基于 WebAssembly 的物联网便携且安全的传感器接口

Botong Ou
bou@purdue.edu 普渡大学，西拉斐特

Baijian Yang
byang@purdue.edu 普渡大学，西拉斐特

arXiv:2601.14555v1 [cs.CR] 21 Jan 2026

摘要

随着物联网连接的不断扩展，全球各地的生活质量得以改善，但与此同时，隐私和安全方面的隐忧也日益加剧。由于缺乏明确的规范来管理对物联网传感器的共享访问和受保护访问，设备可能因此遭到入侵，敏感数据也可能泄露。近年来，WebAssembly 因其高效的应用程序沙箱隔离能力而备受关注，尤其适用于嵌入式系统，因此成为探索安全、便携式传感器接口的理想选择。

本文介绍首个 WebAssembly系统接口（WASI）扩展，可提供安全、便携且占用空间小的沙箱，使不同类型的嵌入式设备能够多租户访问传感器数据。运行时扩展可实现应用程序内存隔离，通过拦截传感器访问确保资源权限适当，并提供 MQTT-SN 接口以支持网络内访问控制。针对 WebAssembly 字节码，并在 Zephyr 实时操作系统之上实现相应的运行时扩展后，我们对传感器访问进行了评估，结果显示，与原生执行相比，延迟开销为 6%，内存占用额外增加 5%。由于 MQTT-SN 请求主要受网络延迟影响，WASI-SN 对 MQTT-SN 的实现仅增加不到 1% 的延迟，内存占用也相近。

关键词

嵌入式设备、设备到设备通信、实时系统、安全与隐私、WebAssembly

ACM 引用格式：

Botong Ou 和 Baijian Yang。2026年。面向物联网的基于 WebAssembly 的便携且安全的传感器接口。载于*ACM* 会议论文集（*Conference'17*）。ACM，纽约，纽约州，美国，12页。https://doi.org/10.1145/ nnnnnnn.nnnnnnn

1 引言

物联网（IoT）部署在从家庭和工业自动化到农业和医疗保健等各个域中迅速增长 [37, 50]。每天都有数千种物联网设备（例如传感器和执行器）投入生产 [28, 48], , 而在其使用过程中，它们往往需要参与人类活动，或收集并报告环境数据 [20]。这些功能要求进行联网部署，以便用户与物联网设备之间能够进行交互式协作访问。

在通常由资源受限设备构成的物联网系统中，确保安全性达到适当水平是一项颇具挑战的任务，因为引入安全策略会增加内存占用、二进制文件大小和运行时延迟。因此，运行在嵌入式设备上的大多数实时操作系统都是轻量级操作系统，会舍弃某些安全防护（例如 AppArmor [16]和 Seccomp [12]），从而使设备容易受到攻击者侵害。这一负担落到了应用程序开发者身上，他们必须深入了解嵌入式系统，才能确保应用程序逻辑符合必要的安全要求 [11]。Apriorit [5]指出，针对嵌入式系统的大多数攻击可分为基于软件的攻击 [32], 、基于网络的攻击 [4, 47] 和侧信道攻击 [15, 34]。我们重点关注基于软件和网络的攻击，因为这两类攻击被认为在嵌入式设备上更为常见，也更容易实施。

为缓解基于软件的攻击，我们主张采用 WebAssembly（Wasm）安全二进制格式 [17]。此类攻击通常涉及利用任意内存访问，因此，限制暴露攻击面的有效策略是采用软件故障隔离（SFI）。目前已有一些工作致力于设计安全执行环境，以隔离物联网设备上的应用程序故障。例如，uSFI [6] 利用基础的内存保护来隔离耦合的应用程序模块。S$\mu\mu$Vw [3] 引入了基于软件的内存虚拟映射，并利用远程证明检测异常行为。此外，还有一种基于 JavaScript 的沙箱方法 [40] ，可保护物联网应用程序并防止故障损害底层系统。WebAssembly 通过建立沙箱环境，将应用程序与底层嵌入式系统隔离开来，提供了这样一种方案。WebAssembly 运行时会为每个独立模块显式分配内存，并拦截访问以确保执行边界检查和类型检查。得益于 WebAssembly 字节码固有的可移植性，嵌入式系统开发者不再需要成为安全专家。

本文介绍了 WASI-SN，这是首个为 Wasm 模块提供安全访问板载传感器功能的 WebAssembly系统接口（WASI）扩展。运行时扩展提供了一个受 SmartThings [41] 启发的传感器功能 API，用于查询、访问和配置连接的传感器。每个请求均由共享运行时管理，从而实现多租户安全访问。WebAssembly运行时会执行函数级签名验证，以确保在原子传感器接口上运行的每个应用程序都具备必要的权限。最后，WebAssembly 缓解了

特此免费授予许可，允许为个人或课堂使用而制作本作品全部或部分内容的数字或纸质副本，但不得以营利或商业利益为目的制作或分发副本，且副本必须附有本声明及首页上的完整引文。对于本作品中由 ACM 以外的其他方拥有版权的组成部分，必须尊重其版权。注明出处后可以摘录。若要以其他方式复制、转载、发布到服务器或分发给列表，须事先获得明确许可和/或支付费用。请通过 permissions@acm.org 申请许可。*Conference'17*，*2017*年7月，美国华盛顿特区© 2026年计算机协会。ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00 https://doi.org/10.1145/nnnnnnn.nnnnnnn

异构设备之间可移植性方面的局限，因为开发者不再需要根据相关部署目标设计平台专用应用程序实现。

基于网络的攻击通常涉及网络协议栈缺乏适当的访问控制；因此，我们主张在应用层网络协议中预先纳入访问控制。对于覆盖资源受限嵌入式设备的物联网网络，MQTT-SN 是一种自然的选择，因为它具备以下特点：(1) 去中心化架构；(2) 依靠 UDP 最大限度地降低能耗；(3) 支持多对多消息传递；(4) 将发布者与订阅者解耦；(5) 仅有 2字节头部开销。因此，我们的 WebAssembly 运行时实现提供了 MQTT-SN API 接口，并配备必要的访问控制原语，以防范相当大一部分常见的基于网络的攻击。为此，我们提出一种遵循通配符身份加密（WKD-IBE）[1]的广播加密方案，可实现数据驱动的端到端轻量级加密，同时支持灵活的密钥委托与撤销。身份管理被定义为一种层级结构，以简化密钥管理过程中的推理。

我们提供了一个与 Zephyr 实时操作系统兼容的 WASI-SN 开源实现，以及相关的 MQTT-SN扩展。我们在一个测试平台上进行了性能评估，该平台由搭载 Cortex-M4F@64MHz 微控制器的 NRF52840-DK开发板组成，采用 Armv7E-M架构，以 IEEE 802.15.4 作为链路层和物理层规范，并运行 OpenThread Thread网络。与原生代码执行相比，传感器访问API带来了 6% 的开销；其中大部分延迟源于运行时启动所需的固定启动时间，启动后即可重复执行模块。编译过程中还增加了 5% 的内存占用。在 Wasm 模块中使用 WASI-SN库仅带来了极少的内存占用，并且运行时延迟开销不到 1%；这是因为请求延迟主要由网络延迟造成。

我们的贡献可概括如下：

- 我们为常用嵌入式设备提供了一种可移植、通用的传感器接口。
- 我们设计了传感器能力的形式化定义，以将传感器接口限制为必要且有效的访问。
- 我们提倡将 WebAssembly 用作构建嵌入式应用的新目标平台，并提供首个支持 Cortex-M处理器、带有传感器接口扩展的 WebAssembly运行时开源实现。
- 我们在 WebAssembly运行时中内置了 MQTT-SN 库，向 WebAssembly 模块公开相应的接口。
- 我们借助通配身份基加密（WKD-IBE）和即时密钥撤销，为不同租户提出了一种基于 MQTT-SN 的访问控制方案。

2 背景与动机

WebAssembly（Wasm）是一种新型可移植二进制格式，旨在作为低级编译目标使用。它最初的设计目的是取代浏览器中的 JavaScript，从而提供轻量级沙箱环境来封装代码，并防止攻击者危害宿主，同时保持接近原生代码的性能。鉴于它在浏览器中的用途，WebAssembly 的设计目标是实现快速编译和实例化。Wasm 让浏览器能够使用流行的 C/C++ 库（如 libopus、libogg、libwebm），同时保持良好的性能、符合标准并支持跨浏览器使用 [24]。更广泛地说，只要具备可用的运行时，并将高级编程语言（C/C++、Rust 等）编译而来，Wasm 就能提供可移植性和灵活性。

栈机器与静态类型。 Wasm [17] 是一种定义完善的栈式虚拟机，采用静态原始类型（i32、i63、f32 和 f64）和预分配的线性内存，这与 Javascript 的动态类型不同。请看下面这段 Wasm 代码，它公开了一个用于将两个值相乘的函数：

