Formal Verification of Clock Synchronization Algorithms

Wilfried Steiner
Outline

• Introduction including Use Cases
• Formal Verification of the Clock Synchronization Algorithm
• Diagnosis to improve Clock Synchronization Quality
• Layered Clock Rate Correction
• Summary and Outlook
• Backup
  • Formal Verification of the Permanence Function
  • Formal Verification of the Compression Master Function
  • Formal Verification of the Startup/Restart Protocol
Introduction including Use Cases
Asynchronous Communication (RC, BE)

- Transmission Points in Time are not predictable
  - Transmission Latency and Jitter accumulate
  - Number of Hops has a significant impact
- Usually solved by High Wire-Speeds & Low Utilization
- Problem of “Indeterminism” remains
Clock Synchronization

Time Master

Enabler for Synchronous Comm.:
- Synchronized Global Time
- Communication Schedule

Ensuring Reliable Networks
Time-Triggered Operation

Time-Division Multiple-Access Communication
Composable network
Complexity reduction and faster integration
Fault tolerant communication system

<table>
<thead>
<tr>
<th>Node A</th>
<th>send</th>
<th>receive</th>
<th>receive</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node B</th>
<th>receive</th>
<th>send</th>
<th>receive</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node C</th>
<th>receive</th>
<th>receive</th>
<th>send</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
<td></td>
</tr>
</tbody>
</table>

Slot
Synchronous Communication (TT)

Exactly one order of messages $M_i$ (in contrast to $\text{PERM}(M_i)$ in async. comm)
Integrated Dataflow Example

Dataflow – Integration
- Time-Triggered (TT)
- Rate-Constrained (RC)
- Standard Ethernet (BE)

TTEthernet Switch is also capable of changing traffic types, e.g. a message received as RC can be relayed as TT
The Future of Human Space Exploration
NASA’s Building Blocks to Mars

Ensuring Reliable Networks

Pushing the boundaries in cis-lunar space

Developing planetary independence by exploring Mars, its moons, and other deep space destinations

U.S. companies provide affordable access to low Earth orbit

Mastering the fundamentals aboard the International Space Station

The next step: traveling beyond low-Earth orbit with the Space Launch System rocket and Orion crew capsule

Missions: 6 to 12 months
Return: hours
Earth Reliant

Missions: 1 month up to 12 months
Return: days
Proving Ground

Missions: 2 to 3 years
Return: months
Earth Independent

Developing planetary independence by exploring Mars, its moons, and other deep space destinations
America’s New Rocket:
Space Launch System
Crew Module
Functional Testing Underway; On Track for May Delivery
Time Triggered Gigabit Ethernet
The Backbone of Orion’s State of the Art, High Reliability Avionics System

48 Network end points  3 planes of connectivity for every device
Orion Network Unification

- The original Orion architecture contained two Ethernet-based data networks:
  - A flight-critical control bus that handled time-sensitive and/or safety critical commands
  - A general-purpose data bus that handled non-critical traffic such as video and personal crew equipment
Orion Network Unification

- However, to reduce vehicle size, weight, and power while maintaining acceptable reliability, Orion collapsed both networks into one, TTEthernet-based infrastructure.

- Critical or time-sensitive data utilized time-triggered or rate-constrained TTEthernet traffic classes.
- Video and personal crew data utilized the best effort TTEthernet traffic class.
This year NASA will fly a spacecraft built for humans farther than any has traveled in over 40 years.

2 Orbits | 20,000 MPH entry | 3,600 Mile Apogee | 28.6 Deg Inclination

Launched Dec/05, 2014 !!

EFT-1 WILL EXERCISE 10 TOP LOSS OF CREW RISKS
Development Agreement Between TTTech and Airbus Safran Launchers

July 08, 2016

TTTech recently signed a long-term development agreement with European space market leader Airbus Safran Launchers to utilize TTEthernet as avionics backbone in the Ariane 6 family of launchers.

