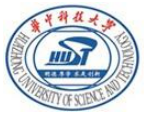
Stop Starving or Stuffing Me: Boosting Firmware Fuzzing Efficiency with On-demand Input Delivery

Shandian Shen, Wei Zhou, Keming Zhao (HUST)

Peng Liu (Penn State) · Chung Hwan Kim (UT Dallas) · **Le Guan (UGA)**



華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

PennState

THE UNIVERSITY OF TEXAS AT DALLAS

UNIVERSITY OF GEORGIA

Background

Healthcare

Industrial Control

Infrastructure

Smart Home

- Monolithic
- Resource limited
- Diverse hardware

↓

- Hard to test on real devices

↓

- Rehosting-based firmware fuzzing

Problems with Firmware Fuzzing

Application Software

```
int main() {

    char input_buf[BUF_SIZE];
    printf("\n[Program] Waiting for external input ...\n");
    // Synchronous POSIX I/O:
    ssize_t bytes_read = read(STDIN_FILENO,
                              input_buf,
                              BUF_SIZE - 1);
```

Input Delivery

Software: Receives input from file/STDIN following the POSIX I/O interface **synchronously**.

Problems with Firmware Fuzzing

Firmware

```
//================ISR Context====================
void UART_IRQHandler() {
    store_char(&Serial, Serial->UART_DR);
}
void store_char(RingBuffer *this, char c) {
    if ((this->Head + 1) & 127 != this->Tail) {
        //R(P): Input retrieval from periperhal
        //Store Input in limited ring ruffer
        this->rx_buffer[this->Head] = c;
        this->Head = (this->Head + 1) & 127;
    }
}
```

```
// ============Main Execution Context==========
void loop(...) { |\label{line:loop}|
    switch(slave.state) {
        case 1: Modbus.poll();
                ...//P1: Input Processing 1
                slave.state++;
                master.state = 1;
            }
        case 2: Modbus.poll();
                ...//P2: Input Processing 2
}
int UARTClass::available(UARTClass *this) {
    return this->Head - this->Tail & 127;
}
```

Consequence:

- The input **arrival time** and **quantity** become undecidable for a fuzzer.

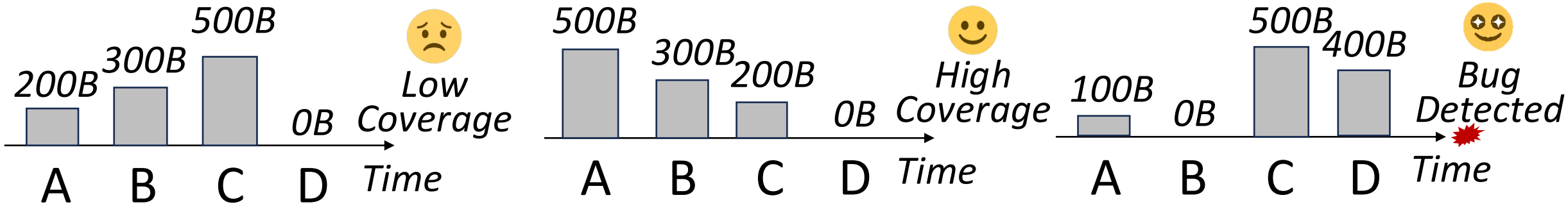
```
    while (this->Head - this->Tail != 0)
        this->port->read();
}
int UARTClass::read(UARTClass *this) {
    result = this->rx_buffer[this->Tail];
    this->Tail = (this->Tail + 1) & 127;
```

```
}
int Modbus::poll(Modbus *const this, ...) {
    // C(A): Input Avaiablity Check
    if (this->port->available() == 0)
        return 0;
    // C(L):Input Length Check
    if (Modbus::getRxBuffer(this) <= 7)
```

Input Delivery

Firmware: Interrupts notify the firmware to retrieve data from peripheral **asynchronously.**

Problems with Firmware Fuzzing



A, B, C are different delivery points.

- The same fuzzing input
- The same firmware
- Different timing and distribution

→ Very different execution outcomes!

Finding/Problem One

```
//================ISR Context==================
void UART_IRQHandler() {
    store_char(&Serial, Serial->UART_DR);
}
void store_char(RingBuffer *this, char c) {
    if ((this->Head + 1) & 127 != this->Tail) {
        //R(P): Input retrieval from periperhal
        //Store Input in limited ring ruffer
        this->rx_buffer[this->Head] = c;
        this->Head = (this->Head + 1) & 127;
    }
}
```

F1: Limited Ring Buffer for Input Retrieval (R_P)

- To reduce latency, an ISR quickly retrieve the data from peripheral and then exit
- Data processing is postponed
- Data is stored in a reserved ring buffer
- The buffer size varies, generally ranging from several bytes to hundreds of bytes

P1 Overfeeding (overwritten/ignored)

- Excessive ISR delivery overflows the ring buffer
- Waste fuzzing input

Finding/Problem Two

```
// ============Main Execution Context===========
void main(...) {
  Modbus::begin(&slave, ...);
  while (1)
  {loop();}
}
void Modbus::begin(Modbus *const this, ...) {
    while (this->Head - this->Tail != 0)
        this->port->read();
}
int UARTClass::read(UARTClass *this) {
    result = this->rx_buffer[this->Tail];
    this->Tail = (this->Tail + 1) & 127;
    return result;
}
```

F2 Buffer Cleanup at Peripheral Reset

- Every time when the peripheral needs to be reset into a predictable and clean state, the associated ISR buffer is cleared

