10/15/2024, CCS'24, Network Security
@Salt Lake City, USA

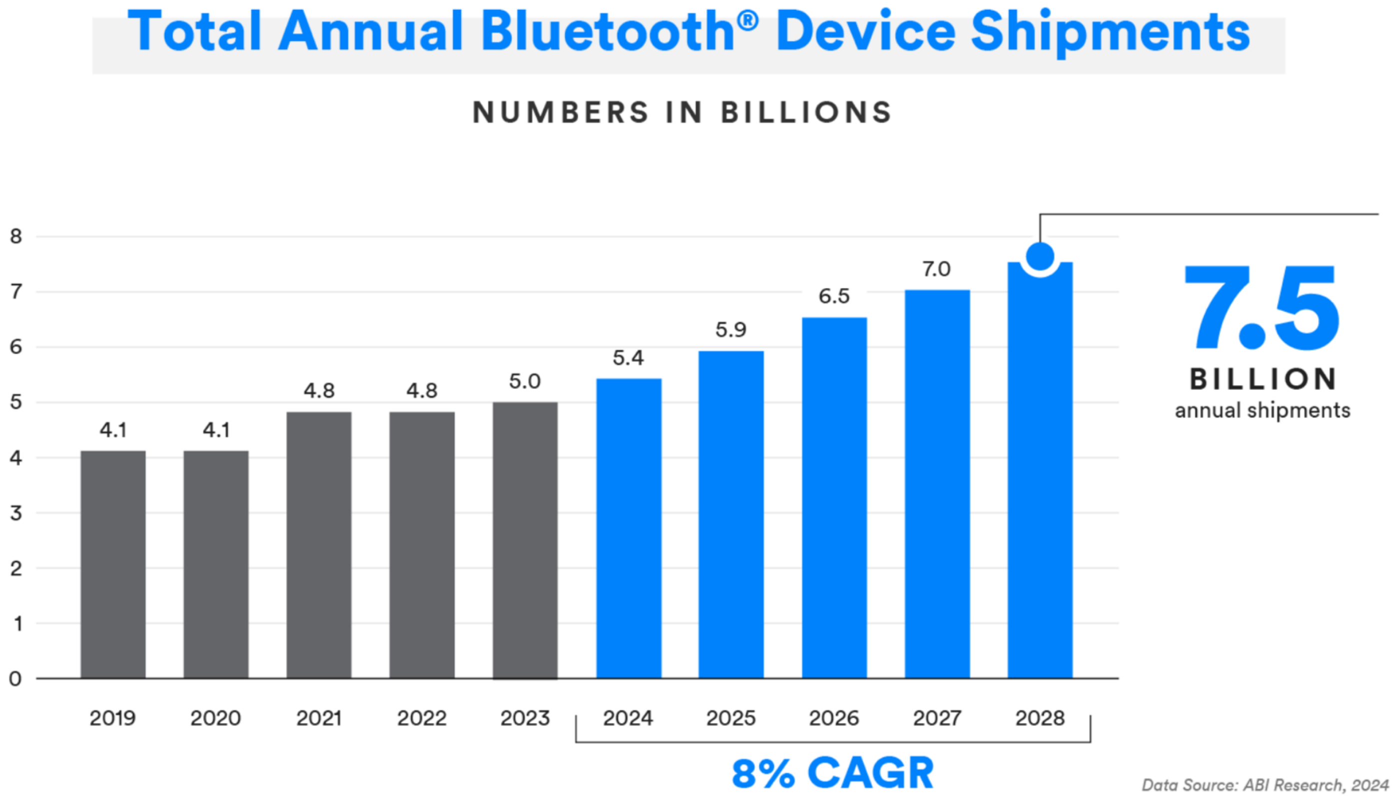
BlueSWAT: A Lightweight State-Aware Security Framework for Bluetooth Low Energy

