IoTSafe: A Safe & Verified Security Controller for Internet-of-Things

Tianlong Yu
Carnegie Mellon University

1 Introduction

The Internet-of-Things (IoT) has quickly moved from hype to reality. Like other disruptive technologies, such as smartphones and cloud computing, IoT has the potential for a societal scale impact by transforming our daily lives. Major companies have already started to pioneer this transformation by introducing a range of products in the areas of smart homes, smart cities, and health care.

While IoT has huge potential, it is also threatened by diverse security threats. Since IoT devices are typically embedded deep inside networks, they are attractive attack targets and may become the "weakest link" for breaking into a secure IT infrastructure [7], or leaking sensitive information about users and their behaviors [8]. In addition, IoT devices have already been used as bots to launch DDoS or spams [1].

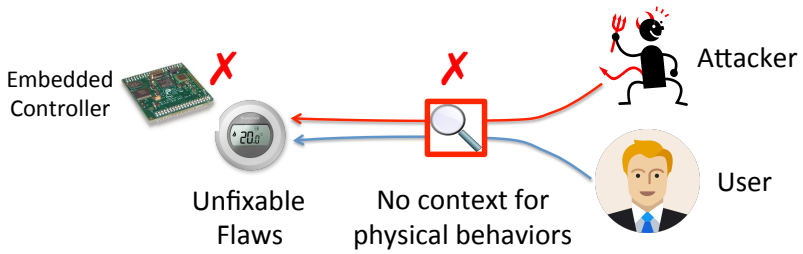

Figure 1: Current embedded controller or security appliances can not secure IoT.

The uniqueness of IoT is posing new challenges to the control and management:

- **Unfixable flaws:** IoT devices tend to have unfixable security flaws for two fundamental reasons. First, there are no host-based defenses (e.g., antivirus) due to resource constraints on these devices (e.g., limited RAM). Second, unlike traditional IT devices, IoT devices lack effective

automated software updates. The current process of patching IoT vulnerabilities is via manual firmware updates, and software updates will likely be unavailable (e.g., a vendor may no longer support updates for an old device). Therefore, these IoT devices can be easily compromised by attacker and the hardware controller on the IoT device itself cannot ensure the device behaves in a safe way.

- **Complex physical behaviors**: IoT devices has a significant physical impact on the environment. For example, the temperature of a baby room is kept in a safe range 20 centigrade to 24 centigrade by a thermostat, and an attacker can compromise the thermostat to overheat the room and endanger the baby. Traditional firewall, IDS or IPS at the gateway is not aware of such physical behavior and thus fail to ensure the physical safety of IoT.

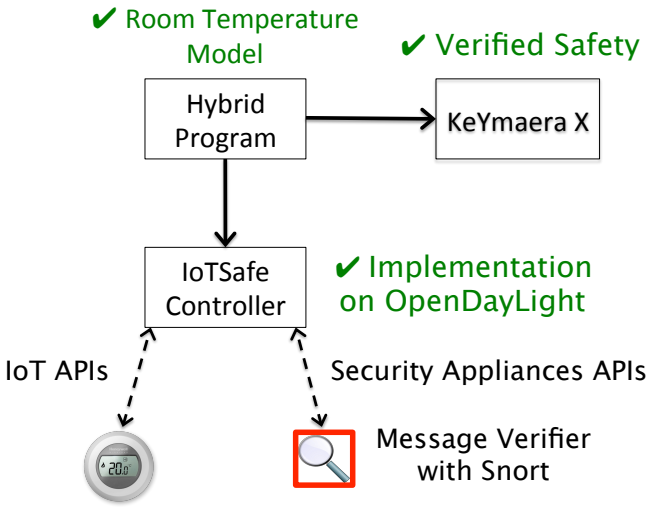

Figure 2: Our contributions in this project.