Main elements of the contract between Airbus Safran Launchers and TTTech include product lifetime technical support, software development (firmware, tooling), and co-funding for the production and qualification of 100/1000 Mbps TTEthernet semiconductors intended to be used in all major subsystems of the launcher. The declared goal of Safran Launchers’ parent company and TTTech is a wider usage of Deterministic technology in various different spacecraft (including satellites) in order to simplify development, maximize reuse, and thus reduce system lifecycle costs across space programs. The radiation-tolerant components support both strictly deterministic (Time-Triggered Ethernet and ARINC A664 compliant) as well as standard, deterministic variants.
Clock Synchronization

Enabler for Synchronous Comm.:
- Synchronized Global Time
- Communication Schedule

Time Master
Fault-tolerant synchronization services are needed for establishing a robust global time base.
Failure Model for High-Integrity Components: Inconsistent-Omission Faulty

Core COM/MON Assumptions:
- COM and MON fail independently
- MON can intercept a faulty message produced by the COM
- COM cannot produce a valid message such that this message appears as two different messages on listen_out and OUT, though it may be valid on listen_out but detectable faulty on OUT or vice versa
- MON cannot itself generate a faulty message, neither by inverting listen_out to an output, nor by toggling the intercept signal
Clock Synchronization Service is executed during normal operation mode to keep the local clocks synchronized to each other.

Startup/Restart Service is executed to reach an initial synchronization of the local clocks in the system.

Integration/Reintegration Service is used for components to join an already synchronized system.

Clique Detection Services are used to detect loss of synchronization and establishment of disjoint sets of synchronized components.
Formal Verification Activities

TTEthernet Executable Formal Specification

- Using symbolic and bounded model checkers *sal-smc* and *sal-bmc*
- Focus on Interoperation of Synchronization Services (Startup, Restart, Clique Detection, Clique Resolution, abstract Clock Synchronization)

Formal Verification of Clock Synchronization Algorithm

- First time by means of Model Checking (*sal-inf-bmc*)

Verification of Lower-Level Synchronization Functions

- Permanence Function
  - verified with the infinite-bounded model checker *sal-inf-bmc*
  - using disjunctive invariant and k-induction
- Compression Function
  - verified with the infinite-bounded model checker *sal-inf-bmc*
  - using abstraction and 1-induction

Finalization and Completion of the formal assessment within the CoMMiCS Project

- Complexity Management for Mixed-Criticality Systems
- European Communities FP7 (FP7/2007-2013) project no. 236701
TTEthernet Formal Verification Activities

TTEthernet Executable Formal Specification

- Using symbolic and bounded model checkers sal-smc and sal-bmc
- Focus on Interoperation of Synchronization Services (Startup, Restart, Clique Resolution, abstract Clock Synchronization)

Formal Verification of Clock Synchronization Algorithm

- First time by means of Model Checking (sal-inf-bmc)

Verification of Lower-Level Synchronization Functions

- Permanence Function
  - verified with the infinite-bounded model checker sal-inf-bmc
  - using disjunctive invariant and k-induction
- Compression Function
  - verified with the infinite-bounded model checker sal-inf-bmc
  - using abstraction and 1-induction

Finalization and Completion of the formal assessment within the CoMMiCS Project

- Complexity Management for Mixed-Criticality Systems
  - European Communities FP7 (FP7/2007-2013) project no. 236701
Formal Verification of the Clock Synchronization
Model-Based Development/Verification

SAL Model

nodes(id:index): MODULE = BEGIN
  INPUT
    membership_in: ARRAY index OF BOOLEAN,
    classification_in: CLASSES,
    sender_in: index
  OUTPUT
    membership_out: ARRAY index OF BOOLEAN,
    classification_out: ARRAY index OF CLASSES
  LOCAL
    asymmetry_memb: ARRAY index OF BOOLEAN,
    asymmetry_clas: ARRAY index OF BOOLEAN
  INITIALIZATION
    (FORALL (i:index): membership_out[i] = TRUE);
    (FORALL (i:index): classification_out[i] = accept);
  TRANSITION
    [ id=faulty
      ->
        membership_out'=[[j:index] IF asymmetry_memb[j] THEN FALSE ELSE TRUE ENDIF];
        classification_out'=[[j:index] IF asymmetry_clas[j] THEN reject ELSE flagged ENDIF];
      ] END;