列表 1：WebAssembly 模块的文本格式示例

```
(模块(函数$multiply(参数$lhsi32)(
参数$rhsi32)(结果i32)获取_局部$lhs
获取_局部$rhsi32.乘)(导出"multiply"(
函数$multiply)))
```

所有访问都具有明确的类型，并且只能从沙箱化的线性内存区域中读取。操作基于最近压入栈中的值执行。如果类型错误或内存访问无效，执行将被停止，以减轻对宿主环境的损害。

控制流完整性。 Wasm 使用函数索引空间来强制执行程序的控制流完整性（CFI）。每个 Wasm 程序由若干结构化的 Wasm 模块组成，这些模块作为部署、加载和编译的部署单元。每次函数调用都必须指定一个经过验证的入口点，以确保函数不会指向其表外部，并且每个调用签名都会进行类型检查。运行时通过维护一个 Wasm 模块无法访问的独立调用栈来提供返回地址完整性。

有界内存访问。 每次加载和存储操作都会由运行时执行边界检查。在读取或写入预分配的线性内存之前，运行时会确保目标地址在操作前后都处于边界范围内。结合控制流完整性，Wasm 因而能够抵御典型的栈溢出和面向返回编程（ROP）攻击，无需采用包括数据执行保护（DEP）[13] 和栈破坏保护（SSP）[29]在内的常见缓解策略。

与原生代码相比，开销极小。 与原生代码相比，Wasm 的性能开销仍然很小。例如，一个“Hello World”程序在 Wasm 中为 214 字节，而在 C 中为 74 字节。尽管 Wasm 需要进行内存边界检查，基准工作负载评估表明，Wasm 程序的运行速度通常不低于原生代码的 1/1.5 [51]。

浏览器之外的 Wasm 尽管 Wasm 主要用于在浏览器中配合 JavaScript，以提升性能和安全性，同时保持可移植性，但人们

表 1：从多个方面比较应用层协议

协议	UDP	去中心化	解耦	通信
COAP	√	×	×	1:1
XMPP	×	√	×	M:N
DDS	√	√	×	1:N
MQTT	×	√	√	M:N
MQTT-SN	√	√	√	M:N

最近开始关注它在浏览器之外的应用 [18, 25, 55]。例如，Fastly [19]和 Cloudflare [49]积极使用 WebAssembly 作为二进制格式，以支持在内容分发网络上托管任意程序。截至本文撰写时，已有多种运行时被推出，可支持在浏览器之外运行 Wasm，包括面向智能手机和微控制器等设备的 Wasm3 [52] (C/C++、Rust、Go)，支持 X86_64 和 AARCH64 的 Wasmtime [9] (Rust 和 C++)，以及专为嵌入式系统设计、占用空间小的 Wasm 微运行时（WAMR） [51], 。WAMR 支持多种架构，包括 X86_32/64、ARM/AARCH64、MIPS 和 XTENSA，并完全符合 W3C Wasm-MVP 规范 [53]。WebAssembly 可提供快速的沙箱执行环境，并有望通过软件故障隔离（SFI）在嵌入式设备上实现程序容器化；WAMR 则提供了可扩展的运行时，以支持传感器访问和平台无关访问控制。

2.1 MQTT/MQTT-SN

在超低功耗物联网场景中，最常用且最实用的无线标准是无线技术，包括蓝牙低功耗（BLE）和 IEEE 802.15.4 [8], 。合适的应用协议有助于最大限度地提升物联网网络所提供的服务质量。此类协议包括 CoAP [7], 、XMPP [42], 、DDS [23], 、MQTT [27], 和 MQTT-SN [45]。受限应用协议（CoAP）旨在通过 UDP 在用户与终端设备资源之间实现面向资源的交互。可扩展消息与在线状态协议（XMPP）是一种基于 XML 语言的面向消息中间件，用于实时交换结构化且可扩展的数据。数据分发服务（DDS）是一种网络中间件，通过实现发布订阅模式，使设备之间能够建立机器间连接 [14]，以进行分布式处理。消息队列遥测传输（MQTT）是一种基于 TCP/IP 的轻量级消息协议，支持通过发布/订阅模式交换数据。表 1 从 UDP 传输、去中心化架构、发送方/接收方解耦以及多播能力等特性对这些协议进行了介绍。

MQTT-SN 是一种基于传统 MQTT 协议的轻量级应用层协议，专为无线传感器网络（WSN）设计 [35]。构成 MQTT 网络的主要实体是**代理**和**客户端**。代理负责维护消息队列，其中包含从各个客户端发布者处接收的数据，并将其分发给相应的客户端订阅者。客户端作为终端用户，既可以向代理发布数据，也可以订阅以接收已发布的消息。连接通过**主题**建立，主题是标识一系列消息的唯一标识符。发布订阅方案的一项主要优势是实现客户端之间的解耦

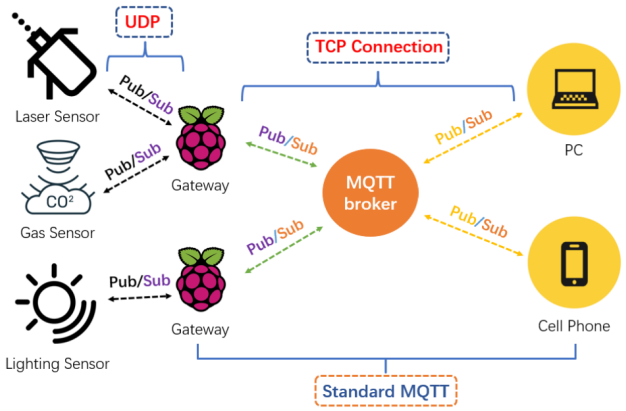


图 1：MQTT-SN 架构（混合布局）。

同时保证服务质量。随后，代理收集消息并将其分发给所有已订阅的客户端。标准 MQTT 架构 [46] 如图 1 右侧所示。

MQTT 存在某些限制，使其无法部署到嵌入式系统中，因此催生了 MQTT-SN 的设计。MQTT-SN 架构的关键组成部分是**网关** 组件，它用于连接资源受限的客户端与 MQTT 代理。首先，众所周知，TCP/IP 网络协议栈并非在资源受限设备上维持长期会话的节能方式。因此，在 MQTT-SN 通信中，设备通过 UDP 连接与网关通信，从而降低总体能耗；随后，网关通过标准 TCP 与代理通信。其次，MQTT 主题定义采用较长的字符串格式，不适用于 IEEE 802.15.4 无线电。MQTT-SN 引入了一个 16 位长度的整数，以取代任意字符串形式的主题名称。随后，终端设备使用新的注册命令，使网关能够将该 16 位整数映射到有效的 MQTT 主题。最后，MQTT-SN 允许客户端进入休眠模式，同时保持活动连接。客户端休眠期间，网关会缓冲下行消息，并在客户端重新唤醒后将消息发送给它们。我们的实现采用了图 3 所示的混合架构，以消除单点故障。

3 设计挑战与目标

将 WebAssembly 直接扩展到嵌入式系统中，以便通过 MQTT-SN 访问传感器并实现应用层网络通信，面临着若干挑战。

交互式嵌入式系统。尽管嵌入式系统通常可以长时间运行给定任务，但不通过物理方式重新刷写设备来更新应用程序是一项不小的挑战。引入支持网络化控制的动态运行时，可以让开发人员远程部署应用程序，而无需亲自接触设备。

定义明确的标准。目前尚无明确标准规定如何将传感器访问接口连接到相关服务。某些实时操作系统（例如 Zephyr[30], VxWorks[10]）遵循 libc 接口设计，以封装特定平台的软件开发工具包提供的底层函数。遗憾的是，libc 标准并未涵盖传感器访问接口，因此不同实时操作系统的特定平台实现各不相同，

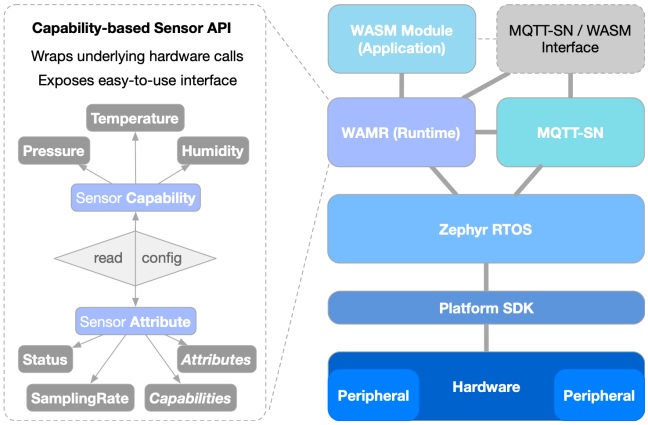


图2：传感器接口示意图（左）用于通过应用程序栈配置和访问设备传感器，应用程序栈图示（右）。

这限制了应用程序二进制文件在不同设备间的可移植性。

最小化开销。 额外的延迟和内存占用之间的权衡通常限制了实时操作系统中安全机制的引入。Wasm 可以通过提供自包含环境来弥补 DEP 和安全扩展页留下的空白。

可扩展的细粒度访问控制。 在嵌入式设备上运行的 Wasm 模块必须能够访问底层服务和应用程序接口，而不能采用粗粒度配置，授予其超出所需范围的权限。因此，需要在函数级别实现细粒度访问控制，既要控制对设备资源的访问，也要控制对 MQTT-SN主题的访问。此外，代表 Wasm 模块身份的签名必须能够由系统验证。最后，鉴于大规模部署的潜在需求，还必须采用可扩展的访问控制方案。