Xijia Che, Yi He, Xuewei Feng, Kun Sun, Ke Xu, Qi Li

Tsinghua University, Beijing, China

Background

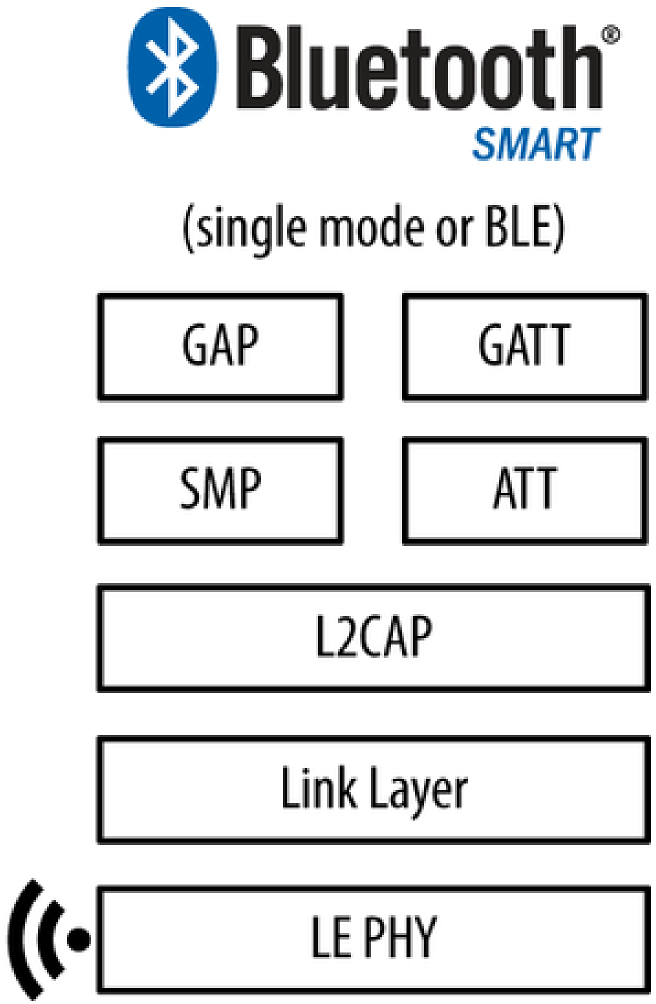
More than 5 million Bluetooth Low Energy (BLE) smart devices are estimated to be in use by 2023, and total annual shipments of Bluetooth devices will reach 7.5 billion by 2028.



