ABSTRACT
Availability of authentication and authorization services is critical for the safety of the Internet of Things (IoT). By leveraging an emerging network architecture based on edge computers, IoT’s availability can be protected even under situations such as network failures or denial-of-service (DoS) attacks. However, little has been explored for the issue of sustaining availability even when edge computers fail. In this paper, we propose an architectural mechanism for enhancing the availability of the authorization infrastructure for the IoT. The proposed approach leverages a technique called secure migration, which allows IoT devices to migrate to other local authorization entities served in trusted edge computers when their authorization entity becomes unavailable. Specifically, we point out necessary considerations for planning secure migration and present automated migration policy construction and protocols for preparing and executing the migration. The effectiveness of our approach is illustrated using a concrete application of smart buildings and network simulation, where our proposed solution achieves significantly higher availability in case of failures in some of the authorization entities.

1 INTRODUCTION
Authentication and authorization play key roles in ensuring safe and secure operations of the Internet-of-Things (IoT) devices in an open, potentially malicious environment. In the context of the IoT, most of existing approaches to providing authorization and authentication rely on remote and centralized cloud servers to provide these services. This means, however, that any device relying on these critical services may be adversely affected by a failure or an attack on one or more of the servers. An example illustrating this risk is the recent Google OnHub incident [14], where a failure in the company’s authentication servers caused IoT gateway devices (called OnHub) to become unavailable, in turn leading to failures in all IoT devices connected to these gateway devices.

An alternative system architecture, in part designed to reduce this dependency on remote, centralized servers, is called edge computing [21] or fog computing [11]. In this approach, individual computing devices called edge computers serve as an Internet gateway to local, neighboring devices; these edge computers may be deployed as a wide range of devices, including smart home routers, laptops, or smart phones. Figure 1 illustrates different characteristics of the cloud servers, edge computers, and IoT devices. As pointed out by Lopez et al. [6], benefits of adopting edge computing include better privacy, lower latency for real-time applications, less dependency on cloud servers and its connections, and better context awareness and manageability of the local systems.

Previously, we proposed an authorization infrastructure of the IoT called Secure Swarm Toolkit (SST) [8]. SST leverages a network architecture with special edge computers called Auths, which provide authorization services to local IoT devices. In this prior work, we showed how SST can provide strong confidentiality and integrity guarantees for communication among IoT devices.

In this paper, building on our prior work, we propose a novel approach for providing resilience against availability attacks or...
failures on an IoT network—e.g., a denial-of-service (DoS) attack designed to render critical services such as authorization unavailable to IoT devices. In particular, our approach leverages a technique called secure migration, which allows IoT devices to continue to receive critical services even when some of the Auth devices become unavailable due to an attack or other failures.

The contributions of this paper are as follows:

- An innovative architectural mechanism for building IoT services that are resilient against availability attacks and other failures, including a secure migration technique which allows IoT entities to migrate to trusted authorization entities to continue IoT services when some of the authorization entities are unavailable.
- A novel approach for finding effective migration policy by leveraging a Integer Linear Programming (ILP) solver and considering access requirements and constraints of IoT networks (Section 3).
- Experiments based on a concrete scenario in smart buildings, demonstrating the effectiveness of our approach against availability attacks (Section 4).

## 2 Problem and Context

IoT devices are heterogeneous, in that they vary over the level of safety and security requirements as well as available computational resources and energy capacity. For example, a smart door lock that authenticates users with their cellphones may rely on secure communication with strong encryption, whereas temperature sensors in the same building may not require the same level of security.

Important parts of an IoT infrastructure are special devices called gateways. A gateway facilitates communication between IoT devices by providing a variety of services such as authentication, authorization, device discovery, and data aggregation. Depending on the design of the infrastructure, a gateway may be deployed as a cloud-based server or as a local network entity (e.g., a router) to which IoT devices can connect to.

### 2.1 Threat Model

We assume the existence of an active network attacker with an ability to discover and send packets to gateway devices on the Internet. In particular, the attacker seeks to disrupt the availability of IoT devices connected to a gateway by carrying out various denial-of-service (DoS) attacks on it, such as flooding or distributed denial-of-service (DDoS) attacks. In our model, we primarily focus on availability attacks and failures, instead of confidentiality or integrity attacks (addressed by our previous work, SST [8]).

In addition, we exclude insider attacks, i.e., malicious acts by an individual who has physical or network access to IoT and gateway devices on the local area network. For instance, we assume that a local gateway placed inside a building is protected from physical tamperings, and that local network users are trustworthy. This allows us to assume that local gateways can be leveraged to perform critical steps in mitigating against an availability attack.