安全且加密的会话通道。 MQTT-SN仅在客户端和代理之间提供 TLS/DTLS 安全会话通道。需要采用保护机制，确保只有适当的客户端才能接收并解码发布的数据。

不可信的 MQTT 代理。传统的 MQTT-SN 设计假设代理是可信的。一旦代理遭到入侵，客户端与代理之间的任何 TLS 都不再安全，发布的消息也可能泄露。所提供的访问控制方案应在设计时避免将代理纳入信任锚。

4 WASI 扩展接口

所提出的传感器接口设计遵循万维网联盟（W3C）组织提供的官方 WebAssembly 系统接口（WASI）。WASI 是一种遵循 POSIX规范的 API标准，旨在为 Wasm 模块提供访问类操作系统功能的途径，例如文件系统、时钟和熵生成器。简而言之，WASI 为原生系统接口提供了通用封装，并通过能力检查将其封装为供 Wasm 模块使用的接口。这种设计模式让人想起三星的 SmartThings；SmartThings 根据传感器具备的能力来定义传感器（例如，音频控制器支持 setVolume、turnUp 和 turnDown）。与专注于云端设备代理表示的 SmartThings 不同，我们这种基于通用能力的传感器接口可将设备外设直接暴露给本地Wasm应用程序。因此，访问传感器与在操作系统中访问文件类似。设备暴露的能力会映射到相应的 Wasm 接口。

当 Wasm 模块执行函数调用时，运行时会拦截该调用，并将其重定向到相应的被调用函数以开始执行。如果模块调用的函数并非由运行时本身实现，而是由操作系统或其他库提供，则控制流会经过验证，并被导向相应的来源。整个执行过程都发生在预先分配给 Wasm 模块的线性内存中。例如，要访问传感器，Wasm 运行时必须将实时操作系统或平台软件开发工具包的底层实现封装到提供给 Wasm 模块的接口中。程序的系统栈和控制流如图2所示。

4.1 传感器接口原语

表 2 概述了传感器接口原语。传感器由一组**能力**和**属性**定义。能力代表底层传感器功能。例如，BME280 可提供温度、湿度和压力能力，而 CCS811 传感器可提供 CO_2 和挥发性有机化合物（VOC）能力。属性表示传感器状态和配置，例如当前采样率或 LCD 显示屏的当前值。为查询这些值，每个传感器都有两个通用属性：**能力** 属性用于指定传感器的能力列表，而**属性** 属性用于指定传感器的属性列表。通过第 5 节所述的网络内访问控制，可明确授予模块对每项能力和属性的读取和/或写入权限。由于此接口旨在用于 Wasm（即由高级语言编译而来），因此接口原语采用低级且通用的形式，从而便于从高级语言进行转换。

getSensors：首先，我们提供一个接口，用于获取嵌入式设备上可用传感器的列表。此接口将读取设备的全局配置，以查看连接了哪些传感器。调用**getSensors** 会返回与已连接传感器关联的传感器标识符列表，例如 BME280、CCS811 或 LCD显示屏。

开启/关闭：此接口用于初始化和反初始化传感器，以便模块随后访问。运行时会拦截这些访问，以确保使用方式适当，并执行底层的 SDK操作。例如，*turnOn("BME280")* 可以初始化传感器（如果运行时尚未初始化），并在需要时将采样率设为 1 秒。为避免多个模块尝试修改同一设备时出现竞态条件，可以选择将*turnOff*保留给运行时，或者将其设为空操作，并通过运行时检查确保没有模块正在注册指定的传感器，正如我们的实现所做的那样。

config：通过 *config* 接口可以更新传感器的属性。该接口可用于配置采样率及其他相关状态变量。配置以键值对的形式提供，用于指示指定的属性值。例如，config("BME280"," SamplingRate", 1000000000) 表示 BME280 传感器应每秒报告一次湿度值，从而覆盖之前指定的（例如默认的）采样

表 2：传感器接口原语概览

接口	参数	返回值	功能
getSensors	*None*	字符串数组	获取可用传感器 ID 列表
turnOn	传感器 ID	布尔值	如有需要，初始化传感器
关闭	传感器 ID	布尔值	取消初始化传感器
config	传感器 ID、属性、值	布尔值	更新传感器的属性（例如采样率）
read	传感器 ID，（能力 | 属性），缓冲区，长度	布尔值	使用当前传感器或属性值更新缓冲区

间隔。运行时会验证模块指定的配置值，以确保这些值不会违反设备状态。通过先公开传感器的属性，再向模块提供对这些值的访问权限，开发者便能够配置传感器状态。每个属性的读写权限都会明确授予各个模块。该接口有意设计为通用接口，从而允许开发者灵活地访问并控制传感器状态。

读取：读取原语会根据指定参数（即请求的传感器或配置）检索当前有效的能力值或属性值。例如，读取具有湿度、温度和压力数据的 BME280 传感器时，会获取当前传感器值。同样，也可以获取采样率和其他状态配置。通过读取两个通用属性，可以获取可用能力和属性的列表，例如 *read("BME280","capabilities")* 和*read("BME280","attributes")*。

状态：此状态属性是一个独特且通用的属性，用于指示传感器的运行状态。当前有效的状态枚举值包括活动、已停止、工作中 和休眠中。状态更新由运行时负责，运行时会维护活动状态，供正在运行的模块读取。传感器启动时，运行时会初始化传感器，并将状态更新为活动。同样，在执行读取或配置更新时，传感器状态会短暂更新为工作中 ，以表明传感器正在使用中。模块的所有访问都会由运行时缓冲，随后执行。

4.1.1 传感器接口示例。以下示例展示了一个连接到 BME280 传感器并读取当前湿度的 C 程序，该程序会被编译为类似的 Wasm 程序：

清单 2：使用传感器接口编译为 Wasm 的 C 程序示例

```
int main(){
    turnOn("BME280");采样率 = 1000000000;
//ns 配置("BME280","采样率",采样率);字
符缓冲区 [] ={0,0,0,0};读取("BME280","湿
度",缓冲区,4);}
```

4.2 网络接口原语

首先，我们在下表中总结了网络接口：Wasm运行时公开的 MQTT-SN 网络接口遵循与 MQTT-SN 本身相同的规范，因此减轻了将高级语言转换为编译后的 Wasm 目标的负担。接口请求会被转换为等效的请求，并由运行时实现。表 3 总结了该接口，其中包括以下内容：

searchGW：此 API 用于通过向网关在特定端口监听的 IPv6 多播地址发送多播消息，查找 MQTT-SN 网关的地址。它还将跳数限制作为参数，用于指定给定消息允许经过的最大跳数。

start： 此 API 用于在终端设备的特定 UDP 端口上启动 MQTT-SN客户端。后续网络消息将通过此端口发送，并且在执行任何网络操作之前，必须先调用此接口。

connect： 此接口允许客户端使用给定地址连接到网关（该地址可以是预先已知的，也可以通过 *searchGW* 接口确定）。默认情况下，连接的保活时间为 30 秒。每条连接消息最多等待 10 秒以接收网关的确认。鉴于 MQTT-SN 基于 UDP，最多重传三次，之后客户端将停止发送额外的连接消息。

register： 此接口用于向代理注册主题。客户端向代理发布数据之前，必须注册主题名称，以便代理建立主题名称与 16 位 MQTT-SN 主题 ID 之间的映射。代理首先发送一条确认消息，表明连接正常，然后发送一条消息，其中包含供客户端和网关使用的转换后主题 ID。

发布/订阅： 这两个接口用于发布和订阅特定主题。使用发布接口发布数据时，必须先将数据转换为字节数组，再发送到主题 ID 映射，以转换为 MQTT 主题名称。与 MQTT-SN 接口类似，用户可以指定所需的服务质量等级：0、1、2。值为 0 表示客户端仅发布一次数据，不检查是否成功送达。值为 1 表示发布者会持续发布，直到收到发布确认（PUBACK）。最后，值为 2 表示代理与客户端之间进行两步握手，以确保幂等发布；在连接有效的情况下，每条消息都保证只发布一次。使用订阅 API 订阅主题时，可以使用主题名称或主题 ID 来订阅指定主题。该接口还支持与发布接口类似的服务质量配置。

hasMessage/getMessage： 由于 Wasm 具有结构化的控制流，运行时会为已发布的 MQTT 消息验证并注册自己的回调。*hasMessage* 接口允许模块查询运行时是否有消息可用；若有，模块便可通过 *getMessage* API 访问该消息，前提是

