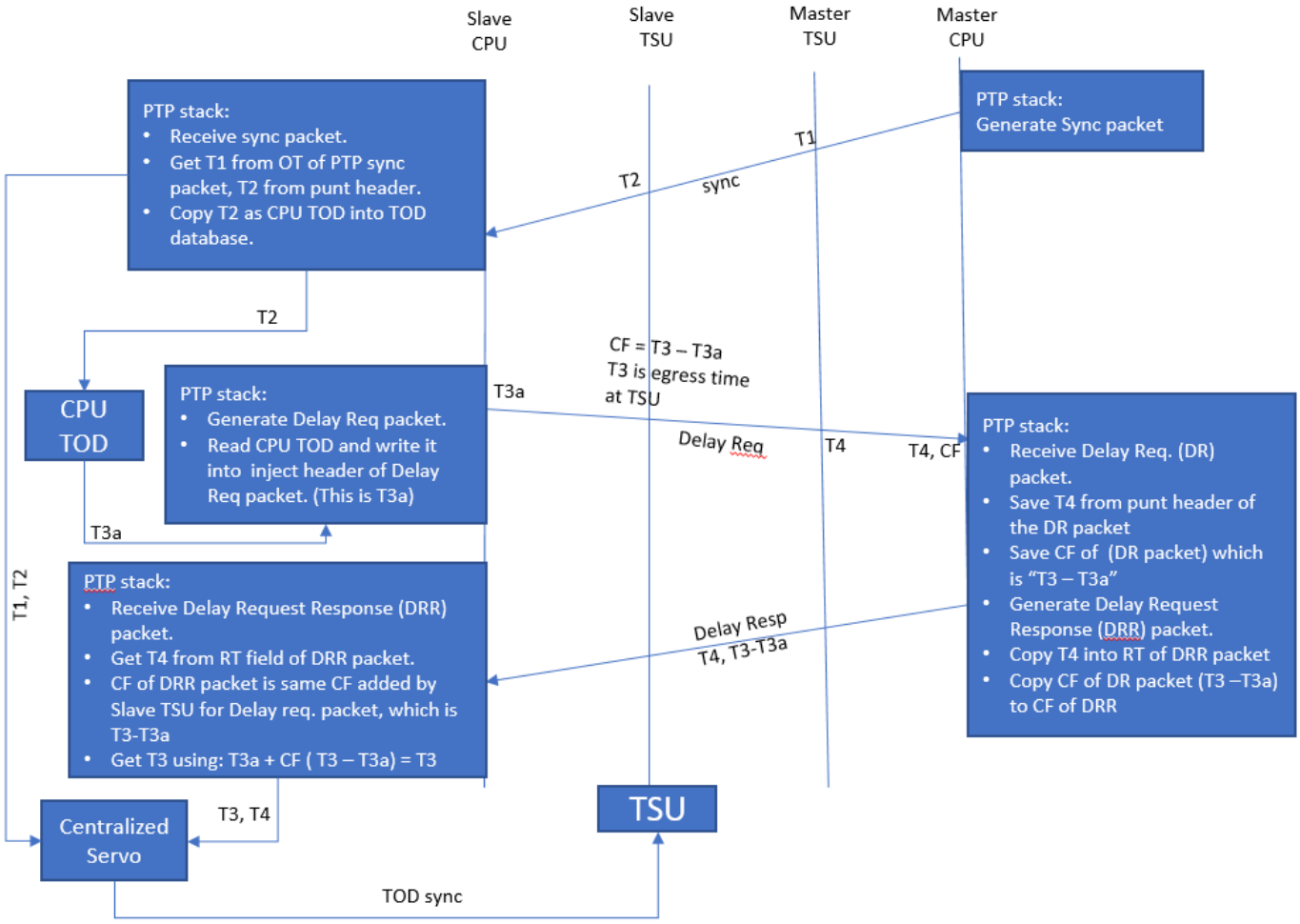

Figure 1

The "X" value (e.g., T3a) is used to timestamp packets sent out from this 1588 protocol stack. Thus, all event packets, including delay request packets, sent from the slave CPU between updates of "X" will have the same timestamp. For example, if the slave session is run at 16 packets per second (pps), "X" value will be updated approximately every 62.5 milliseconds (ms) and the CPU TOD may lag the TSU's TOD by approximately 62.5 ms in a worst case scenario (and, if more than one slave session is running, X gets updated more frequently and will reduce the maximum lag time). However, with the techniques presented herein, the slave CPU's TOD need not be very accurate (e.g., need not be as accurate as the slave TSU's TOD) because the slave TSU will update the CF of PTP packet based on the TSU TOD. For example, if the TOD difference between the slave TSU and "X + time taken for packet to reach egress of TSU after its timestamped in CPU"

is less than 500ms, the slave TSU can update the CF with proper handling of wrap around conditions.

Since delay request packets can be timestamped based on CPU TOD at the slave CPU 1588 protocol stack (e.g., timestamped T3a instead of T3) the slave CPU need not read a timestamp from the FIFO of the slave TSU. Instead, per IEEE 1588, the master node will copy CFs of delay request packets, which allows the slave CPU to receive the accurate timestamp T3 in a delay request response (along with a timestamp T4 of the time the delay request was received at the master TSU). Specifically, the slave CPU can determine T3 by combining timestamp T3a with the CF (e.g. delta of timestamp T3 and T3a) received in a delay response (e.g., T3 = T3a + CF = T3a + (T3-T3a)).

If, at some point, the distributed network experiences a TOD update, a clock servomechanism (servo) running on the slave node detects it and updates TSUs. Then, the CPU TOD will be stale until the next sync packet arrives, since the CPU will continue to use the TOD of the last received sync packet's T2 (before TOD is updated by servo). This may cause the slave CPU's X (e.g., timestamp T3a) and the slave TSU's TOD to differ by more than one second. This can cause boundary condition detection issues, render the CF improper. However, the techniques presented herein avoid these issues by programming the servo to avoid timestamps after TOD change for few seconds. Consequently, packets with improper timestamps will be ignored by servo.

Overall, with these techniques avoid reading the TOD of timestamp T3 (the timestamp of a delay request packet) at TSU FIFO data structures. Likewise, the techniques negate the need for timestamps to be read from FPGA, which helps save FPGA resources while also ensuring the techniques are operable when an FPGA is not connected with required 1588 clocks, operating at 1pps, etc. Additionally, the techniques may reduce the load on a slave CPU at least because the slave CPU does not need to synchronize its TOD counter the TOD counter of the slave TSU. Collectively, these advantages significantly improve 1588 session scale.

In summary, the techniques presented herein maintain real-time software clocks in a distributed network by approximating a synchronized timestamp for a delay request sent by a slave node based on a timestamp of a sync request received at the slave node. Then, a TSU associated with the slave node may update the correction factor of the timestamp to

adjust the synchronized timestamp appropriately. Notably, this avoids the need to read timestamps from FIFO data structures and/or FGPAs and, thus, may significantly improve the scale of 1588 sessions while preserving slave node resources.