P2 Input Discard Due to Incorrect Timing

- Bytes injected before buffer cleanup are not read

Finding/Problem Three

```
// ============Main Execution Context==========
void loop(...) { |\label{line:loop}|
    switch(slave.state) {
        case 1: Modbus.poll();
                ...//P1: Input Processing 1
int UARTClass::available(UARTClass *this) {
    return this->Head - this->Tail & 127;
}
int Modbus::getRxBuffer(UARTClass *this) {
    BufferSize = 0;
    while (this->port->available()) {
        //R(B): Input retrieval from Buffer
        auBuffer[BufferSize] = this->port->read();
        BufferSize++;
    }
    return BufferSize;
}
int Modbus::poll(Modbus *const this, ...) {
    // C(A): Input Avaiablity Check
    if (this->port->available() == 0)
        return 0;
    // C(L):Input Length Check
    if (Modbus::getRxBuffer(this) <= 7)
        return 1;
    ....
}
```

F3 Availability / Length Checks

- The ring buffer is filled by ISR functions asynchronously
- The firmware must check if there is new data available before using it
- In addition, some protocols require a minimal amount of data before they can be processed

P3 Unnecessary Availability & Length Check

- Existing fuzzers repeatedly fail availability check (C_A) and length check (C_L)
- Wasted execution time

Finding/Problem Four

```
// ============Main Execution Context==========
void loop(...) { |\label{line:loop}|
    switch(slave.state) {
        case 1: Modbus.poll();
                ...//P1: Input Processing 1
                slave.state++;
                master.state = 1;
            }
        case 2: Modbus.poll();
                ...//P2: Input Processing 2
}
```

```
int Modbus::poll(Modbus *const this, ...) {
    // C(A): Input Avaiablity Check
    if (this->port->available() == 0)
        return 0;
    // C(L):Input Length Check
    if (Modbus::getRxBuffer(this) <= 7)
        return 1;
    ....
}
```

F4 Multiple Processing Points with Different Calling Contexts

- The input processing functions may be invoked at multiple places under different contexts
- The consuming points encountered can vary depending on the input data

P4 Input is not distributed across all calling contexts

- Difficulty in exploring new behaviors
- Some input routes remain unexplored

Existing Firmwrae Fuzzers Suffer from These Problems

Existing fuzzers are unaware of the real firmware needs:

Fuzzer	Timing	Quantity	Problems

Optimal Input Delivery Strategy:

- Timing: deliver input only when the firmware **needs** it
  - Avoid P2 and P4
- Quantity: should be **within** the range of required min. processing length and the buffer size
  - Avoid P1 and P3

Timing

- Round-robin/Fuzz: Fixed/Fuzzed intervals (BB execution) ISR trigger
- Manual Spec. Points: user-specified delivery points
- Waiting-state: inferred interrupt-waiting states

Quantity

- 1B: one byte per ISR trigger
- All Input: whole test case at once
- Manual Spec.: undefined / manually configured quantity

Problems

- P1: buffer overwrite
- P2: cleanup discard
- P3: repeated checks
- P4: missed routes

Modeling Input Handling (CRP)

- Availability Check (C_A):
  - Firmware checks the ring buffer populated by the ISRs.
- Data Retrieval (**R**):
  - Main firmware logic retrieves data from the ISR buffer (R_B).
  - The ISR reads from the peripheral's data I/O and stores it in the buffer (R_P).
- Processing (**P**):
  - Firmware operations involving the input.

```
//=================ISR Context====================
void UART_IRQHandler() {
    store_char(&Serial, Serial->UART_DR);
}
void store_char(RingBuffer *this, char c) {
    if ((this->Head + 1) & 127 != this->Tail) {
        //R(P): Input retrieval from periperhal
        //Store Input in limited ring ruffer
        this->rx_buffer[this->Head] = c;
        this->Head = (this->Head + 1) & 127;
    }
}
// ============Main Execution Context==========
void main(...) {
  Modbus::begin(&slave, ...);
  while (1)
  {loop();}
}
void Modbus::begin(Modbus *const this, ...) {
    while (this->Head - this->Tail != 0)
        this->port->read();
}
int UARTClass::read(UARTClass *this) {
    result = this->rx_buffer[this->Tail];
    this->Tail = (this->Tail + 1) & 127;
    return result;
}
```

R_P

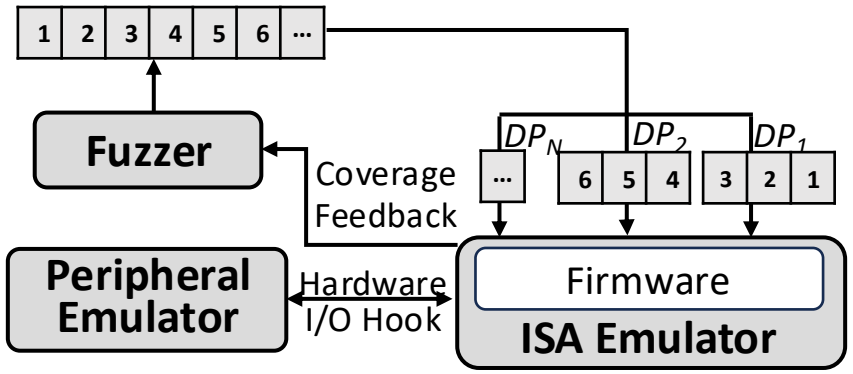
```
// ============Main Execution Context==========
void loop(...) { |\label{line:loop}|
    switch(slave.state) {
        case 1: Modbus.poll();
                ...//P1: Input Processing 1
                slave.state++;
                master.state = 1;
            }
        case 2: Modbus.poll();
                ...//P2: Input Processing 2
}
int UARTClass::available(UARTClass *this) {
    return this->Head - this->Tail & 127;
}
int Modbus::getRxBuffer(UARTClass *this) {
    BufferSize = 0;
    while (this->port->available()) {
        //R(B): Input retrieval from Buffer
        auBuffer[BufferSize] = this->port->read();
        BufferSize++;
    }
    return BufferSize;
}
int Modbus::poll(Modbus *const this, ...) {
    // C(A): Input Avaiablity Check
    if (this->port->available() == 0)
        return 0;
    // C(L):Input Length Check
    if (Modbus::getRxBuffer(this) <= 7)
        return 1;
    ....
}
```