表 3：网络接口原语摘要

接口	参数	返回值	功能
启动	Port	绑定端口	启动 MQTT-SN客户端
stop	*None*	布尔值	停止 MQTT-SN客户端
搜索网关	广播地址、端口、最大跳数	网关地址	通过广播识别网关地址
连接	客户端标识符、保持连接、网关、端口	布尔值	连接到网关并设置会话时长
注册	主题名称	主题标识符	在代理中注册主题并获取映射的标识符
发布	主题标识符、服务质量、有效载荷	布尔值	以指定的服务质量级别向某个主题发布数据
订阅	主题标识符、服务质量	布尔值	将模块订阅到指定主题
有消息	主题标识符	布尔值	指示运行时是否有可用消息
获取消息	主题标识符	字节数组	返回下一条已发布的消息
休眠	长度（可选）	布尔值	在网关暂存消息
唤醒	*None*	布尔值	恢复接收消息

模块请求有效，且已通过适当的访问控制完成注册。

休眠/唤醒：休眠命令会暂停设备上的 MQTT-SN 客户端，并在运行时中保存连接状态。唤醒命令将根据已保存的状态恢复 MQTT-SN 连接，使设备能够在保持 MQTT-SN 连接活动的同时长时间休眠。网关会暂存休眠期间发布的消息，并在设备唤醒后将其转发给客户端。

4.2.1 网络接口示例。以下示例展示了一个 C 程序（可编译为功能等效的 Wasm 模块），该程序启动 MQTT-SN 客户端、连接到网关、注册主题、发布一个值，然后断开连接。

清单 3：使用 MQTT-SN 网络接口编译为 Wasm 的 C 程序示例

```
int main(){int 保持连接 = 30;//秒 字符串
客户端标识符="WASM" 字符串 IP="fdde:
ad00..." int 端口 = 47193;启动(1000);连接(
客户端标识符,保持连接,IP,端口);int 主题标
识符 =注册("湿度");发布(主题标识符,2,数据)
;断开连接();}
```

5 访问控制

同时提供传感器接口和 MQTT-SN 接口，可让 Wasm 应用程序访问板载传感器并连接远程设备。然而，由于大多数微控制器和实时操作系统都没有提供完善的访问控制方案来限制应用程序的行为，我们引入了一种基于 WKD-IBE 的加密方案以及针对 MQTT-SN 的访问控制。基于通配符的加密具有显而易见的优势：模式委托可以串联和分组，因此天然适用于 MQTT-SN 的多对多消息传递模型。

5.1 WKD-IBE

WKD-IBE 是通配符身份基加密的缩写。这是一种使用模式加密消息的加密方案。WKD-IBE 使用的密钥与模式绑定，而不是使用与不同实体相关联的公钥/私钥对。模式定义如下：$P = (P_1, \ldots\ldots, P_l) \in (\{0, 1\}^* \cup \{*\})^l$ 其中 $l \le L$。$*$ 是一个特殊的通配符，L 是 WKD-IBE 方案的最大深度。P 表示由 l 个分量组成的模式向量 P_i，$1 \le i \le l$ 模式 P^* 与 P' 相匹配，记作 $P^* \in_* P'$，当且仅当对每个 $i \in \{1, \ldots\ldots, l\}$ $P^*(i) = \{*\}$ 或 $P^*(i) = P'(i)$。具体来说，只有当模式 P^* 中的每个元素都等于特殊符号 $*$（表示任意值）或 P' 中相同位置的元素时，才表示 P^* 与 P' 相匹配。反过来则不一定成立，即 P' 与 P^* 相匹配。

WKD-IBE 为每种模式指定一个密钥。模式 P^* 的密钥可以解密使用模式 P^* 加密的消息。如果 P^* 匹配 P'，同一个密钥还可以解密使用模式 P' 加密的所有消息。如果需要为模式 P' 生成新的密钥，则当 P^* 与 P' 匹配时，也可以使用其密钥生成 P' 的密钥。一般而言，使用 WKD-IBE方案的流程如下：

初始化：在初始化过程中，根身份随机选择 $params = g$、h_0、$\ldots h_L \in \{0, 1\}^*$ 作为用于推导主公钥和私钥的参数。主公钥为 $mpk \leftarrow (g^\alpha, params)$，其中 $\alpha \in \mathbb{Z}_n$ 是一个素数。至于主密钥，则通过 $msk \leftarrow h_0^\alpha$ 推导得到

密钥派生：接下来，派生特定模式对应的密钥。密钥派生函数以 $params$、$secret\ key$ 和 $P = (P_1, P_2, \ldots\ldots, P_l)$ 作为输入，其中这些内容构成目标模式，并据此派生模式 P 对应的秘密密钥 P_{sk}。对于 $secret\ key$，它可以是 **初始化** 阶段派生的 msk，也可以是派生出的 P_{sk}，用于为匹配的新模式 P' 派生新密钥 P'_{sk}。有关密钥派生的更多详情，请参阅 [1]。

加密：要加密消息 m，加密方法需要将 $params$ 和模式 P 作为输入。例如，如果消息 m 是针对模式 $P' = (P'_1, \ldots\ldots, P'_l) \in_* P$ 加密的，那么发送方（已拥有 P_{sk}）会使用 $mpk = (params, g^\alpha)$ 计算出若干密文块，并将它们组合成最终密文。这些密文块的计算方式如下：

(1) $C_0 = g^t$ 其中 t 是随机选取的素数

(2) $C_i = H_i(P'_i)^t$ where $i \in \{1, ..., l\}$. H_i is the L independent random oracles mapping $\{0,1\}^* \rightarrow \mathbb{G}$ where $\mathbb{G}$ is group of prime order p
(3) $C_i = h_i^t$ for every $i \in \{l+1, ..., L\}$
(4) $C_{L+1} = m \times \hat{e}(g_1, h_0)^t$

最终密文组装为 $C = (C_0, \ldots\ldots, C_{L+1})$。

解密：解密由模式 P 加密的消息时，接收方应使用私钥 P_{sk} 或派生密钥 P'_{sk}。

如文中所述，基本加密方案用于确保所有消息均经过端到端加密，且只有获准访问的客户端才能解密已发布的消息。然而，WKD-IBE 未解决密钥委托问题；因此，需要采用分层密钥结构来委托和撤销密钥。

5.2 资源和请求者的层级结构

WKD-IBE 专为单一层级的密钥委托而设计。其预期用途是委托通配符密钥，例如电子邮件地址中的兜底地址。将 WKD-IBE 应用于嵌入式系统时，嵌入式资源的结构可以与电子邮件地址的结构相似。对于远程请求者，资源可以表示为由多个域组成的层级结构。例如，位于工厂中的温度传感器可以用表 4 中的 A 表示，它同时作为描述远程资源的统一资源标识符。

为了更好地管理不同请求者所使用的不同密钥，并能够在必要时撤销某个子群组的密钥，还需要使用层次结构来表示统一身份标识符（UII）。同时，我们引入结构化用户身份，具体如下。例如，名为 Alice 的用户需要从本地工厂的 BME280 传感器获取温度值。于是，Alice 的结构化身份变为 /*entity*/*location*/*Alice*/*NULL*。 *entity* 与用户姓名之间的域表示请求者的地理位置（用户姓名由请求者控制）。*delegated User* 域表示被 *Alice* 授予权限的目标实体（NULL 表示未进行委托）。

在分别为资源和请求者建立层次结构后，需要一种网络级密钥交换方案，使远程嵌入式设备能够向已授权的群组委托密钥。

5.3 密钥建立

在大多数情况下，最好不要求信任代理；理想情况下，只有已授权的终端用户才能访问消息。因此，WKD-IBE方案的解密密钥应仅对已授权群组可访问。在这种情况下，我们在 MQTT-SN 应用层协议之上传输层中构建了一种类似于传输层安全协议所采用的密钥交换方案。我们首先定义针对远程用户请求的不同能力和属性的密钥建立过程。

以 BME280 传感器为例。BME280 能够感知环境的温度、湿度和压力。因此，BME280 的远程资源统一资源标识符为 /*resources*/*device*01/*BME*280/*temperature*，用于表示温度功能。当终端设备采集温度值并将其发布到代理时，应对值内容进行加密，使其仅可由远程用户访问。用于加密该值的密钥是通过 WKD-IBE 派生的主公钥 mpk。然而，对于 WKD-IBE 加密和解密中使用的模式，由于它应仅对授权组开放访问，密钥派生也应将统一资源标识符和结构化身份都作为模式输入参数。

