Local-First Smart Home Applications with HubOS

Igor Zavalyshyn
UCLouvain
Louvain-la-Neuve, Belgium

Axel Legay
UCLouvain
Louvain-la-Neuve, Belgium

Annanda Rath
SIRRIS
Brussels, Belgium

Etienne Rivière
UCLouvain
Louvain-la-Neuve, Belgium
etienne.riviere@uclouvain.be

***Abstract*—Many smart home applications collect sensitive data and interact with remote services. The local-first principle minimizes data propagation by favoring local processing and communicating with remote services only when necessary. HubOS is an operating system for smart home hubs supporting developers in the design and execution of privacy-conscious, local-first applications. HubOS applications can combine smart home computations and automation modules of diverse complexity, from simple trigger-action rules to CPU-intensive operations such as inference from machine-learned AI models. For the latter, it leverages WebAssembly, enabling portability and near-native performance. All HubOS application modules are strictly isolated and controlled in their access to resources and data. We contribute a novel Trigger-Action-Based Access Control model (TABAC) that allows context-sensitive and user-understandable control of access conditions. We demonstrate the interest of HubOS by detailing the implementation of three use case applications: a voice assistant, a security alarm based on face recognition, and a fall detector with remote video assistance for elderlies. Our evaluation shows that even for smart hubs with modest resources, HubOS applications have sufficient performance to contend with cloud-only alternatives.**

***Index Terms*—smart homes, privacy, local-first, WebAssembly.**

I. Introduction

Smart home applications are increasingly popular and diversified, expanding from home automation scenarios to advanced, AI-based applications. These applications all depend on the collection, storage, and processing of a wide variety of data from sensors, such as cameras, presence detectors, microphones, etc., and on the interaction with a diverse set of online services. Commercial systems for smart homes such as Samsung SmartThings [1], Google Home [2], or Amazon Alexa [3] follow a *cloud-only* model in which *all* data from smart home appliances is forwarded to the cloud for storage and processing. This model enables a simple development canvas for smart home applications but is intrinsically problematic for privacy. Smart home data may allow inferring information about the habits, home presence, and activities of homeowners and tenants [4]–[7]. Examples of privacy violations include leakages of voice assistants streams [8] or Amazon Ring [9] smart home security services disclosing private video streams to third parties [10].

We aim to support developing, deploying, and running smart home applications following the *local-first* principle [11], [12]. This model promotes local processing and using cloud services only when necessary, with minimum data exposure. In a smart home, the local-first model builds upon a *smart hub* device (e.g., a Raspberry PI or NUC) that connects smart appliances and sensors, offers a unified view of the system, and is the only access point to the Internet [13]. Our ambition is to enable the principled development of local-first applications that can later be installed and run by ordinary smart home users, with solid isolation guarantees and explicit control of data, devices, and network access.

Figure 1 presents three applications designed under the local-first principle. A1 is a smart security camera that can be a local-first replacement to the cloud-only Amazon Ring [9]. In A1, a single module accesses the video stream of a porch camera and runs a face detection algorithm against a database of known faces. Only when an unknown face is detected does a second module receive temporary access to a snapshot of the stream and is authorized to access an online notification service. A2 is a local-first alternative to Amazon Alexa [3]. Cloud-hosted, powerful voice recognition models cannot be deployed on a smart hub. Yet, many commands are simple enough to be recognized by a lighter model (e.g., "*switch on the lights*"). The first module hosts such a light model and

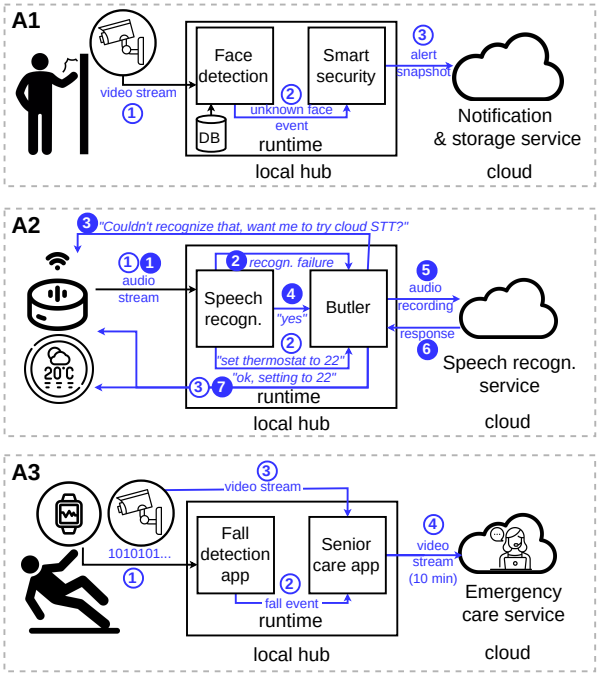


Fig. 1. Three local-first applications. **A1:** Smart security camera, **A2:** Voice-controlled butler, and **A3:** Fall detection and remote assistance for elderlies.

has full-time audio stream access. Upon detection of simple commands, appliances are activated directly. When detection fails, it asks the user (using voice synthesis) to send the recording to an online service. The second module (the *butler*) only gets access to the recording and the internet upon this consent. Finally, A3 supports fall detection, a critical application for the safety of older citizens wishing to live in the comfort of their homes [14]. A3 combines two modules developed by different companies. A first module gets access to privacy-sensitive data flows such as motion sensors, cameras, or wearables such as wristbands [15], but not to the internet. The detection technology provider typically develops this module. A second module gets temporary access to a video camera if and only if the first suspects a fall and connects to emergency services. This second module may be developed and commercialized by a local assistance company.

Enabling local-first smart home applications sets the following requirements. First, application modules must be strictly isolated. They should only access resources (smart devices and appliances, the network, storage, or communication with other modules) when authorized rather than by default (e.g., they should not be able to open network connections freely). Second, the conditions to enable (or disable) such accesses are by nature *context-sensitive*, i.e., they depend on the status of smart sensors but also the output of other application modules. Access control must allow the expression of such contextual conditions. Users must inspect and agree with these; hence, readability is a priority. Finally, modern smart home applications combine simple automation rules with CPU-intensive computation such as AI model inference. The programming model must support this variety.

