

Reactor-Based Modeling of Safe Interrupt-Driven Communication for Distributed Embedded Systems

Tejeswini Jayaramareddy^{1,3}, Hokeun Kim², and Hoeseok Yang¹

¹Santa Clara University ²Arizona State University ³Lucid Motors

Time-Centric Reactive Software Workshop at ESWEEK 2026

Barcelona, Spain

October 8, 2026

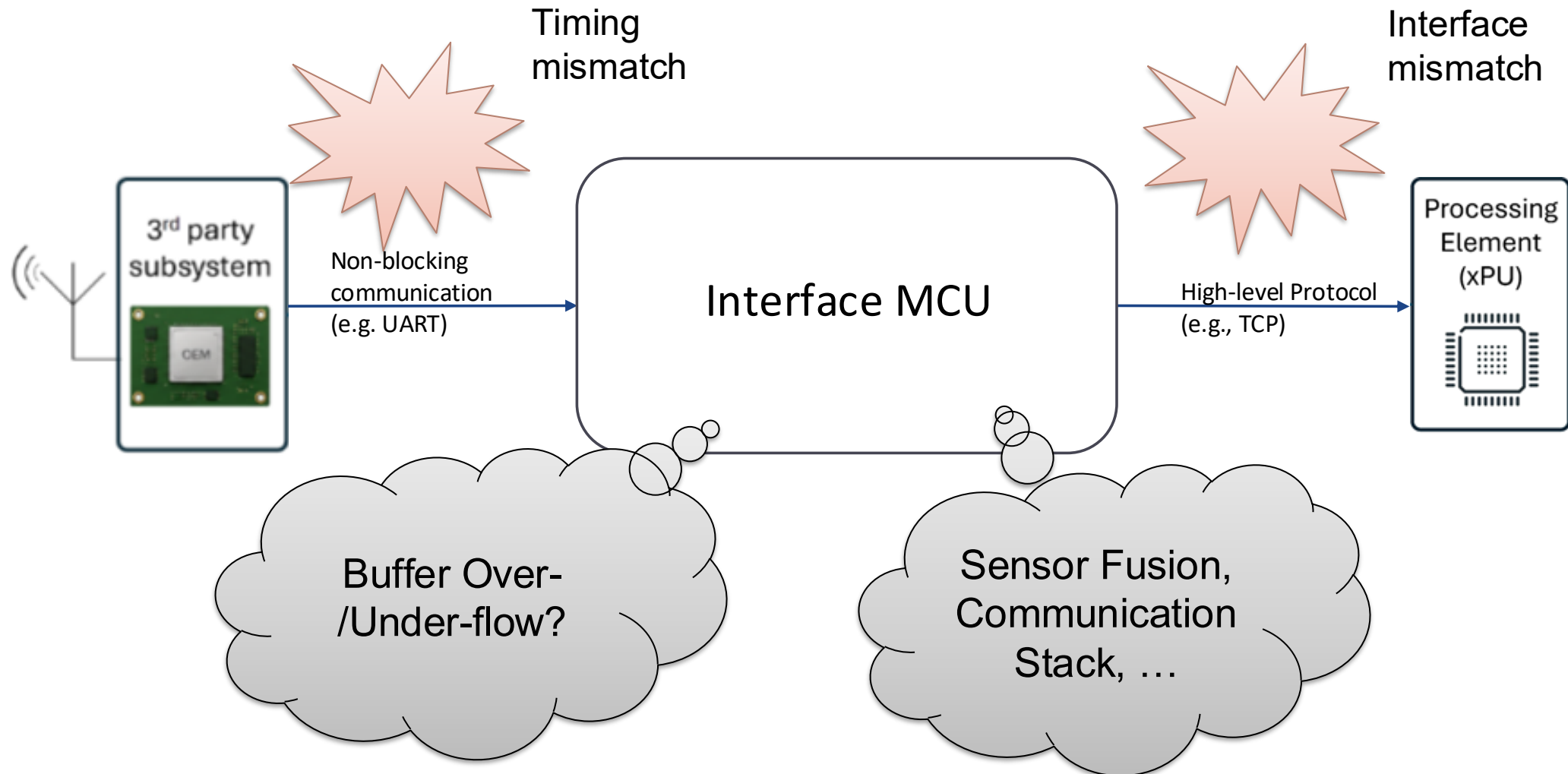
In this talk, I would like to illustrate

- System integration creates **timing-sensitive interfaces**
 - Timing can affect correctness, yet interface timing is often handled ad hoc.
- **No single execution model** can govern the entire system
 - E.g., LF-specified sub-system integrated with AUTOSAR
- Timing parameters often encode **arbitrary engineering choices**
 - Periods, thresholds, and timeouts may be heuristic, overly conservative, or unnecessary sources of uncertainty.
- **Reactor-based modeling** can make these assumptions explicit and **improve predictability and portability**.
 - even when applied to only part of the system

Outline

- GNSS Pipeline and Interface MCU Challenges
- Proposed Lingua Franca Modeling and Dynamic Thresholding
- Evaluations and Takeaways

Integrating OEM products that do not care about others ...



Threshold-based HW-FIFO

28.7.2 Operation Modes

28.7.2.1 FIFO Operation Mode

In normal FIFO operation mode the content of the FIFO is written and read in first-in first-out order, where the data is destroyed after it is delivered to the system bus or the F2A sub-module (see chapter “FIFO to ARU Unit”).

The upper and lower watermark registers (registers **FIFO[i]_CH[x]_UPPER_WM** and **FIFO[i]_CH[x]_LOWER_WM**) are used for controlling the FIFO’s fill level. If the fill level falls below the lower watermark or it exceeds the upper watermark, an interrupt signal is triggered by the FIFO sub-module if enabled inside the **FIFO[i]_IRQ_EN**.