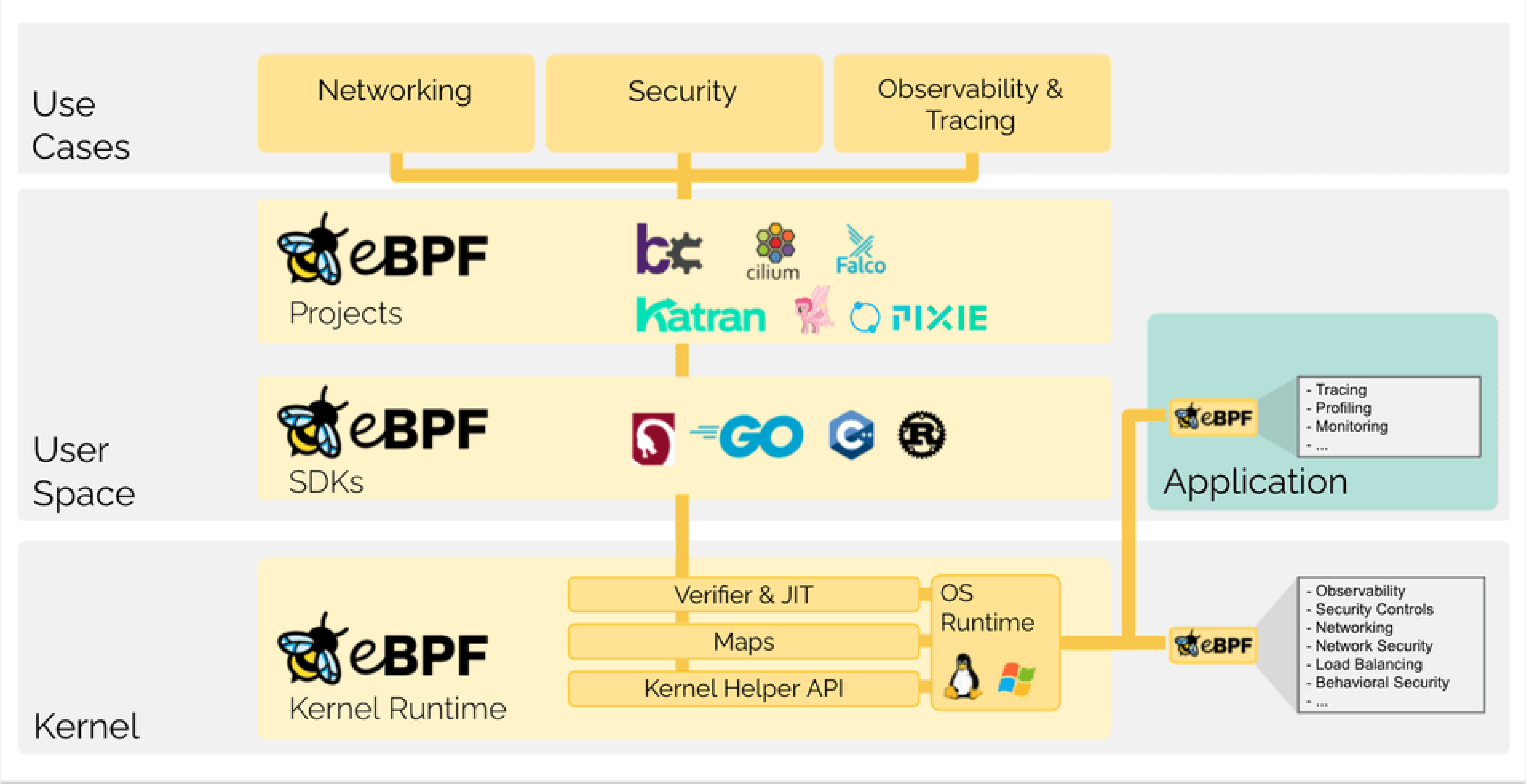
Bluetooth Low Energy

BLE is designed for low-cost communication on resource-constrained devices.

- Link Layer (LL)
  - ◆ fundamental procedure of BLE session
- Security Management Protocol (SMP)
  - ◆ BLE security mechanism, e.g. pairing, encryption
  - ◆ TARGETED BY MOST PROTOCOL STUDIES