我们假设有三个标识符，即../*Loc*1/*Alice*、.. *Loc*1 *Bob* .. *Loc*2 *Sarah* / /和/ /，希望访问传感器 BME280 的温度属性。同时，我们还假设 *Loc*1 中的人员有权访问数据。在这种情况下，用于加密传感器数据的模式如下：$P = (P_1, P_2)$，其中 P_1 = /*entity*/*Loc*1/∗ 和 P_2=/*resources*/*device*01/*BME*280/*temperature*。结合该模式 P、终端设备所拥有的主公钥 mpk 以及传感器值的数据 m，即可对其进行加密。至此，加密阶段已完成。

在生成与模式 P对应的解密密钥时， *Alice* 和 *Bob*之间的密钥派生过程有所不同。为了派生 *Alice*的解密密钥，设备使用主密钥 msk和模式 $P' = (P'_1, P'_2)$，其中 P'_1 = /*entity*/*Loc*1/*Alice* 和 $P'_2 = P_2$，以生成解密密钥 P'_{sk}。在为 *Bob*派生密钥时，模式变为 $P'' = (P''_1, P''_2)$，其中P''_1 = /*entity*/*Loc*1/*Bob* 和 $P''_2 = P_2$，以派生密钥 P''_{sk}。 P'_{sk}和 P''_{sk} 都能够解密由模式 P 加密的消息，因为 $P'_1 \in_* P_1$ 和 $P''_1 \in_* P_1$，这表明两种模式都与加密模式匹配。

5.4 密钥交换

当终端设备接收到远程用户请求访问某个传感器值的权限时，就会建立解密密钥。这里介绍通过 MQTT-SN 使用类似 TLS 的方法来建立密钥交换。用户需要发布一个请求，申请授权并获取特定统一资源标识符的数据解密密钥。我们将用户应发布请求的主题定义为表 4 中所示的 请求负载 。 *key*之前的所有域名都表示属性的统一资源标识符。请求者将请求的能力插入消息载荷中，该载荷由多个字段组成：请求者希望在设备上执行的接口、能力和属性；按照前述定义语法表示的用户标识符 I_U ； 以及请求者的公钥 U_{pk}。完整载荷的结构类似于 *read* ； /*entity*/*Loc*1/*Alice*； $Alice_{pk}$。如果用户 *Alice* 希望请求 *BME*280 传感器的温度值，则将其发布到主题 /*resources*/< *deviceID* > /*BME*280/*temperature*/*request* 。如果用户希望获得 *config* 访问权限，载荷中唯一的区别是将 *read* 替换为 *config*|*sensorID*, *attribute*。

请求者生成有效载荷后，为避免将代理作为框架中的信任锚，有效载荷内容会使用设备的公钥 D_{pk} 加密，且只有设备本身可以访问该公钥。这里假设请求者和设备的公钥均由 PKG 中心等可信第三方生成（这些密钥仅用于请求和授予权限）。数据加密后，会发布到代理，并由设备接收。在设备端，为了接收权限请求，设备应订阅相同的主题。为确保框架可用，设备应始终订阅主主题 */resources/device01/**。每当用户针对任意传感器属性发布权限请求时

资源	层级结构
BME280的能力	/resources/<设备 标识符>/BME280/<能力>
BME280的属性	/resources/<设备标识符>/BME280/<属性>
请求者 UII 请求负载	/entity/<位置>*/<请求者标识符>/<受委托用户> <接口|参数>;<请求者>;<请求者的公钥>

表 4：访问控制资源和请求者层级示例

主题	主题格式
BME280 的能力请求	/resources/<设备标识符>/BME280/<能力>/request
BME280 的属性请求	/resources/<设备标识符>/BME280/<属性>/request
用于从 BME280 签发密钥的主题:	/resources/<设备标识符>/BME280/<能力>|<属性>/issue
用于接收以下内容的返回值的主题: 来自 BME280 的接口:	/resources/<设备标识符>/BME280/<能力>|<属性>/ret

表 5：表 4 所述示例中使用的 MQTT-SN 主题

在 *device*01 下，终端设备始终可以收到请求。收到请求后，终端设备可以使用自己的私钥 D_{sk} 解密有效载荷并提取请求信息。随后，设备可以独立判断是否应授权用户获得相应权限。

现在，*Alice* 已获准访问设备的能力和属性，便使用 WASI-SN 提供的接口开发程序，并通过主题 */resources/ device*01/*BME*280 发送该程序，同时附上由她的私钥 U_{sk} 签署的签名，该私钥由 PKG 中心颁发。签名消息由 Alice 的统一身份标识符和设备公钥加密的有效载荷解密密钥 D'_p 组成，该设备公钥 D_{pk} 如上所述。有效载荷对应的解密密钥将按 D'_p 推导得出，再使用设备公钥 D_{pk} 加密并放入签名中。当指定设备上运行的 Wasm运行时收到有效载荷后，会检查签名以确认请求者有效，并使用其私钥 D_{sk} 解密有效载荷，从而获取 Wasm 模块。在执行前及执行过程中，运行时会检查模块中使用的所有接口函数调用，以确保对能力和属性的访问有效且已获准，从而实现交互式且安全的传感器网络部署。

5.5 密钥撤销

5.5.1 完全子树广播加密。对于密钥撤销方案，我们直接采用同时支持密钥过期和即时密钥撤销 [26]的 JEDI 方案。其基本思路是为不同实体管理一棵完全子树（CS）。前文介绍的请求者层级结构旨在帮助构建完全子树。每个用户对应一个叶节点，其所有父节点分别代表统一身份标识符（UII）中的不同域，并且每个节点都包含其所属域各自独立的 WKD-IBE 密钥对。因此，每个用户都拥有根叶路径上所有节点对应的全部秘密密钥。为了加密一条仅能由部分用户解密的消息，应从包含所有必要叶节点的子树集合中选择加密密钥。随后，使用每棵子树根节点各自不同的公钥对消息进行多次加密。

5.5.2 即时撤销。当密钥被撤销时，发行者（即终端设备）会搜索完全子树，找出需要撤销的叶节点范围。终端设备会选取包含其余有效叶节点的子树，并使用这些叶节点各自祖先节点的公钥来执行新的加密。密钥被撤销后，所有被移除叶节点对应的用户将无法再解密已发布的消息。同样的方法也适用于密钥过期，但每台设备会先监控是否有叶节点过期，然后再执行密钥撤销。

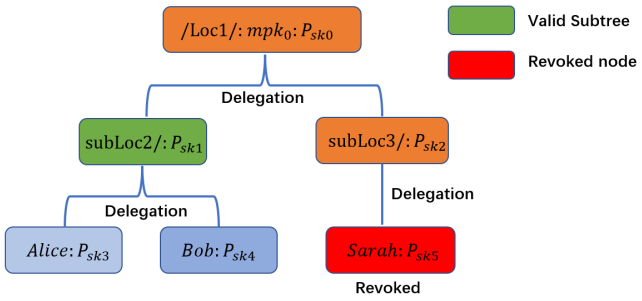


图 3：完全树的密钥管理。当 Sarah 的密钥被撤销时，subLoc2/is 是覆盖所有其余有效节点的剩余子树根节点

6 评估

我们将方案应用于真实的嵌入式设备测试平台，评估 Wasm 扩展接口的性能和访问控制开销，并将我们的 WebAssembly

应用程序	Wasm 模块	原生代码
访问传感器	10 代码行数（357B）	23 代码行数（893B）
MQTT-SN客户端	16 代码行数（523B）	97 代码行数（3,003B）

表 6：所评估 Wasm 模块与原生代码的代码大小

实现与原生代码进行比较。Cortex-M4F@64 MHz 微控制器的NRF52840-DK开发板上运行，该微控制器采用 Armv7E-M架构并配有 256KB SRAM。开发板使用 IEEE 802.15.4 作为链路层和物理层规范进行通信。在以下结果中，标记为 native 的所有数据表示程序在 Zephyr 实时操作系统中的执行结果；未标记为 native 的数据则表示 Wasm 模块在 Zephyr 实时操作系统中运行的结果，其使用的是我们基于英特尔的 Wasm 微运行时（WAMR）定制的运行时。为初始化实验，我们选择配备 Cortex-A53@1.4GHz CPU 的 Rpi 3B+ 作为网关，该网关连接到运行在配备英特尔酷睿 i7-6700@4 GHz CPU 的个人计算机上的代理。为初始化实验，终端设备与网关之间采用混合模式连接，以避免单点故障。为了在嵌入式设备与网关之间使用 IP 作为网络协议，我们选择 OpenThread 组建 Thread网络，并使用 NRF52840-dongle 作为网络协处理器（NCP）来管理连接。以下实验所使用的传感器为 BME280 传感器。实验初始化设置如下图4所示。

表6展示了评估程序的代码大小和代码行数，这些程序分别是：（1）编译为 WebAssembly 的 C 程序；（2）原生C应用程序。与原生应用程序不同，大多数 WebAssembly 功能由 Wasm运行时处理，从而减轻了应用程序开发者的编码和维护负担。此外，由于 Wasm模块可在不同设备间移植，因此无需针对设备进行定制实现，从而实现代码复用。