P

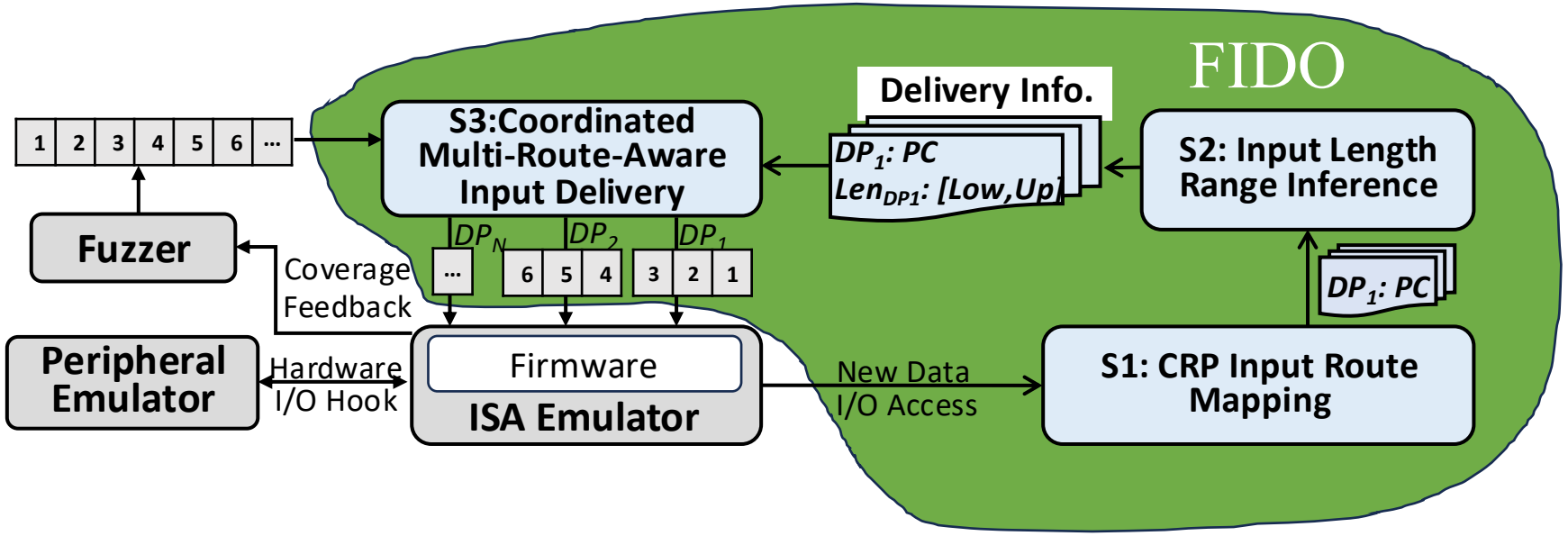
R_B

C_A

Solution Overview: FIDO: Fuzzing Input Delivery Optimizer



Solution Overview: FIDO: Fuzzing Input Delivery Optimizer



S1 CRP Input Route Mapping:

- Recovering CRP input routes.
- Use C_A (Availability Check) sites as delivery points

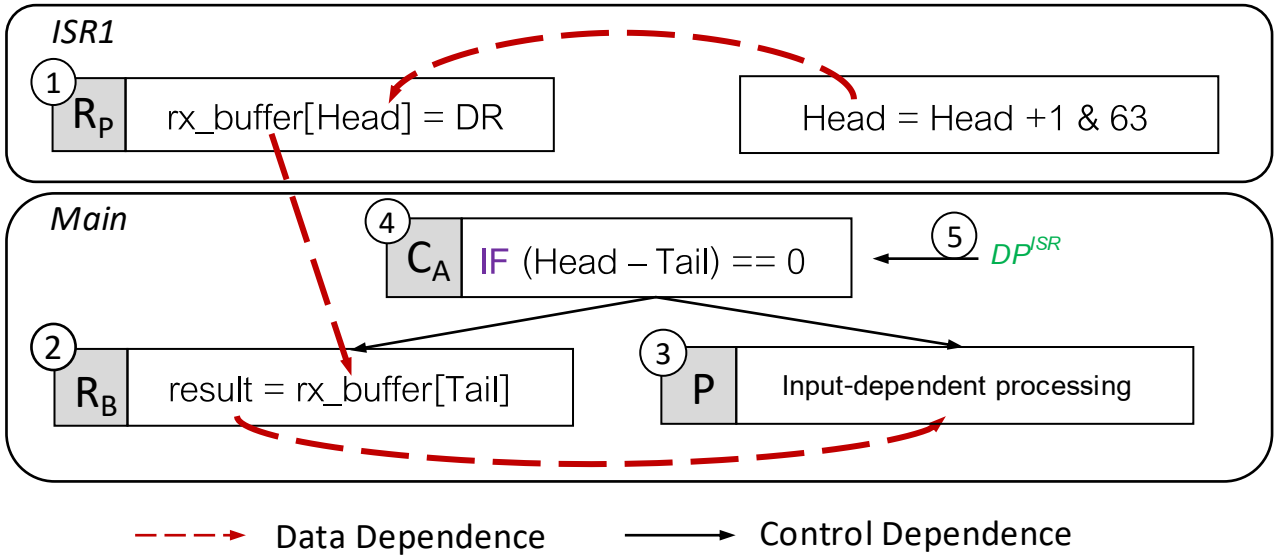
S2 Input Length Range Inference:

- Inferring the valid length range [Low, Up].