Because of these security challenges, we think the current hardware controller on device or the firewall at the gateway is insufficient to ensure the physical safety for IoT devices [11]. To address the two challenges, our idea is to build a security controller that controls both the IoT device and the security appliances such as firewall, IDS/IPS and guarantees the physical safety constrains (e.g., the temperature of the baby room is between 20 centigrade and 24

centigrade) even when the adversary presents (attacker can change the setting of the temperature setting of the thermostat). More specifically, we make the following contributions in this project, as shown in Figure 2:

- We model the baby room temperature with thermostat, attacker and security appliance (message verifier) presents and design a hybrid program for the security controller.
- We determine the safety property and the controller action to ensure the safety, and use KeYmaera [2] to prove the safety constrains always holds for the hybrid program.
- We implemented a real security controller based on the verified hybrid program on top of commodity SDN controller OpenDayLight [3], which can be deployed and used in a real smart home network.

2 CPS Scenario

Let's start by describing the CPS scenario we study here. We consider a baby room with temperature T_1 and thermostat that can change the room temperature. The safe temperature for a baby is 20 centigrade to 24 centigrade. The temperature outside is T_e. The user can set this thermostat with a desired temperature T_d through device APIs. Suppose this API is exposed and the attacker can also set the desired temperature T_d to overheating/overcooling the room to endanger the baby. Next we show how to capture the room temperature change using physics.

Newton's law of cooling: The Newton's law of cooling says the heat gain/loss between two objects is proportional to the temperature difference between the two objects.

$$T_1' = K(T_2 - T_1) \tag{1}$$

Assume the thermostat is using a common embedded heating mechanism, producing heat $h_{thermostat} = K_1(T_d - T_1)$. The intuition behind this mechanism is the heat generated is proportional to the difference between the desired temperature T_d and current room temperature T_1, and if they are equal, produce no heating.

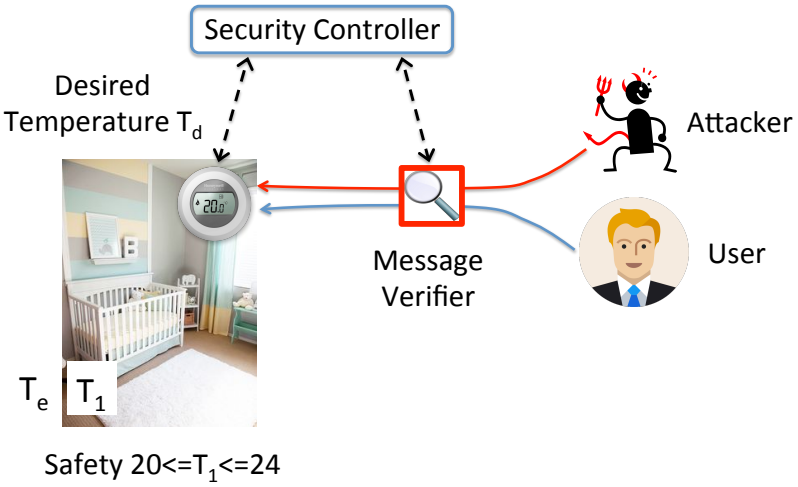

Figure 3: CPS model of a baby room.

Then according to Newton's law of cooling, the temperature of the baby room T_1 can be described by:

$$T_1' = K_e(T_e - T_1) + K_1(T_d - T_1) \tag{2}$$

where K_e is a constant decided by the physical property of the room (e.g., wall) and K_1 is a constant decided by the thermostat.

Security Controller: To protect the baby and ensures the safety constrains, the controller can perform two actions: (1) dynamically configure the message verifier to decide when the user or attacker can change the desired temperature T_d on the thermostat; (2) reset the desired temperature T_d to a safe value. Note that the reset action just uses the APIs provided by the device to control T_d and requires no modification on the device itself.

3 Hybrid Program for Security Controller

In this section, let's try to design a Hybrid Program for the security controller, reason about the design choices and decisions, and verify The intuition for design the controller is that if the room temperature gets too close to the upper bound 24 centigrade or lower bound 20 centigrade, the controller will reset the desired temperature of the thermostat.

3.1 Hybrid Program Design

Event-triggered or Time-triggered? The first question we need to ask is wether this controller should be event-triggered or time-triggered. Event-triggered here means the thermostat actively check the temperature and message the controller when temperature reaches a certain point. Time-triggered means the controller request the temperature from the thermostat periodically. For IoT devices we need to design time-triggered security controller for the following reason. Here an important observation we made is the IoT devices are usually passive, for example, Philips Hue passively respond to the get/post message from control. Since we assume we do not change the hardware or firmware of these IoT devices, we cannot change a passive IoT device to active, for example, actively send alerts when the temperature reaches a threshold.