Open-source systems for smart hubs such as OpenHab [16], Home Assistant [17], Domoticz [18], or HomeGenie [19] allow enthusiast, expert users to install and configure home automation workflows. These systems provide, however, no proper isolation between applications [20]–[22]: Everything runs under the permissions of a unique user on the local OS, without fine-grain restrictions to networking or device access. As such, these systems are unfit to deploy third-party-provided local-first applications.

Contributions and outline. We propose HubOS, an operating system for developing and deploying local-first smart home applications. The contributions of HubOS are a programming model for local-first applications, a runtime supporting this model, and the associated access control mechanisms.

HubOS assists developers in designing local-first applications as a collection of isolated modules. Some of these modules can be simple *trigger-action rules* as found in commercial and open-source smart home platforms. For complex actions, HubOS supports interpreted scripts (in Javascript) or code compiled to WebAssembly (from one of the many languages supporting WebAssembly as a target, e.g., C++, Java, or Rust). The latter is instrumental in supporting CPU-intensive workloads such as AI model inference. Modules communicate and consume sensory data exclusively via HubOS-provided flow and event channels. We present HubOS system services and programming model in Section II.

Access control in HubOS expresses a tradeoff between the *utility* of accessing resources and the risk for *privacy* of such access. Application modules must follow the principle of least privilege, and users must agree with access control rules upon installing them. To allow user-understandable and context-sensitive access control, HubOS contributes a novel access control model called TABAC, for *Trigger-Action-Based Access Control*. TABAC rules extend trigger-action rules to express access control, with simple "if this then that" rules that can be presented visually to users as part of the installation process. We detail TABAC in Section III.

We present the implementation of HubOS in Section IV. In Section V, we detail how we implemented the local-first applications of Figure 1. Our experimental evaluation in Section VI uses these three applications. We show that, while not designed to outperform powerful cloud services, local-first alternatives over HubOS represent valid contenders while intrinsically improving privacy. We finally review related work in Section VII and conclude in Section VIII.

II. HubOS programming model and runtime

HubOS targets a single-tenant smart home with several devices (sensors and actuators) and a small computer system, or *hub*, as the sole connection point for all devices [13]. We exclude direct communication between devices and the cloud.

HubOS promotes an alternative, local-first eco-system to cloud-only, commercial offerings [1]–[3]. While such systems can co-exist with HubOS, their closed nature prevents from deploying application modules with solid guarantees about isolation and data protection. Our focus is, therefore, on applications solely deployed on the HubOS-controlled hub, interacting with open sensors and appliances from the large variety supported by OpenHab [16] open-source drivers.

Security assumptions and objectives. The HubOS ecosystem consists of developers who wish to build privacy-conscious applications and users who select these applications for installation in their smart homes. Due to errors in the code or voluntary data misuse, users do not blindly trust developers' willingness to enforce the local-first principles. Developers must, therefore, strictly compartmentalize their applications and define access control that follows the *least-privilege* principle. HubOS provides the canvas to enable and enforce such compartmentalization and strict access control. Detecting malware attempting to bypass these controls, e.g., using side-channel attacks [23], is orthogonal to our objective. HubOS is complementary to solutions for detecting such behavior [24].

Modules. Applications must be split into independent *modules* subject to distinct access control rules. Modules with unrestricted access to smart home data are generally not expected to access local storage or the external network. In contrast, modules with networking access should only see a curated or limited version of this data, such as simple notifications from modules with full data access. In all cases, modules access resources only when authorized and solely through HubOS

APIs. This includes communication with other modules and with modules of co-located applications.

Programming model. Smart home applications have diverse needs in terms of computation, from simple reactions to sensory data in the smart home to stateful, CPU-intensive operations. In addition, programming models should be similar to those used by developers in commercial, cloud-only systems to foster the adoption of the local-first principle. HubOS offers three complementary choices, representing different tradeoffs between simplicity, expressiveness, and performance.

First, HubOS supports the de-facto standard mechanism for home automation of *trigger-action rules* (TAR), as present in cloud- and local-only systems [1], [16]–[19], [25]. TARs are sufficient for many smart home tasks, e.g., triggering lights based on motion sensors, and have the added value that they can be verified systematically [26]. A TAR engine supports their evaluation, not requiring spawning a specific user-space process. A TAR has the following syntax:

```
rule ⟨name⟩
IF ⟨trigger cond.⟩ [or ⟨trigger cond.⟩ [or ...]]
THEN ⟨action⟩ [and ⟨action⟩ [and ...⟨action⟩]
```

The TAR's `name` must clearly express the rules' purpose. A TAR is triggered by conditions (cond.) applying to environmental variables, i.e., readings of smart home sensors. These conditions can be an arbitrary boolean function, here in disjunctive normal form, with operators such as $<$, $\geq$, $=$, $\subset$, $\in$, etc. When conditions evaluate true, actions are performed, typically as calls to communication and I/O APIs.

Second, HubOS supports Javascript (Node.js), also a well-established programming language in smart home contexts, and a commonly used language in cloud-only solutions allowing scripting [27]–[29]. In contrast with TARs, JavaScript modules are supported by actual user-based processes.

Third, HubOS supports executing binary code compiled to WebAssembly [30], a low-level, byte-code execution runtime initially designed to address the limitations of JavaScript in web browsers. WebAssembly is hardware- and platform-independent and based on formally-defined semantics. It also aims to be language-independent, supporting compilation from multiple languages, such as C/C++, Rust, Java, or Go. Compiled WebAssembly code generally outperforms JavaScript in raw performance and energy efficiency [31], although there is room for optimization [32].

WebAssembly modules in HubOS are "embedded" in a JavaScript enclosing wrapper and interact with that module for access to HubOS's services. They support demanding computations such as AI-based voice or image recognition. WebAssembly modules have two additional advantages. First, they may allow adapting existing, legacy code of a microservice from a cloud backend to a module on the hub with relatively minor re-engineering. Second, in contrast with JavaScript, they avoid distributing source code at the client side, a feature that may matter for some application providers.