S3 Coordinated Multi-Route-Aware Delivery:

- Slice the testcase and deliver segments on demand.

S1: CRP Input Route Mapping



S1.1: Start from R_P
- Locate ISR-side peripheral data retrieval

S1.2: Track data to R_B
- Find where main logic reads the buffer

S1.3. Identify P
- Find input-dependent processing

S1.4. Locate C_A
- Find checks that control R_B and P

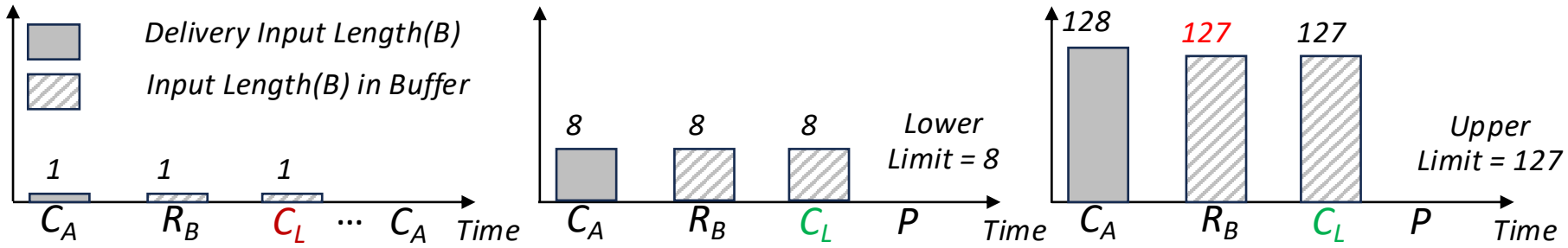
S1.5: Identify Delivery Point (DP)
- The **initial availability check** for each input route as DP

S2: Input Length Range Inference

Goal: find a valid delivery length range: len ∈ [Low, Up]

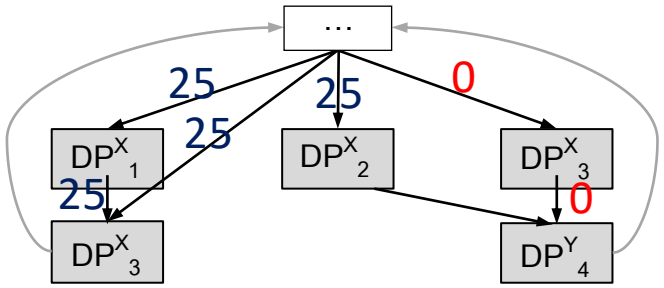
- Low: minimum bytes to get processed
- Up: maximum bytes before buffer is overwritten

1. **Increase delivery length at DP**
   Trigger extra ISRs with dummy input.
2. **Infer Low Limit**
   First length that reaches input processing (P).
3. **Infer Up Limit**
   First length before old buffer data is overwritten.



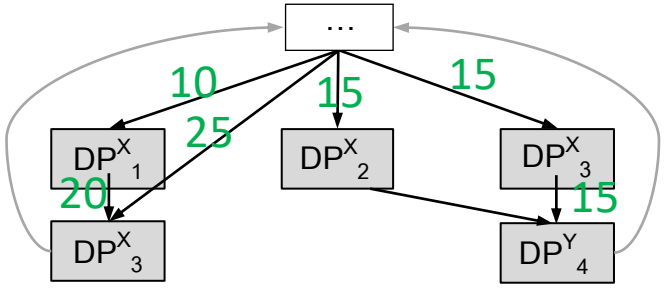
S3: Coordinated Multi-Route-Aware Delivery

Random schedule of 100B Input
Some routes receive 0B



Schedule

Schedule of 100B Input with FIDO
Each route receives a valid slice



DP^X_i: i-th delivery route of input source X.

Algorithm: don't starve or stuff any delivery point

- **L2–9** Reserve minimum bytes for selected routes
- **L16–20** Pad zeros to satisfy the lower bound
- **L21–26** Choose Slice within $[Low_p, UP_p]$
- **L23–24** FI-based seed for reproducibility
- **L27–32** Update Pos / Len_{FI} / Len_R
- **L35–36** Dynamic DP update

Algorithm 1 Coordinated Multi-Route-Aware Delivery

Input: Entry Point (EP), Fuzzing Input(FI), Delivery Point (DP)

1: $Pos = 0$, $PC = EP$
2: $N = \sum^{n} |CFG(DP_n)|$
3: $X = \min_{p \in DP} LOW_p$
4: $Len_R = N * X$
5: **while** $Len_{FI} < Len_R$ **do**
6: $N = N - 1$;
7: $Len_R = N * X$;
8: **end while**
9: $Len_{FI} -= Len_R$
10: **while** true **do**
11: $PC = Execute(PC)$;
12: **if** $PC == p \in DP$ **then**
13: **if** $Len_{FI} + Len_R == 0$ **then**
14: **return**;
15: **else**
16: $\Delta = Len_{FI} + X - LOW_p$;
17: **if** $\Delta \leq 0$ **then**
18: $Len_{SI} = Len_{FI} + X$;
19: $SI = FI_{[Pos, Pos+Len_{FI}+X]} \,\|\, \underbrace{00\ldots0}_{|\Delta|}$;
20: $Delivery(SI)$;
21: **else**
22: $t = \min(Len_{FI} + X, UP_p) - LOW_p$;
23: $Len_{SI} = Rand(FI_{[Pos]}) \mod t + LOW_p$;
24: $SI = FI_{[Pos, Pos+Len_{SI}]}$;
25: $Delivery(SI)$;
26: **end if**
27: $Len_{FI} -= Len_{SI} - X$;
28: $Len_R -= X$;
29: $Pos = Pos + Len_{SI}$;
30: **if** $Len_R == 0$ **then**
31: $X = 0$;
32: **end if**
33: **end if**
34: **end if**
35: **if** $PC == New\ R_P\ or\ R_B$ **then**
36: $DP += DPIdentify(PC)$
37: **end if**
38: **end while**

Evaluation

Research Questions

RQ1 CRP Route Identification
Can FIDO automatically identify input routes in both unit tests and real-world firmware?