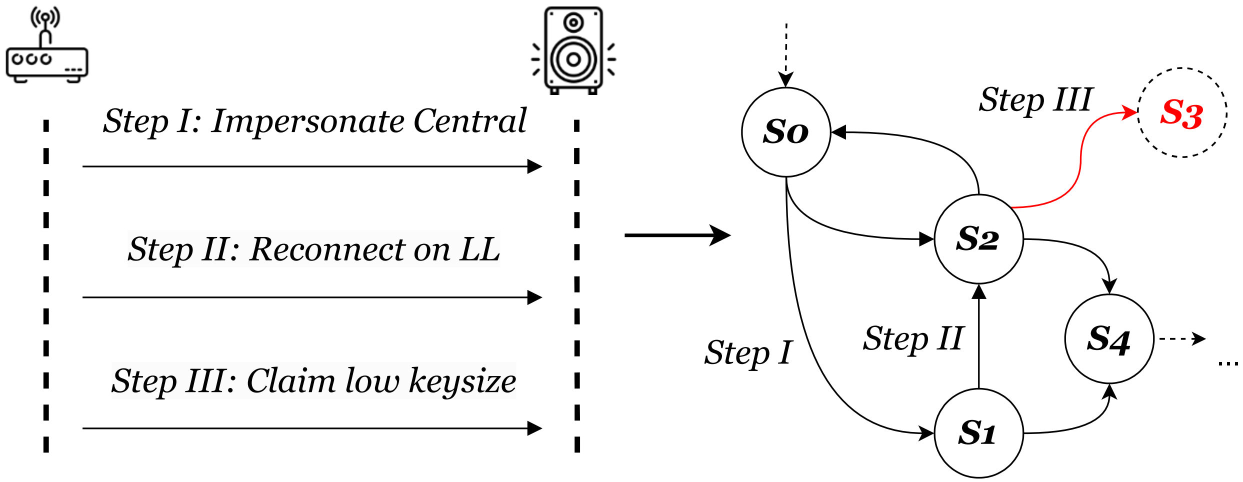


Extended Berkeley Packet Filter (eBPF)



A Motivating Example – BLE KNOB Attack

Attacks: Packet-based and Session-based



The pattern of the KNOB session is modeled as a malicious transition path in FSM.

Limitations:

1. Only Inspecting Individual Packets (LBM 2019)

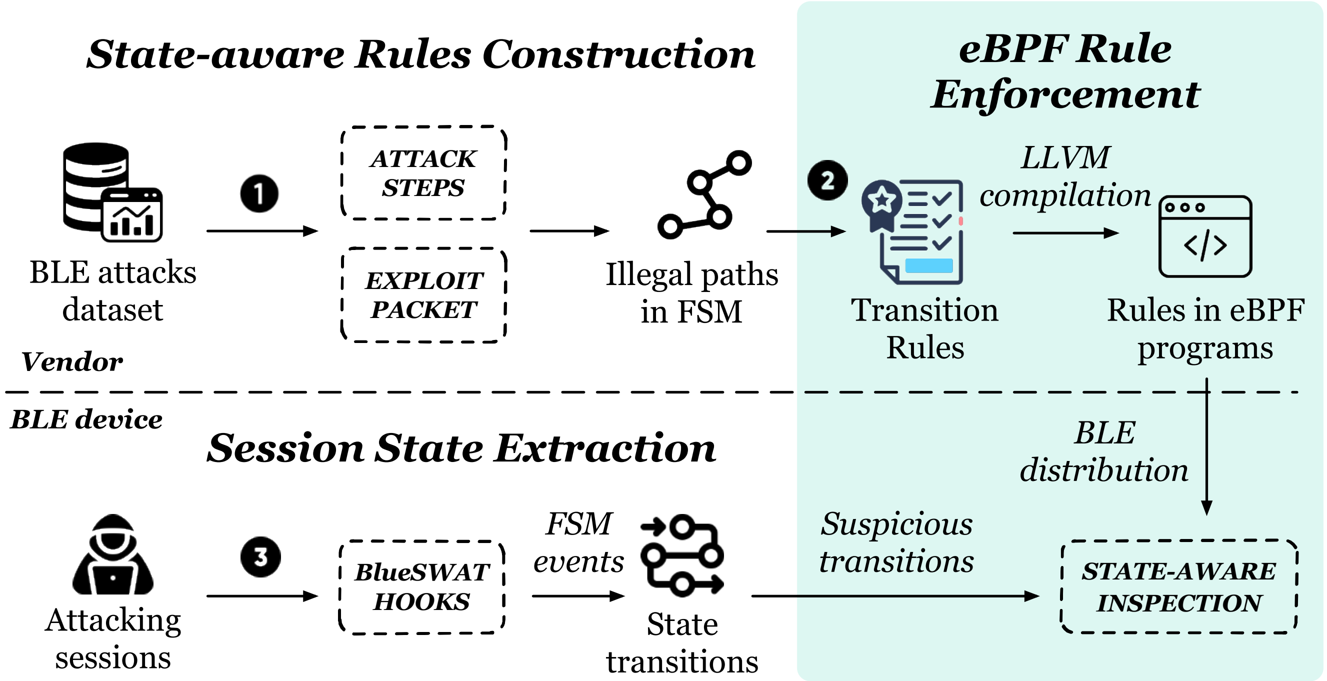
   Vulnerable to BLE session-based attacks.