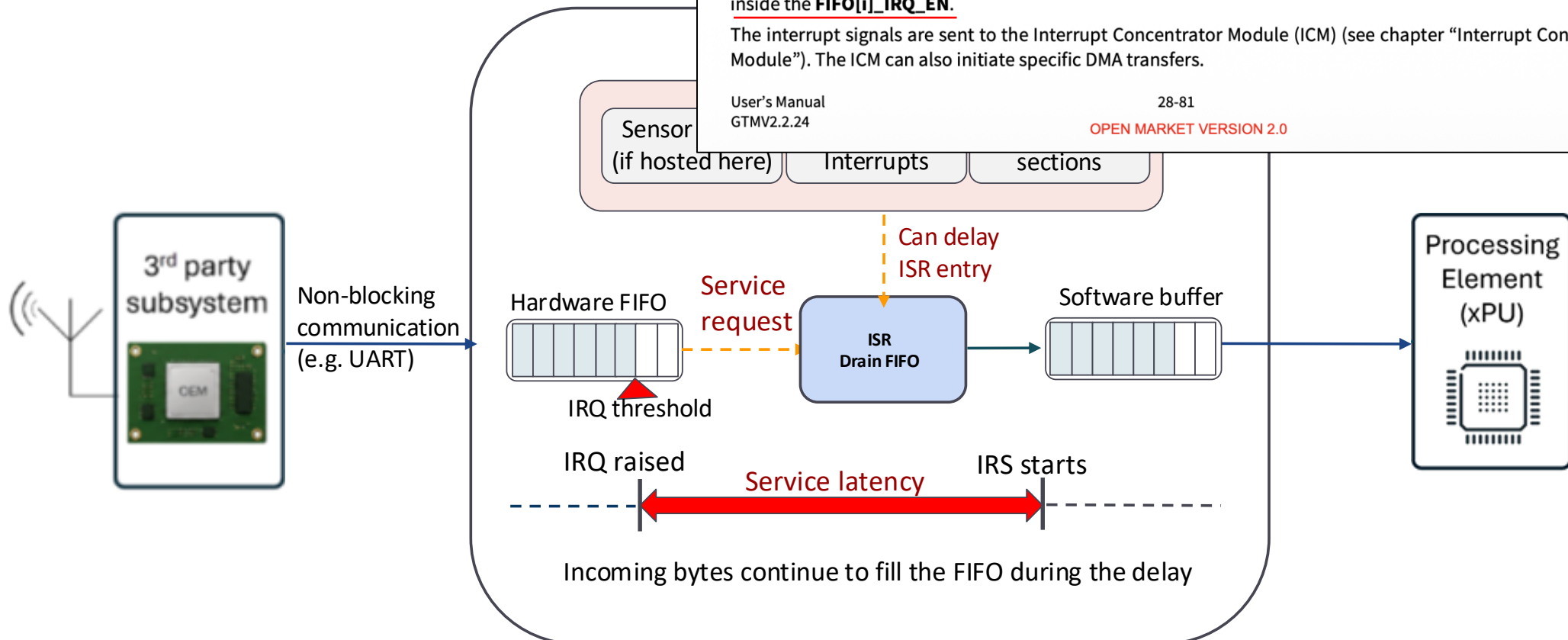
The interrupt signals are sent to the Interrupt Concentrator Module (ICM) (see chapter “Interrupt Concentrator Module”). The ICM can also initiate specific DMA transfers.

User’s Manual
GTMV2.2.24

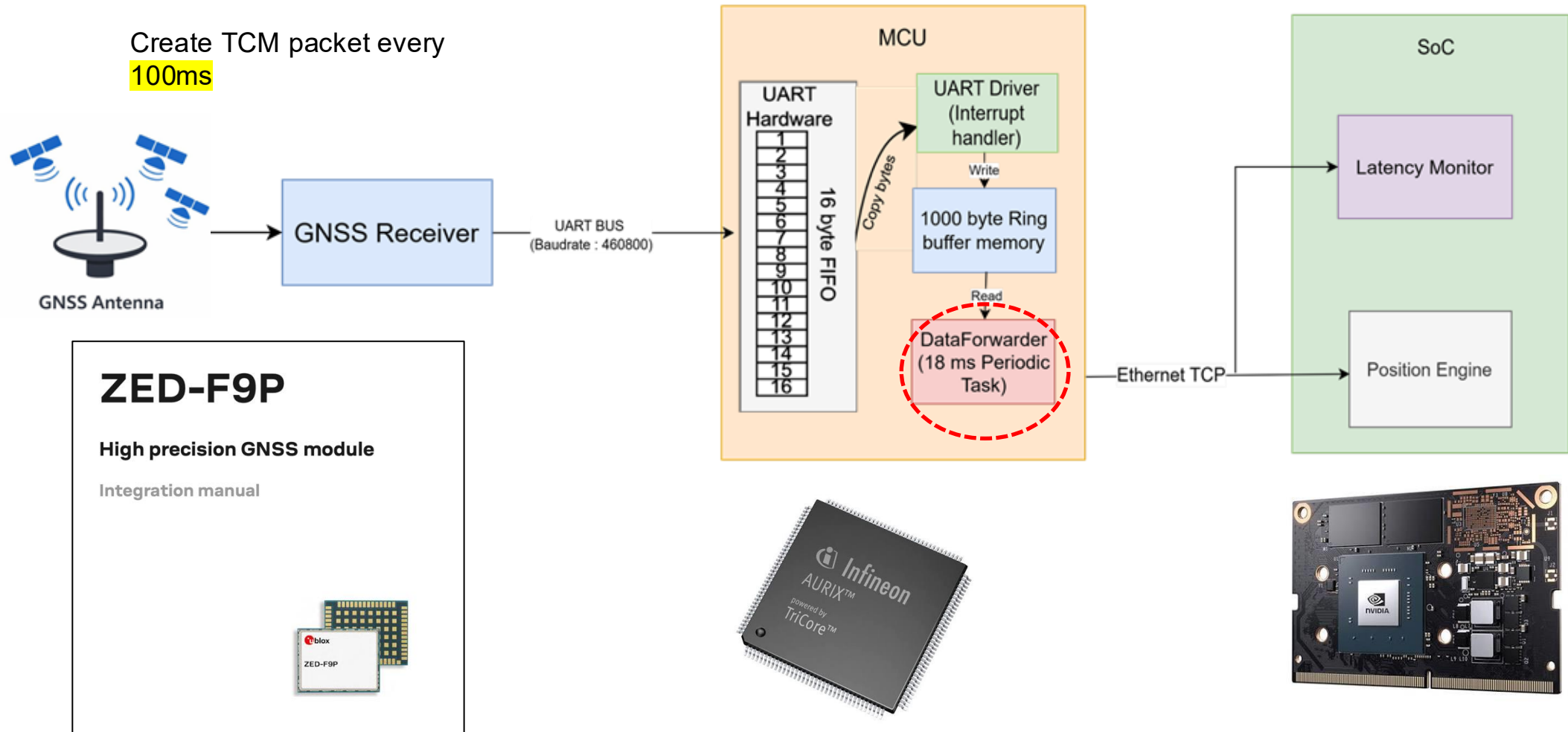
28-81

OPEN MARKET VERSION 2.0

V2.0.0
2021-02



A Real-World Automotive Pipeline – GNSS Case Study



T. Jayaramareddy et al., “Ensuring Deterministic Timing in a Federated GNSS Correction Pipeline with Lingua Franca,” , ReCPS 2026