JavaScript or WebAssembly modules run in isolated Linux processes. We modify the JavaScript runtime to disable file system access, network access, and the importation of external modules (`npm`). As it does not need file-system isolation, HubOS chooses not to use Linux containers (e.g., Docker) but only process-level isolation, using directly the `cgroups` and `namespaces` Linux features. We do not use (lightweight) virtual machines as they require hardware capabilities that some target devices, e.g., Raspberry Pis, may not have.

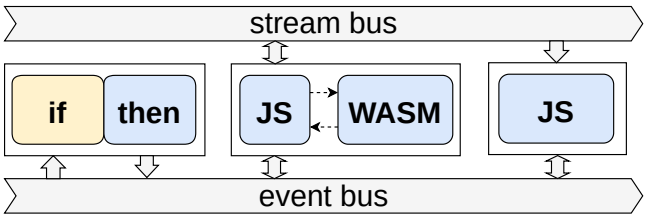


Fig. 2. Stream- and event-based communication between modules.

Support services. HubOS provides communication and storage services to application modules.

HubOS unifies inter-module communication and I/O using two complementary modes of publish/subscribe information exchange: instantaneous events and continuous data flows. For instance, a presence sensor may generate an event when a person passes in front while a camera or wearable generates a continuous stream. Events also play the role of triggers and enable inter-module workflow and synchronization.

All communication to and from modules uses one of the two buses illustrated by Figure 2. Events and streams are named after a human-understandable string, e.g., "*bedroom presence*" or "*porch camera*". The event bus allows modules to register a handler, which will be called upon a trigger event of the given name, as in serverless systems [33]. Events are the only possible trigger for TAR modules. JavaScript and WebAssembly modules may also run continuously.

HubOS does not give modules access to the local file system. JavaScript and WebAssembly modules can store state using HubOS embedded storage service. This service is a simple key/value store, but integrating other options, such as relational databases, would be straightforward. Each module is associated with a unique storage bucket, and buckets cannot be shared between modules.

III. Trigger-Action-Based Access Control

Access control in our local-first model enables users to decide when the utility of letting an application (or one of its modules) access sensory data, send commands to smart devices, or connect to the Internet outweighs potential risks to privacy. Each application is expected to declare in its manifest the accesses each of its modules needs for fulfilling its purposes and that the user must accept for the application to operate. All other accesses are disabled by default.

In existing smart home systems, access control is a binary decision: users allow or deny access to a given device or resource at application installation time [13]. Previous research has shown significant gaps in user perceptions of smart home devices and applications activities, resulting in poor access

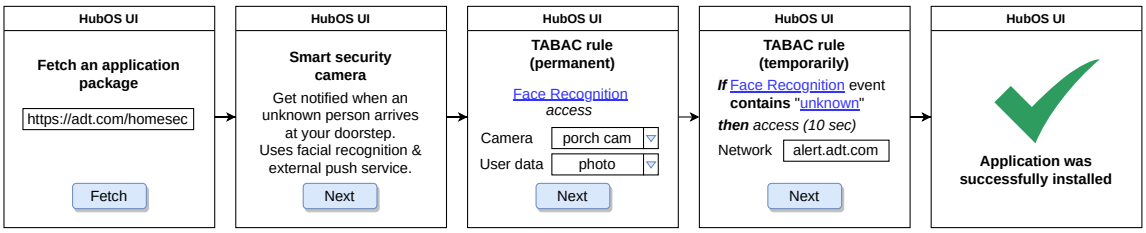


Fig. 3. A HubOS application installation workflow and consent collection.

control choices [34]. In addition, general allow/deny permissions fail to capture the *context* in which permission is granted, as noted for mobile applications [35], [36] It is preferable in smart homes to offer a clear and concise definition of temporary access context rather than an ambiguous and often misleading permanent access [37].

HubOS access control rules must be easy for humans to understand and rooted in a specific physical context. This can be a challenge, as contextual information may take multiple forms, such as a sensor reading a particular value, a signal from a smart device, or a specific output from one of the modules of a running application. Attribute-Based Access Control [38], in contrast with more traditional access control models such as RBAC (Rule-Based Access Control), allows the integration of elements of context. Still, existing implementations are complex to reason upon for users [39] in a single-tenant, smart home setting [39].

HubOS contributes a novel access control model that targets simplicity and understandability for users while retaining the ability to express context-sensitive access control rules. We observe that trigger-action rules (TAR) already fulfill many of our criteria: They follow a simple pattern of action/reaction, can be chained by reacting to the output of other application modules, and are easy to understand by humans [40], [41]

Building on TAR, we design the Trigger-Action-Based Access Control model (TABAC). The rationale behind TABAC is that TAR triggers can be used *as is* to authorize, respectively deny, access to specific resources and operating systems services. The trigger element of a TAR rule is intrinsically cyber-physical and dependent on context. As it can integrate elements (e.g., the emission of an event) linked with the output of applications' modules, it also allows this context to integrate computations' results. TABAC rules have the following syntax:

```
TABAC rule ⟨name⟩
IF ⟨trigger cond.⟩ [or ⟨trigger cond.⟩ [or ...]]
THEN
⟨allow/deny⟩ ⟨access⟩ ⟨resource⟩ ⟨period⟩ [and ...]
```

The first elements of a TABAC rule are the same as those of a TAR: a unique name and a boolean expression used as a trigger condition. A condition can be evaluated upon events published on the event bus by device drivers or application modules. Several environmental conditions (time of day, etc.) are also available. Instead of triggering *actions*, however, a TABAC rule can allow or deny *access* to a resource or service for a specific (optional) *period*.

The range of resources onto which TABAC rules apply in HubOS include (1) the access to the network, possibly restricted to a specific IP or DNS name; (2) the ability to subscribe to a stream or specific events or to produce a stream or events with a particular name; (3) the access to all or a subset of the API of the driver allowing to control a smart device; (4) the use of services such as scheduled triggers; (5) access to local storage. A TABAC rule can be attached to and applied to one or more application modules. When attached to a TAR module, a TABAC rule can prevent the rule from triggering if access to one of the necessary elements for its evaluation is denied. When attached to a JavaScript or WebAssembly module, a TABAC rule dictates how accesses to the HubOS service API are served. Network connections or stream subscriptions subject to a time limitation are automatically terminated at the end of the authorization period. If two rules conflict in allowing or denying access to a resource, HubOS conservatively favors the latter.

