MOST (Modeling of SpW Traffic) Applications & improvements

Felix Siegle
SpW WG meeting #18
24/04/2012

Outline

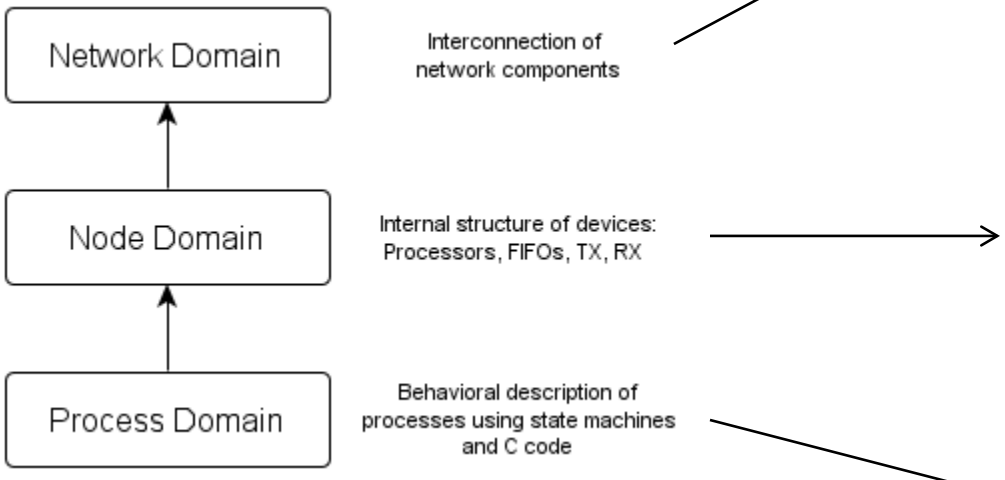
1. Introduction
2. Refresher on OPNET / MOST
3. Limitations of current version
4. Application 1: BepiColombo
5. Redevelopment of basis building blocks
6. Application 2: time-code propagation jitter (incl. hardware cross validation)
7. Conclusion

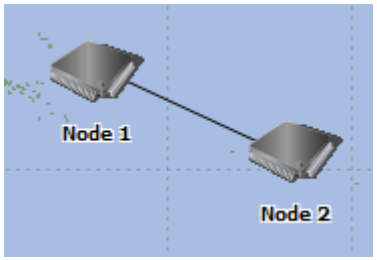
Introduction

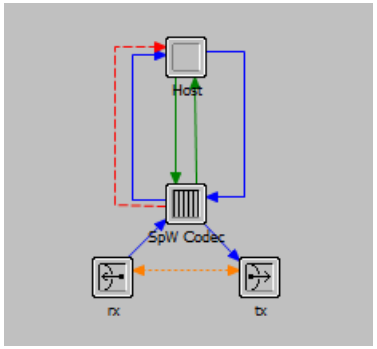
- Student at the University of Applied Sciences Jena, Germany in Master's programme "space electronics"
- Internship and Bachelor's thesis at Astrium Friedrichshafen
- Now intern at ESTEC for 3 months
- As a space enthusiast, I am proud to be able to spend time at ESTEC and contribute something useful to ESA activities

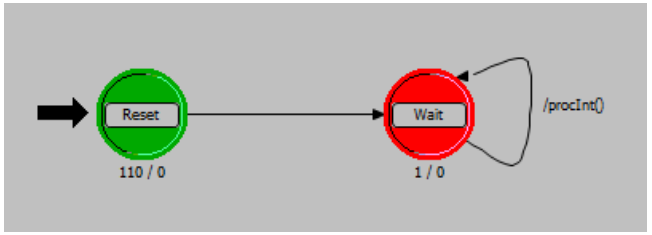
Refresher on OPNET Modeler

- One of the leading software tools for intercommunication network simulation and analysis
- Event-driven simulation







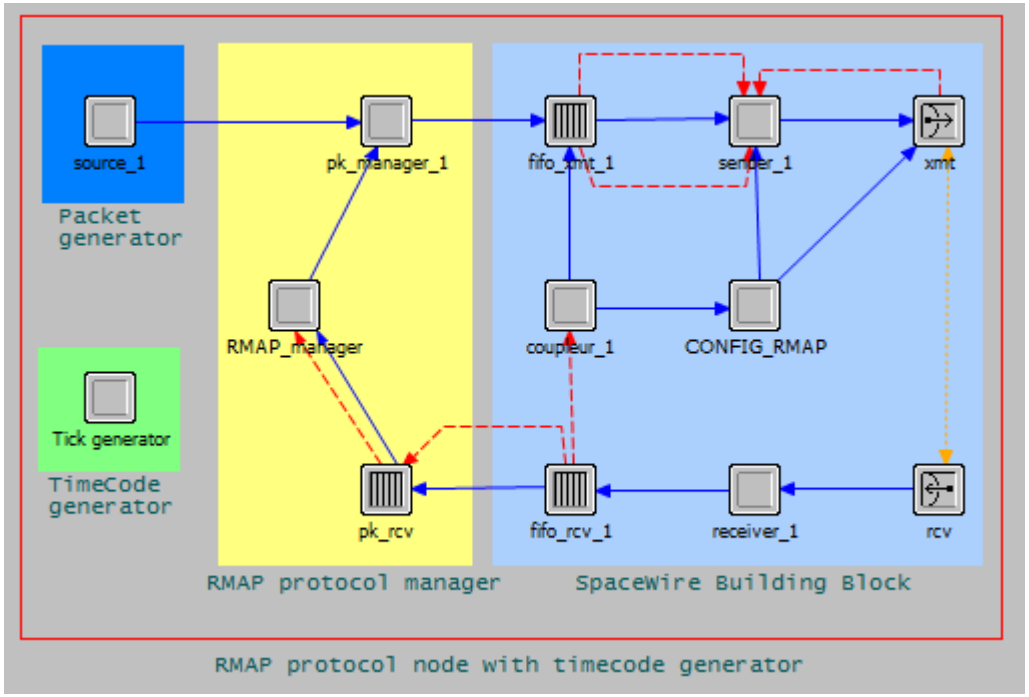


Refresher on MOST

- Developed by ThalesAlenia Space and delivered to ESA in Dec. 2011
- The library contains all important SpaceWire components:
    - Basics: SpaceWire Codec
    - Protocols: CCSDS PTP, RMAP
    - Devices:
        - Traffic Generator / Sink
        - RTC Node
        - SMCS116SPW
        - SMCS332SPW
        - SPW10X Routing Switch