Timing and Design Parameters

- 1 Epoch Time**
100 ms - GNSS receiver period operating at 10 Hz
- 2 UART Baudrate**
460,800 baud - Serial communication rate for data transfer
- 3 HW Buffer**
16 bytes - UART FIFO hardware buffer size
- 4 Ring Buffer**
1,000 bytes - Software buffer capacity for data storage
- 5 Data Forwarder period**
18 ms - Network transmission period for data forwarding

From Application

start + 1B + end

$$T_{\text{UART,max}} = \frac{800 \text{ bytes} \times 10 \text{ bits/byte}}{460,800 \text{ bits/s}} \approx 17.36 \text{ ms}$$

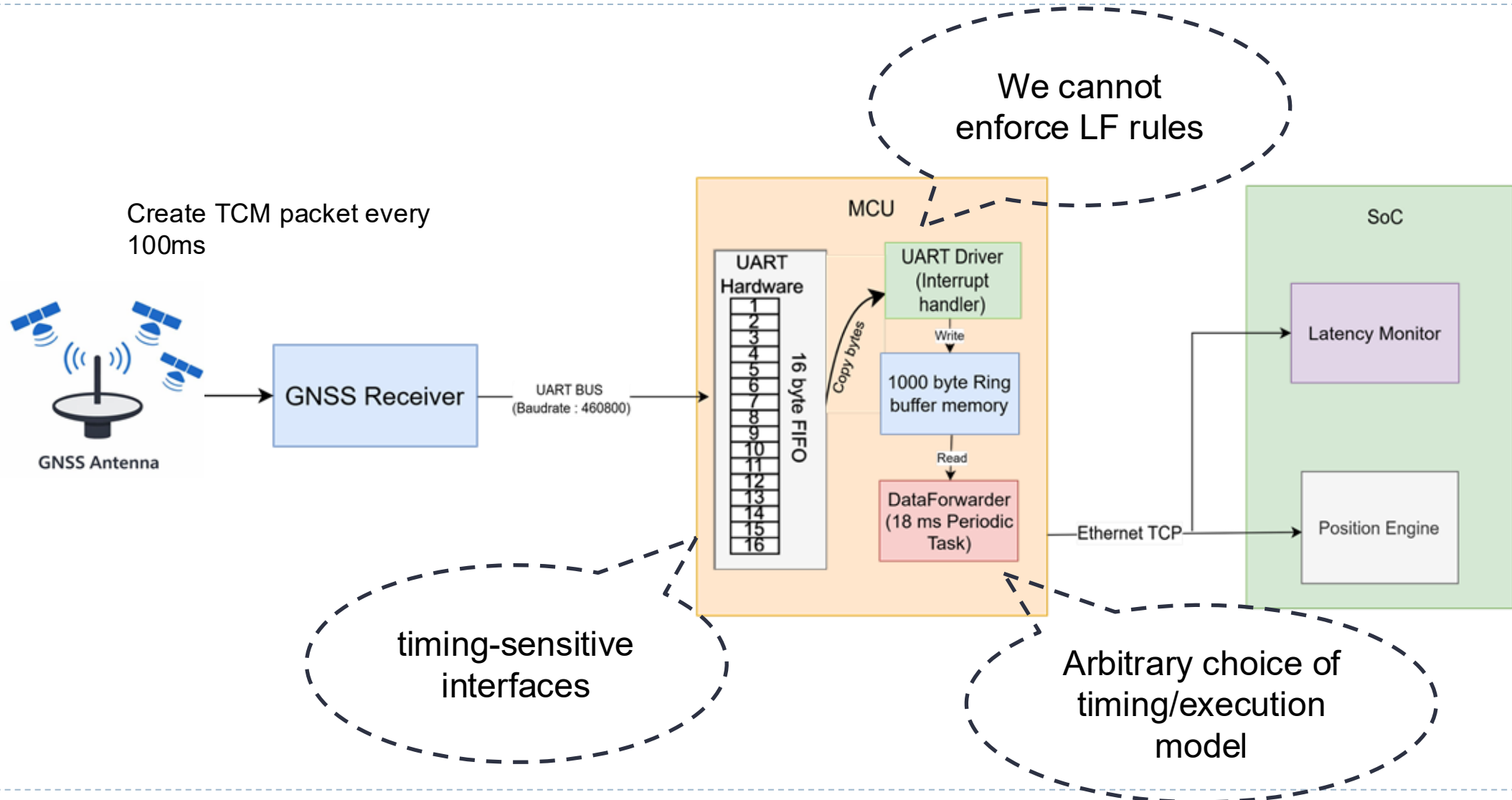
From UART spec

$$T_{\text{Forwarder}} \approx 18 \text{ ms}$$

“Does it even need to be periodic?”

“Yes, because ...”

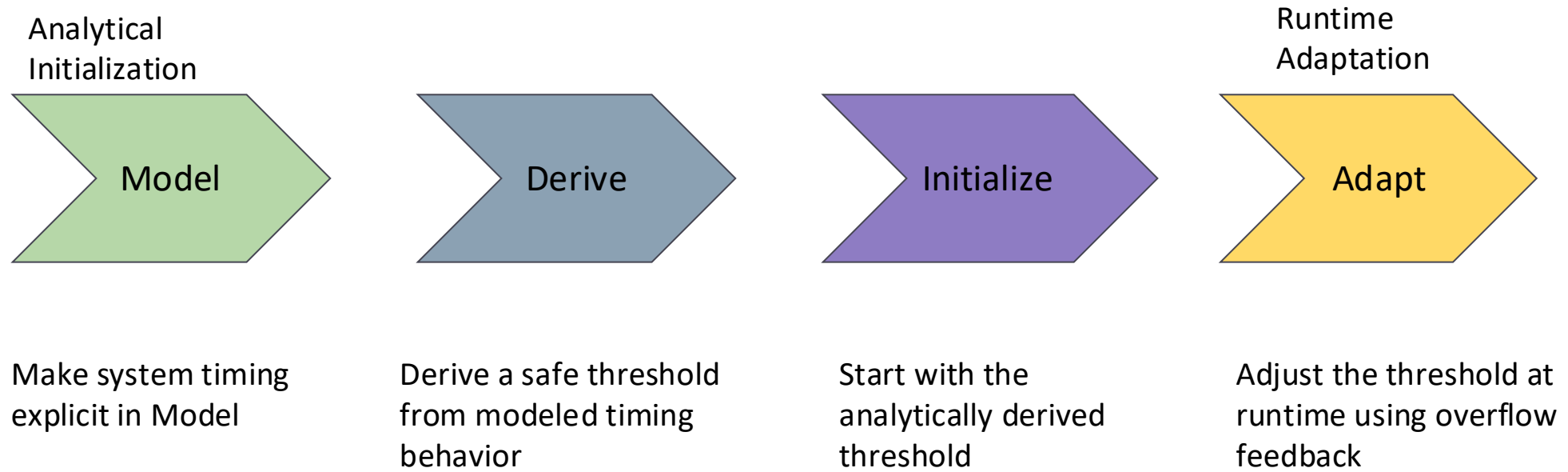
Observations



Outline

- GNSS Pipeline and Interface MCU Challenges
- **Proposed Lingua Franca Modeling and Dynamic Thresholding**
- Evaluations and Takeaways

How can we select the threshold systematically?