Counter Example

asymmetry_memb[1][2] = false;
asymmetry_memb[1][3] = false;
asymmetry_memb[1][4] = false;
asymmetry_memb[1][5] = false;
asymmetry_memb[2][1] = false;
asymmetry_memb[2][2] = false;
asymmetry_memb[2][3] = false;
asymmetry_memb[2][4] = false;
asymmetry_memb[2][5] = false;
asymmetry_memb[3][1] = true;
asymmetry_memb[3][2] = false;
asymmetry_memb[3][3] = false;
asymmetry_memb[3][4] = false;
asymmetry_memb[3][5] = true.
Clock Synchronization Algorithm

Algorithm Specification

- one SM clock: \( \text{compressed\_clock} = \text{SM\_clock}_1 \)
- two SM clocks: \( \text{compressed\_clock} = \frac{\text{SM\_clock}_1 + \text{SM\_clock}_2}{2} \)
- three SM clocks: \( \text{compressed\_clock} = \text{SM\_clock}_2 \)
- four SM clocks: \( \text{compressed\_clock} = \frac{\text{SM\_clock}_2 + \text{SM\_clock}_3}{2} \)
- five SM clocks: \( \text{compressed\_clock} = \text{SM\_clock}_3 \)
- more than five SM clocks: take the average of the \( (k+1)\)th largest and \( (k+1)\)th smallest clocks, where \( k \) is the number of faulty SMs that have to be tolerated.

- one CM clock: \( \text{SM\_clock} = \text{CM\_clock}_1 \)
- two CM clocks: \( \text{SM\_clock} = \frac{\text{CM\_clock}_1 + \text{CM\_clock}_2}{2} \)
- three CM clocks: \( \text{SM\_clock} = \text{CM\_clock}_2 \)
Step 1: SMs send messages to CMs

Compression
Master

Acceptance Window
(of SM 2/5)

Reference Point
Step 2: CMs send voted clock values back

Compression Master

Synchronization Master 1

Synchronization Master 2

Synchronization Master 3

Synchronization Master 4

Synchronization Master 5

Acceptance Window (of SM 2/5)

Reference Point

Dispatch

Permanence

SM1

SM2

SM3

SM4

SM5

t_0

t_1,

t_2

t_4,

t_5
Step 2: Multiple Channels/CMs

Multiple Channels/CMs are required for fault-tolerance.

Synchronization Masters (SMs) receive synchronization messages from all non-faulty Compression Masters (CMs)

SMs use either the median or the arithmetic mean on the redundant messages from the CMs.
Step 2: **Faulty** CMs send to some SMs

Only one Channel and one CM are shown. Typically replicated.
Synchronized Clocks Property

At any point in real time, when the system is synchronized, then the difference of clock readings of any two non-faulty clocks in the system does not deviate more than a defined value. We call this value the **precision**.

- Late Clock
- Perfect Clock
- Early Clock
Clock synchronization algorithm is formalized in SAL as state-transition system of the form \((S, I, \rightarrow)\), where

- \(S\) defines a set of systems states \(\sigma_i\)
- \(I\) defines a set of initial states with \(I\) subset of \(S\)
- \(\rightarrow\) defines the set of transitions between system states

We use the bounded infinite-state model checker

**sal-inf-bmc**

- We can model real time as a continuous entity

We use \(k\)-induction to prove the upper bound on the precision, \(k\)-induction is a generalized form of the regular induction:

- Base case: show that all the states reachable from \(I\) in no more than \(k-1\) steps satisfy \(P\)
- Induction step: For all trajectories \(\sigma_0 \rightarrow \sigma_1 \rightarrow \ldots \rightarrow \sigma_k\) of length \(k\), show that
  
  \((\sigma_0 \models P \text{ and } \sigma_1 \models P \text{ and } \ldots \text{ and } \sigma_{k-1} \models P) \Rightarrow \sigma_k \models P\)
Abstraction / Monitor

We model check:

- all $\sigma_i$ belong to either $\Sigma_{\text{sync}}$ or $\Sigma_{\text{send}}$
- all concrete transitions are mapped to an abstract transition (trivial in this case)
POSREAL: TYPE = \{x: REAL | x\geq 0 \};

max_drift: POSREAL; % maximum drift offset is uninterpreted

max_SM: NATURAL = 5; % system consists of five SM

max_CM: NATURAL = 2; % system consists of two channels

Precision is expressed by \( \text{FACTOR} \times \text{max\_drift} \)

Problem: find \( \text{FACTOR} \)
Model Fragments – Fault Free Case

%Synchronization Master

...  
[  state=sync -->  
state'=send;  
  interval_ctr'=interval_ctr + 1;  
  clock'=[[j:TYPE_CM] clock[j] + drift];  
]

[] state=send -->  
state'=sync;  
  clock' IN  
    {x: ARRAY TYPE_CM OF TYPE_clock |  
      average(list_compressed_clock[1],  
       list_compressed_clock[2],x[1])};

%Compression Master (for 4 SM)

...  
compressed_clock IN  
  {x: TYPE_clock | average(clocks_cm[order[2]],  
                         clocks_cm[order[3]], x)}

...  

drift IN {x: TYPE_drift | x>=-max_drift AND x<=max_drift}

average(value1, value2, avg: TYPE_clock): BOOLEAN = avg=(value1+value2)/2

order IN {x: ARRAY TYPE_SM OF TYPE_SM | sort(clocks_cm, x)};
sort_part() predicate

sort_part( unsorted_list: ARRAY TYPE_SM OF TYPE_clock,  
          sorted_list: ARRAY TYPE_SM OF TYPE_SM): BOOLEAN =

  sorted_list[max_SM]=1  %1 is the faulty SM

  AND
  (FORALL (i:TYPE_SM): i<max_SM-1 =>
    unsorted_list[sorted_list[i]] <= unsorted_list[sorted_list[i+1]])

  AND
  (FORALL (i,j:TYPE_SM): sorted_list[i] = sorted_list[j] => i=j);

input

unsorted_list

<table>
<thead>
<tr>
<th>2.7</th>
<th>2.9</th>
<th>2.5</th>
<th>2.8</th>
<th>3.0</th>
</tr>
</thead>
</table>

TYPE_clock [TYPE_SM]

output

sorted_list (sort_part)

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>2</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
</table>

TYPE_SM [TYPE_SM]

unsorted_list [sorted_list [1]] = 2.5
unsorted_list [sorted_list [2]] = 2.8
unsorted_list [sorted_list [3]] = 2.9
unsorted_list [sorted_list [4]] = 3.0
( unsorted_list [sorted_list [5]] = 2.7 )
%Faulty Synchronization Master

... 

[ state=sync \[ state, =send; interval\_ctr'=interval\_ctr + 1; clock'= [[j:TYPE\_CM] clock[j]] + drift + failure[j]]; 

[] state=send \[ state, =sync; clock' IN 

{x: ARRAY TYPE\_CM OF TYPE\_clock | x[1]=x[2] AND average(list\_compressed\_clock[1], list\_compressed\_clock[2], x[1])}; 

... 

[] state=send AND id=FAULTY\_SM \[ state' =sync; ]

%Faulty Compression Master

... 

compressed\_clock IN 

{x: TYPE\_clock | 

... 

average(clocks\_cm[order[2]], clocks\_cm[order[3]], x) OR 

average(clocks\_cm[order[2]], clocks\_cm[order[4]], x) OR 

average(clocks\_cm[order[3]], clocks\_cm[order[4]], x) 

})} 

... 

failure IN {x: ARRAY TYPE\_CM OF TYPE\_drift | IF id = FAULTY\_SM THEN TRUE ELSE x[1]=0 AND x[2]=0 ENDIF
Property of Interest

BIG = (FORALL (i:TYPE_SM): list_states[i]=send) AND
(FORALL (i,j:TYPE_SM): list_clocks[i][1]>list_clocks[j][1] =>
(list_clocks[i][1]-list_clocks[j][1] <= FACTOR * max_drift));