Refresher on MOST

- Internal structure of an end node in MOST (incl. codec):



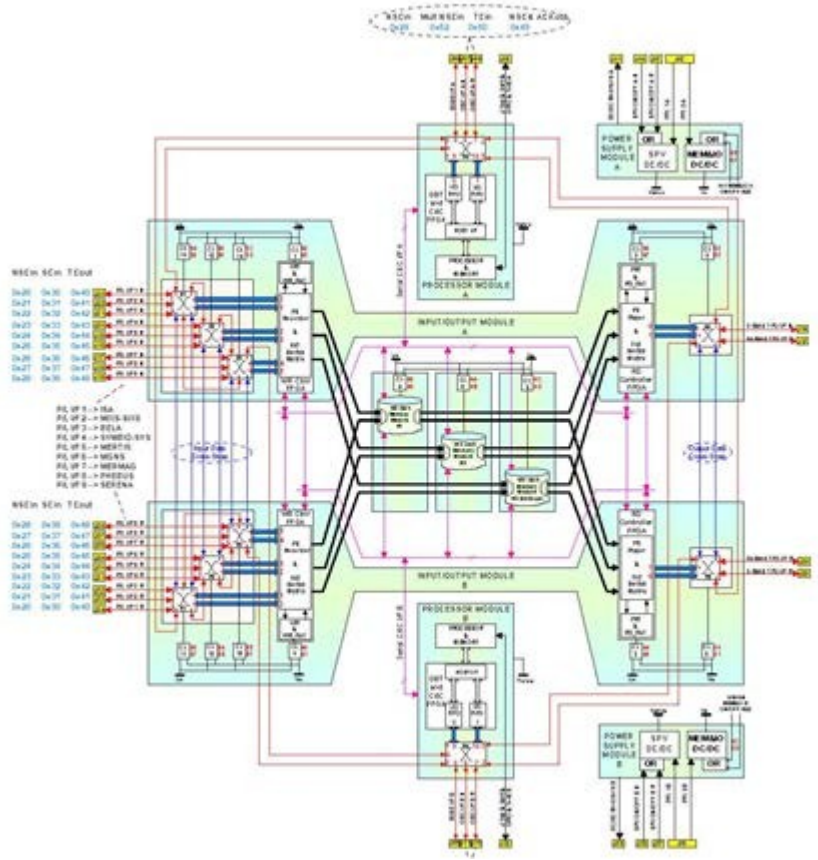
Limitations of current version

- No link disconnect mechanism
  - Partly implemented on Exchange Level
  - Not implemented on Network Level (Spilling of packets etc.)
- No parallel ports in 10X routing switches
- Characters does not carry information
- Minor problems with several statistics
  (e.g. missing TC jitter measurement)

- Main problem: difficult to understand, maintain and extend!
  - Models
  - Documentation
- Clearly Alpha level (SW TRL 3)

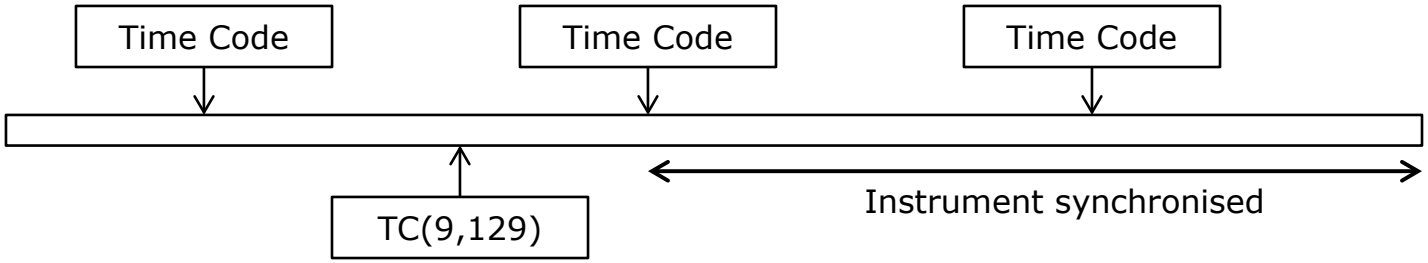
Application 1: BepiColombo

- BepiColombo payload data system
  - Fully SpW-based incl. C&C (no MIL-1553-STD bus)
  - 10 switches (incl. redundancy)
  - 48 endpoints (SpW I/F)
    - Spread over 38 nodes
    - …and 13 units
- Running 2 SpW-based protocols:
  - CPTP transporting PUS packets
  - Time synchronisation protocol TC (PUS) + Time Codes



Application 1: BepiColombo

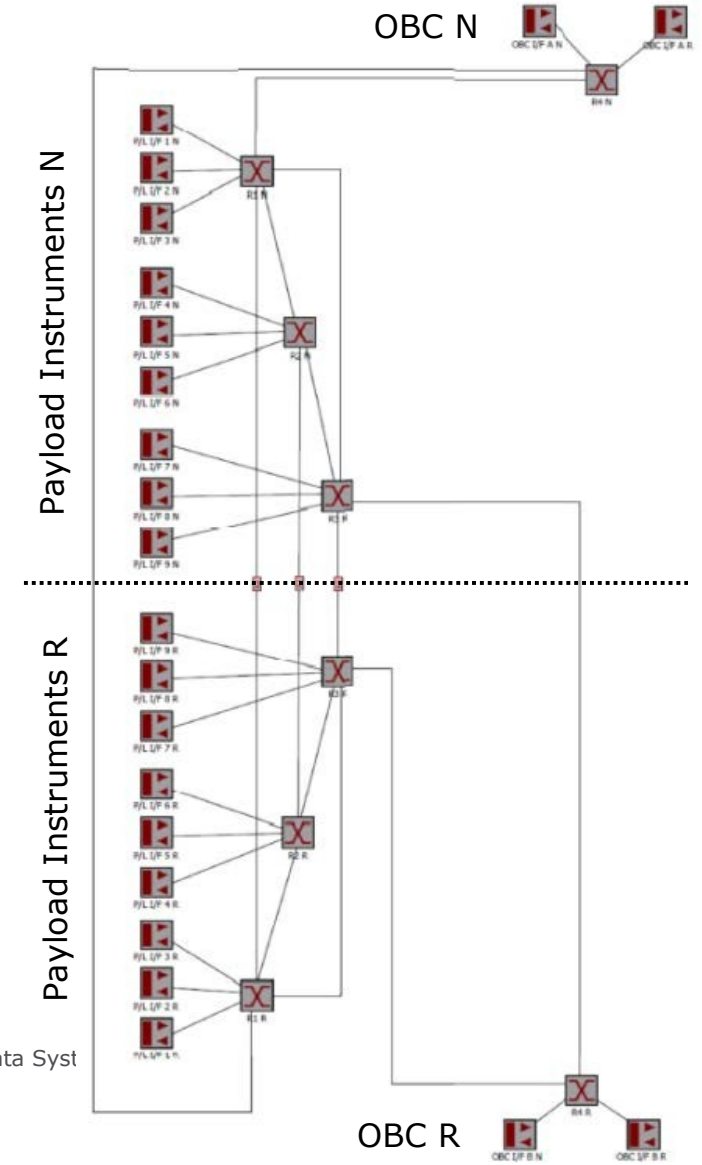
- Time Synchronisation Protocol:



- TC(9) carries Spacecraft Elapsed Time which is valid with next Time Code
- If Time Code value matches SCET seconds, the instrument time is updated
- Time Codes are sent every seconds, value is copied to seconds field

Application 1: BepiColombo

- Focus on the “left” part of the network:
  - 9 redundant instruments
  - 1 redundant SSMM supervisor
  - 1 redundant OBC
- Required improvements to the current version of MOST:
  - Implementation of OBC master clock
  - Implementation of instruments slave clock
  - Modelling of TC(9,129)
  - Link disconnect mechanism



Application 1: BepiColombo
Example Scenario: Random TC initial values

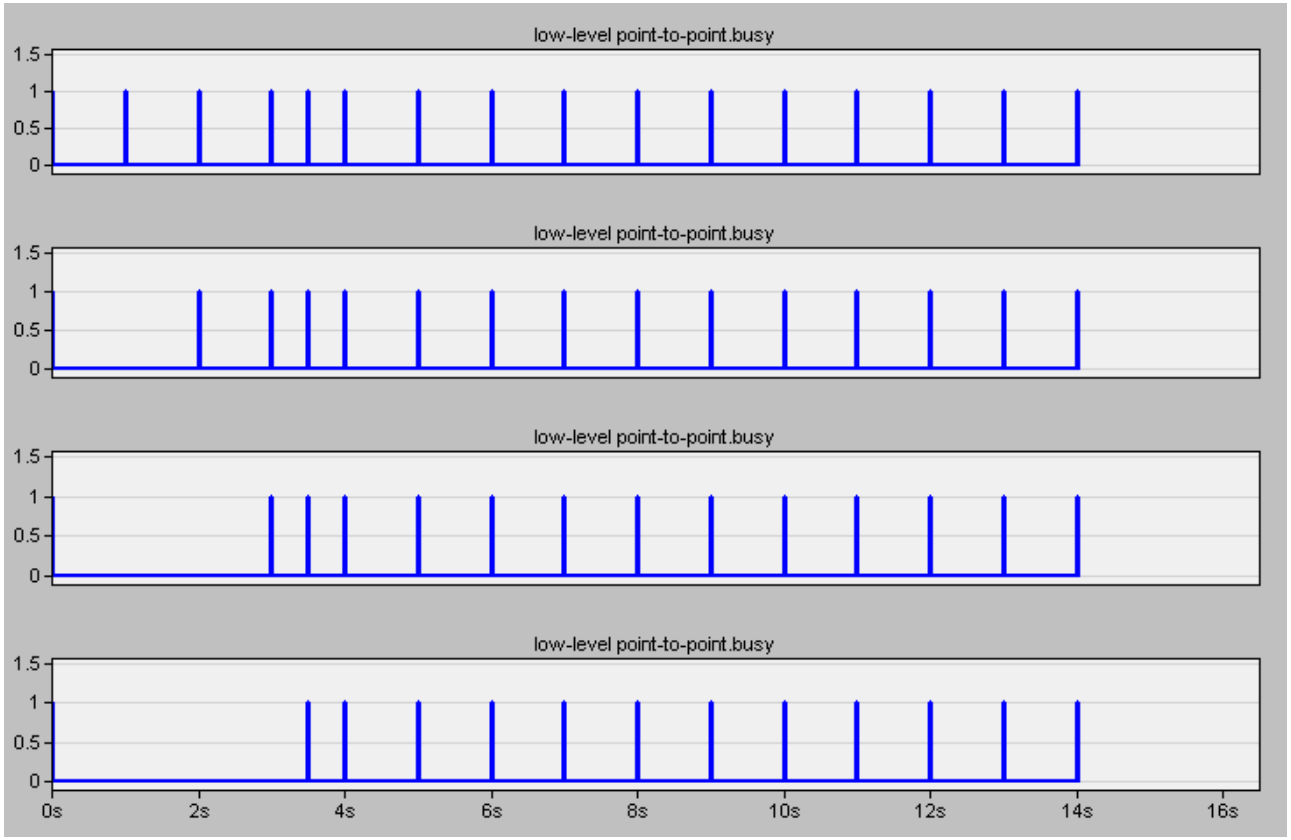
Simulation Time [s]	0	1	2	3	3.5	4	5	6	7	8	9	10
SpW Time Code	T(0)	T(1)	T(2)	T(3)		T(4)	T(5)	T(6)	T(7)	T(8)	T(9)	T(10)
OBC Master Clock	0	1	2	3		4	5	6	7	8	9	10
					TC(9,129) announcing Master Time Value 4							
P/L I/F 1 N				T(3)		T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 2 N				T(3)		T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 3 N				T(3)		T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 4 N						T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 5 N						T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 6 N						T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 7 N						T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 8 N						T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 9 N						T(4) M(4)	T(5) S(5)	T(6) S(6)	T(7) S(7)	T(8) S(8)	T(9) S(9)	T(10) S(10)
R4	T(4)	T(1)	Tick	Tick		Tick						
R1	T(4)	T(4)	T(2)	Tick		Tick						
R2	T(4)	T(4)	T(4)	T(3)		Tick						
R3	T(4)	T(4)	T(4)	T(3)		Tick						