RQ2 Fuzzing Improvement
How much does FIDO improve coverage and bug finding over SOTA ad-hoc delivery strategies?

RQ3 Interrupt-driven Comparison
How does FIDO compare with AidFuzzer in addressing P1–P4 delivery issues?

RQ4 Ablation Study
How much does each component, S1/S2/S3, contribute to fuzzing effectiveness?

Firmware Dataset

28 unit-test samples

- From large-scale empirical study
- GPIO, UART, I2C, ADC drivers
- STM32, NXP, Arduino HALs
- RTOS kernels: RIoT, Nuttx

25 real-world firmware samples

- 22 tested by SEmu, Fuzzware, AidFuzzer, and MULTIFUZZ
- +2 MbedOS BLE GATT server examples
- +1 Zephyr BLE-HCI example
- At least one data peripheral in IRQ mode

Delivery Information Identification (RQ1)

Unit Test

- Each of sample has a single input route
- Manual verification after one main loop completion

Real-world firmware:

- Manual verification after 24h fuzzing

Efficiency

- S1 (DP identification): 3–10s, avg. **8.84s**
- S2 (length inference): 2–7s, avg. **5.06s**
- < **20s** per sample
- **One-time effort** per input route

TABLE 10: Details and input routes of 25 real-world firmware samples (Underline input routes in polling mode, others in interrupt mode. Bold input routes that always occur in main loop; other routes occur only in specific conditions.)

Firmware	MCU	OS/Sys lib.	Total	CRP Route(Peripheral.:#:[Lower:Upper],...))
Console	NXP K64F	NXP HAL	2,251	**UART:1:[1:64]**
Steering_Control	SAM3X	Arduino	1,835	**UART:2:[1:128]**
Gateway	STM32F103	Arduino	4,921	**UART:1:[1:64]**,I2C:2:[1:128],GPIO:4,ADC:1
Heat_Press	SAM3X	Arduino	1,837	**UART:6:[8:64]**
PLC	STM32F429	Arduino	2,304	**UART:1:[8:64]**
Soldering_Iron	STM32F103	FreeRTOS	3,657	**I2C:1:[1:128]**,**GPIO:1**,**ADC:1**
GPS_Tracker	SAM3X	Arduino	4,194	**UART:1:[1:256]**,UART:4:[1:256]
LiteOS_IoT	STM32L431	LiteOS	2,423	**UART:1:[20:100]**,UART:5:[20:100]
3Dprinter	STM32F103	STM32 HAL	8,045	**UART:1:[1:64]**,**USB:1**,GPIO:2
SocketCan	STM32L432	Zephyr	5,943	**CAN:1:[1:64]**
μTasker_USB	STM32F429	μTasker	3,491	UART:1:[1:516],USB:1:[1:512],**GPIO:1**
Bootstrap(UART)	nRF52840	Zephyr	4,972	**UART:1:[1:5]**,GPIO:1
Bootstrap(SPI)	nRF52840	Zephyr	4,949	**SPI:1:[1:5]**,GPIO:1
Echo_Server	SAM4E	Zephyr	7,007	**SPI:1:[1:132]**
L2cap_Processor	TICC2538	Contiki-NG	4,002	**RADIO:1:[1:128]**
Snmp_Server	TICC2538	Contiki-NG	3,080	**RADIO:1:[1:128]**
CCN-Lite-Relay	nRF52832	Nordic HAL	12,675	**UART:1:[1:128]**, Radio:1:[1:168]
Gnrc_networking	STM32F303	STM32 HAL	6,448	**UART:2:[1:128]**
Filesystem	STM32F303	STM32 HAL	2,429	**UART:1:[1:128]**,UART:4:[1:128]
6Lowpan_Receiver	SAM R21	Contiki	6,988	UART:1, **Radio:1**, I2C:1
6Lowpan_Transmitter	SAM R21	Contiki	6,988	UART:1, **Radio:1**, I2C:1
P2IM_Drone	STM32F103	Bare-Metal	2,754	UART:1:[1:2], **I2C:4**
Client-Gattupdate	nRF52840	MBedOS	13,888	**SPI:2:[1:256]**,**GPIO:1**
Server-Gattupdate	nRF52840	MBedOS	13,826	**SPI:2:[1:256]**,**GPIO:1**
BLE-HCI	nRF52840	Zephyr	7,470	**UART:1:[1:7]**

Fuzzing Efficiency Improvement (RQ2)

Coverage Improvement

TABLE 3: Code coverage (median of 5 trials after 24-hours) using *FIDO* compared to original delivery methods. Shaded areas means indicating additional coverage from bug exploits. Growth Rates (GRs) that have significant changes are marked in bold (based on a Mann-Whitney U test with a 0.05 significance threshold).