SMALL = (( ... <= FACTOR_small * max_drift));

distance: LEMMA world |- 

G(FORALL (i,j: TYPE_SM):
(list_states[i]=send AND list_clocks[i][1]>list_clocks[j][1] =>
list_clocks[i][1]-list_clocks[j][1]) <= FACTOR*max_drift));

When the Synchronization Masters are in the “send” state, the difference in their local clocks is less than or equal to FACTOR*max_drift

• In the send state a drift in [-max_drift, max_drift] is added to the local clocks
• Faulty clocks have to be excluded from this property (e.g., by i /= Faulty_SM)
• FACTOR is instantiated before calling the model checker
Verification Process and Results

`> sal-inf-bmc clocksync distance --depth=3 -i`

<table>
<thead>
<tr>
<th>Property</th>
<th>No Faults</th>
<th>Faulty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CM</td>
<td>SM</td>
</tr>
<tr>
<td>FACTOR distance</td>
<td>2</td>
<td>(8/3) 2</td>
</tr>
<tr>
<td>abstraction distance</td>
<td>0.5</td>
<td>0.58 0.49</td>
</tr>
<tr>
<td>distance + abst.</td>
<td>0.34</td>
<td>0.36 0.38</td>
</tr>
</tbody>
</table>

`> sal-inf-bmc clocksync abstract_invar --depth=3 -i`

`> sal-inf-bmc clocksync distance -I abstract_invar --depth=3 -i`
Algorithm Specification

- one SM clock: \( \text{compressed\_clock} = \text{SM\_clock}_1 \)
- two SM clocks:
  \[ \text{compressed\_clock} = \frac{\text{SM\_clock}_1 + \text{SM\_clock}_2}{2} \]
- three SM clocks: \( \text{compressed\_clock} = \text{SM\_clock}_2 \)
- four SM clocks:
  \[ \text{compressed\_clock} = \frac{\text{SM\_clock}_2 + \text{SM\_clock}_3}{2} \]
- five SM clocks: \( \text{compressed\_clock} = \text{SM\_clock}_3 \)
- more than five SM clocks: take the average of the \((k+1)^{th}\) largest and \((k+1)^{th}\) smallest clocks, where \(k\) is the number of faulty SMs that have to be tolerated.

- one CM clock: \( \text{SM\_clock} = \text{CM\_clock}_1 \)
- two CM clocks:
  \[ \text{SM\_clock} = \frac{\text{CM\_clock}_1 + \text{CM\_clock}_2}{2} \]
- three CM clocks: \( \text{SM\_clock} = \text{CM\_clock}_2 \)
Byzantine Faulty Clock

Clock Synchronization Two Failures Scenario
Ensuring Reliable Networks
Diagnosis to improve Clock Synchronization Quality
Layered Algorithm to Exclude Faulty CMs

The TTEthernet clock synchronization algorithm is a two-step state correction algorithm.

Failures in the SMs or CMs are transparently compensated by arithmetic. The quality of the precision decreases with the severity and number of faulty components.

A diagnosis algorithm executed in the SMs that detects inconsistent CMs and excludes the CM from the clock synchronization algorithm improves the quality of the precision.
Detection and Notification of a faulty CM

Algorithm 4 Diagnosis Algorithm executed by SM $i$

1: if $CM\_clock_j \neq \perp$ then
2: \hspace{1em} active[$j$] $\leftarrow$ TRUE
3: end if
4: for $j = 1 \rightarrow |CM|$ do
5: \hspace{1em} if $CM\_clock_j = \perp \land active[j]$ then
6: \hspace{2em} accused[$i$][$j$], $AC_i\_accused[j] \leftarrow$ TRUE
7: \hspace{1em} end if
8: end for

Remember when a CM has been active.