Hardware :

- Bytes arrive independently
- Fixed Communication rate

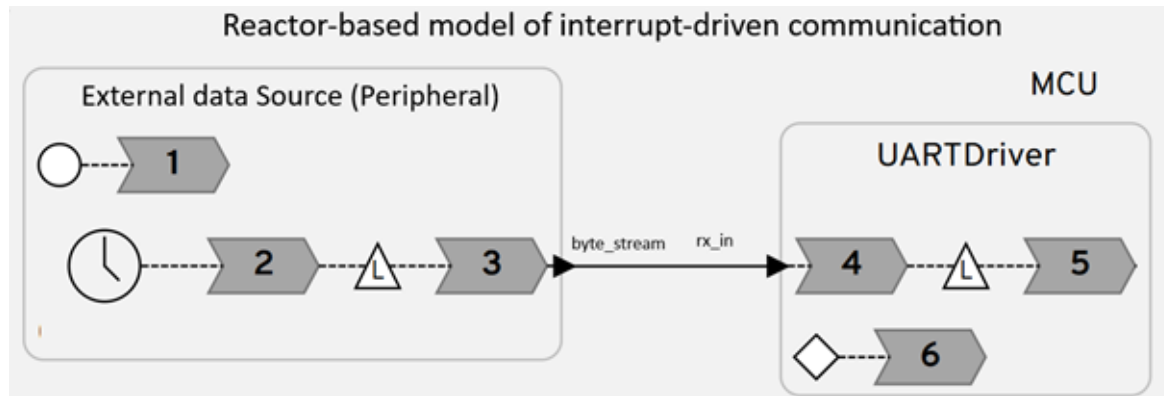
Software :

- ISR executes after variable latency
- CPU workload affects response time

Reactor model :

- Makes both event streams explicit
- Gives deterministic timing semantics

Reactor-Based Communication Model in Lingua Franca



Each transmitted byte is modeled as an individual logical event, faithfully capturing UART byte-serial timing within deterministic LF execution semantics.

Peripheral Side (Reactions ①–③)

- ① Startup initialization
- ② Periodic packet generation via timer trigger
- ③ Emits one logical byte event per UART byte, separated by **byte interval** τ_b

MCU / UARTDriver (Reactions ④–⑥)

- ④ Increments FIFO occupancy; schedules interrupt at **Threshold** T
- ⑤ ISR: drains FIFO into software memory after **Interrupt response latency** L_s
- ⑥ Overflow detection feedback loop

Analytical Safe Threshold Derivation

Step 1 : Byte Transmission Interval :

$$\tau_b = \frac{10 \times 10^b}{b} \dots\dots\dots \text{Eq (1) where } b \text{ is the baud rate. Factor of 10 accounts for start bit, 8 data bits, and stop bit}$$

Step 2 : Total FIFO Service Latency :

$$L_{\max} = L_s + L_e \dots\dots\dots \text{Eq (2)}$$

L_s = Interrupt response latency

L_e = ISR execution time to drain FIFO.

Step 3 : Maximum Safe Latency Condition :

$$L_{\max} \leq (N + 1 - T)\tau_b - 1 \dots\dots\dots \text{Eq (3)}$$

N : FIFO Size, T : Interrupt threshold.

Overflow is avoided only if the ISR drains FIFO before remaining $N - T$ locations are exhausted.

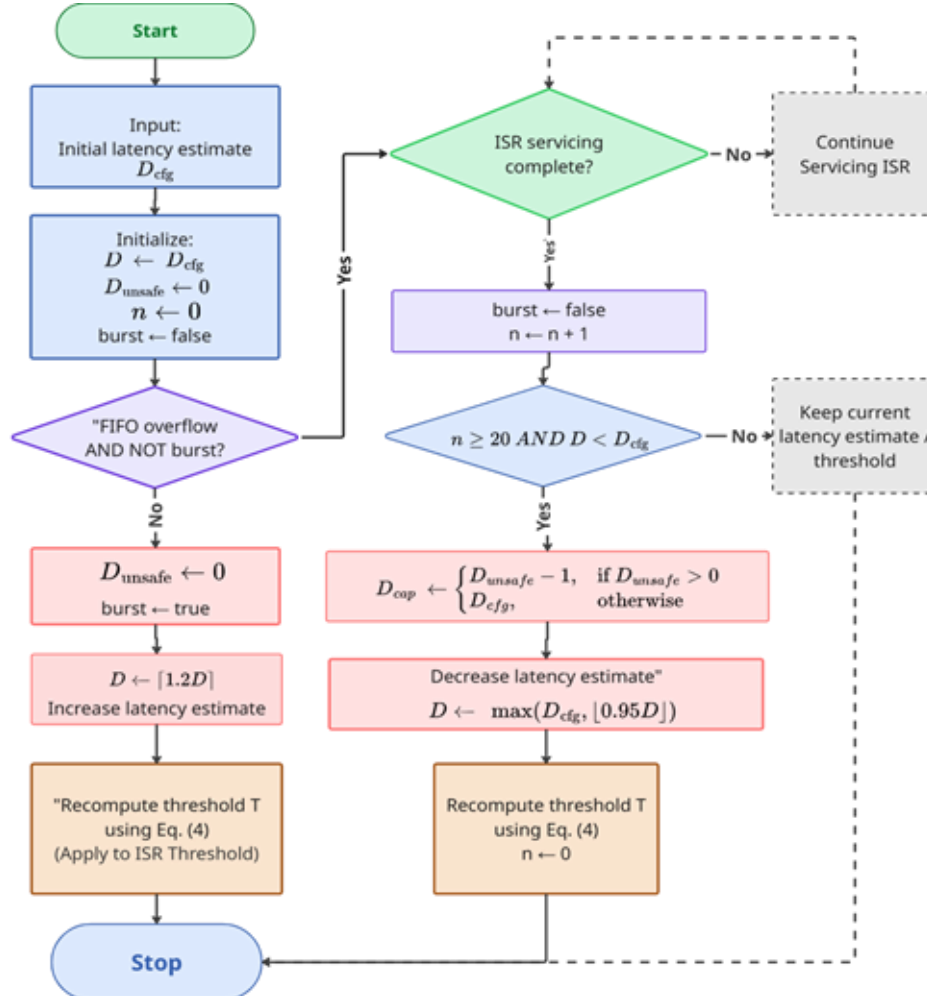
Step 4 : Maximum Safe Threshold :

$$T^* = \lfloor N + 1 - (L_{\max} + 1) / \tau_b \rfloor \dots\dots\dots \text{Eq (4)}$$

Highest overflow-free threshold for the assumed worst-case latency. Subtraction of 1 enforces strict inequality.