Application 1: BepiColombo
Example Scenario: Random TC initial values

[3.500000] [OBC I/F A N] Sending a packet. Destination: 64. Bytes: 22.
[3.500026] [P/L I/F 1 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000026. Master Clk: 4.
[3.500050] [OBC I/F A N] Sending a packet. Destination: 65. Bytes: 22.
[3.500076] [P/L I/F 2 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000026. Master Clk: 4.
[3.500100] [OBC I/F A N] Sending a packet. Destination: 66. Bytes: 22.
[3.500126] [P/L I/F 3 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000026. Master Clk: 4.
[3.500150] [OBC I/F A N] Sending a packet. Destination: 67. Bytes: 22.
[3.500177] [P/L I/F 4 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000027. Master Clk: 4.
[3.500200] [OBC I/F A N] Sending a packet. Destination: 68. Bytes: 22.
[3.500226] [P/L I/F 5 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000026. Master Clk: 4.
[3.500250] [OBC I/F A N] Sending a packet. Destination: 69. Bytes: 22.
[3.500277] [P/L I/F 6 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000027. Master Clk: 4.
[3.500300] [OBC I/F A N] Sending a packet. Destination: 70. Bytes: 22.
[3.500327] [P/L I/F 7 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000027. Master Clk: 4.
[3.500350] [OBC I/F A N] Sending a packet. Destination: 71. Bytes: 22.
[3.500376] [P/L I/F 8 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000026. Master Clk: 4.
[3.500400] [OBC I/F A N] Sending a packet. Destination: 72. Bytes: 22.
[3.500427] [P/L I/F 9 N] Sink destroyed packet. Source: OBC I/F A N. Delay: 0.000027. Master Clk: 4.

[4.000006] [P/L I/F 2 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000006] [P/L I/F 2 N] Timecode received. Val: 4. Slave Time: 4
[4.000006] [P/L I/F 3 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000006] [P/L I/F 3 N] Timecode received. Val: 4. Slave Time: 4
[4.000007] [P/L I/F 1 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000007] [P/L I/F 1 N] Timecode received. Val: 4. Slave Time: 4
[4.000008] [P/L I/F 4 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000008] [P/L I/F 4 N] Timecode received. Val: 4. Slave Time: 4
[4.000008] [P/L I/F 5 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000008] [P/L I/F 5 N] Timecode received. Val: 4. Slave Time: 4
[4.000008] [P/L I/F 6 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000008] [P/L I/F 6 N] Timecode received. Val: 4. Slave Time: 4
[4.000008] [P/L I/F 7 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000008] [P/L I/F 7 N] Timecode received. Val: 4. Slave Time: 4
[4.000008] [P/L I/F 8 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000008] [P/L I/F 8 N] Timecode received. Val: 4. Slave Time: 4
[4.000008] [P/L I/F 9 N] Timecode received. Val: 4. Master Time: 4 --> MATCH!
[4.000008] [P/L I/F 9 N] Timecode received. Val: 4. Slave Time: 4

Application 1: BepiColombo
Example Scenario: Random TC initial values



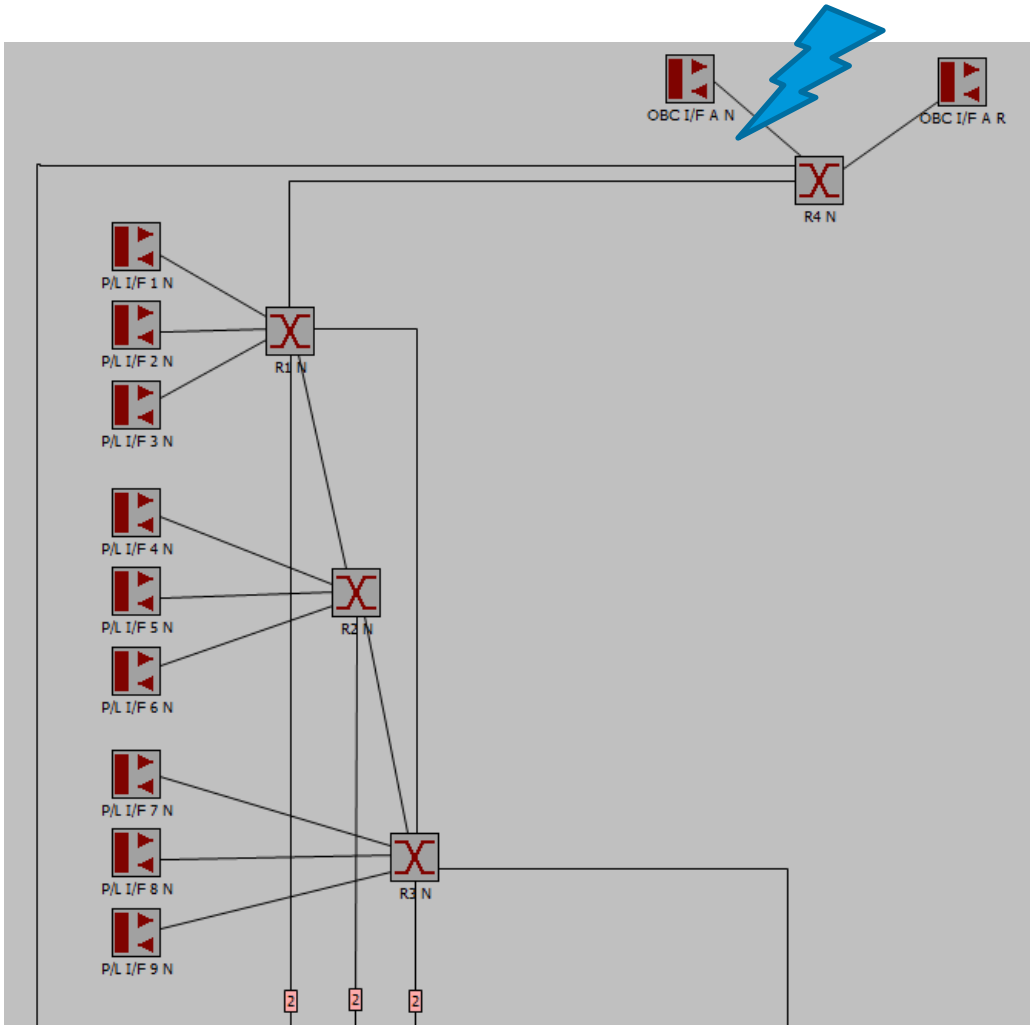
OBC ☾ R4

R4 ☾ R1

R1 ☾ P1

R2 ☾ P4

Application 1: BepiColombo
Example Scenario 2: Link disconnect



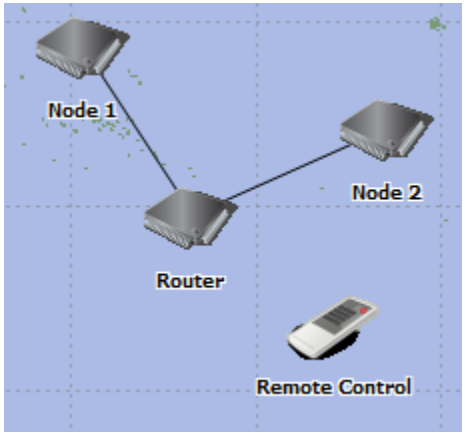
Application 1: BepiColombo
Example Scenario 2: Link disconnect