Firmware	Feature	*Fuzzware*							*MultiFuzz*						
		RR	Fuzz	w.*FIDO*	Problem	GR(RR)	Problem	GR(Fuzz)	RR	Fuzz	w.*FIDO*	Problem	GR(RR)	Problem	GR(Fuzz)
3DPrinter	F1,F2,F3,F4	786	780	**931**	P2-P4	**+18.4%**	P2-P4	**+19.3%**	4,193	3,642	**4,411**	P2-P4	+5.2%	P2-P4	**+21.1%**
Bootstrap(SPI)	F1,F3,F4	956	950	**998**	P3	+4.4%	P3	+5.1%	982	1,184	**1,198**	P3	**+22.0%**	P3	+1.2%
Bootstrap(UART)	F1,F3,F4	994	951	**1,878**	P1,P3-P4	**+88.9%**	P1,P3-P4	**+97.5%**	1,289	1,986	**1,986**	P1,P3-P4	**+54.1%**	P1,P3-P4	+0.0%
CCN-Lite-Relay	F1,F3,F4	491	556	**1,054**	P3-P4	**+114.7%**	P3-P4	**+89.6%**	4,077	4,445	**4,472**	P3-P4	**+9.7%**	P3	+0.6%
μtasker_USB	F1,F3,F4	1,269	1,253	**1,518**	P2-P4	**+19.6%**	P2-P3	**+21.1%**	1,995	1,924	**2,129**	P2-P4	+6.7%	P2-P3	+10.7%
Console	F1,F2,F3	711	712	**794**	P2-P3	+11.7%	P2-P3	+11.5%	1,165	1,161	**1,171**	P2-P3	+0.5%	P2-P3	**+0.9%**
Echo_Server	F3	2,854	2,852	**2,905**	P3	+1.8%	P3	+1.9%	3,553	3,567	**3,569**	P3	+0.5%	P3	+0.1%
Gateway	F1,F3,F4	2,362	2,712	**2,756**	P3-P4	**+16.7%**	P1-P4	+1.6%	2,968	2,882	**3,188**	P3-P4	**+7.4%**	P1-P4	**+10.6%**
Gnrc_networking	F1,F3,F4	421	416	**668**	P3-P4	**+58.7%**	P3-P4	**+60.6%**	1,849	1,779	**2,136**	P3-P4	**+15.5%**	P3-P4	**+20.1%**
GPSTracker	F1,F2,F3,F4	661	977	**1,011**	P2-P4	**+53.0%**	P2-P4	+3.5%	1,227	1440	**1,589**	P2-P4	**+29.5%**	P2-P4	**+10.3%**
Heat_Press	F1,F2,F3,F4	551	555	**570**	P2-P4	**+3.4%**	P2-P4	**+2.7%**	573	580	**601**	P2-P4	+4.9%	P2-P4	+3.6%
L2cap_Processor	F3	**1,001**	1,001	**1,001**	P3	+0.0%	P3	+0.0%	1,002	1,021	**1,170**	P3	**+16.8%**	P3	**+14.6%**
LiteOS_IoT	F1,F3,F4	738	746	**1,333**	P3-P4	**+80.6%**	P1-P4	**+78.6%**	1,375	1,380	**1,377**	P3-P4	**+0.1%**	P1-P4	-0.2%
PLC	F1,F2,F3	638	640	642	P2-P3	+0.3%	P2-P3	+0.3%	640	640	**1,838**	P2-P3	**+187.2%**	P2-P3	**+187.2%**
Filesystem	F1,F3,F4	-	-	-	-	-	-	-	1,374	1,352	**1,414**	P3	+2.9%	P3	+4.6%
Snmp_Server	F3	1,032	1,032	**1,045**	P3	+1.3%	P3	+1.3%	1,066	1,083	**1,297**	P3	**+21.7%**	P3	**+19.8%**
Soldering_Iron	F1,F3,F4	2,177	**2,280**	2,267	P3	+4.1%	P3	-0.6%	2,675	2,799	**3,271**	P3	**+22.3%**	P3	**+16.9%**
Steering_Control	F1,F3,F4	587	594	**606**	P3	+3.1%	P3	+1.9%	652	655	**660**	P3	+1.2%	P3	+0.8%
Zephyr_SocketCan	F1,F3	2,583	**2,662**	2,660	P3	+3.0%	P3	-0.1%	3,334	2,880	**3,341**	P3	+0.2%	P3	**+16.0%**
Client-Gattupdate	F1,F3,F4	-	-	-	-	-	-	-	4,333	3,051	**7,454**	P3-P4	**+72.0%**	P3-P4	**+144.3%**
Server-Gattupdate	F1,F3,F4	-	-	-	-	-	-	-	10,117	10,846	**11,018**	P3-P4	+8.9%	P3	+1.6%
BLE-HCI	F1,F3,F4	2,523	2,576	**2,784**	P3-P4	+10.3%	P3-P4	**+8.1%**	-	-	-	-	-	-	-

-: The original firmware fuzzer fails to reach the stage where firmware receives external input (*e.g.*, getting stuck in initialization stage after 24 hours).

Improve SoAT solutions: Fuzzware **(+115%),** MULTIFUZZ **(+54%),** SEmu **(+19%)**

Fuzzing Efficiency Improvement

Bug Finding Enhancement

- Table 4: 48h crash detection
- Count = unique true crashes
- Time = minimum discovery time
- Fac. = FIDO speedup
- Green = newly discovered bugs