### 2.2 Proposed Architectural Mechanism

The key distinguishing feature of SST is a locally centralized, globally distributed architecture [9]: It consists of a set of local authorization entities called Auths [10], each of which is deployed on edge computers. An Auth enforces access policies and provides security services to local IoT devices (entities). Figure 2 (a) shows an example of a small IoT network in normal operation using SST, with two Auths ($A_1$ and $A_2$) and 6 entities ($t_1$ through $t_6$). If $A_1$ in the current SST design fails due to a DoS attack, authorization services for registered entities will become unavailable as shown in Figure 2 (b). This will also affect availability of other entities. For example, communication between $t_4$ and $t_5$ will be disrupted.

However, this does not fully utilize the globally distributed architecture of SST. Our goal is to provide enhanced availability of IoT authentication and authorization services through a mechanism that leverages SST’s globally distributed architecture. The architectural mechanism proposed in this paper includes backing up communication between/authenticity and migrating IoT entities to continue the IoT services even in case of Auth failures as shown in Figure 2 (c). The proposed mechanism is detailed in Section 3.

## 3 Approach

We propose an approach called a secure migration technique for restoring the availability of IoT devices during failures in some Auths by exploiting the architectural characteristics of SST. In particular, we accomplish this goal by having Auths take over other Auths’ authorization tasks when the latter group becomes unavailable due to an attack. Before any failure occurs in Auths, each Auth backs up information for authorization services to other trusted Auths and securely migrates IoT entities to continue the IoT services even in case of Auth failures as depicted in Figure 2 (c) so that they can continue to use authorization services.

### 3.1 Considerations for Migration Policies

Although the high-level idea behind migration may appear straightforward, determining migration policies is a non-trivial problem due to a number of factors that need to be considered:

**Auth trust relationships:** SST introduces an explicit notion of trust between a pair of Auths, and enables an entity from Auth $A$ to communicate to an entity in another Auth $A'$ if and only if $A$ and $A'$ trust each other. This communication constraint imposed by trust must be maintained during migration while satisfying individual entities’ requirements. Consider Figure 3 (a), where entities $t_4$ and $t_5$ are required to communicate with each other. Suppose that $A_1$ becomes unavailable. If we arbitrarily decide to migrate $t_4$ to $A_2$, then $t_4$ will no longer be allowed to communicate with $t_5$, since...
When an Auth becomes unavailable due to an attack, we say that the Auth is located in a restricted-access area for extra security requirements of high-criticality entities. A migration policy may involve migrating a trusted Auth to a neighboring Auth. Note that the trust relationship among Auths is not necessarily transitive.

Managing Auth’s workload for authorization: Let us consider a different scenario where entities who have lost connection with their original Auths are placed within the signal ranges of both Auths, thus they can migrate to either of them. An arbitrary migration decision may lead to a situation where all entities will have migrated to only one Auth. However, this type of decision may adversely affect the performance of the overall network, depending on the operational conditions of Auths. Each Auth may have an inherent limit in its resource capacity to perform authorization tasks for entities, for example, and may be required to run other computationally-intensive tasks, or it may be running on a battery-powered edge computer with a limited energy budget. Therefore, assigning entities within a signal range to another Auth may lead to a greater performance for the overall network.

Characteristics of Auth: Another important factor to be considered in a migration policy is the level of security guaranteed by an Auth, compared with those required by an entity. These include whether the Auth is equipped with hardware security support such as a TPM (Trusted Platform Module) or Intel’s SGX (Software Guard Extensions) and whether the Auth is located in a restricted-access area for extra physical security.

3.2 Policy Construction

When an Auth becomes unavailable due to an attack, we say that entities belonging to that Auth are dangling. A migration policy is an assignment of each dangling entity to one of the remaining Auths in the network. As discussed in Section 3.1, a valid policy must ensure that the new network resulting from migration satisfies a number of hard constraints (e.g., trust relationships).

We formulate the problem of constructing a valid migration policy as a problem in Integer Linear Programming (ILP). An ILP problem has two parts: a set of linear terms (each term being a product of an integer variable by a coefficient; e.g., c_i x_i for some variable x_i) whose sum is to be maximized, and a set of linear constraints over those variables. In our formulation, variables are Boolean values indicating whether an entity is connected to an Auth (thus, there are in total E × A number of variables, where E and A are the total numbers of entities and Auths in the network, respectively).

Due to limited space, we omit a detailed formalization of the ILP problem, and instead briefly describe the set of constraints that must be satisfied by a solution:

- Each entity can have security requirements that must be provided by an Auth. These requirements include hardware support (e.g., TPM/SGX), restricted physical location, cryptography specification (e.g., ephemeral keying for perfect forward secrecy).
- Each Auth has a threshold for authorization tasks in terms of upper bounds for authorization requests per minute or upper bounds for session keys cached in Auth. In other words, there is a threshold of registered entities for each Auth. When there are more registered entities than this threshold the Auth will experience degradation in authorization.
- Entity-to-entity communication requirements. Some entities need to be authorized to communicate with certain entities even after migration. This requires trust relationship between Auths with which those entities are registered.
- There exist multiple criticality levels for the communication requirements among entities, yielding a mixed-criticality system [4]. This means we should prioritize communication requirements of high-criticality entities.