Simulation Time [s]	0	1	2	3	3.5	4	5	6	7	8	9	10
SpW Time Code	T(0)	T(1)	T(2)	T(3)		T(4)	**T(5)**	**T(6)**	**T(7)**	**T(8)**	**T(9)**	**T(10)**
OBC Master Clock	0	1	2	3		4	5	6	7	8	9	10
					TC(9,129) announcing Master Time Value 4							
P/L I/F 1 N				T(3)		T(4) M(4)				T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 2 N				T(3)		T(4) M(4)				T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 3 N				T(3)		T(4) M(4)				T(8) S(8)	T(9) S(9)	T(10) S(10)
P/L I/F 4 N						T(4) M(4)					T(9) S(9)	T(10) S(10)
P/L I/F 5 N						T(4) M(4)					T(9) S(9)	T(10) S(10)
P/L I/F 6 N						T(4) M(4)					T(9) S(9)	T(10) S(10)
P/L I/F 7 N						T(4) M(4)					T(9) S(9)	T(10) S(10)
P/L I/F 8 N						T(4) M(4)					T(9) S(9)	T(10) S(10)
P/L I/F 9 N						T(4) M(4)					T(9) S(9)	T(10) S(10)
							^ Disconnect between OBC I/F A N and R4 N					
R4	T(4)	T(1)	Tick	Tick		Tick	T(4)	T(6)	Tick	Tick	Tick	
R1	T(4)	T(4)	T(2)	Tick		Tick	T(4)	T(4)	T(7)	Tick	Tick	
R2	T(4)	T(4)	T(4)	T(3)		Tick	T(4)	T(4)	T(4)	T(8)	Tick	
R3	T(4)	T(4)	T(4)	T(3)		Tick	T(4)	T(4)	T(4)	T(8)	Tick	

Application 1: BepiColombo

- Next steps:
  - Complete the payload network
  - Get switching tables
  - Get PUS profiles (size and frequency)
  - Simulate real traffic on network
- Requires update of MOST to include:
  - Link Disconnect
  - External ports in 10X routing switches
  - Characters carrying real information
  - Measurement of Time Code jitter etc…
- Difficult to add these features to the current version of MOST
  - These features are already implemented in my basic building blocks.

Redevelopment of basis building blocks



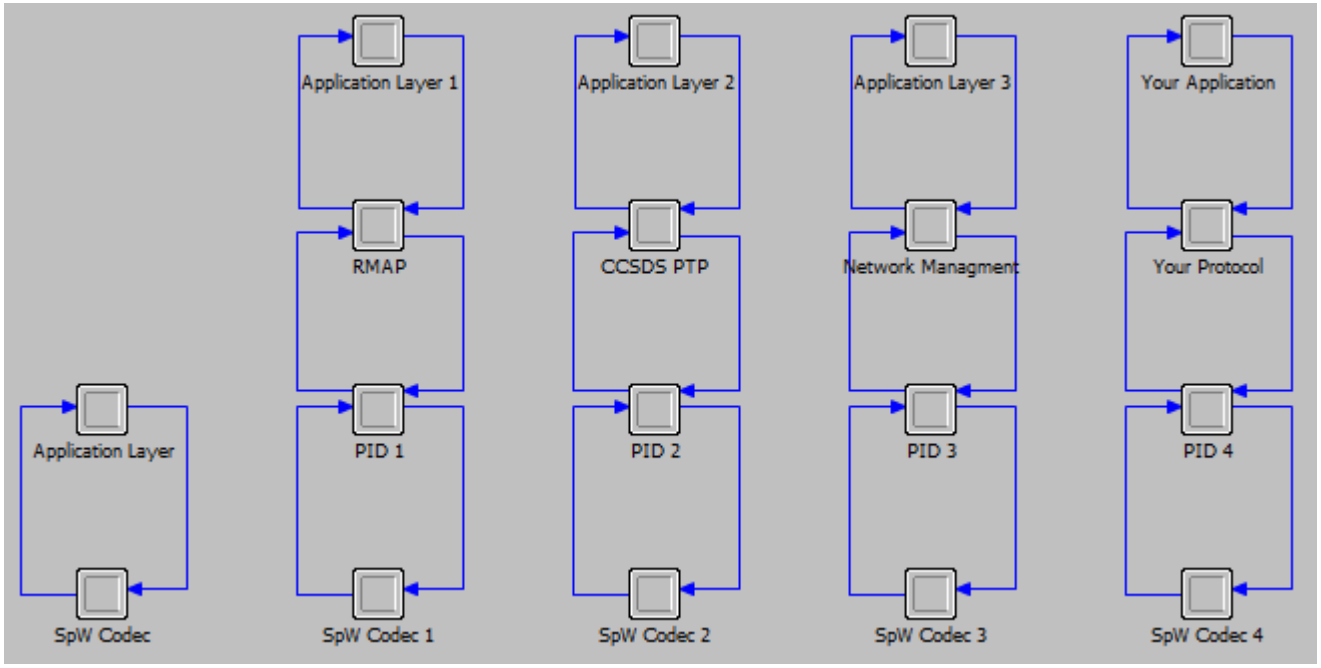
- Basic idea: **Keep it simple!**
  - Less code, less events
    - Improves simulation speed
  - One coding style, functional design with reusable code blocks
    - Easier to maintain, extend and adapt to new devices
  - Sticks to the ECSS standard as much as possible
    - Easier to understand, more intuitive
  - Creation time of Time Codes preserved
    - Allows jitter measurement validation of time distribution protocol
  - Link disconnect mechanism implemented by design

Redevelopment of basis building blocks

- Basic idea: **Keep it simple!**
  - Simple and well-documented service interfaces
    - Allows implementation of protocol stacks
  - **SpW characters carries real information**
    - Decision within network devices are based on this information
    - Allows PUS on-board communications (BepiColombo, MTG)