FIDO **improves bug finding** across fuzzers

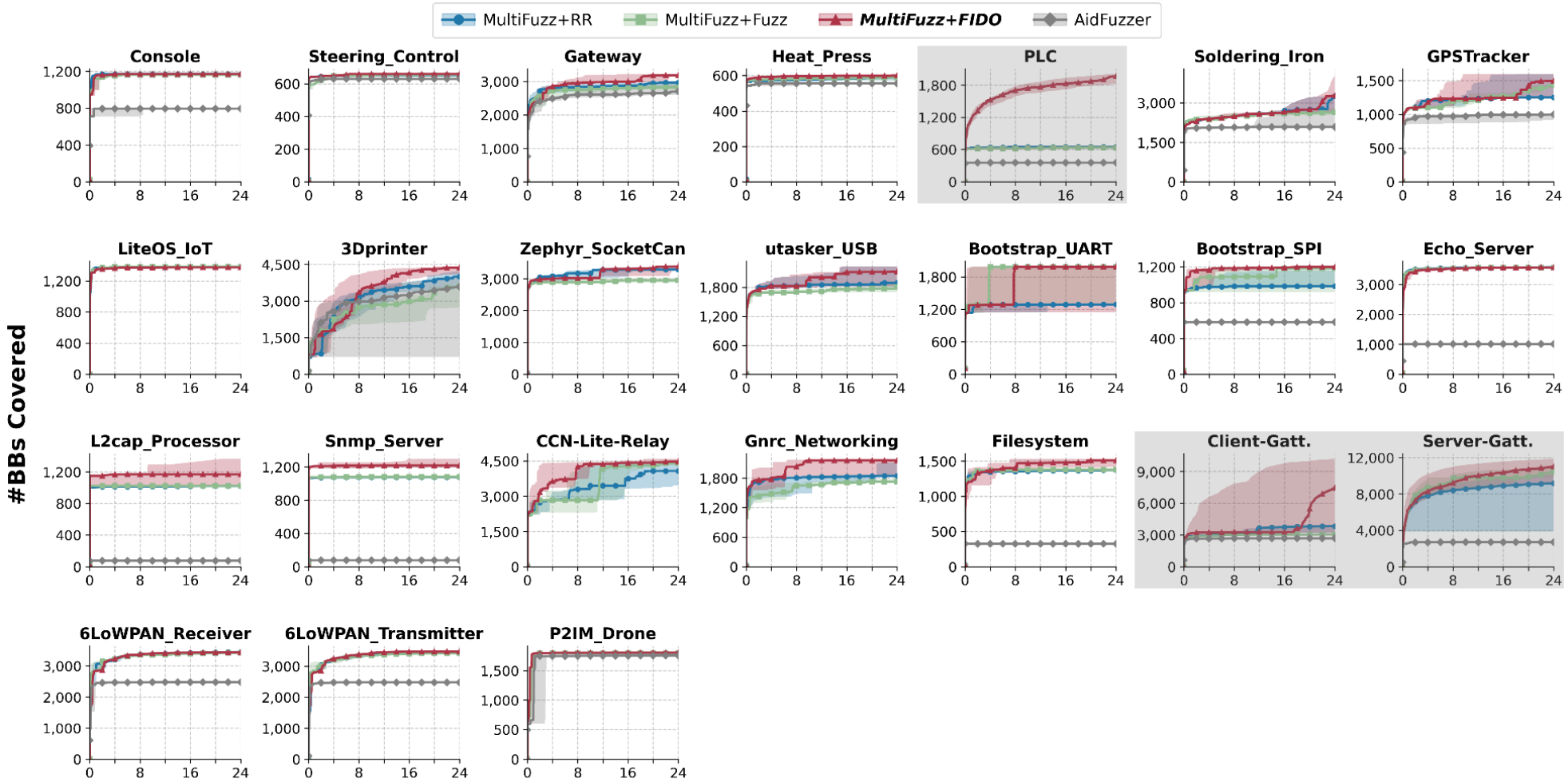
- SEmu **2.6×** more crashes, **67×** faster
- Fuzzware **6×** more crashes, **7×** faster
- MultiFuzz **2×** more crashes, **6×** faster

New bugs

- **10+ bugs than SoTA and 5** previously unknown bugs
  - 2 Zephyr 4.1.0 BLE Subsystem
  - 1 Mbed OS LE Cordio implementation
  - 2 Gateway I2C handler

