An IEEE-1588 Compatible RADclock

Matthew Davis¹, Benjamin Villain², Julien Ridoux¹, Anne-Cécile Orgerie³, Darryl Veitch¹

¹ Electrical & Electronic Engineering Dpt, The University of Melbourne, Australia
{matt, julien}@synclab.org, dveitch@unimelb.edu.au

² Université de Nantes, IRCCyN UMR CNRS 6597, Nantes, France
benjamin.villain@gmail.com

³ Ecole Normale Supérieure, Lyon, France
annececile.orgerie@ens-lyon.fr

This work was supported in part by a research grant from Symmetricom Inc.
Introduction

Present some new features and recent work on RADclock

- Robust Absolute and Difference clock
- Software clock that relies on a feed-forward paradigm

RADclock aims at being an “ideal” software clock, capable to:

- Use any synchronization protocol (IEEE 1588, NTP)
- Use software timestamps but be reliable and accurate
- Use NIC hardware timestamps when available
- Be robust to latency variability of network and OS
- Be robust to high system load (consistent performance)
Motivation

Previous work
- ISPCS 2008, RADclock (NTP) vs. ptpd (IEEE 1588) perfs
- Demonstrated impact of network delay on clock performance
- Advocated for a robust synchronization algorithm to filter noise

Purpose of this talk
- RADclock and ptpd have improved, work should be revisited
- Increasing hardware timestamping support on commodity NICs
- Compare to commercial solution (TimeKeeper by FSMLabs)
## Comparing Contenders

<table>
<thead>
<tr>
<th></th>
<th>ntpd</th>
<th>ptpd</th>
<th>TimeKeeper</th>
<th>RADclock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Linux</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>*BSD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NTP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IEEE 1588</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>S/W timestamps</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>H/W timestamps</td>
<td>✓</td>
<td>✓*</td>
<td>✓</td>
<td>✓*</td>
</tr>
</tbody>
</table>

- Not possible to compare all solutions across all dimensions
- Present most interesting comparisons only (using IEEE 1588)
RADclock IEEE 1588 Support

Early Support / Proof of Concept

- Runs as slave only (reasonable for software clock)
- End-To-End only
- No support for Announce, Signalling or Management Message

Use 1588 Packets as Needed by Feed-Forward Algorithm

![Diagram showing the interaction between the master and slave with labels for SYNC, FOLLOW_UP, DELAY_REQ, and DELAY_RESP at time points t1, t2, t3, t4, and t5.]

- DELAY_REQ and DELAY_RESP only
- Use bi-directional paradigm, for reliable RTT based filtering
- Ignore SYNC and FOLLOW_UP
- Much less input data than ptpd or TimeKeeper
Comparison Methodology

- DAG card provides the hardware time reference
- Compare two clocks at a time, each against DAG timestamps
- Clocks timestamp UDP test packets (almost) simultaneously
  - BPF/libpcap timestamps if we are conducting a software timestamping experiment
  - Hardware timestamps if available
Clocks Share:

- Host load and latency
- Network conditions
- Time server quality

...but attain timestamps from different locations

- `ntpd SW`: Userland
- `ptpd SW`: SO_TIMESTAMP (socket layer)
- `TimeKeeper HW`: NIC timestamps converted by kernel
- `TimeKeeper SW`: ???
- `RADclock SW`: BPF/libpcap timestamps
- `RADclock HW`: NIC oscillator RAW counter
We modified the Intel i350 NIC driver (Linux)

- Export raw i350 time counter from the NIC up to userland
- This hardware value can be read from userland via the Linux socket API
Experiments

- Clocks converged for 2+ hours then 2 hour stress period
- Stress period was a large data transfer across 100 Mbps hub
  - Transfer rate capped at 45Mbps
  - RTT of timing packets increases with large outliers

<table>
<thead>
<tr>
<th></th>
<th>NTP</th>
<th>PTP SW</th>
<th>PTP HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>90th prctile [µs]</td>
<td>480</td>
<td>340</td>
<td>240</td>
</tr>
<tr>
<td>90th pctlile stress [ms]</td>
<td>10.7</td>
<td>10.8</td>
<td>0.40</td>
</tr>
</tbody>
</table>
**RADclock (NTP) vs. timekeeper (NTP) on Linux**

- **RADclock**’s median error is consistent with the network asymmetry seen by the DAG
- **timekeeper** affected by stress, but quickly recovers
**RADclock (NTP) vs. ptpd (1588) on Linux**

- **RADclock** remains stable despite the network stress
- **ptpd** demonstrates a considerable level shift (2 ms) during the stress period
**RADclock (NTP) vs. ptpd (1588) on BSD**

- **ptpd** keeps accumulating error after stress period
- **RADclock** hardly affected by stress
**RADclock (1588) vs. ptpd (1588) on Linux**

- **RADclock** comparable to **ptpd** during non-stressed periods
- Both affected by stress, **RADclock** discards input (implementation problem)
**RADclock (1588, hardware)** vs. **timekeeper (1588, hardware)** on Linux

- **timekeeper** outperforms **RADclock** during non-stress period
- **timekeeper** affected by stress (IQR from 2 $\mu$s to 495 $\mu$s)
- Oscillations of **RADclock** due to server-room air conditioning and feed-forward algorithm sliding windows
• We investigated other client software solutions and compared them against *RADclock*

• *RADclock* shows high stability during periods of stress

http://www.synclab.org/radclock/