Consent collection. Installing a HubOS application requires the user to consent to TABAC rules associated with each module. Currently, HubOS presents TABAC rules as text. While installation time consent is permanent for a given TABAC rule, applications can request one-time consent to perform a specific action (e.g., send a voice command to a cloud when speech recognition fails in A2).

In the example of Figure 3, application A1 consists of two modules. The first implements face recognition logic and requires permanent access to a camera stream. The HubOS UI presents the TABAC rule for this module and asks the user to select the necessary camera device from the list. Then, the user is prompted with a TABAC rule for the second module. It specifies a face recognition event emitted by the first module as a trigger for temporary access (10 sec) to a remote notification and storage service. The rule triggers only if the event contains an *"unknown"* value, i.e., a person's face in front of the door is recognized that is not part of the model of known faces.

IV. HubOS Implementation

Figure 4 shows HubOS components and their interactions. HubOS builds upon GNU/Linux and OpenHab [16].

Connection to and from smart devices. Interaction with devices and sensors requires appropriate drivers. We take advantage of OpenHab's extensive support for diverse families of smart devices and reuse its driver components codebase (which includes networking support, e.g., via Z-Wave or Zigbee). We

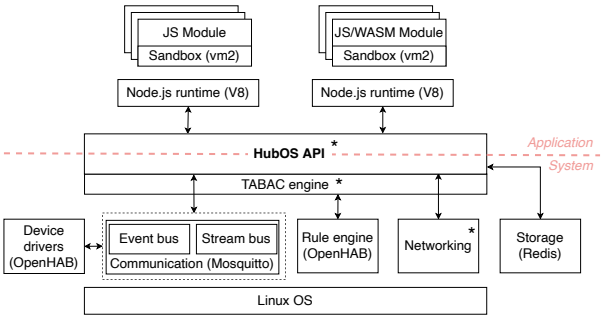


Fig. 4. HubOS's custom-built (marked with $*$) and OpenHab components.

modified drivers to use HubOS APIs, which required only minimal changes. For instance, video camera drivers share video streams via the stream bus, disallowing direct connection by application modules. Drivers also allow sending commands to devices via HubOS event bus.

Runtimes and isolation. We use the rule engine of OpenHab for TAR modules. TARs are evaluated when events, specified upon registration, are received from the event bus. Triggered rules may generate new events for internal use or for issuing commands to device drivers. For JavaScript and WebAssembly, we leverage Google's open-source V8 engine [42].

We isolate JavaScript modules (and, as a result, embedded WebAssembly modules) using Simek's `vm2` [43] sandbox library. We deny any external import of libraries, `eval()` calls, and file system access.

We implement resource isolation using Linux's `cgroup` and `namespaces`. Services run as a collection of Linux processes in the same namespace, while application modules run in isolated namespaces under resource control. Application modules have no network access and a limited share of CPU and RAM (according to the user-approved manifest).

TABAC orchestration and enforcement. Access control is enforced between modules and HubOS services at the API level. A TABAC engine maintains each module's current set of authorized or denied services and resource access. It extracts trigger components from new TABAC rules and replaces allow/deny components with actions that generate intermediate events to be shared via the event bus. Resulting TAR rules are registered with the OpenHab rule engine, requesting a re-evaluation for any event that appears in their trigger conditions. The TABAC engine subscribes to generated events and updates authorizations.

Communication. The event and stream buses use the Mosquitto publish/subscribe broker [44] (as used by OpenHab and implementing the MQTT protocol, a standard in smart environments). All communication between OpenHab and HubOS, between HubOS components, and between application modules happen through Mosquitto channels. A specific HubOS component handles external networking; communication with this component is the only possible way for the application module to contact external services.

Storage. We use Redis for local storage [45]. Modules authorized to use local storage have access to a JavaScript wrapper that accesses the official Node.js Redis client and filters out requests for keys outside the module's bucket.

```
### TABAC for smart security camera application (A1)
[{"if": {
  "event": "FaceRecognitionEvent",
  "operator": "contains",
  "value": ["unknown"]},
"then": [
  {"access":"Doorcam",
  "type": "device",
  "context":{"period":"30"}}]},
  {"access": "NetworkClient",
  "type": "service",
  "context": {"period": "30"}}]

### TABAC for voice-controlled butler application (A2)
[{"if": {
  "event": "SpeechRecognitionEvent",
  "operator": "containsAny",
  "value": ["set thermostat to",
            "make it warmer", ...]},
"then": [
  {"access":"Thermostat",
  "type": "device",
  "context":{"period":"5"}}]},
{"if": {
  "event": "SpeechRecognitionEvent",
  "operator": "equals",
  "value": [""]}, # unrecognized command
"then": [
  {"access": "NetworkClient",
  "type": "service",
  "context": {"period":"10",
              "allow":"speech.googleapis.com"}},
  {"access": "Thermostat",
  "type": "device",
  "context": {"period": "15"}}]}]

### TABAC for fall detection and
    assistance application (A3)
[{"if": {
  "event": "FallDetectionEvent",
 "operator": "equals",
  "value": [true]},
"then": [
  {"access": "NetworkClient",
  "type": "service",
  "context": {"period": "600"}},
  {"access":"LivingRoomCam",
  "type": "device",
  "context":{"period":"600"}}]}]
```

Listing 1. Example of TABAC rules in the three use cases.

V. Use cases implementation

We detail proofs-of-concept implementations of the three local-first applications presented in our introduction (Figure 1), using libraries and frameworks often used for cloud back-ends. Modules combine JavaScript and WebAssembly compiled from C++ using the Emscripten [46] compiler. The TABAC rules for the three applications are in Listing 1.