AIMD Adaptation for Unknown Latency

AIMD (Additive Increase, Multiplicative Decrease)



AIMD Notation

D_{cfg} : Initial configured ISR latency estimate

D : Current ISR latency estimate.

D_{unsafe} : Last latency estimate associated with overflow.

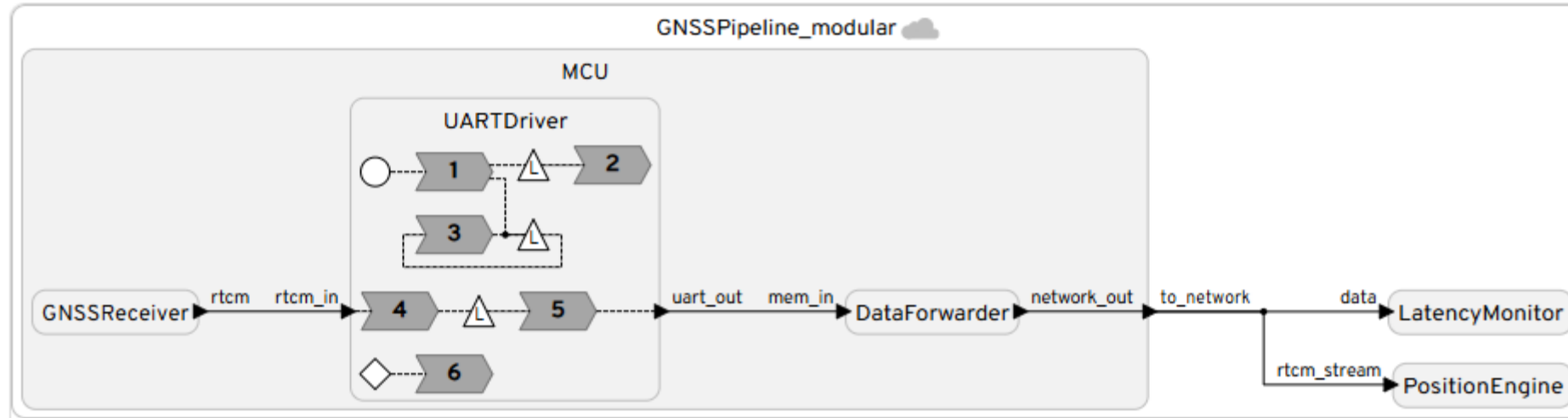
D_{cap} : Maximum allowed latency estimate

n : Consecutive overflow-free ISR completions

T : FIFO interrupt threshold

burst : Overflow-burst flag

Federated LF Implementation



Initialization and Control flow(Reactions ①–③)

- ① Initializes FIFO, computes interrupt threshold and sets latency bound
- ② Starts the validation phase to check whether the chosen threshold is safe
- ③ Runs threshold sweep and validates the maximum latency boundary

Data handling and validation(④–⑥)

- ④ Receives RTCM bytes, writes to the FIFO, handles overflow, and schedules the hardware interrupt
- ⑤ Drains FIFO, emits the packet downstream, and adapts threshold after stable operation.
- ⑥ Prints final report

Federated LF Model Structure

Top-Level Topology



```
1 federated reactor GNSSPipeline {  
2   mcu = new MCU()  
3   soc = new PositionEngine()  
4   monitor = new LatencyMonitor()  
5  
6   // Network connection with deterministic timing  
7   mcu.to_network -> soc.rtcn_stream  
8   mcu.to_network -> monitor.data  
9 }  
10
```

Decentralized Coordination :

```
coordination: decentralized  
advance-message-interval: 10 msec
```

The 10 ms AMI bounds logical-time divergence between federates while allowing independent advancement. Since AMI is one-tenth of the 100 ms GNSS epoch, timing observations remain accurate within a single correction cycle.

Composing the MCU pipeline

1. Component instantiation

```
1 gnss = new GNSSReceiver(  
2     convergence_epochs = convergence_epochs,  
3     validation_epochs   = validation_epochs,  
4     sweep_epochs        = 800 // 16 probes x 50 epochs for threshold verification sweep  
5 )  
6 forwarder = new DataForwarder()
```

GNSS message generation runs every 100 ms.
The forwarder runs every 18 ms. Both are separate reactors.

2. Interrupt-latency configuration

```
1 uart = new UARTDriver(convergence_epochs = convergence_epochs,  
2                       validation_epochs   = validation_epochs,  
3                       max_interrupt_latency_us = 1000)
```

The UART latency bound starts at 1000 μ s.
Shared parameters align the learning and validation phases of GNSS and UART

3. Explicit port connections

```
1 gnss.rtcn -> uart.rtcn_in  
2 uart.uart_out -> forwarder.mem_in  
3 forwarder.network_out -> to_network
```

MCU exposes forwarded packets through its to_network output

Modeling GNSS byte-arrival timing

1. Periodic packet generation

```
1 logical action emit_byte: data_packet_t
2   timer epoch(0, 100 msec)
```

The epoch timer triggers packet generation at 10 Hz

2. Model UART serialization

```
1 // Emit bytes one-by-one so UARTDriver consumes one byte per event.
2 // Spacing matches real UART byte time: 10 bits / baudrate.
3 // UART framing: 1 start + 8 data + 1 stop = 10 bits per byte.
4 int byte_time_us = (10 * 1000000) / self->baudrate;
5 // const data_packet_t* packet = &emit_e2e->value;
6 for (int i = 0; i < packet.bytes; i++) {
7   data_packet_t one_byte = {
8     .bytes = 1
9   };
10  one_byte.data[0] = packet.data[i];
11
12  lf_schedule_copy(emit_byte, USEC(i * byte_time_us), &one_byte, 1);
13 }
```

10 bits per byte: start bit + 8 data bits + stop bit.