2. Long Patching Window
   - Bluetooth SIG updates the Specification
   - Manufacturers develop a patch
   - Vendors test, recompile and update the corresponding user products.

Design Overview

BlueSWAT monitors patterns of session-based attacks with Finite State Machine (FSM).

1. Vendors abstract attack patterns and model them as illegal transition paths in FSM.
2. Vendors compile transition rules into eBPF programs and distribute them to BLE devices.
3. BlueSWAT captures session events and inspects FSM transitions at runtime.



Session State Extraction

Hooks are common to different vendor stacks and require minimal engineering efforts.

BlueSWAT hooks at **LL** and **SMP**

- LL is where plaintext data can be accessed in the stack for the first time.
- SMP serves as the core architecture of BLE security mechanisms, such as device pairing and encryption.

```
// nimble/controller/src/ble_ll_conn.c:
void ble_ll_conn_rx_data_pdu(struct os_mbuf *rxpdu,
struct ble_mbuf_hdr *hdr){
  ...
  if (IFW_DC_LL_CTRL_PARSER(connsm, rxpdu)){
    goto conn_rx_data_pdu_end;
 }
  ...
}
```

(a) LL RX parser for control PDUs.

```
// nimble/host/src/ble_sm.c:
int ble_sm_rx(struct ble_l2cap_chan *chan){
  ...
    if (IFW_SMP_PARSER(chan)){
        return BLE_HS_EUNKNOWN;
    }
  ...
}
```

eBPF-based Rule Enforcement

We develop a lightweight eBPF framework for IoT platforms:

1. **Usability** - Patch update does not require firmware recompilation and device reboot.

   eBPF programs can be transmitted via BLE and dynamically loaded by BlueSWAT.

2. **Compatibility** - BlueSWAT is compatible in the fragmented IoT environment.

   eBPF bytecode can be executed across different chips regardless of architectures.

2. **Practicality** - eBPF programs introduce minimal runtime overhead and memory consumption.

Security Analysis

We systematically collect 101 real-world BLE vulnerabilities by November 2023.

- Around 54% of them are session-based, which is left unstudied by previous research.
- BlueSWAT can successfully mitigate 87.1% of them, including 76.1% of session-based and 96.4% of packet-based attacks.

Table 5: Comparison of BlueSWAT and LBM for mitigating real-world BLE vulnerabilities in our dataset.

Category	Impact	LBM		BlueSWAT	
		S	P	S	P
Design Flaw	Pairing Compromise	0 / 5	0 / 0	5 / 5	0 / 0
	Illegal Service Access	0 / 10	0 / 0	7 / 10	0 / 0
Function Error	Authentication Bypass	0 / 10	1 / 2	7 / 10	2 / 2
	Key Compromise	0 / 4	0 / 1	4 / 4	1 / 1
	Encryption Failure	0 / 3	0 / 1	2 / 3	1 / 1
	Denial of Service	0 / 6	0 / 0	6 / 6	0 / 0
Runtime Error	Bounds Check Missing	0 / 1	15 / 24	1 / 1	24 / 24
	Buffer Overflow	0 / 1	9 / 18	1 / 1	18 / 18
	Logic Error	0 / 6	5 / 9	2 / 6	7 / 9
Overall	-	0 / 46	31 / 55	35 / 46	53 / 55
Proportion	-	0	56.4%	76.1%	96.4%

S: Session-based vulnerabilities. P: Packet-based vulnerabilities.

Evaluation

- We implement BlueSWAT on 5 real-world devices with mainstream BLE stacks and architectures.
- BlueSWAT encompasses around 2k lines of C code and 1k lines of Python code.