Firmware	Bug[Report] Type-Func./CVE-20..	# Crash Count Original	# Crash Count w.*FIDO*	Minium Discovery Time Original	Minium Discovery Time w.*FIDO*	Fac.
	***SEmu* (Original = MSP)**					
Heat_Press	OOB-FC3 [43]	Fail	**175**	-	00:09:31	**>302**
PLC	OOB-FC1 [44]	24	**112**	03:23:18	00:44:24	**4.58**
	OOB-FC3 [47]	Fail	**17**	-	01:56:40	**>24**
	OOB-FC15 [45]	135	**175**	00:14:23	00:08:52	**1.62**
	OOB-FC16 [46]	100	**185**	01:53:08	00:33:48	**3.35**
	***Fuzzware* (Original = The better results of RR and Fuzz)**					
Heat_Press	OOB-FC3 [43]	1236	**1765**	00:56:12	00:19:52	**2.83**
PLC	OOB-FC1 [44]	258	**399**	00:42:35	00:46:27	0.92
	OOB-FC3 [47]	68	**129**	05:07:08	04:48:25	1.06
	OOB-FC15 [45]	244	**385**	00:10:23	00:09:56	1.05
	OOB-FC16 [46]	88	**1329**	01:28:57	00:21:26	**4.15**
Echo_Server	21-3319 [25]	1590	**1620**	00:27:34	00:25:13	1.09
	21-3320 [26]	1	**18**	02:33:05	03:42:53	0.69
	20-10064 [22]	2	**8**	10:34:24	11:30:48	0.92
Bootstrap(UART)	21-3329 [27]	Fail	**1**	-	35:26:12	>1.41
Bootstrap(SPI)	20-10065 [23]	66	**774**	07:31:43	00:07:02	**64.23**
	20-10066 [24]	15	**225**	05:18:11	03:45:45	**1.41**
BLE-HCI	NPD-ull_conn.	Fail	**1**	-	08:16:22	**>6.04**
	***MultiFuzz*(Original = The better results of RR and Fuzz)**					
Heat_Press	OOB-FC3 [43]	127	**302**	00:04:02	00:01:12	**3.36**
PLC	OOB-FC1 [44]	91	**374**	00:00:41	00:00:20	**2.05**
	OOB-FC3 [47]	7	**15**	04:44:55	04:31:55	1.05
	OOB-FC15 [45]	6	**9**	00:26:22	00:02:34	**10.27**
	OOB-FC16 [46]	6	**20**	00:28:17	00:26:24	1.07
GPSTracker	NPD-strtok_xx [10]	5	**5**	02:01:33	01:28:02	1.38
	NPD-strstr_xx [11]	Fail	**3**	-	05:01:49	**>9.6**
	NPD-strstr_xx [12]	2	**3**	09:51:50	05:45:03	1.72
Gateway	ING-Sysex. [6]	6	**8**	00:03:55	00:05:46	0.68
	OOB-setPin. [40]	**30**	20	00:02:39	00:01:51	1.43
	UPD-TxCplt [41]	**10**	8	00:00:01	00:00:01	1
	NPD-pwm_start [42]	97	**764**	00:17:20	00:13:07	1.32
	OOB-decode.	Fail	**1**	-	08:00:43	**>6**
	NPD-processSysex.	Fail	**5**	-	06:30:13	**>7.38**
Echo_Server	20-10064 [22]	90	**95**	03:23:34	01:38:02	**2.08**
Bootstrap(UART)	NPD-net_buf_simple.	Fail	**1**	-	05:50:42	**>8.23**
Bootstrap(SPI)	20-10065 [23]	10	**15**	00:11:28	00:02:35	**4.44**
L2cap_Processor	20-12140 [38]	Fail	**5**	-	00:53:16	**>57**
Snmp_Server	20-12141 [39]	**3**	2	08:57:22	06:21:03	1.41
CCN-Lite-Relay	RC-ble_isr [13]	20	**25**	00:01:12	00:01:46	0.68
	NPD-ccnl [8]	Fail	**2**	-	18:35:26	**>2.69**
	NPD-evtimer [9]	4	**5**	09:50:23	06:11:44	1.59
	UAF-ccnl_xx [15]	20	**28**	03:49:21	01:48:32	**2.11**
SocketCan	NPD-net_pkt [14]	6	**8**	16:10:02	12:37:22	1.28
	NPD-pwm_shell	3	**9**	17:42:07	05:08:16	**3.45**
Gattupdate*	24-22095	40	**43**	07:27:16	00:15:32	**28.79**

Comparison to Interrupt-driven Firmware Fuzzers (RQ3)



Experiment

- 27 samples tested with AidFuzzer
- Compare against MultiFuzz + FIDO
- 24h fuzzing campaign

Result **Duration(h)**

- AidFuzzer produced outputs on 16 / 27 samples
- **Failures:** unsupported memory-map alignment or initial seed crashes
- On all 16 successful samples, MultiFuzz + FIDO **achieves higher coverage**
- Improvement: **5%** to >**1500%**

Ablation Study (RQ4)

TABLE 6: Ablation study for the effect of S1-S3, using *Fuzzware* in RR mode as the baseline. ↑ indicates changes in median coverage and average crash count compared to the previous configuration. Changes below 0.1% are not displayed, and significant changes are marked in bold (based on a Mann-Whitney U test with a 0.05 significance threshold).

Firmware	RR		S1					+S2					+S3				
	Cov. Med.	Crash Count	Coverage Med.	↑	*p-value*	Crash Count	↑	Coverage Med.	↑	*p-value*	Crash Count	↑	Coverage Med.	↑	*p-value*	Crash Count	↑
Gateway	2362	0	2,512	+6.4%	0.056	0		2,558	+1.8%	0.548	0		2,756	+7.7%	0.095	0	
Heat_Press	551	247.2	563	+2.2%	0.010	235.6	-4.7%	565	+0.4%	0.666	259	+9.9%	570	+0.9%	0.916	353	+36.3%
CCN-Lite-Relay	491	0	1,054	+114.7%	0.011	0		1,054		0.796	0		1,054		1.000	0	
Gnrc_Networking	421	0	666	+58.2%	0.014	0		666		0.821	0		668	+0.3%	0.564	0	
3Dprinter	786	0	902	+14.8%	0.056	0		901		0.916	0		931	+3.3%	0.140	0	
GPSTracker	661	0	948	+43.4%	0.032	0		952	+0.4%	1.000	0		1,011	+6.2%	0.012	0	
LiteOS_IoT	738	0	739	+0.1%	0.083	0		1,079	+46.0%	0.408	0		1,333	+23.5%	0.292	0	

Reading Guide

- RR: Fuzzware round-robin baseline
- +S1: add CRP route mapping
- +S2: add length range inference
- +S3: add coordinated delivery

Key Findings

- S1 gives the largest coverage jump: CCN-Lite-Relay **+114.7%,** Gnrc_Networking **+58.2%**
- S2 helps length-sensitive firmware: LiteOS_IoT **+46.0%**
- S3 improves multi-route delivery: Heat_Press crash count **+36.3%,** Gateway coverage **+7.7%**

Conclusions

New Problem

Identify **input delivery (timing & quantity)** as a **critical** but often **overlooked** factor in existing firmware fuzzers.

Key Insight

CRP (Check–Retrieve–Process) model reveals the input demand, enabling optimal delivery timing and quantity.

Implementation

Implemented as a plugin for Fuzzware, MULTIFUZZ and Semu.

Results

FIDO consistently enhances code coverage for various fuzzers, detects bugs more rapidly, and identifies five previously unknown bugs.

Thank you!

Questions?

weizhou_sec@hust.edu.cn

Open source at https://github.com/IoTS-P/FIDO

華中科技大學 HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

PennState

THE UNIVERSITY OF TEXAS AT DALLAS

UNIVERSITY OF GEORGIA