Application A1: Smart security camera. A1 uses two modules. The first module performs face recognition on an input camera feed using Human [47], a library for face recognition in TypeScript using the Tensorflow.js [48] wrapper. Its WebAssembly part is C++ Tensorflow with the MediaPipe BlazeFace face detection model [49] pre-trained using our visages. The second module sends a notification to a service running on a VM on AWS upon reception of a *FaceRecog-*

nitionEvent event from the first module. The TABAC rule on lines 2–12 authorizes access to networking and the camera stream only when the content of this event is "unknown".

Application A2: Voice-controlled butler. A2 also uses two modules. The first subscribes to a microphone audio stream. It uses Coqui-STT [50] and TensorFlow compiled from C++ to WebAssembly to detect a wake word and recognize speech. It can only emit *SpeechRecognitionEvent* containing a known command or a raw audio recording towards the second module (we leave the implementation of a user approval dialogue to future work). The second module is subject to the TABAC rule of lines 15–33. Depending on the trigger event, this rule grants access to the API of devices for actuation (lines 19–22) or the authorization to contact Google's Speech-to-Text API [51].

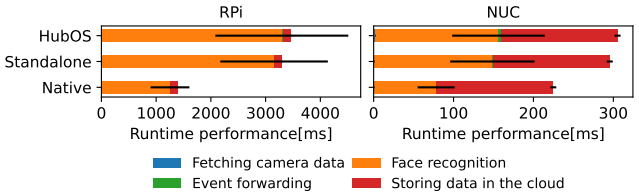
Application A3: Fall detection and remote assistance. A3 features two separate applications, with one module each. The first application's module subscribes to a video stream. We use the MoveNet *pose-estimation* model [52] as a basis for our fall detection script written in JavaScript, using the TensorFlow library as in previous use cases. Detecting a fall pose in a frame triggers a *FallDetectionEvent*, to which the second application's module subscribes. The TABAC rule on lines 36–46 provides access to the network and the camera feed for 10 minutes upon the reception of this event. We forward this stream to a sink service running on a VM on AWS.

VI. Evaluation

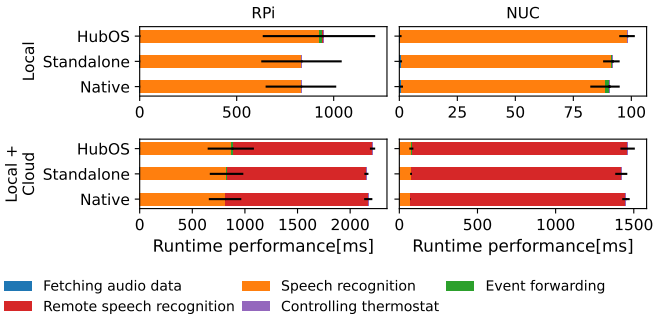
Our evaluation aims to answer two research questions: 1) Can local-first applications on resource-constrained smart hubs provide acceptable performance compared to cloud-based applications with unlimited data access? 2) What is the overhead of the HubOS runtime and isolation mechanisms?

In addition to the legacy "HubOS" applications described in the previous section, we consider three baselines. "Standalone" versions run on the hub and use WebAssembly but run modules non-isolated as standard processes with direct communication. They allow evaluating the impact of HubOS runtime support. "Native" versions further replace WebAssembly modules with non-portable native code, highlighting the impact of using portable WebAssembly. Standalone and native correspond to deploying the applications over a Linux box running OpenHab, as would a hobbyist. "Cloud-only" are native versions running in a VM in the cloud, for which the hub only acts as a data forwarder.

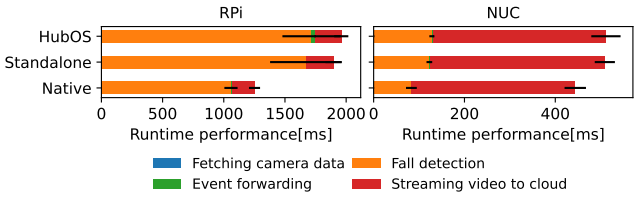
We use two platforms representing the two ends of the spectrum for smart home hub capabilities. The first is a Raspberry Pi 3B device (RPi) with a 4-core 1.2GHz ARM Cortex-A53 CPU, 1GB of RAM, and 32GB of slow SD storage. The second is an Intel mini-PC (NUC) with a 4-core 2.30GHz Intel mobile-class i5-8259U CPU, 16GB of RAM, and 1TB of fast SSD storage. We use a 4-vCPU VM on AWS with 16GB of RAM to host back-end cloud services (e.g., for receiving the video stream after a fall detection in A3) or Cloud-only instances. The smart hubs connect to the cloud using a consumer-grade cable connection, measured at a capacity of 9.88 Mbps for upload and 95.2 Mbps for download.



(a) A1 (smart security camera) operation breakdown on the two smart hubs.



(b) A2 (voice-controlled butler) operation breakdown. Top: voice command is recognized locally. Bottom: it needs to be sent to the cloud for recognition.



(c) A3 (fall detection) operation breakdown on the two smart hubs.

Fig. 5. Operation breakdowns for the local versions of each application.

Our primary evaluation metric is the execution latency of the three use case applications. We collect a complete latency breakdown of their workflows, as indicated by the numbers in Figure 1. We script human interactions and count time from the launch of the script. Because interactions with cloud services in A1 and A3 are asynchronous, only data transfer costs are included in the breakdown. For A3, calls to the Google Speech-to-Text API are part of the request-response loop. We use the following triggers. For A1, we use a stream of camera frames and start counting time when a frame featuring the face of an unknown person is provided. For A2, we use two voice command recordings: one that can be recognized locally and one that requires a speech-to-text cloud service. We count the time from when the complete recording is available for application processing to ignore user latency. A3 is similar to A1 except that we count time from the provision of the frame presenting a person lying on the floor. We report the time breakdown for every operation averaged over 20 runs, together with the standard deviation unless negligible.

Local-first applications. We compare the three application versions running on the two hubs (6 configurations per application). Figure 5 presents the operation breakdowns.