One timed event per byte captures FIFO filling during ISR delay

Modeling UART FIFO and ISR latency

1. FIFO occupancy and overflow

```
1 reaction(rtc_in) -> hw_interrupt {=  
2     if (rtc_in->value.generation_time > 0)  
3         self->last_generation_time = rtc_in->value.generation_time;  
4  
5     // No space in FIFO – drop byte and report loss.  
6     if (self->fifo_level >= self->hw_buffer_size) {  
7         self->total_bytes_lost++;  
8         self->overflow_events++;  
9         return; // byte was dropped – do not write to FIFO  
10    }  
11    // Write byte into FIFO.  
12    self->fifo_data[self->fifo_tail] = rtc_in->value.data[0];  
13    self->fifo_tail = (self->fifo_tail + 1) % self->hw_buffer_size;  
14    self->fifo_level++;
```

Each arrival fills FIFO, dropped if FIFO is full

3. FIFO drain at ISR service

```
1 int bytes_to_drain = self->fifo_level;  
2 self->fifo_level = 0;  
3  
4 if (bytes_to_drain > 0) {  
5     data_packet_t out_packet = {0};  
6     out_packet.bytes = bytes_to_drain;  
7     out_packet.generation_time = self->last_generation_time;  
8  
9     for (int i = 0; i < bytes_to_drain; i++) {  
10        out_packet.data[i] = self->fifo_data[self->fifo_head];  
11        self->fifo_head = (self->fifo_head + 1) % self->hw_buffer_size;  
12    }  
13  
14    lf_set(uart_out, out_packet);
```

ISR forwards buffered bytes and frees FIFO space

2. Threshold-triggered interrupt with delay

```
1  
2 // Trigger hardware interrupt when FIFO reaches threshold OR packet is complete.  
3 if (!self->interrupt_pending && (self->fifo_level >= self->interrupt_threshold || is_last_byte)) {  
4     int latency_us = is_last_byte ? 0 : rand() % (self->current_max_latency_us + 1);  
5     self->interrupt_pending = true;  
6     lf_schedule_int(hw_interrupt, USEC(latency_us), self->fifo_level);  
7 }
```

Sampled delay models ISR latency. Bytes continue arriving while an interrupt is pending.

Modeling periodic data forwarding

1. Periodic forwarding

```
timer period(0, 18 msec)
  reaction(period) -> network_out, forwarding_watchdog
```

18 ms forwarding period. The timer triggers processing of bytes in the ring buffer.

2. Packet integrity

```
parse_e2e_packet(&out_packet);
  if (!validate_e2e_crc(&out_packet)) {
    self->total_messages_corrupted++;
  } else {
    lf_set(network_out, out_packet);
  }
```

Check for CRC failures and forward valid packets

3. Timing and liveness monitoring

```
1 reaction(period) -> network_out, forwarding_watchdog {=
2   lf_watchdog_start(forwarding_watchdog, 0); // Pet watchdog
3
4   // LF TIME TRACKING: Measure execution lag
5   instant_t logical_now = lf_time_logical();
6   instant_t physical_now = lf_time_physical();
7   interval_t execution_lag = physical_now - logical_now;
```

Logical time meets physical time.

Lag measures physical-time delay.

100 ms watchdog monitors refresh

Outline

- GNSS Pipeline and Interface MCU Challenges
- Proposed Lingua Franca Modeling and Dynamic Thresholding
- Evaluations and Takeaways

Experimental Setup and Baseline

Methodology



UART bytes arrive
at 460,800 bps



Buffered in 16-
byte HW FIFO



Interrupt at
threshold T



ISR after latency
drains FIFO



Packets verified
(CRC-32), E2E



PASS (No overflow)



FAIL (Overflow)

Applying the analytical formula for a worst case latency of 200 μ s:

$$T = 16 - \left\lfloor \frac{200}{21} \right\rfloor - 1 = 6$$

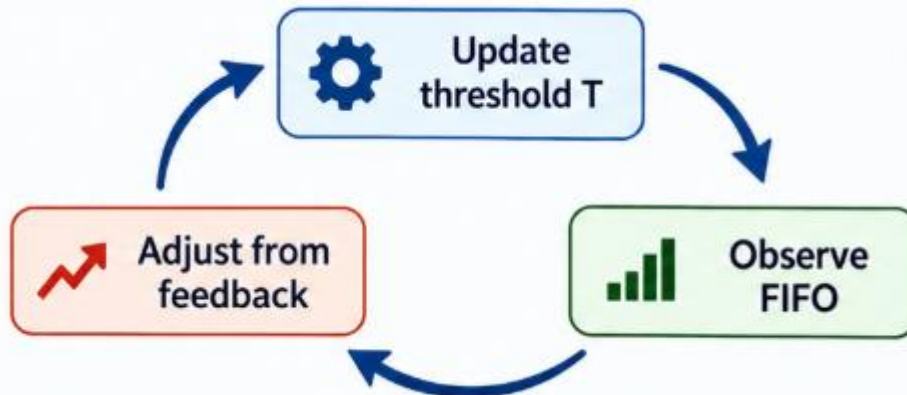
- This yields a conservative analytical threshold of **$T^* = 6$** , corresponding to a maximum tolerable latency of **$\text{safe_ceil}(6) = 230 \mu\text{s}$** .
- $\text{safe_ceil}(T)$ denotes the maximum FIFO service latency that can be tolerated without overflow for interrupt threshold T.

Blind AIMD Threshold Discovery

Scenario

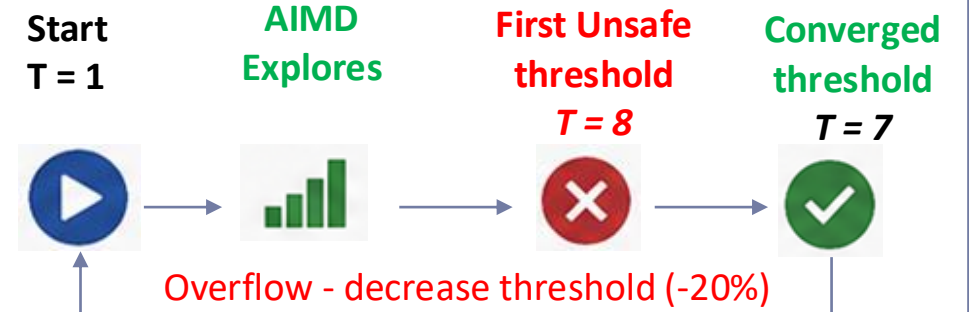
Controller starts from the most conservative threshold ($T = 1$) with **no prior knowledge** of ISR latency (true value: $200\text{ }\mu\text{s}$). FIFO overflow events are the only feedback signal.