Redevelopment of basis building blocks

- Basic idea: **Keep it simple!**
  - Protocol stack implementation
    - One block for codec, one block for RMAP etc…
    - User must only design application layer (API provided)
    - Could become a prototyping platform for new protocols

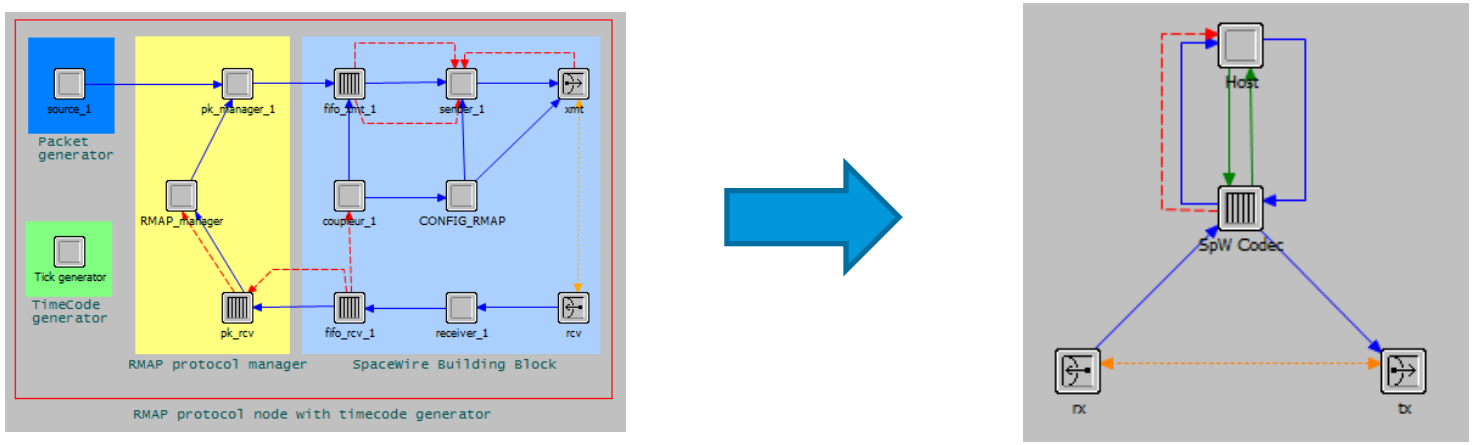


Your Application

Your Protocol

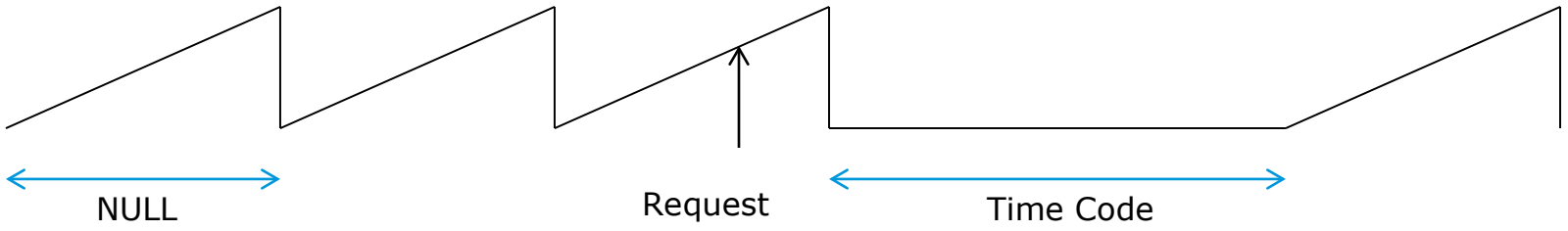
Redevelopment of basis building blocks

- Current development state:
  - Codec finished and under testing
  - Prototype of a traffic generator / sink host system
  - Working on generic routing switch in progress



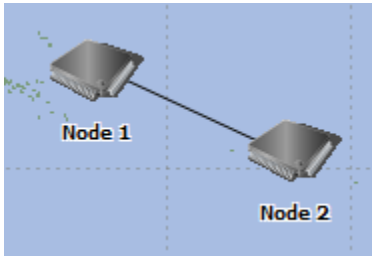
Application 2: time-code propagation jitter (incl. hardware cross validation)

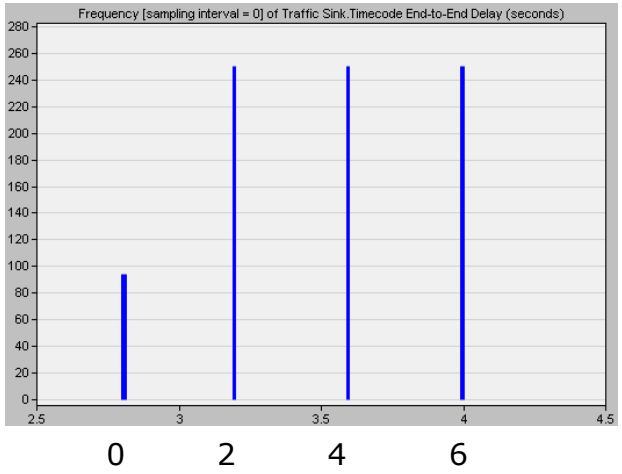
- Question: Is it possible to simulate a realistic TC jitter?
- Simulation scenario:
  - Only NULLs on line (idle state)
  - Periodic Time Code generation
  - Time delay for a waiting Time Code between 0 .. 7 bits
- Problem:
  - "Coherent" clock in simulation
  - Distribution of TC waiting time depends on several parameters, e.g. Frequency of Time Code requests, data link rate
  - The distribution of the waiting time is not uniform



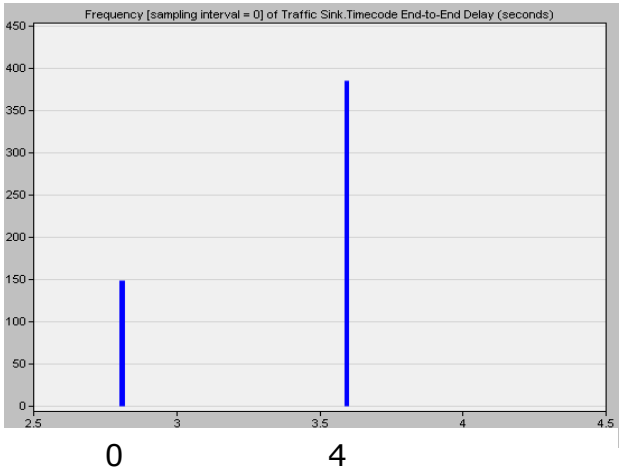
Application 2: time-code propagation jitter (incl. hardware cross validation)