Therefore, our controller is a time-triggered controller running a loop execution, checking the temperature every tc time. Suppose the time in each execution is denoted by t, the change of time is denoted by t', and the evolution domain of each loop execution is $0 < t < tc$.

User/Attacker Actions? We assume the user or attacker can change the desired temperature T_d to an arbitrary value in $0 < t < tc$. We also assume that the desired T_d is within range $T_d^{min} < T_d < T_d^{max}$ because of the heating capability constrain of the thermostat (e.g. limited power). And this constrain is embedded at the thermostat and can not be changed. Also, whether the user/attacker can change the desired temperature is controlled by the message verifier. We model message verifier control as a test of a message verifier state $?FW = 0$, which means the user/attacker can change the temperature at state $FW = 0$. We model this user/attacker setting of desired temperature as a non-deterministic variable with a range constrain: $?FW = 0; T_d := *; ?(T_d > T_d^{min} \& T_d < T_d^{max})$;.

Controller Actions? At the beginning of a loop execution, when a controller predicts that the potential attacker can overheat or overcool the room, the controller can reset the desired temperature to a safe value and set the message verifier to prevent the potential attacker override the desired temperature to break the safety, which is like to put a lock on the desired temperature. There are two key questions here: (1) how to predict whether the potential attacker can overheat or overcool the room? (2) what is the safe value to set the desired temperature to?

From the differential equation $T_1' = K_e(T_e - T_1) + K_1(T_d - T_1)$, we calculate the maximum temperature increase Δ_+^{max} and maximum temperature decrease

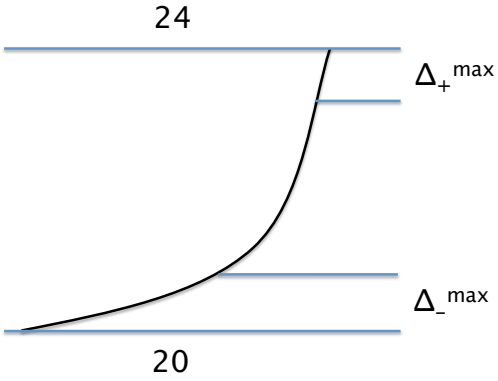

Figure 4: Estimate whether the potential attacker can overheat or overcool the room.

Δ_{-}^{max}. From the calculation, we learn that:

- The condition for the potential attacker to overheat the room is $?(T_e + T_d^{max} - (Te + T_d^{max} - 2 * T_1) * e^{(-2*tc))/2} > 24;$.
- The condition for the potential attacker to overcool the room is $?(T_e + T_d^{min} - (Te + T_d^{min} - 2 * T_1) * e^{(-2*tc))/2} < 20;$.

Then if the potential attacker can overheat or overcool the room, we can set T_1' to 0 to prevent the temperature increase or decrease in the following tc. To do so, we should set $T_d := 2 * T_1 - T_e$;

```
/*
 * HybridProgram
 * safe temperature 20 - 24 for baby room
 */

Functions.
  R Ke.
  R K1.
End.

ProgramVariables.
  R t. /* time */
  R tc. /* control interval */
  R Te. /* temperature outside */
  R T1. /* temperature room 1 */
  R T1d. /* desired temperature room 1 */
  R FW1. /* lock desired temperature */
End.
```

Figure 5: Hybrid Program Variables.

```
Problem.
(T1 >= 20 & T1 <= 24 & Te = 40 & tc = 0.05 & Ke = 1 & K1 = 1 & T1d > 20 & T1d < 24) /* Pre condition */
->
[
        {
                /* environment update */
                t := 0;

                /* security admin control */
                {
                        /* check if temperature can break constrain */
                        FW1 := 0;
                        ? (Te+40 - (Te+40 - 2*T1)*2.72^(-2*tc))/2 > 24; FW1 := 1; T1d := 2*T1 - Te;
                        ? (Te+0 - (Te+0 - 2*T1)*2.71^(-2*tc))/2 < 20; FW1 := 1; T1d := 2*T1 - Te;
                };

                /* user attacker control */
                {
                        ?FW1 = 0; T1d :=*; ?(T1d > 0 & T1d < 40);
                };

            /* continuous dynamics */
            { T1' = Ke*(Te-T1)+K1*(T1d-T1), t' = 1 & t < tc }
        }*@invariant(T1 >= 20 & T1 <= 24)
]
(T1 >= 20 & T1 <= 24)
End.
```

Figure 6: Hybrid Program.