Another aspect of policy construction is that of optimization; that is, generating a migration policy that results in a network with a minimal amount of overall communication costs between Auths and things. For instance, communication costs between a pair of nodes may depend on a number of factors, such as the physical distance between them and the energy required to perform cryptographic operations. Our formulation of policy construction as an ILP is intended to enable optimization of migration policies as well. This task, however, requires a detailed modeling of communication and energy costs, which is beyond the scope of this paper.

For our current implementation of SST, we use Gurobi [7] as the underlying ILP solver.

3.3 Secure Migration Procedure

In SST, each entity is authorized by an Auth by receiving a session key, which is a temporary symmetric cryptographic key for accessing a certain service or communicating with another entity (or entities). Session keys need to be confidential; they must be only known to the entities participating in a certain access activity. Thus, these session keys must always be encrypted with another symmetric key called a distribution key shared between Auth and each entity. A distribution key can be updated using public-key cryptography. For this, Auth and the registered entity should have already exchanged public keys during the registration (initialization) process of SST. For secure migration, Auth must prepare the migration before it
fails. When a failure occurs, Auth’s registered entities should detect the failure and migrate to trusted Auths. The approach described here is implemented as part of SST and available in SST’s open-source repository (https://github.com/iotauth).

3.3.1 Preparing Migration. Before a failure occurs, Auth should provide both its registered entities and the trusted Auths with information for migration. We call this operation a backup operation, described in Figure 4 (a). For its registered entity $t_1$, Auth $A_1$ gives out a list of its trusted Auths ($A_2$ and $A_3$) and their host names (IP addresses) and port numbers. This is first given during the entity registration and then updated when the information changes. Then, $A_1$ constructs the migration policies for its registered entities, either by itself or by receiving migration policies from other trusted Auths, using the method shown in Section 3.2.

With the constructed migration policies, $A_1$ prepares the migration token for each of its registered entities. A migration token must be able to establish the trust relationship between entities and the new Auth, $A_3$, to which $t_1$ will migrate in case of $A_1$’s failure. $A_1$ includes the name (a unique identifier in string) and the public key of $t_1$ so that $A_3$ can trust $t_1$ when it migrates to $A_3$, as shown in Figure 4 (a)-(1). $A_1$ also issues a migration certificate (cert) shown in Figure 4 (a)-(2) for $A_3$. This includes $A_3$’s public key signed by $A_1$’s private key and this will be verified by $t_1$ which already has $A_1$’s public key.

3.3.2 Detection and Migration. When Auth $A_1$ fails, its registered entity $t_1$ will detect that $A_1$ is not reachable due to a failure or an attack and will try to migrate to $A_3$’s trusted Auths. When $A_1$ does not respond for more than a threshold times, $t_1$ starts a migration operation shown in Figure 4 (b). Since $A_1$ had provided $t_1$ with a list of $A_1$’s trusted Auths which were $A_2$ and $A_3$, $t_1$ tries these Auths in the order specified in the list. Thus, $t_1$ sends MIGRATION_REQ to $A_2$, an Auth trusted by $A_3$, but not the one that $t_1$ is supposed to migrate to. Since $A_2$ does not have the required credentials for $t_1$, $A_2$ sends a response, MIGRATION_RESP indicating that the request is rejected and $t_1$ should try another trusted Auth in its list. After receiving MIGRATION_RESP (rejected) from $A_2$, $t_1$ tries the next Auth in its list, $A_3$. When $A_3$ receives MIGRATION_REQ from $t_1$, it notices that $t_1$ can migrate to it and responds with a MIGRATION_RESP message indicating that the request has been accepted.

A MIGRATION_REQ message includes the entity’s name and a verification token. This verification token includes a digital signature of the entity if it uses public-key cryptography for authorization. The Auth that the entity is supposed to migrate to will be able to verify the MIGRATION_REQ with the entity’s public key. This is because the Auth should have been backed up with the entity’s public key from the Auth that the entity was previously registered with. The accepted MIGRATION_RESP contains the certificate of the $A_3$ signed by $A_1$. The whole MIGRATION_RESP should be authenticated by the Auth’s private key, so that the entity can verify the MIGRATION_RESP message upon receiving it. When the migration request and response are successful, both the Auth and entity update the counterpart’s credentials for further authorization.