Figure 5(a) focuses on A1. The different steps of the

workflows are colored: fetching the frame from the stream bus, running face recognition, forwarding via the event bus to the second module, and sending the frame to the cloud. Face recognition dominates latency on the RPi, while data forwarding dominates on the NUC. There is about one order of magnitude latency difference between the two platforms. The RPi processes one frame every 4 seconds (with high variability due to slow SD storage). The NUC allows sub-second-frequency analysis. Native performs better, and the overhead of WebAssembly is more pronounced on the NUC. Compared to the standalone one, the HubOS version has a slight overhead for using HubOS runtime, but the difference (about 35 ms avg.) is small compared to the total running time. Overall, we observe viable performance for the considered application, even on the low-power system, but with a relatively significant impact of using WebAssembly.

Figure 5(b) considers two scenarios for A2. On top (Local), we consider a case where the voice command is recognized locally and triggers an API command to a smart device (a thermostat). On the bottom (Local+Cloud), the command needs processing in the cloud. As for A1, in the Local case, the processing time dominates on both platforms. In the Local+Cloud case, the sending time of the audio excerpt and its processing using Google's powerful model does. Processing commands locally is much faster, especially on the NUC, although the model capability cannot be compared to Google's. Overheads of HubOS compared to standalone is slightly higher than for A1: 11.5% (RPi) and 6.5% (NUC) additional latency overall. Calls to HubOS services (including those relaying device commands) have negligible latencies: we can only observe the communication between the first and second modules (in green) followed by the concise execution of the second module (in purple), all taking but a few milliseconds. In contrast with A1, the performance of AI inference is very close between standalone and native, illustrating the performance variability of WebAssembly [53].

We finally present the breakdown for A3 in Figure 5(c). The observations are similar as for A1: a slight overhead for using HubOS services compared to standalone, a faster execution in native than in standalone due to the overhead of WebAssembly, and forwarding latency dominating on the NUC when processing latency dominates on the RPi. Performing detection every few seconds is perfectly sufficient for this application, and both platforms can support it.

These experiments show that local-first applications are viable options for smart home applications while constrained by the limited hub resources compared to cloud services. HubOS overheads are negligible. WebAssembly has a more substantial impact, but we are hopeful the performance gap will shorten as the technology progresses.

Comparison to cloud-only. We compare local-first applications on the NUC to cloud-only counterparts. For A1 and A3, we use the same AI model locally and in the cloud. For cloud-only A2, we directly use Google's text-to-speech service and bypass the small model. Figure 6 compares the latency breakdowns between local-first applications on the

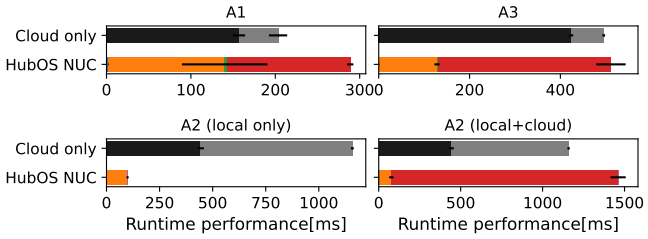


Fig. 6. *Local-first* vs. *cloud-only* mode. For cloud-only, the black portion represents the communication to the cloud; the gray portion represents the actual processing (locally for A1 and A3 and calling Google's speech-to-text service in A2). Other legends are shared with Figure 5.

NUC and cloud-only variants. For A1 and A3, computation time in the cloud (gray) is 33 to 55% lower than on the hub, as expected from the server-class hardware on EC2. This difference is acceptable given the gain in privacy. When weighing in the video sending time in A3, the local-first is close to cloud-only. In A2, the latency of using Google's model is higher than the local model, although again their capacities are not comparable. Sending unrecognizable commands from the hub to Google's service has slightly higher latencies than sending all commands indistinguishably to the cloud. These experiments show that the local-first approach compares favorably to a non-privacy-preserving, cloud-only approach regarding the tradeoff between performance and privacy (or lack thereof).

VII. Related work

Our survey comprehensively reviews research on smart home hubs [13]. Early work proposing centralizing smart home operations on a hub includes HomeOS [54] and BOSS [55]. These works focus on trigger-action rules and do not aim to support arbitrary computation. Open-source platforms cited in our introduction [16]–[19] follow such early efforts. They allow running arbitrary code but, in contrast to HubOS, provide no isolation [20]–[22] and do not support developers in designing compartmentalized, local-first applications.

Programming models for smart homes with a focus on data isolation include Homepad [56], dSpace [57], and PatrIoT [58] (the latter requiring a trusted execution environment). FlowFence [59]'s programming model separates between *quarantined* and ordinary modules. Taint-tracked *handles* allow to follow sensitive data propagation but enforce specific programming patterns. Adding taint tracking capabilities in HubOS without enforcing specific code patterns would be an interesting perspective.

SandTrap [60] improves the sandboxing and isolation of JavaScript in popular cloud-based TAP platforms such as IFTTT [25]. Similarly to HubOS, SandTrap leverages the Node.JS `vm` module and allows access-control policies.

The local-first model promotes disconnected operations and local storage in web applications [11]. Other examples of local-first approaches include Privacy capsules [61] for Android mobile applications, Privacy mediators [62] for IoT, and

Databox [63] for personal data management. HubOS complements these with specific support for smart home applications, such as context-aware access control.

Access control models taking contextual information into account include SecurehomeABAC [64], ABAC-CC [65], HABAC [66], and EGRBAC [67]. Based on ABAC (Attributed-Based Access Control), these models require handling a specific, pre-established taxonomy of attributes separate from the events used as triggers by applications. TABAC directly corresponds between the two and allows for the expression of contextual access rules that depend on applications' outcomes.

ContexIoT [68] adds guards to the source code of an IoT application, prompting the user upon actions requiring sensitive data. Prompting is impractical for smart homes and general user groups (e.g., elderly care in application A3). HubOS presents TABAC rules for agreement as part of the application installation.