Put Together: We now put together the above pieces and generate a Hybrid Program as shown in Figure 6. In the initial stage, the controller predict whether potential attacker can overheat the room. The the controller allow the user to change the desired temperature in range $T_d^{min} < T_d < T_d^{max}$. Finally, the controller uses the differential equation to describe the temperature change in tc as $\{T1' = K_e * (T_e - T_1) + K_1 * (T_d - T_1), t' = 1 \& t < tc\}$, where the evolution domain is $t < tc$.

3.2 Proof with KeYmaera X

After we obtain the Hybrid Program, we use the KeYmaera X [2] to verify the controller would always satisfy the safety condition. Here we do not seek much contribution on manually find complex and hard-to-obtain proofs or proof rules. This is because our long term goals is not to obtain a delicate and complex manually proof. Instead, we restrict ourself to just use general proof rules, which gives us chance to automate the proof for not only the room temperature for one particular house, but a large scale of diverse houses.

The rules we use here are general. We observed that the invariant in all loop executions is $T_1 > 20 \& T_1 < 20$, and select this as the loop invariants. Then we use the `loop` rule to break the loop:

We use the `randomb` rule to resolve the nondeterministic variables:

$$\texttt{loop} \qquad \text{ind}\ \frac{\Gamma \vdash j(\bar{x}), \Delta \quad j(\bar{x}) \vdash [a]j(\bar{x}) \quad j(\bar{x}) \vdash P}{\Gamma \vdash [a^*]P, \Delta}$$

Figure 7: Loop rule [2].

$$\texttt{randomb}\ \ [:*]\ \ [x := *]p(x) \leftrightarrow \forall x\, p(x)$$

Figure 8: Randomb rule [2].

And we use the `DW` rule (differential weakening) to prove the differential equation part:

$$\texttt{diffWeaken}\ \text{DW}\ \frac{\Gamma \vdash \forall x\, (q(x) \rightarrow p(x)), \Delta}{\Gamma \vdash [x' = f(x)\,\&\,q(x)]p(x), \Delta}$$

Figure 9: Differential weakening rule [2].

Then we can prove the Hybrid Program step by step by using `KeYmaera X` and with the following tactic.

4 Implement IoTSafe on Commodity SDN Controller

We have implemented a fully functional IoTSafe controller from the Hybrid Program on top of commodity Software-Defined-Network (SDN) controller OpenDayLight, consisting of around 8K lines of code [4].

First we provide a quick walkthrough of how our IoTSafe controller works.

1. At initial stage, using Network-Function-Virtualization (NFV) techniques, we create a number of virtualized security appliances at a security server connected to the IoT gateway, *e.g.*, a customized message verifier for the thermostat. Then using SDN technique, each device's traffic is tunneled to/from, and processed, in the micro-security appliances called micro-middleboxes.

2. Then our IoTSafe controller will establish control connections with the IoT devices and security appliances. The IoTSafe controller then actively send a message to check the room temperature and desired temperature every tc time, using a loop execution. We implemented a checking logic

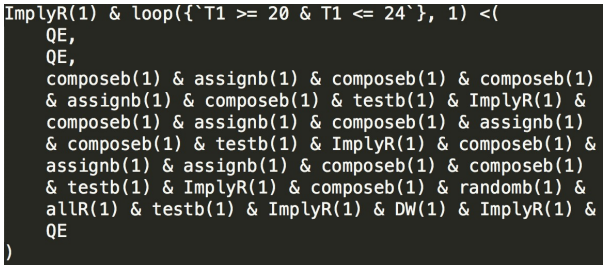

Figure 10: Tatic to prove the Hybrid Program.

with a Finite State Machine and bind each state with a action in micro-middlebox. For example, when the state is "will be overheated" - $(T_e + T_d^{max} - (Te + T_d^{max} - 2 * T_1) * e^{(-2*tc))/2} > 24$, then the controller will send a control message to the message verifier to check on and prevent any control message from user (potential attacker) send to the thermostat. In this way, the temperature is "locked" for safety in next tc time. After that, the controller will send a control message to the thermostat to reset the desired temperature to $2 * T_1 - T_e$ to overwrite the potential overheating T_d value.