4 EXPERIMENTS AND RESULTS

In this section, we carry out experiments to demonstrate the effectiveness of our migration approach for maintaining availability. As an experimental scenario, we take a door controller and door opening application in a smart building. This is inspired by a prototype door controller deployed on the 5th floor of Cory Hall at UC Berkeley. We assume a virtual environment where the door controllers are deployed on currently card-key accessed doors, door opening mobile phone apps run on user smartphones, and Auths are deployed on some of the existing WiFi access points. Figure 5 illustrates this virtual environment. We also assume Auths trust each other only if their research centers have trust relationships, and the Things (door controllers and user smart phones) are registered with Auth as shown in Figure 5. We measure availability
We ran simulations with 4 Auths, 19 door controllers, and 26 user WiFi connections and LXCs for IoT entities have WiFi connections. As a simulation platform, we use Ubuntu Linux 16.04.2 LTS. LXCs on which Auths are running have both the wired and wireless interfaces and are connected to the network simulated in ns-3. The LXCs’ virtual Ethernet interfaces are connected to the wireless network interfaces of Auths and IoT entities using the proposed approach in Section 3.2 (denoted as “ILP-Based Migration” in the results).

As shown in Figure 7 (a), when one Auth fails the availability drops down to 69% without any migration, while the naive migration and ILP-based migration policies recover the availability up to 92% and 100%, respectively. The naive migration policy could not recover 100% availability due to the fact that it did not appropriately consider communication requirements among IoT entities together with trust relationships between Auths. In fact, with the naive migration policy, some clients ended up migrating to Auths that do not have trust relationships with the Auths for the servers with which the clients were supposed to communicate.

In Figure 7 (b), we can see that the availability drops down to 50% without any migration and fluctuates around 58% with the naive solution, while the proposed ILP-based solution recovers 81% of availability. Figure 7 (c) shows decreased availability around 27% with no or the naive migration policy and recovered availability of 54% with the proposed technique. The fluctuation with the naive migration policy was mainly due to the interference caused by infeasible migration requests and unbalanced workload among functioning Auths. The proposed ILP-based solution was not able to achieve 100% availability because some entities were out of the signal range of Auths. However, the proposed solution still achieved significantly higher availability compared to the other two cases.

5 RELATED WORK

We are aware of only a couple of other approaches that rely on an edge computing architecture for IoT security. In order to provide robust authorization of medical IoT devices without dependency on remote servers, SEAuthentication and Authorization Architecture) [13] uses distributed smart e-health gateways as local authorization centers based on DTLS (Datagram TLS). TACIoT (Trust-aware Access Control for the IoT) [3] employs an architecture based on entities called IoT bubbles, which act as local units of authorization for the IoT, similar to Auths. As far as we know, while these approaches ensure confidentiality and integrity of communication among IoT devices, they do not provide resilience against availability attacks.

Beside the two mentioned in the previous paragraph, a number of other authorization and authentication services for IoT devices have been proposed [1, 5, 15, 22, 24]. However, all of these approaches rely on remote, cloud servers to provide the critical security services. We believe that the local, distributed nature of Auths enables a more lightweight, flexible defense against availability attacks: Our
Figure 7: Availability results with different numbers of failing Auths (a) 1 Auth fails (b) 2 Auths fail (c) 3 Auths fail, for three different migration policies using SST simulated on the ns-3 simulator and Linux containers

approach allows an Auth to continue providing critical services to its local network without requiring a connection to the Internet. There exists a large body of work that address DoS attacks in the network security literature (a comprehensive survey can be found in [26]). These works can be categorized as (1) preventing an DoS attack before it takes place [12, 23, 27], (2) limiting the effect of an on-going attack by filtering malicious connections [16, 20], or (3) performing forensics after an attack to identify compromised IoT devices [17, 18, 25]. As far as we know, none of these approaches involve providing resilience against a DoS attack through migration of critical services from an attacked node to another.

6 CONCLUSIONS

Availability of IoT services can be critical for safety. By leveraging emerging network architecture based on edge computing and SST’s distributed authorization infrastructure, the proposed approach achieves much higher availability even under failures of local authorization entities running on edge computers. Our secure migration approach will be appropriate especially for the Internet of Things under safety-critical environments including medical centers, manufacturing systems, and electric power grids, so that we can maintain as much availability as possible.

Moving forward, we plan to investigate the use of an ILP solver to construct a migration policy that is not only valid but also optimal with respect to the overall network costs. We also plan to carry out larger scale experiments to further evaluate the effectiveness and limitations of our migration mechanism. In addition, we plan to develop a mathematical model of an attacker, and explore possible relationships between the knowledge and capability of the attacker and the guarantees provided by our mitigation mechanism.

ACKNOWLEDGMENTS

This work was supported in part by TerraSwarm, one of six centers of STARNet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

REFERENCES