When a CM has been active, but it did not provide a clock value, then:
- classify CM as faulty
- notify all other SMs

Algorithm 5 Accusation Message Message Reception in SM $i$

1: if receives($AC_k$) then
2: \hspace{1em} for $j = 1 \rightarrow |CM|$ do
3: \hspace{2em} if $AC_k\_accused[j]$ then
4: \hspace{3em} accused[$k$][$j$] $\leftarrow$ TRUE
5: \hspace{2em} end if
6: \hspace{1em} end for
7: end if

Record which SM accuses (i.e., classifies as faulty) which CM

$\{CM\_clock \mid pcf\_membership\_new \geq \text{accept\_threshold} \land CM\_clock_j \not\Rightarrow \neg\text{accused}[k][j] \land CM\_clock_j \not\Rightarrow \exists i_1, \ldots, i_z : \text{accused}[i_1][j] \land \ldots \land \text{accused}[i_z][j]\}$

Use the clock value of a CM only if the SM did not accuse it itself or if the SM did not receive accusations from a significantly number of other SMs for the CM.
Quality Improvement of the Precision

- CM 1 skips SM 2
- SM 2 accuses CM 1
- CM 1 skips SM 3
- SM 3 accuses CM 1
- Sufficient SMs accused CM 1 and all SMs exclude CM 1
Theorem 2.

\[ \exists i_1, \ldots, i_z : \text{accused}[i_1][j] \land \ldots \land \text{accused}[i_z][j] \Rightarrow \text{precision} \leq 2 \times \max(\text{drift}) \]

Abstraction \( \mathcal{A} \)

<table>
<thead>
<tr>
<th></th>
<th>( k )</th>
<th>time</th>
<th></th>
<th>( k )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction ( \mathcal{A} )</td>
<td>5</td>
<td>180.00</td>
<td>Precision ( \square P )</td>
<td>4</td>
<td>3.55</td>
</tr>
</tbody>
</table>
Layered Clock Rate Correction
The TTEthernet clock synchronization algorithm is a two-step state correction algorithm.

A clock-rate correction algorithm can be executed in the SMs that monitors the clock-state correction for a configurable number of cycles and adjusts the speed (i.e., the rate) of the clock as a function of the correction terms. A clock-rate correction algorithm improves the precision in a synchronized system when clock drifts are stable to some extent.

When clock drifts are not stable we need to proof that the precision is still bounded.
Algorithm 6 Rate-Correction Algorithm executed by SM $i$

1: if $cycle \leq rate_{obs_{nr}}$ then
2:   $drift_{obs}[cycle] \leftarrow act_{corr}$
3: end if
4: if $cycle = rate_{obs_{nr}}$ then
5:   $corr \leftarrow \left( \sum_{l=1}^{l \leq rate_{obs_{nr}}} drift_{obs}[l] \right) \div rate_{obs_{nr}}$
6:   if $corr > \max(drift)$ then
7:     $corr = \max(drift)$
8:   else if $corr < -\max(drift)$ then
9:     $corr = -\max(drift)$
10: end if
11: $clock_{rate} \leftarrow clock_{rate} - corr$
12: end if
13: $cycle \leftarrow cycle + 1$

Store the actual clock correction terms.

After a number of observation cycles, use the state-correction terms to determine a rate-correction term.

Force an upper bound on the rate-correction term.

Apply the rate-correction term.
Rate-Correction with Stable Clock Drifts

Store 1\textsuperscript{st} state-correction term

Store 2\textsuperscript{nd} state-correction term

Calculate and apply rate-correction term
Rate-Correction with Unstable Clock Drifts

Store 1\textsuperscript{st} state-correction term

Store 2\textsuperscript{nd} state-correction term

Calculate and apply rate-correction term
Theorem 3.

\[ \text{precision} \leq \frac{8}{3} \times 2 \times \max(\text{drift}) \]

Abstraction \( A \)

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \text{time} )</th>
<th>( k )</th>
<th>( \text{time} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>5</td>
<td>1432.69</td>
<td>Precision ( \Box ) ( P )</td>
</tr>
</tbody>
</table>
Summary and Outlook
Summary and Outlook

• In the past clock synchronization algorithms have been formally verified by heavy-duty theorem provers, like PVS.

• Today clock synchronization algorithms can be formally verified by means of model checking (using k-induction).

• In our model time is represented as real value and the model progresses discrete.

• We are working on a hybrid model of the algorithms.

• First results have been presented:
  • A Benchmark of a Fault-Tolerant TTEthernet Network, Sergiy Bogomolov, Christian Herrera, and Wilfried Steiner, ARCH16