Next, we briefly our implementation and the extensions we made to open source tools to enable the IoTSafe vision.

We choose `OpenDaylight` a popular industry-grade SDN controller as our starting point. Since `OpenDaylight` only focuses on simple forwarding functions, to support IoTSafe, we add several extensions:

1. *Event handler:* `OpenDaylight` natively only handles `OpenFlow` messages from SDN switches. We extend it to handle events from common security appliances; *e.g.*, Snort IDS alerts. We also extend it to get the physical context from IoT devices from IoT devices' APIs, *e.g.*, the actual temperature and the desired temperature from the thermostat.

2. *micro-middlebox control APIs:* We also implement custom control channels between the IoTSafe controller and the individual micro-middlebox. They implement micro-middlebox-specific messages for reconfiguring policies; *e.g.*, installing and updating micro-middlebox-specific configurations [5].

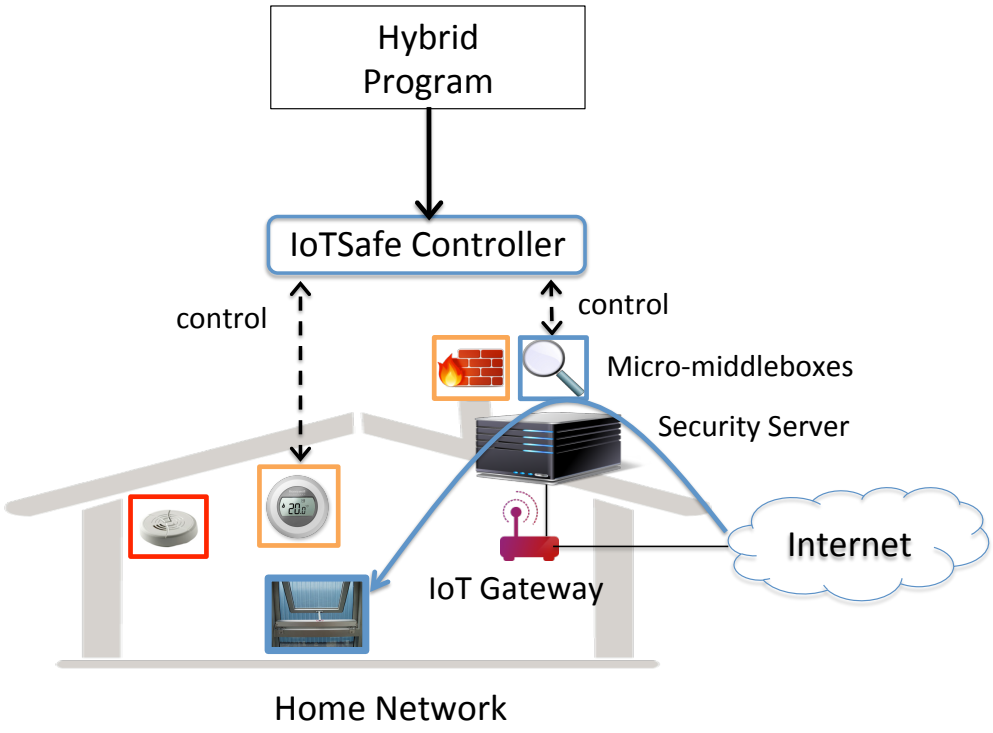

Figure 11: A real security IoT controller implementation that controls both IoT devices and network security appliances.