How the controller learns



Runtime Threshold Discovery

Unknown ISR latency = $200\text{ }\mu\text{s}$ (No prior knowledge)



	Learned Threshold (Stable)	$T = 7$
	First Unsafe Threshold	$T = 8$
	Learning Epochs	400
	Learning Overflows	42
	Verification Epochs	1,000
	Verification Overflows	0

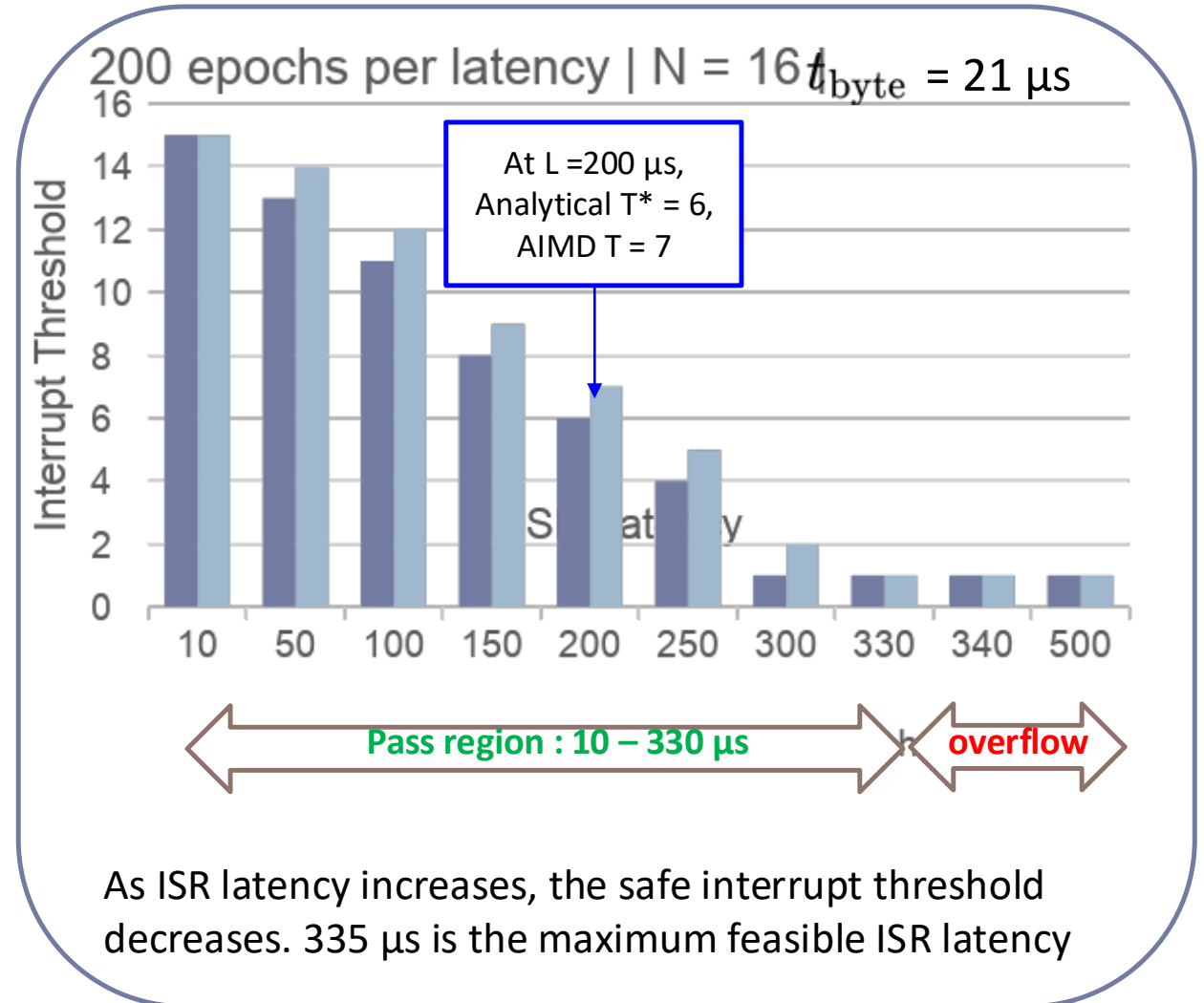
Formula Validation Under Varying ISR Latencies

AIMD Exploits Timing Margin :

AIMD consistently converges to thresholds equal to or one level higher than the analytical prediction, safely exploiting remaining timing margin.

Hardware Infeasibility Identified :

Once ISR latency exceeds $\text{safe_ceil}(1) = 335 \mu\text{s}$, overflow is unavoidable regardless of threshold — a condition threshold tuning alone cannot fix.



Empirical boundary Validation

Empirical boundary validation at Interrupt latency 200 μ s

Every threshold from 1 to 16 tested

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
PASS	PASS	PASS	PASS	PASS	PASS	PASS	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL

Interrupt threshold T

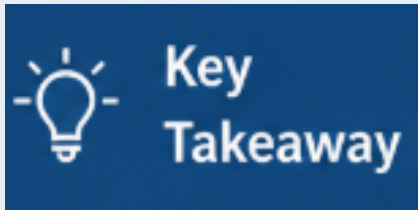
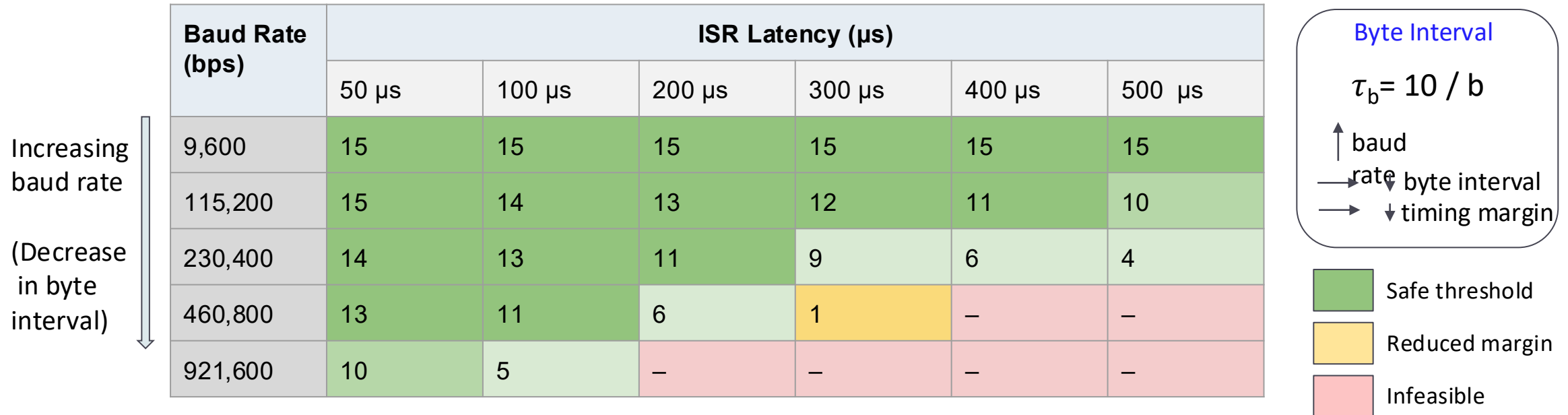
$$T \leq 7$$