- Simulation Model: Examples

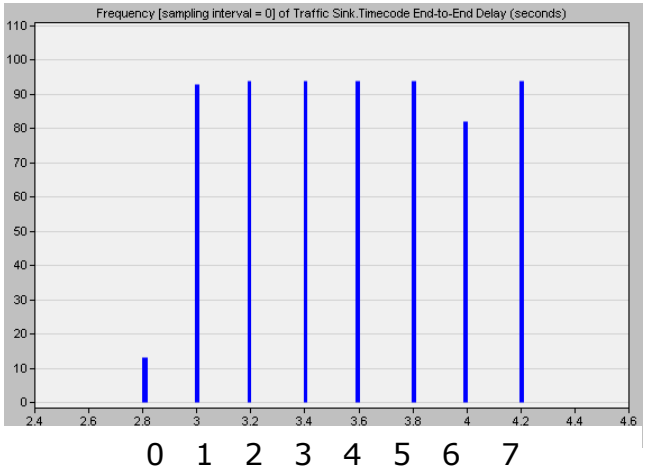




TC period: 100 µs
@ 5 Mbit/s



TC period: 130 µs
@ 5 Mbit/s

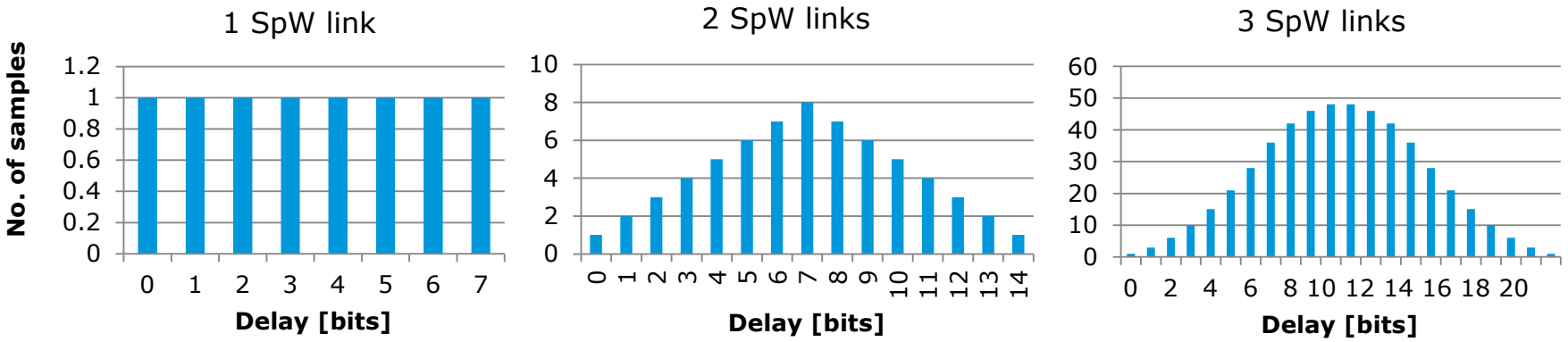


TC period: 133.3 µs
@ 50 Mbit/s

Application 2: time-code propagation jitter (incl. hardware cross validation)

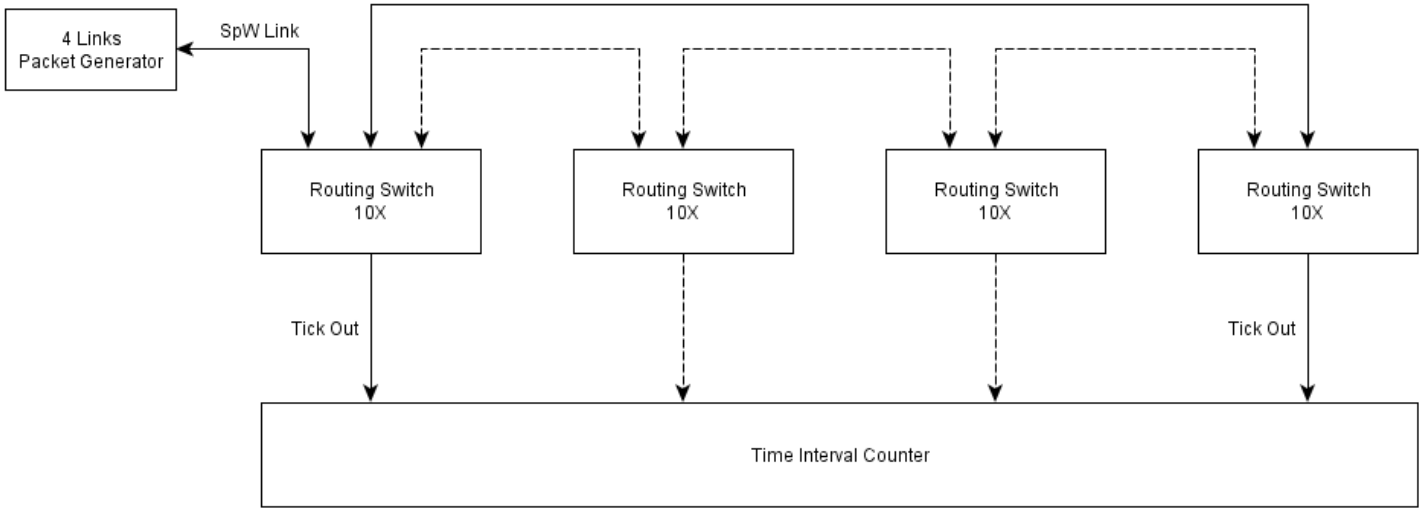
- In real hardware the distribution of the waiting time in the first codec should be more or less uniform due to internal jitter

- Triangle shaped distribution on second link
- ∞ links: limited Gaussian distribution



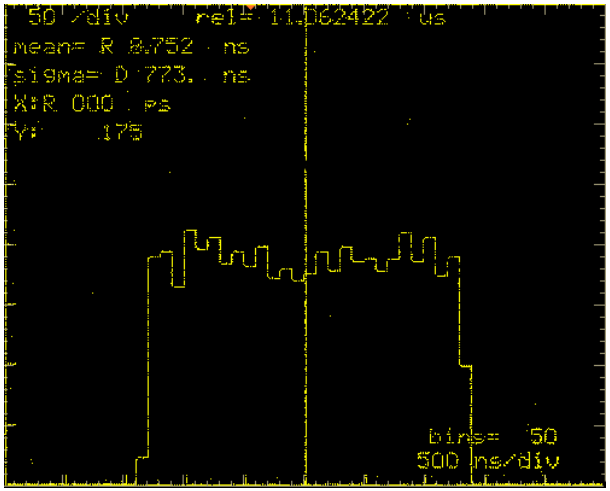
Application 2: time-code propagation jitter (incl. hardware cross validation)