3. *micro-middlebox* To realize each micro-middlebox, we use virtual machines. We currently support several open source security functions; *e.g.*, NAT/firewalls using iptables, IDS/IPS using Snort [10], proxies using Squid [6]. Each micro-middlebox runs inside a VM; we use Centos 6.5 as the host OS running the KVM hypervisor. We set up simple tunnel-based forwarding rules at the ingress switch to steer the traffic to the security server. These micro-middleboxes need two minimal extensions to integrate with IoTSafe. First, we extend the micro-middlebox implementations to support the addition of tags to outgoing packets to enable the tag-based forwarding [9]. Second, we need to forward events/alerts to the IoTNetS controller. We implement this by having a light-weight IoTNetS client program that parses these alerts (*e.g.*, Snort alerts) and forwards them to the controller. This alert parsing program is configurable and can be customized to only forward relevant events.

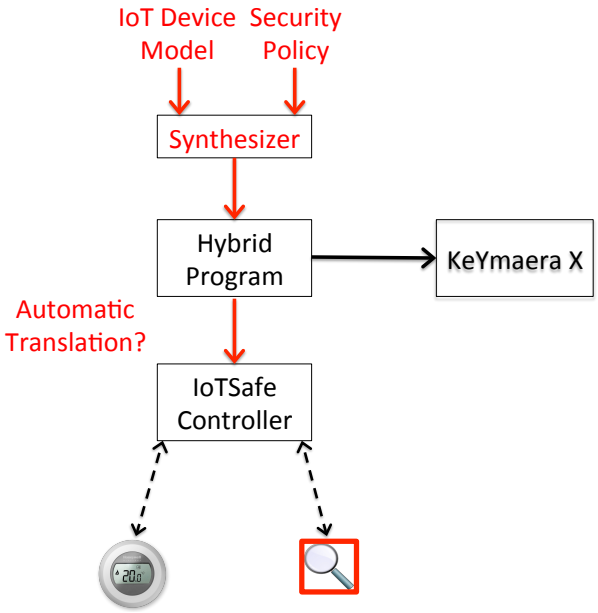

Figure 12: Future works.

5 Discussion and Future Works

In this section, we discuss the problems we encounter in this project (prove a double room setting) and critical pieces for future works.

- **Two room, two thermostats controls:** Initially we were trying to provide controller for a two room, two thermostats setting, where each room has it's own safety temperature range, and the temperature for room 1 is $T_1' = K_e(T_e - T_1) + K_{1-e}(T_d - T_1) + K_{1-2}(T_2 - T_1)$ and the temperature for room 2 is $T_2' = K_e(T_e - T_2) + K_{2-e}(T_d - T_2) + K_{2-1}(T_1 - T_2)$. However, this requires a potentially asymmetric control for each of the thermostat, and we have a hard time to clarify what is the right control assignment for the desired temperature for each thermostat and what's the right way to model the asymmetric control.

- **Synthesizer:** Currently we are write the hybrid program manually. However in reality, the users or security administrators of the IoT network do not have the knowledge to write the hybrid program. The reasonable input we require from these users or security administrators can be some

IoT models $T_1' = K_e(T_e - T_1) + K_1(T_d - T_1)$ and security policy specifications ($20 < T_1 < 40$). Therefore, a critical piece of work is to write a synthesizer to automatically translate the IoT models and policy specifications into reasonable hybrid program.

- **Automatic Hybrid Program Translator:** Now we are manually implementing the real controller based on the Hybrid Program for one scenario. This does not scale to diverse IoT scenarios given the potential combination of devices and environment is infinite. We need a Hybrid Program translator that can automatically translate the Hybrid Program into real controller and security appliances implementations such as OpenDayLight implementation.

References

[1] Fridge sends spam emails as attack hits smart gadgets. http://www.bbc.com/news/technology-25780908.

[2] KeYmaera: A Hybrid Theorem Prover for Hybrid Systems. http://symbolaris.com/info/KeYmaera.html.

[3] OpenDayLight. http://www.opendaylight.org/.

[4] Psi. https://github.com/PreciseSecurity/PSI.git.

[5] Pulledpork. https://code.google.com/p/pulledpork/.

[6] Squid. http://www.squid-cache.org/.

[7] The Internet of Things Is Wildly Insecure - And Often Unpatchable. http://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/.

[8] Will giving the internet eyes and ears mean the end of privacy? http://www.theguardian.com/technology/2013/may/16/internet-of-things-privacy-google.

[9] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proc. NSDI*, 2014.

[10] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[11] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2015.