Securing the Smart Home via a Two-Mode Security Framework

Devkishen Sisodia
dsisodia@cs.uoregon.edu
Computer and Information Science
University of Oregon

Abstract

The growth of Internet of Things (IoT) faces the growing concern of cyber attacks toward IoT. Unfortunately, the constrained resources of IoT devices and their networks make many traditional attack detection methods less effective or even not applicable. In this paper, we present a framework for a smart home environment that considers the unique properties of IoT networks. This framework utilizes a two-mode adaptive security model to enable users to balance performance, security, and overhead. We show the efficacy of our framework in two case studies that address two types of attacks in the smart home environment: sinkhole attack and distributed denial-of-service (DDoS) attack. We examine two existing intrusion detection and prevention systems and transform both of them into new, improved systems using our framework. Our evaluations show that our framework is not only friendly to resource-constrained devices, but can also successfully detect and prevent the two types of attacks, with a significantly lower overhead and detection latency than the existing systems.

I. Introduction

The Internet of Things (IoT) continues to pervade our lives. In 2016, 6.4 billion devices were connected to the Internet [1]. This number is expected to increase to 30 billion by 2020 [2]. However, as IoT devices are connected by the Internet, they also suffer from the same types of attacks that plague traditional Internet-connected machines. In October 2016, for example, the Mirai IoT botnet, which comprised of up to 100,000 infected IoT devices, launched multiple large-scale distributed denial of service (DDoS) attacks [3]. This botnet created a 1.2 terabits per second attack which resulted in the inaccessibility of many popular websites, such as Twitter, Reddit, Netflix, GitHub, and Airbnb. This attack is the largest DDoS attack on record. In

November 2016, the same botnet was responsible for successfully attacking Liberia's Internet infrastructure and taking the entire country offline [4].

While IoT devices and traditional machines suffer from the same types of attacks, IoT devices tend to be harder to secure due to some unique properties. IoT devices are often harder to patch and update due to largely non-existent automatic update systems. Also, they tend to have scarce CPU and memory resources and limited battery capacity. IoT devices can have anywhere from a few gigabytes to a few kilobytes of memory. Furthermore, with many different types of IoT devices, IoT networks are far more diverse and heterogeneous than wired networks. These unique properties, which differentiate IoT devices from traditional machines, hinder the deployment of existing security mechanisms in IoT environments.

Cryptographic protocols and intrusion detection/prevention systems (IDSes/IPSes), developed for the traditional Internet, are designed without the assumption of extremely limited resource and computing power. Even systems that are considered extremely lightweight cannot be installed on memory-constrained devices that have less than 1 MB of available memory [5]. For example, the cryptographic protocol SSL/TLS, a traditional Internet security standard, on average adds an overhead of almost 2 KB of RAM and 40 KB of ROM, while IoT devices may have only as little as 4 KB of RAM and 48 KB of ROM [6]. If a security solution needs to probe devices they protect, most devices in an IoT environment may either lack the battery power or network bandwidth to respond to every probe, or simply wish to stay dormant most of the time. Sometimes a security solution may impose some minor penalties on benign devices while mitigating an attack (e.g., dropping traffic from devices to mitigate a DDoS attack), doing so in an IoT environment can be a significant pain to those benign devices.

In this paper, we focus on the smart home environment where security and privacy are especially important, and address the ineffectiveness of traditional security mechanisms in the smart home. We introduce a security framework called TWINKLE that supports individual security applications that handle specific attacks in the smart home. By enabling each security application to run in two distinct modes, TWINKLE not only preserves the salient features of classic security solutions, but also addresses the resource limitations that IoT devices face. Every security application, while plugged into TWINKLE, will be running in regular mode for the most time and incur a minimal amount of resource consumption, but when it detects any suspicious behavior that an attack must display, it can readily switch to vigilant mode and engage in

sophisticated routines for a short time window during which to cope with the suspicious behavior with strong competence. By only running the heavyweight routines when needed, TWINKLE saves precious resources over methods that run these routines either continuously or periodically.

We further apply the TWINKLE framework to transform two prior attack solutions for the smart home environment. We convert the SVELTE solution [7] that detects sinkhole attacks to SVELTE+, which does not consume network and battery resources unless a suspicious behavior is detected and further adds a routine to remove a sinkhole node once it is detected. We also convert the D-WARD solution [8] that handles DDoS attacks from source networks to D-WARD+; unlike D-WARD, D-WARD+ does not drop packets from benign devices while still effectively keeping the DDoS traffic to an unharmful level. Our evaluation further demonstrates that SVELTE+ and D-WARD+ incur much less overhead than SVELTE and D-WARD, respectively, while achieving equal or better efficacy in handling the attacks.

The rest of the paper is organized as follows. In Section II, we describe the TWINKLE framework, including its two modes and its architectural design. In Section III, we describe the sinkhole attack in 6LowPAN networks and the prior SVELTE solution, and presents how we convert SVELTE into the SVELTE+ solution running on TWINKLE. In Section IV, we describe how D-WARD addresses the DDoS attack from their source and how we design D-WARD+ to address the drawbacks of D-WARD in the smart home environment. We present the evaluation results of both case studies in Section V, showing that the TWINKLE framework can help reduce the resource consumption in the smart home, compared to SVELTE and D-WARD. In Section VI, we discuss the feasibility of deploying TWINKLE, present possible extensions to TWINKLE, and consider open issues that will be addressed in future work. Lastly, we survey related work in Section VII and conclude the paper in Section VIII.