No overflow observed

$$T = 8$$

First unsafe threshold

Higher Baud Rates Shrink the Safe Operating Region



As baud rate increases, **bytes arrive faster**, reducing the available ISR response time, At 460,800 baudrate and 921,600 some latency budgets are infeasible regardless of the threshold selection.

Analytical vs AIMD Complementary Strengths

Analytical Formula :

Provides a **conservative, guaranteed-safe** threshold when latency is known. Directly computable from FIFO depth, baud rate, and worst-case latency. Identifies hardware-infeasible configurations at design time.

AIMD Adaptation :

Operates without prior latency knowledge. Converges to the **tightest safe threshold** using only overflow feedback. AIMD found $T = 7$ vs. analytical $T^* = 6$ — a 17% reduction in interrupt overhead.

Why Reactor Based Model Matters

Deterministic Timing Abstraction :

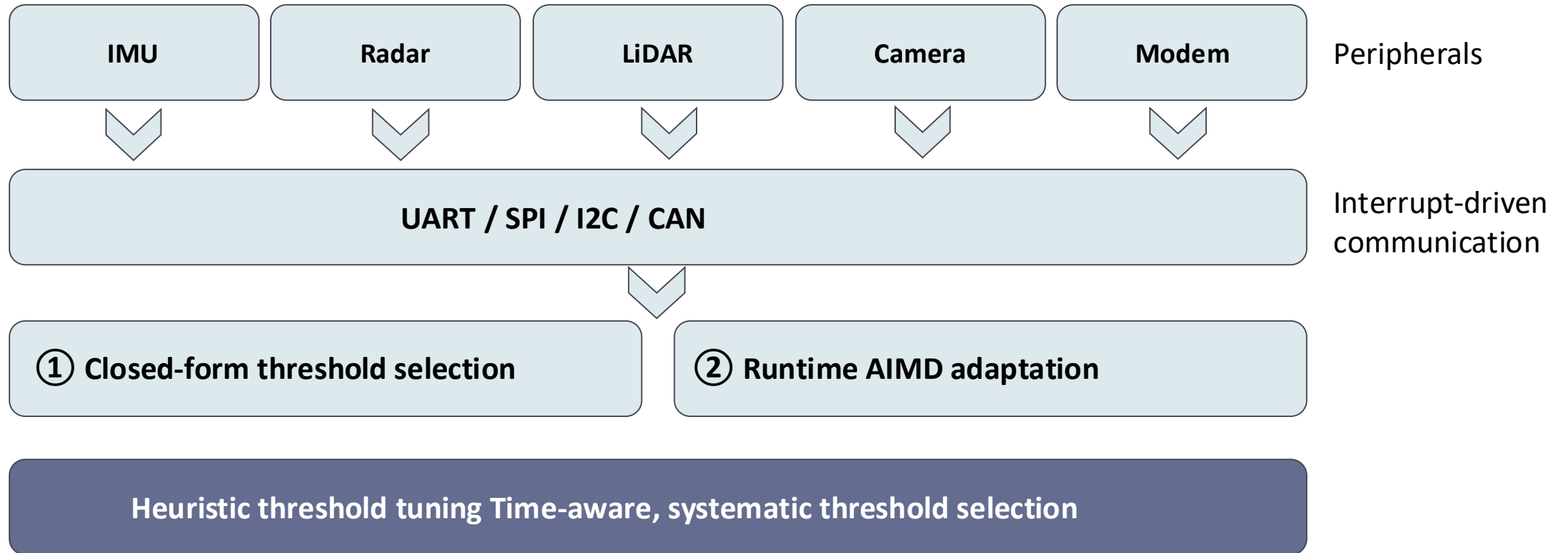
Lingua Franca represents byte arrivals and ISR scheduling as independent logical events, making concurrent hardware-software timing explicit and analyzable — not buried in implementation details.

Executable System Model :

The same LF model used for analysis is also the running implementation. Varying only FIFO threshold and ISR latency parameters inside UARTDriver enables non-intrusive, reproducible evaluation across all configurations with a fixed random seed.

Generalizability Beyond GNSS use case

The same timing-aware methodology extends to a broad class of interrupt-driven, FIFO-based interfaces



In this talk, I would like to illustrate

- System integration creates **timing-sensitive interfaces**
 - Timing can affect correctness, yet interface timing is often handled ad hoc.
- **No single execution model** can govern the entire system
 - E.g., LF-specified sub-system integrated with AUTOSAR
- Timing parameters often encode **arbitrary engineering choices**
 - Periods, thresholds, and timeouts may be heuristic, overly conservative, or unnecessary sources of uncertainty.
- **Reactor-based modeling** can make these assumptions explicit and **improve predictability and portability**.
 - even when applied to only part of the system

Conclusion

- Finite buffers create servicing deadlines: delayed servicing can cause data loss in sensor pipelines.
- **Analytical timing analysis** determines the maximum safe interrupt threshold from FIFO depth, data rate, and interrupt latency.
- **Runtime adaptation** handles unknown or time-varying interrupt latency without manual tuning.
- An executable LF model makes timing explicit: byte arrivals, FIFO occupancy, and interrupt servicing support reproducible evaluation.
- Evaluation confirms the **predicted overflow boundary** and maintains **overflow-free operation**.
- GNSS case-study results validate the approach: adaptation found $T = 7$ at $200\ \mu\text{s}$, with zero overflows over 1,000 verification epochs following exploration.
- The methodology provides a **systematic alternative to heuristic threshold selection** for FIFO-based peripheral interfaces.

Thank you.

hoeseok.yang@scu.edu