Table 1: Real-world devices used in evaluation. Stacks with * are partly closed-source.

Device	Manufacturer	Processor	Architecture	BLE Stack
nRF51833 DK	Nordic.	Cortex-M0	ARMv7-M	NimBLE
CC2640R2	TI.	Cortex-M3	ARMv7-M	SimpleLink*
nRF52840 DK	Nordic.	Cortex-M4	ARMv7-M	Zephyr
ESP32	Espressif.	Xtensa LX6	Xtensa	ESP-IDF*
Sipeed M0P	Bouffalo	BL618	RSIC-V	Bouffalo*

Table 2: Hooks and lines of code inserted into the stacks.

BLE Stack	# LL Hooks	# SMP Hooks	LoC Inserted
NimBLE	3	2	8
SimpleLink	3	2	8
Zephyr	3	3	9
ESP-IDF	3	2	8
Bouffalo	2	2	6

Memory Consumption

We conduct performance evaluations with the Zephyr stack on a Nordic nRF52840 DK, which has a Cortex-M4 SoC running at 64 MHz, 1 MB Flash, and 256 KB SRAM.

- *On average, one ePBF program takes up 137.2 B Flash memory (less than 0.08% overhead) and 217.8 B dynamic memory, which can be considered controllable.*

Runtime Latency

1. Micro benchmark

- We load 10 rules and generate 1k packets on the Bluetooth RX path.

The average latency in interpretation mode is 1.266 us (81 MCU cycles) while with JIT it drops to 1.094 us (70 MCU cycles).

2. Macro benchmark

- We test two real-world BLE applications

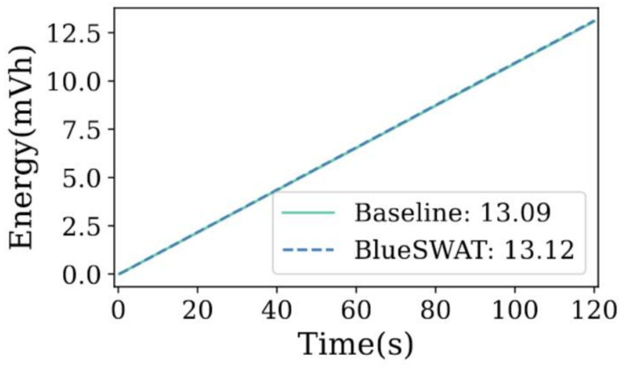
Battery Level Service: average baseline RTT 3489.7 us

Heart Rate Service: average baseline RTT 3487.8 us

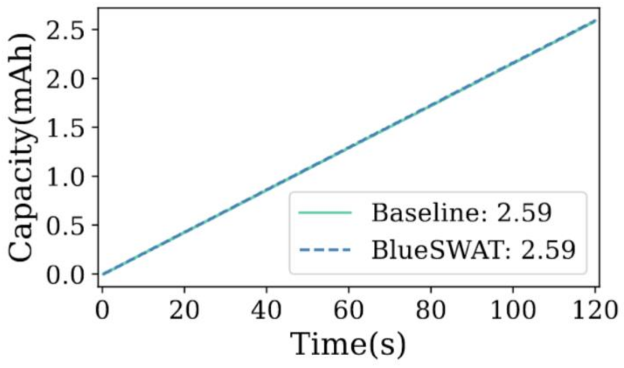
Power Performance

We access the power and energy performance over a 120-second window, encompassing four phases: 20s of connection, 40s of BAS, another 20s of connection, and 40s of HRS.

- *BlueSWAT introduces an average of 0.0009 W more power than the baseline, representing a 2.29% increase.*



(b) Energy usage.



(c) Capacity usage.

Thank you!

Xijia Che, Tsinghua University

cxj22@mails.tsinghua.edu.cn

Code: https://github.com/RayCxggg/BlueSWAT