- How does realistic TC jitter looks like? ☾ Hardware cross validation:
  - Basic setup using four 10X routing switches
  - 2500 Time Codes generated by Packet Generator (4Links)
  - Time delays are measured between Tick-Out signals with a high accurate Time Interval Counter (Stanford SR620)

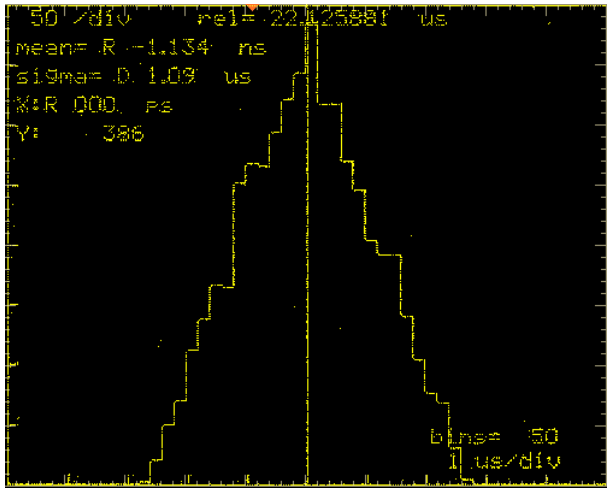


Application 2: time-code propagation jitter (incl. hardware cross validation)

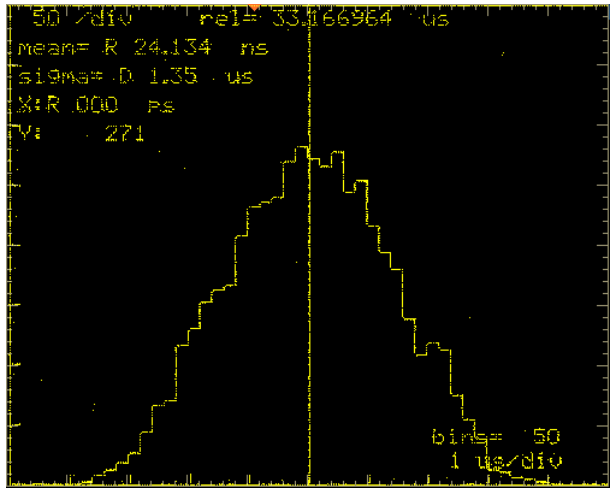
- With uniform distribution on first link, it can be shown that the distribution on the other links follows the statistical theory (here @ 3 Mbit/s)



1 link



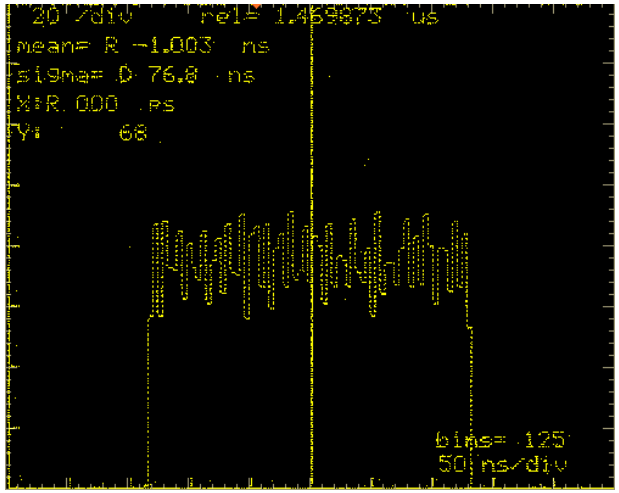
2 links



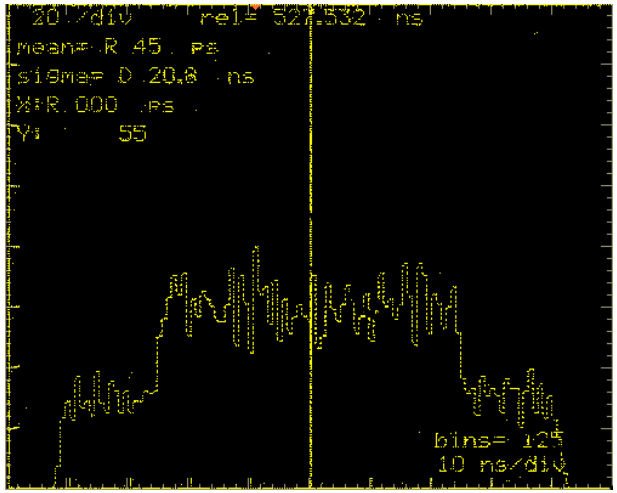
3 links

Application 2: time-code propagation jitter (incl. hardware cross validation)

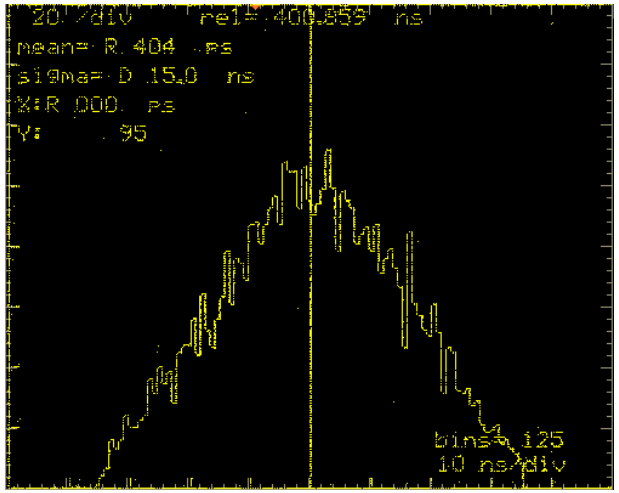
- Distribution of waiting time on one link:



30 Mbit/s



120 Mbit/s



200 Mbit/s

Why?? Suggestions welcome!

Conclusion

- Simulation is useful and necessary
- OPNET Modeler is a good choice as simulation framework
- MOST is usable in the current development state but still misses some important features

- Thank you very much for your attention! -