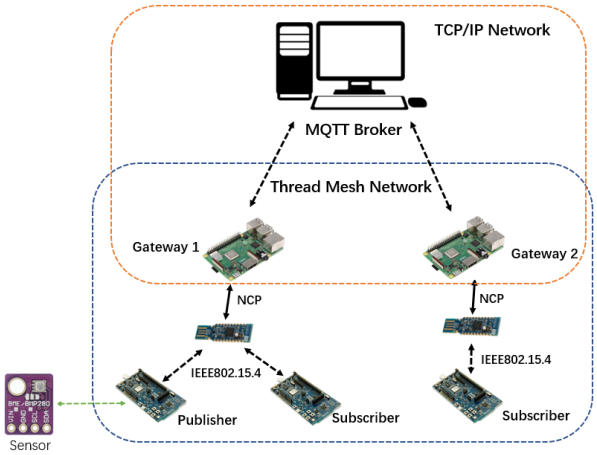


图4：实验初始化

6.1 WASI传感器接口开销

内存占用：我们首先测量将 WebAssembly 用作目标二进制格式所引入的额外内存开销

环境	编译	运行时
原生代码	125,162B	*N/A*
WASI-SN	131,500B	10,189B
WASI-SN +MQTT-SN	212,176B	10,189B

表 7：WASI-SN WAMR 实现（也包含 MQTT-SN）的内存消耗与原生代码相比。应用程序可以选择预编译为 Wasm，以消除编译开销。

。为突出传感器接口的内存消耗开销，应用程序的主函数仅包含第 3.3 节中定义的传感器接口。在这项首要指标测试中，应用程序只初始化传感器并尝试获取一次温度值。结果如表 7 所示。

与原生编译相比，WASI-SN 在编译期间会带来 5% 的内存开销。鉴于部署这些程序的开发板只有 256KB SRAM，这种开销通常应该可以接受。此外，由于 Wasm 被有意设计为可移植的字节码，应用程序可以在其他主机上预编译，从而完全消除编译所需的内存。本结果使用了分配给运行时、用于编译的 100kB 内存池。为公平比较，我们也为原生应用程序静态分配了 100KB 内存。

模块执行期间，WASI-SN 仅消耗 10kB，与编译时静态预分配的开销相比，这一开销相对较小。运行时测得的动态内存占用保持在约 10437B。由于原生程序是在 Zephyr 实时操作系统上执行的，其执行方式决定了我们无法报告其用量，因此未报告原生程序的结果。MQTT-SN 库在编译期间会带来显著的内存占用，这主要是由于库本身的代码大小；不过，对于程序中未使用 MQTT-SN 的模块，添加该库不会影响运行时。

应用延迟：接下来，我们测量了通过 Wasm 接口访问传感器时相对于原生应用的延迟。我们通过多次读取 BME280 传感器的温度值来分析延迟，以展示加载 WAMR 运行时所产生的固定启动开销。结果如图 5 所示。随着重复访问传感器，通过传感器接口调用所产生的延迟开销逐渐降低；连续访问 32 次后，延迟差异稳定在约 6% 的开销。此外，使用 MQTT-SN 库和不使用该库时，运行时的延迟开销几乎相同。这是预期的，因为在本次评估中，Wasm 程序并未使用 MQTT-SN 接口。

Wasm 运行时延迟：为进一步分离 WASI-SN 运行时的启动开销，我们对运行时初始化、执行和销毁各阶段进行了生命周期分析。我们测量了表 8 中描述的每个阶段，以全面审视各个独立环节。需要注意的是，这些阶段大多只在运行时首次初始化时产生；此后便可连续执行多个模块。所使用的主函数非常简单，只执行返回操作。结果如图 6 所示。

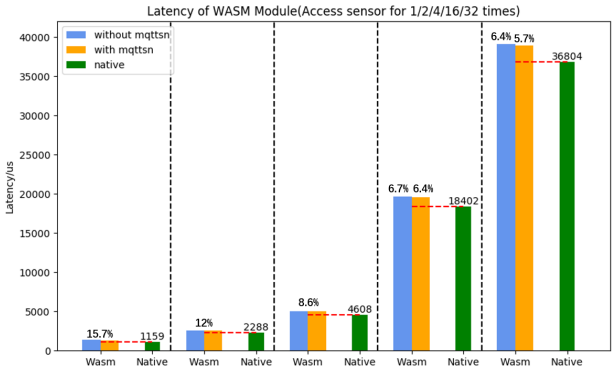


图 5：访问传感器不同次数时，WASI-SN 与原生代码的延迟对比

表 8：Wasm 运行时生命周期

阶段	功能
运行时初始化	初始化运行时并分配内存
加载模块	加载 Wasm 模块
模块初始化	初始化模块的函数
创建执行环境	为模块创建沙箱
调用 main	执行模块
销毁模块	按顺序销毁变量
销毁运行时	终止运行时，清理内存

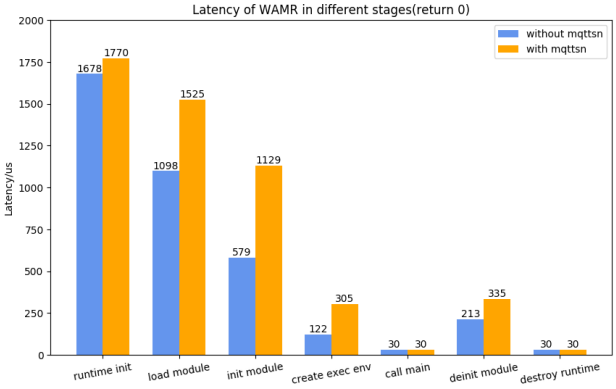


图 6：WASI-SN 运行时生命周期的延迟分析。除了主要阶段之外，所有阶段都可以由运行时复用于后续模块。阶段说明见表 8。

大部分启动开销都发生在加载带有相应沙箱环境的 Wasm 模块之前（即运行时"启动"阶段）。加入 MQTT-SN 库显著增加了这些前期成本，这可能是因为需要处理额外的符号并加载相关库（例如网络接口）。然而，只要模块不访问这些特定扩展，MQTT-SN扩展就不会在模块执行期间增加延迟。图 7 展示了各个运行时阶段的完整细分情况，如

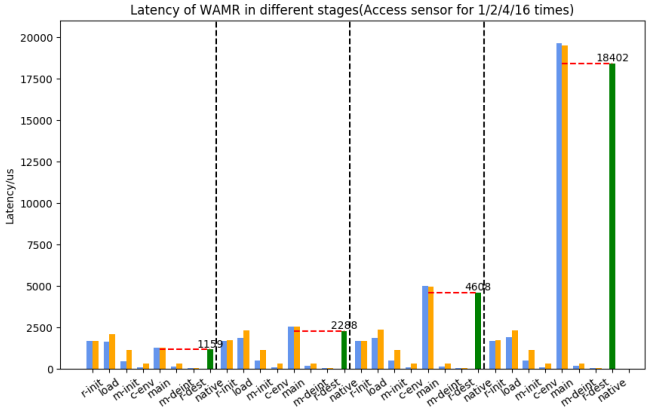


图 7：传感器访问次数可变时 WASI-SN 运行时生命周期的延迟分析。随着 Wasm 模块执行时间的增加，运行时和模块初始化的固定开销会降低。

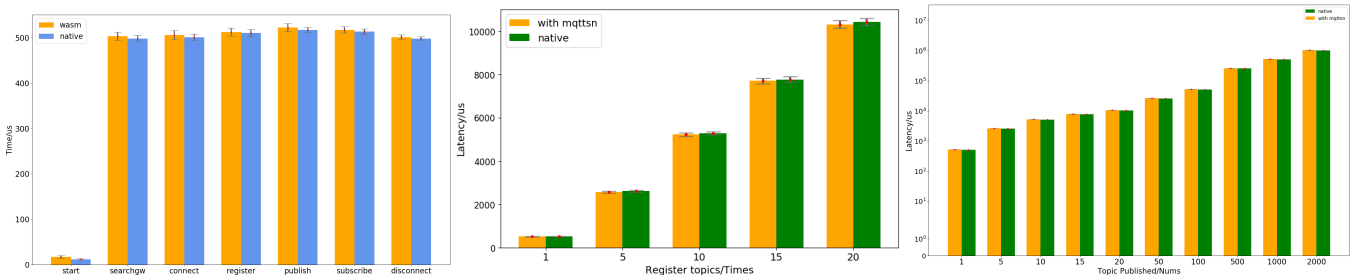
传感器访问次数增加。如我们所见，随着模块复杂度或执行时间的增加，启动开销不会发生变化，并且 WASI-SN 扩展相对于原生执行提供了类似的 6% 开销。在模块可能被多次执行后，运行时只需反初始化模块，而无需销毁自身，即可开始运行另一个模块，这与嵌入式应用中常见的主循环类似。