WebAssembly [30], initially proposed for efficient code offloading in a Web browser, has gained momentum in several other contexts including the edge-cloud continuum [53], [69], [70], serverless platforms [33], embedded systems [71], decentralized services [72], and the IoT [69]. Wasmachine [69] is a IoT operating system merging WebAssembly modules with a minimalistic kernel in Rust for performance. ThingSpire OS [73], WiProg [74], and WAIT [75] are other examples of WebAssembly runtimes for the thing-to-edge continuum, implementing various performance optimizations and portability features. Unlike HubOS, however, these systems do not target multi-application settings or inter-module communication and do not focus on systems services and access control.

VIII. Conclusion

We presented HubOS, an operating system enabling the development, deployment, and execution of local-first smart home applications. This work opens several perspectives. First, HubOS could provide automatic snapshot-restore of local applications to encrypted storage in the cloud. Support cloud services could monitor the hub liveness and trigger reactions for sensitive applications such as A3. Second, compartmentalization in HubOS is for the moment binary (allow/deny). We wish to explore the integration of the systematic use of privacy-enhancing technologies into our programming model. This can include, e.g., face blurring in video flows or general data degradation such as differential privacy.

Acknowledgments

This work was supported by the Brussels Institute for Research and Innovation (Innoviris) under the "Smart and Social Home Care" research project. We thank the anonymous reviewers for their precious time and for their valuable suggestions and comments.

References

[1] "Samsung smartthings." [Online]. Available: https://www.smartthings.com/

[2] Google Home. [Online]. Available: https://developers.google.com/home

[3] Amazon Alexa. [Online]. Available: https://www.amazon.com/smart-home-devices

[4] J. S. Edu, J. M. Such, and G. Suarez-Tangil, "Smart home personal assistants: a security and privacy review," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, 2020.

[5] S. Dong, Z. Li, D. Tang, J. Chen, M. Sun, and K. Zhang, "Your smart home can't keep a secret: Towards automated fingerprinting of iot traffic," in *15th ACM Asia Conference on Computer and Communications Security*, ser. AsiaCCS, 2020, pp. 47–59.

[6] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec, 2020, pp. 207–218.

[7] R. Xu, Q. Zeng, L. Zhu, H. Chi, X. Du, and M. Guizani, "Privacy leakage in smart homes and its mitigation: IFTTT as a case study," *IEEE Access*, vol. 7, pp. 63 457–63 471, 2019.

[8] C. Gao, V. Chandrasekaran, K. Fawaz, and S. Banerjee, "Traversing the quagmire that is privacy in your smart home," in *Workshop on IoT Security and Privacy*, 2018.

[9] Amazon Ring. [Online]. Available: https://www.amazon.com/Ring/

[10] A. Agnihotri and S. Bhattacharya, "Ring: The new Amazon subsidiary and the social, privacy, and security issues it generates," in *SAGE Business Cases*. SAGE Publications, 2021.

[11] M. Kleppmann, A. Wiggins, P. Van Hardenberg, and M. McGranaghan, "Local-first software: you own your data, in spite of the cloud," in *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward!, 2019.

[12] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[13] I. Zavalyshyn, A. Legay, A. Rath, and E. Rivière, "SoK: Privacy-enhancing smart home hubs," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 4, 2022.

[14] K. Maswadi, N. B. A. Ghani, and S. B. Hamid, "Systematic literature review of smart home monitoring technologies based on IoT for the elderly," *IEEE Access*, vol. 8, 2020.

[15] P. Bet, P. C. Castro, and M. A. Ponti, "Fall detection and fall risk assessment in older person using wearable sensors: A systematic review," *International journal of medical informatics*, vol. 130, p. 103946, 2019.

[16] "openhab." [Online]. Available: https://www.openhab.org/

[17] "Home assistant." [Online]. Available: https://www.home-assistant.io/

[18] Domoticz: Home automation system. [Online]. Available: https://www.domoticz.com

[19] Home Genie: The open source, programmable, home automation server for smart connected devices and applications. [Online]. Available: https://homegenie.it

[20] Home Assistant Community Forum. [Online]. Available: https://community.home-assistant.io/t/add-on-permissions-system-coming-soon/282544

[21] OpenHAB Community Forum. [Online]. Available: https://community.openhab.org/t/openhab-sudo-exec-binding/34988

[22] Domoticz Wiki. [Online]. Available: https://www.domoticz.com/wiki/Scripting_in_Domoticz

[23] Z. Wang, D. Liu, Y. Sun, X. Pang, P. Sun, F. Lin, J. C. Lui, and K. Ren, "A survey on IoT-enabled home automation systems: Attacks and defenses," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2292–2328, 2022.

[24] A. K. Sikder, L. Babun, and A. S. Uluagac, "Aegis+ a context-aware platform-independent security framework for smart home systems," *Digital Threats: Research and Practice*, vol. 2, no. 1, pp. 1–33, 2021.

[25] "Ifttt." [Online]. Available: https://ifttt.com

[26] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering end users in debugging trigger-action rules," in *ACM Conference on Human Factors in Computing Systems*, ser. CHI, 2019.

[27] Google, "Samples and libraries for actions on google." [Online]. Available: https://github.com/actions-on-google

[28] Amazon, "Alexa ask sdk for node.js." [Online]. Available: https://developer.amazon.com/en-US/docs/alexa/alexa-skills-kit-sdk-for-nodejs/overview.html

[29] ——, "Alexa samples collection." [Online]. Available: https://github.com/orgs/alexa-samples

[30] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," in *38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI, 2017.

[31] J. De Macedo, R. Abreu, R. Pereira, and J. Saraiva, "Webassembly versus javascript: Energy and runtime performance," in *International Conference on ICT for Sustainability*, ser. ICT4S. IEEE, 2022.

[32] A. Jangda, B. Powers, E. D. Berger, and A. Guha, "Not so fast: Analyzing the performance of WebAssembly vs. native code," in *USENIX Annual Technical Conference*, ser. ATC, 2019.

[33] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in *USENIX Annual Technical Conference*, ser. ATC, 2020.

[34] S. Zheng, N. Apthorpe, M. Chetty, and N. Feamster, "User perceptions of smart home iot privacy," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–20, 2018.

[35] B. Shebaro, O. Oluwatimi, and E. Bertino, "Context-based access control systems for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 150–163, 2014.

