The Cost of Variability
(or the importance of a robust servo)

Julien Ridoux  jridoux@unimelb.edu.au
Darryl Veitch  dveitch@unimelb.edu.au
Introduction

- **A perfect world**
  - Constant delays
  - Symmetric paths
  - No disconnection, no power outage
  - No change of network topology

- **Distribution of time becomes extremely easy**

- **Of course the reality is slightly different**
  - Network heterogeneity
  - Cheap, low quality, incompatible equipment
  - Evil IT guys
Why we should care about delays

- **Even for hardware solutions latency is an issue**
  - Variable delays are unavoidable (hence transparent clocks)
  - Lower quality hardware
  - Unexpected events (congestion, disruption, ...)

- **Software component can be introduced**
  - If slave does not require sub-microsecond accuracy
  - But still benefits from gold class IEEE 1588 Master clock

- **Servo design is crucial when facing destabilising events**
  - We compare PTPd against TSCclock
    - Ideal conditions, Network Congestion, Disruption event
    - Focus on variability and robustness
  - Advocate for robust and stable servo design
    - Feedback vs. Feedforward design
    - Robust delay filtering
PTPd

- **Software implementation of IEEE 1588**
  - [http://ptpd.sourceforge.net](http://ptpd.sourceforge.net)
    - Correll et al., “Design Considerations for Software only Implementations of the IEEE 1588 Precision Time Protocol”, ISPCS 2005
  - Easy to install (Thanks!)

- **Servo used is a Proportional-Integral (PI) controller**
  - Feedback algorithm
  - Tries to correct clock rate and clock offset simultaneously
  - Difficult “tuning” between fast convergence and oscillations
    - Short term tracking vs. rate stability
  - Stability not guaranteed in noisy environments
The TSCclock

- The TSCclock is
  - the result of a long lasting research effort at the UoM
  - a set of algorithms and filtering tools
  - a software implementation of a Robust Absolute and Difference clock (RADclock)

- Main objective: replace the ntpd daemon
  - Works on LAN and WAN
  - Designed for noisy environments, emphasis on robustness

- The TSCclock is complementary to IEEE 1588
  - is a servo controller, not a protocol
  - can be used with different master clocks
The TSCclock

- **TSCclock uses the CPU cycle Time-Stamp Counter (TSC)**
  - Runs on common hardware PC architecture

- **Feedforward algorithm**
  - No feedback loop “locking onto” the input signal
  - Estimate the long-term average oscillator rate
    - Non-Linear filtering
    - High stability
  - Tracks drift continuously
    - Not fed back into the filtering and rate estimate
    - Correction applied when needed

![Graph showing offset estimates over time](image)
Testbed

Comparison of each clock against DAG

Internal Monitor

External Monitor

Host

SW-GPS

SW-NTP

TSCclock

PTPd

Atomic Clock

DAG-GPS

Unix PC

SW-GPS

PTPd

NTP Server

Stratum 1

GPS Receiver

UDP Sender & Receiver

PPS Sync.  NTP flow  IEEE 1588 flow  UDP flow  Time Request
Fair comparison

- **Same hardware support for each solution**
  - Clocks run in parallel
  - Same system noise
  - Same temperature environment

- **Use of defaults parameters**
  - No tuning of the servo
    - IEEE 1588 objective: "The default behavior of the protocol will allow simple systems to be installed and operated without requiring the administrative attention of users"
  - Shared master polling frequency
    - Period = 16 seconds
Ideal conditions

- **Focus on stability / robustness of clocks**
  - IQR of PTPd error larger than the TSCclock one
  - Both clock suffer from higher system noise in FreeBSD host

- **Average error not the focus of the talk**
  - Asymmetry issues (see later)

### Linux

![Clock Error Graph](image)

- **PTPd**
  - Median: 59.3
  - IQR: 31.6

- **TSCclock**
  - Median: 38.3
  - IQR: 8.1

### FreeBSD

![Clock Error Graph](image)

- **PTPd**
  - Median: 23.5
  - IQR: 34.9

- **TSCclock**
  - Median: 50.7
  - IQR: 17.7
Linux Client
- Starts alone on network

Startup convergence
- PTPd: 90 minutes
- TSCclock: ~immediate
System / Hardware imperfection

- **FreeBSD client**
  - Unintentional events
  - System specific
  - Unusual large noise

- **Reaction from clocks:**
  - PTPd: large jump and oscillation
  - TSCclock: unaffected
Network Congestion

- Add extra host to create cross-traffic

- Bi-directional SCP transfers
  - Each capped 15 Mbit/s
  - 750 MB files (~7mn)
  - Pause 10 sec
  - Monitoring hub 100Mb/s
Network Congestion

- **Linux: Network congestion only**
  - Large impact on PTPd
  - TSCclock barely affected (robustness)

- **FreeBSD: Network and Host congestion**
  - Impact on PTPd even worse
  - TSCclock still not affected

<table>
<thead>
<tr>
<th></th>
<th>PTPd</th>
<th>TSCclock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>31.6 µs</td>
<td>8.1 µs</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>34.9 µs</td>
<td>17.7 µs</td>
</tr>
</tbody>
</table>

Previous IQR (ideal conditions)

**Linux**

**FreeBSD**
Disconnection scenario

- Back to ideal conditions
- Let the clocks converge
- Disconnect from the time server
## Disconnection

Disconnected for 2 hours
- PTPd: local sudden change with worse error -300 ms !!
- TSCclock: local stability and gradual drift of the oscillator

## Reconnection
- PTPd: needs more than an hour to recover, plus oscillation
- TSCclock: ~instant recovery
Disconnected

Disconnected for 2 hours
- PTPd: local sudden change with worse error -300 ms !!
- TSCclock: local stability and gradual drift of the oscillator

Reconnection
- PTPd: needs more than an hour to recover, plus oscillation
- TSCclock: ~instant recovery
**Disconnected for 2 hours**
- PTPd: local sudden change with worse error -300 ms !!
- TSCclock: local stability and gradual drift of the oscillator

**Reconnection**
- PTPd: needs more than an hour to recover, plus oscillation
- TSCclock: ~instant recovery
Conclusion

Direct results from the comparison
- TSCclock much less variable than PTPd under ideal conditions
- TSCclock more robust than PTPd under noisy conditions
  - System hiccups, congestion, disconnection
  - Cause can be traced to servo design

More generally
- IEEE 1588 implementations may suffer from variability
  - Could use transparent clocks ($$)
- Latency variability of a component on the path puts servo at risk
  - Expensive master / transparent clock pointless if slave servo not robust
  - PI controllers may perform badly with variable latencies
  - A feedforward servo (TSCclock)
    - offers greater stability and performance
    - suitable for hardware and software implementations of IEEE 1588