6.2 WASI 网络接口开销

接下来，我们评估了 MQTT-SN 接口相较于原生接口的端到端延迟。为确定端到端延迟，我们测量了调用接口到收到网关确认之间的时间。从开发者的角度来看，端到端延迟能够实际反映 WASI-SN 的影响，因此我们选择它作为评估指标。首先，我们测量了使用第 4 节中定义的不同网络接口时的延迟。结果如图 8 所示。

运行**启动**接口所涉及的延迟微乎其微，因为它不涉及任何网络传输。相反，它只是在 WebAssembly 沙箱内的特定端口启动一个 MQTT-SN 客户端。在执行其他需要向网关发送数据包的网络接口时，WASI-SN 实现的延迟几乎与原生接口完全相同。这主要是因为网络延迟在延迟结果中占主导地位。无论请求哪种网络接口，WebAssembly 带来的开销都可以忽略不计。

图 8b/c 展示了更常见的场景：应用程序可以注册多个主题，并发布多条消息。注册多个主题所涉及的延迟如预期般与主题数量呈线性关系，与原生代码的差异小于 1%。即使应用程序注册了 20 个主题，产生的延迟仍约为 1 秒，这通常是可以接受的；注册完成后，无需额外注册即可推送多条消息。随着发布的消息数量增加，延迟也呈现类似的线性增长，而额外开销可以忽略不计。



（a）执行不同 MQTT-SN 网络接口请求的延迟。

（b）注册不断增加的主题数量的延迟。

(c) 发布温度值的延迟次数不断增加。

Fi图 8：WASI-SN 中各种 MQTT-SN 网络接口操作的延迟测量

总而言之，采用我们提出的 WASI-SN 实现及其相关 MQTT-SN扩展，在传感器访问和网络接口开销方面，性能差异都微乎其微。因此，将 WebAssembly 用作网络化嵌入式系统的软件故障隔离（SFI）解决方案，对应用程序性能的影响极小，同时还能带来多项安全和可移植性优势。

7 相关工作

面向物联网设备的 WebAssembly：WebAssembly 已引起业界和学术界的广泛关注。将 Wasm 用作嵌入式设备的可移植二进制目标已成为近期关注的热点 [33, 36]。Elliott Wen 等人 [54] 设计了一种轻量级操作系统，通过基于 Rust 的内核在物联网设备上运行 WebAssembly，以保障 Wasm 应用程序的安全。Jacobsson 等人 [21]探索了在不同架构的多种嵌入式设备上运行 WebAssembly 的可能性。英特尔等公司 [51]也通过创建 Wasm 运行时来扩展 Wasm 二进制格式的安全功能，并在运行时强制执行内存检查。随后，Alexandru [38] 提出了一种 WebAssembly 框架，不仅能够在物联网设备上运行 WebAssembly，还支持动态更新，以修复 Wasm二进制文件中的缺陷。

基于 MQTT/MQTT-SN 的安全性：许多现有研究提出了 MQTT-SN 的各种安全方案，以缓解缺乏端到端加密和访问控制的问题。Ousmane 等人 [39] 提出将 ChaCha20-Poly1305 带关联数据的认证加密（AEAD）作为轻量级加密方案，用于保护通过 MQTT/MQTT-SN 传输的数据。Amaran, M. 等人 [2] 评估了 MQTT-SN 上不同加密方案的性能，例如 LBLOCK、AES 和 KLEIN，并得出结论：总体而言，LBLOCK 的性能最佳。Muhammad, Ahsan 等人 [31] 遵循 OneM2M 标准架构实现了一种安全 MQTT 绑定，以在物联网设备之间提供互操作性和安全通信信道。在其他研究中，Sochor 等人 [44] 研究了 MQTT-SN 协议的攻击面。他们成功地针对嵌入式设备实施了 MQTT-SN 分布式反射拒绝服务（DRDoS）攻击，并提出了相应的对策。Singh 等人 [43] 利用密钥策略/密文策略属性基加密（KP/CP-ABE），提出了 MQTT 和 MQTT-SN 的安全版本（SMQTT/SMQTT-SN），以保护传输中的数据。

物联网中的基于身份的加密：随着物联网设备的快速发展，业界开始探索将 IBE 用作物联网网络加密方案的可能性，以轻量级方式保障安全性。Yang Liu 等人 [56] 将 IBE 与 PKI/CA 相结合，构建了物联网网络安全架构，从而通过网络实现安全认证和数据加密。Jebri 等人 [22] 提出了一种适用于物联网和无线传感器网络的通用安全模型，借助 IBE 和基于假名的加密（PBC），保障设备间的匿名性和保密性。Kumar 等人 [26] 最近提出了一种基于通配符基于身份的加密（WKD-IBE）的多对多端到端加密协议 [1]，并通过支持快速密钥委托以及密钥到期/即时撤销方案改进了该设计；该方案应用于我们的 MQTT-SN 访问控制方法。

8 结论

本文提出了 WASI-SN，这是首个扩展 WebAssembly系统接口的实现，使运行在嵌入式设备上的 Wasm 应用程序能够通过统一的传感器接口定义实现网络访问控制。所提出的接口遵循先前定义的基于能力的方案，用于对传感器进行分类、配置和访问。此外，我们还实现了首个可在 Zephyr 上运行的 WebAssembly MQTT-SN 库，并设计了一种端到端的基于通配符的身份加密（WKD-IBE）方案，在不将代理和网关纳入信任链的情况下保护数据隐私。总体而言，我们的框架有望提升嵌入式应用的可移植性和安全性，尤其适用于需要访问传感器和网络的应用，同时开销接近原生代码。通过定义良好的接口，将 WebAssembly 与嵌入式设备结合起来，这一设计有望推动在异构平台上运行安全、可移植应用的发展，从而促进下一代物联网和边缘计算的发展。我们坚信，WebAssembly 很快将成为未来嵌入式领域新的主流二进制格式。

参考文献

[1] Michel Abdalla、James Birkett、Dario Catalano、Alexander W Dent、John Malone-Lee、Gregory Neven、Jacob CN Schuldt 和 Nigel P Smart。2011。通配符身份基加密。密码学杂志24，1（2011），42–82。[2] M Amaran、M Rohmad、L Adnan、N Mohamed 和 H Hashim。2018。MQTT-SN 的轻量级安全。《国际工程与技术杂志》（阿联酋）