[36] O. Oluwatimi, D. Midi, and E. Bertino, "A context-aware system to secure enterprise content," in *21st ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT, 2016.

[37] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE symposium on security and privacy*, ser. S&P. IEEE, 2016.

[38] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[39] D. Servos and S. L. Osborn, "Current research and open problems in attribute-based access control," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–45, 2017.

[40] L. Shi and D. W. Chadwick, "A controlled natural language interface for authoring access control policies," in *ACM Symposium on Applied Computing*, ser. SAC, 2011.

[41] S. Goffinet, D. Schmitz, I. Zavalyshyn, A. Legay, and E. Riviere, "Controlling security rules using natural dialogue: an application to smart home care," in *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*, 2021.

[42] Google, "V8 JavaScript and WebAssembly engine." [Online]. Available: https://v8.dev

[43] P. Simek, "VM2: a sandbox that can run untrusted code with whitelisted node's built-in modules." [Online]. Available: https://github.com/patriksimek/vm2

[44] E. Foundation, "Eclipse mosquitto: open source (epl/edl licensed) message broker that implements the mqtt protocol." [Online]. Available: https://mosquitto.org

[45] "Redis: open source, in-memory data store." [Online]. Available: https://redis.io

[46] "Emscripten: a complete compiler toolchain to webassembly, using llvm, with a special focus on speed, size, and the web platform." [Online]. Available: https://emscripten.org

[47] V. Mandic. . [Online]. Available: https://github.com/vladmandic/human

[48] Tensorflow.js. [Online]. Available: https://github.com/tensorflow/tfjs

[49] Mediapipe. [Online]. Available: https://mediapipe.dev/

[50] Coqui-STT. [Online]. Available: https://github.com/coqui-ai/STT

[51] Google Speech-To-Text. [Online]. Available: https://cloud.google.com/speech-to-text/

[52] G. Research. Next-generation pose detection with movenet and tensorflow.js. [Online]. Available: https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html

[53] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, "Webassembly as a common layer for the cloud-edge continuum," in *2nd Workshop on Flexible Resource and Application Management on the Edge*, ser. FRAME, 2022.

[54] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *9th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI, 2012.

[55] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, "BOSS: Building operating system services," in *10th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI, 2013.

[56] I. Zavalyshyn, N. O. Duarte, and N. Santos, "Homepad: A privacy-aware smart hub for home environments," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 58–73.

[57] S. Fu and S. Ratnasamy, "dSpace: Composable abstractions for Smart Spaces," in *28th ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP, 2021.

[58] I. Zavalyshyn, N. Santos, R. Sadre, and A. Legay, "My house, my rules: A private-by-design smart home platform," in *MobiQuitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2020, pp. 273–282.

[59] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical data protection for emerging IoT application frameworks," in *25th USENIX Security symposium*, 2016, pp. 531–548.

[60] M. M. Ahmadpanah, D. Hedin, M. Balliu, L. E. Olsson, and A. Sabelfeld, "SandTrap: Securing JavaScript-driven Trigger-Action Platforms," in *30th USENIX Security Symposium*, 2021.

[61] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee, "Privacy capsules: Preventing information leaks by mobile apps," in *14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys, 2016.

[62] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy mediators: Helping IoT cross the chasm," in *17th international workshop on mobile computing systems and applications*, 2016, pp. 39–44.

[63] R. Mortier, J. Zhao, J. Crowcroft, L. Wang, Q. Li, H. Haddadi, Y. Amar, A. Crabtree, J. Colley, T. Lodge *et al.*, "Personal data management with the databox: What's inside the box?" in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, 2016, pp. 49–54.

[64] B. Bezawada, K. Haefner, and I. Ray, "Securing home iot environments with attribute-based access control," in *3rd ACM Workshop on Attribute-Based Access Control*, ser. ABAC, 2018.

[65] S. Bhatt and R. Sandhu, "ABAC-CC: Attribute-based access control and communication control for internet of things," in *25th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT, 2020.

[66] S. Ameer, J. Benson, and R. S. Sandhu, "An attribute-based approach toward a secured smart-home iot access control and a comparison with a role-based approach," *Information*, vol. 13, no. 2, p. 60, 2022.

[67] S. Ameer and R. Sandhu, "The habac model for smart home iot and comparison to egrbac," in *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, 2021, p. 39–48.

[68] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards providing contextual integrity to appified iot platforms," in *Network and Distributed System Security Symposium*, ser. NDSS, 2017.

[69] E. Wen and G. Weber, "WasMachine: Bring the edge up to speed with a WebAssembly OS," in *13th International Conference on Cloud Computing (CLOUD)*, ser. CLOUD. IEEE, 2020.

[70] N. Mäkitalo, T. Mikkonen, C. Pautasso, V. Bankowski, P. Daubaris, R. Mikkola, and O. Beletski, "Webassembly modules as lightweight containers for liquid IoT applications," in *International Conference on Web Engineering*, ser. ICWE, 2021.

[71] F. Scheidl, "Valent-blocks: Scalable high-performance compilation of webassembly bytecode for embedded systems," in *International Conference on Computing, Electronics & Communications Engineering*, ser. iCCECE. IEEE, 2020.

[72] Z. Zhang, M. Król, A. Sonnino, L. Zhang, and E. Rivière, "EL PASSO: efficient and lightweight privacy-preserving single sign on," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 2, pp. 70–87, 2021.

[73] B. Li, H. Fan, Y. Gao, and W. Dong, "ThingSpire OS: a WebAssembly-based IoT operating system for cloud-edge integration," in *19th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys, 2021.

[74] B. Li, W. Dong, and Y. Gao, "WiProg: A webassembly-based approach to integrated iot programming," in *IEEE Conference on Computer Communications*, ser. INFOCOM, 2021.

[75] B. Li, H. Fan, Y. Gao, and W. Dong, "Bringing WebAssembly to resource-constrained IoT devices for seamless device-cloud integration," in *20th Annual International Conference on Mobile Systems, Applications and Services*, ser. MobiSys, 2022.