7，4（2018），223–226。[3] Mahmoud Ammar、Bruno Crispo、Bart Jacobs、Danny Hughes 和 Wilfried Daniels。2019。安全微型监视器：一种经过形式化验证、面向物联网的基于软件的安全架构。《IEEE 可信与安全计算汇刊》16，5（2019），885–901。[4] Syaiful Andy、Budi Rahardjo 和 Bagus Hanindhito。2017。物联网系统中 MQTT 通信协议的攻击场景与安全分析。载于《2017 年第 4 届电气工程、计算机科学与信息学国际会议（EECSI）》。IEEE，1–6。[5] Apriorit。2020。嵌入式系统的 12 种常见攻击及其防范方法。https://www.apriorit.com/dev-blog/690-embedded-systems-attacks [6] Zelalem Birhanu Aweke 和 Todd Austin。2018。uSFI：面向物联网级设备的超轻量级软件故障隔离。载于《2018 年欧洲设计、自动化与测试会议暨展览会（DATE）》。IEEE，1015–1020。[7] Carsten Bormann、Angelo P Castellani 和 Zach Shelby。2012。CoAP：面向数十亿微型互联网节点的应用协议。《IEEE 互联网计算》16，2（2012），62–67。[8] Hannah Brunner、Rainer Hofmann、Markus Schuß、Jakob Link、Matthias Hollick、Carlo Alberto Boano 和 Kay Römer。2020。商用 Wi-Fi、BLE 和 IEEE 802.15.4 设备之间的跨技术广播通信。载于 EWSN。176–177。[9] Bytecodealliance。2019。使用 Cranelift 的 WebAssembly 独立即时编译式运行时。https://github.com/bytecodealliance/wasmtime [10] DANIEL CASINI。2016。支持 VxWorks 实时操作系统中的分层调度。（2016）。[11] Dong Chen、Phuthipong Bovornkeeratiroj、David Irwin 和 Prashant Shenoy。2018。物联网设备的私密回忆：在物联网时代保护用户隐私。载于《2018 年 IEEE 第 38 届分布式计算系统国际会议（ICDCS）》。IEEE，1327–1336。[12] Jonathan Corbet。2009。Seccomp 与沙箱隔离。LWN.net，5 月 25 日（2009）。[13] Ghada Dessouky、Tigist Abera、Ahmad Ibrahim 和 Ahmad-Reza Sadeghi。2018。Litehax：对程序执行进行轻量级硬件辅助证明。载于《2018 年 IEEE/ACM 计算机辅助设计国际会议（ICCAD）》。IEEE，1–8。[14] Marisol García-Valls、Javier Ampuero-Calleja 和 Luis Lino Ferreira。2017。数据分发服务与树莓派的集成。载于《绿色、普适与云计算国际会议》。施普林格，490–504。[15] Dennis RE Gnad、Jonas Krautter 和 Mehdi B Tahoori。2019。泄漏噪声：混合信号物联网设备中的新型侧信道攻击途径。《IACR 密码学硬件与嵌入式系统汇刊》（2019），305–339。[16] Andreas Gruenbacher 和 Seth Arnold。2007。AppArmor 技术文档。[17] Andreas Haas、Andreas Rossberg、Derek L Schuff、Ben L Titzer、Michael Holman、Dan Gohman、Luke Wagner、Alon Zakai 和 JF Bastien。2017。借助 WebAssembly 提升 Web 性能。载于《第 38 届 ACM SIGPLAN 编程语言设计与实现会议论文集》。185–200。[18] Adam Hall 和 Umakishore Ramachandran。2019。边缘无服务器函数的执行模型。载于《物联网设计与实现国际会议论文集》。225–236。[19] Pat Hickey。2019。使用 Rust 和 WebAssembly 进行边缘编程。https://www.fastly.com/blog/edge-programming-rust-web-assembly[20] Asif Iqbal、Faman Ullah、Hafeez Anwar、Ata Ur Rehman、Kiran Shah、Ayesha Baig、Sajid Ali、Sangjo Yoo 和 Kyung Sup Kwak。2020。用于人体活动识别和医疗保健的可穿戴物联网平台。《国际分布式传感器网络杂志》16，6（2020），1550147720911561。[21] Martin Jacobsson 和 Jonas Willén。2018。基于 WebAssembly 的可穿戴设备虚拟机执行。载于 EAI 身体区域网络国际会议。施普林格，381–389。[22] Sarra Jebri、Mohamed Abid 和 Ammar Bouallegue。2015。物联网匿名通信的高效方案。载于《2015 年第 11 届信息保障与安全国际会议（IAS）》。IEEE，7–12。[23] Woochul Kang、Krasimira Kapitanova 和 Sang Hyuk Son。2012。RDDS：面向信息物理系统的实时数据分发服务。《IEEE 工业信息学汇刊》8，2（2012），393–405。[24] kbumsik。2020。opus-media-recorder。https://github.com/kbumsik/opus-media-recorder[25] Paul Krill。2019。Bytecode Alliance 专注于浏览器之外的 WebAssembly。
https://www.infoworld.com/article/3453716/bytecode-alliance-focuses-on-webassembly-outside-the-browser.html [26] Sam Kumar、Yuncong Hu、Michael P Andersen、Raluca Ada Popa 和 David E Culler。2019。{JEDI}：面向物联网的多对多端到端加密与密钥委托。载于第 28 届 {USENIX} 安全研讨会（{USENIX} Security 19）。1519–1536。[27] Roger A Light。2017。Mosquitto：MQTT 协议的服务器和客户端实现。《开源软件杂志》2，13（2017），265。[28] Gilad David Maayan。2020。2020 年物联网概览：统计数据、风险与解决方案。https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx [29] Héctor Marco-Gisbert 和 Ismael Ripoll-Ripoll。2019。SSPFA：面向 Android 操作系统的有效栈破坏保护。《国际信息安全杂志》18，4（2019），519–532。[30] Gustav Midéus 和 Antonio Morales Chavez。2020。RISC-V 线程隔离：使用 Zephyr 实时操作系统。[31] Ahsan Muhammad、Bilal Afzal、Bilal Imran、Asim Tanwir、Ali Hammad Akbar 和 Ghalib Shah。2019。基于 OneM2M 架构的 Mbed OS 安全 MQTT 绑定。载于 2019 年 IEEE 欧洲安全与隐私研讨会（EuroS&PW）。IEEE，48–56。[32] Gary Mullen 和 Liam Meany。2019。对物联网操作系统中基于缓冲区溢出的攻击进行评估。载于 2019 年全球物联网峰会（GIoTS）。IEEE，1–6。[33] Jonah Napieralla。2020。探讨在硬件受限的物联网设备上使用 WebAssembly 容器进行边缘计算。[34] Jungmin Park 和 Akhilesh Tyagi。2017。利用功耗线索入侵物联网设备：功耗侧信道可用于指令级反汇编。《IEEE 消费电子杂志》6，3（2017），92–102。[35] Pangun Park、Sinem Coleri Ergen、Carlo Fischione、Chenyang Lu 和 Karl Henrik Johansson。2017。控制系统的无线网络设计：综述。《IEEE 通信综述与教程》20，2（2017），978–1013。[36] Gregor Peach、Runyu Pan、Zhuoyi Wu、Gabriel Parmer、Christopher Haster 和 Ludmila Cherkasova。2020。eWASM：用于可靠嵌入式设备的实用软件故障隔离。载于嵌入式软件国际会议（EMSOFT）。[37] Christo Petrov。2020。47 个令人惊叹的物联网统计数据 *2020 [*物联网的崛起*]*。https://techjury.net/blog/internet-of-things-statistics/#gref [38] Alexandru Radovici。 [无日期]。面向物联网的安全执行框架。（[无日期]）。[39] Ousmane Sadio、Ibrahima Ngom 和 Claude Lishou。2019。面向 MQTT/MQTT-SN 协议的轻量级安全方案。载于 2019 年第六届物联网：系统、管理与安全国际会议（IOTSMS）。IEEE，119–123。[40] Ayush Sahu 和 Ashima Singh。2016。使用基于 JavaScript 的沙箱保护物联网设备。载于 2016 年 IEEE 电子、信息与通信技术近期趋势国际会议（RTEICT）。IEEE，1476–1482。[41] 三星。2020。能力参考。https://smartthings.developer.samsung.com/docs/api-ref/capabilities.html [42] S Seleznev 和 V Yakovlev。2019。工业应用架构：物联网及 AMQP、MQTT、JMS、REST、CoAP、XMPP、DDS 协议。《开放信息技术国际期刊》7，5（2019），17–28。[43] Meena Singh、MA Rajan、VL Shivraj 和 P Balamuralidhar。2015。面向物联网的安全 MQTT。载于 2015 年第五届通信系统与网络技术国际会议。IEEE，746–751。[44] Hannes Sochor、Flavio Ferrarotti 和 Rudolf Ramler。2020。利用 MQTT-SN 发起分布式反射型拒绝服务攻击。载于数据库与专家系统应用国际会议。施普林格，74–81。[45] Andy Stanford-Clark 和 Hong Linh Truong。2013。传感器网络 MQTT（MQTT-SN）协议规范。国际商业机器公司（IBM），版本 1，2（2013）。[46] Marco Steger、Carlo A Boano、Kay Römer、Michael Karner、Joachim Hillebrand 和 Werner Rom。2017。Cesar：用于评估无线汽车软件更新效率的测试平台基础设施。载于第 20 届 ACM 无线与移动系统建模、分析与仿真国际会议论文集。311–315。[47] Deris Stiawan、Mohd Idris、Reza Firsandaya Malik、Siti Nurmaini、Nizar Alsharif、Rahmat Budiarto 等。2019。研究物联网网络中的暴力破解攻击模式。《电气与计算机工程杂志》2019（2019）。[48] Priceonomics 数据工作室。2020。物联网数据爆发：物联网数据市场规模有多大? https://priceonomics.com/the-iot-data-explosion-how-big-is-the-iot-data-market/ [49] Kenton Varda。2018。Cloudflare Workers 上的 WebAssemblyy。https://blog.cloudflare.com/webassembly-on-cloudflare-workers/ [50] Malvina Vega。2020。物联网统计数据、事实与预测。https://review42.com/internet-of-things-stats/ [51] Wenyong Huang.Xi Wang。2019。面向物联网设备的 Wasm 微运行时。https://github.com/bytecodealliance/wasm-micro-runtime [52] Wasm3。2019。最快的 WebAssembly 解释器。https://github.com/wasm3/wasm3[53] Webassembly。2020。WebAssembly 设计文档。https://github.com/WebAssembly/design [54] Elliott Wen 和 Gerald Weber。2020。Wasmachine：借助 WebAssembly 操作系统加速物联网。载于 2020 年 IEEE 普适计算与通信国际会议研讨会（PerCom 研讨会）。IEEE，1–4。[55] Christian Weyer。2020。浏览器之外的 WebAssembly：在 .NET Core 应用程序中使用 WASI 和 Wasmtime 运行 WASM。https://www.thinktecture.com/en/webassembly/with-dotnet/ [56] Liu Yang、Peng Yu、Wang Bailing、Bai Xuefeng、Yuan Xinling 和 Li Geng。2013。基于 IBE 与 PKI/CA 集成的物联网安全传输。《控制与自动化国际期刊》6，2（2013），245–254。