II. TWINKLE: A Two-Mode Security Framework for the Smart Home

Many security solutions developed for the traditional Internet, if deployed in an IoT environment such as a smart home, would incur more computing power, resources and energy than what IoT devices can handle. We design a two-mode security framework called **TWINKLE** that will not only preserve the salient features of classic security solutions, but also address the resource limitations that IoT devices face. We describe our design in this section.

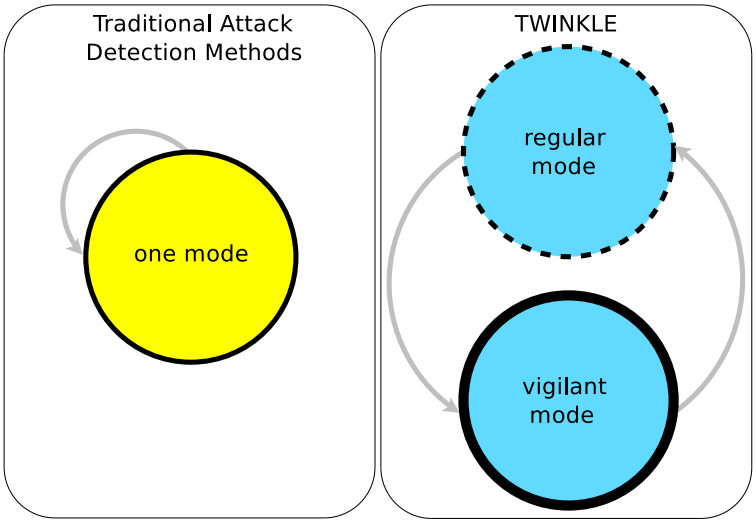


Fig. 1: State diagrams of traditional defense methods and TWINKLE

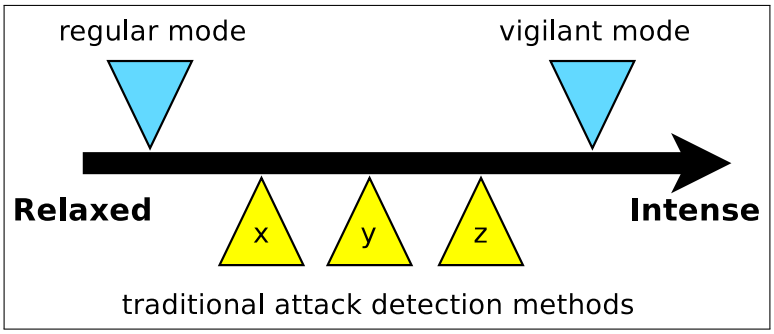


Fig. 2: Resource consumption for the two modes as compared to traditional defense methods

A. Basic Design with Two Modes

A smart home requires many types of security applications. It may face various malicious attacks such as an eavesdropping attack that can spy on the traffic between the smart home devices, a sinkhole attack that can misdirect traffic of devices to a sinkhole, a wormhole attack that can reroute data from the smart home to an attacker outside, or an attack that compromises devices at the smart home and turns them into nodes of a botnet. Worse, a smart home may also initiate attacks, such as launching a distributed denial-of-service (DDoS) attack or a phishing campaign through compromised devices at home.

The TWINKLE framework thus aims to support various security applications for the smart home, where every security application handles a specific type of attack. For every security application, the user can plug it into the framework when needed, or remove it when it is no

longer necessary.

The central dilemma facing these security applications is that they must address the inadequacy of computing power and resources at smart home devices without compromising their efficacy. If a security application runs directly on a smart home device, it may demand sufficient resources from a device that is possibly unavailable; otherwise, a security application may still need devices to respond to its requests, sometimes causing a stretch in the resources at those devices.

We therefore design the TWINKLE framework to address this dilemma. It supports every security application to operate in two distinct modes: *regular mode* for most of the time which has a low resource consumption rate and *vigilant mode* that potentially incurs a high overhead but is infrequent. In regular mode, a security application invokes functions from the TWINKLE to detect suspicious behavior that an attack, if occurring, must display, where those functions must also be lightweight. Once it detects suspicious behavior, i.e., an attack *may be* occurring, the security application will enter vigilant mode to inspect closely whether an attack is *indeed* occurring and if so conduct other security operations such as sending an alert of the attack, mitigating the attack, or recovering from the attack. After the attack is handled or the smart home is no longer under this attack, the security application goes back to regular mode. As shown in Figure 1, this two-mode design differs from many traditional attack detection methods which either continuously or periodically run in one mode, which is more resource-consuming than TWINKLE's regular mode.

TWINKLE thus supports every security application to switch between these two modes. By staying in regular mode most of the time, the security application will incur a minimal amount of resource overhead. By transitioning into vigilant mode for a short period only when needed, the security application can engage in sophisticated operations, including those that may be resource-consuming, to detect or handle an attack in question. This concept is depicted in Figure 2, which represents the resource consumption of the two modes as compared to traditional attack detection methods.

B. Architecture of TWINKLE

As shown in Figure 3, TWINKLE is composed of three main components: *manager*, *policy checker*, and *watchdog*. In general, the manager and policy checker will be running at a central node, such as the border router of a smart home, and the watchdog can be running at every device.

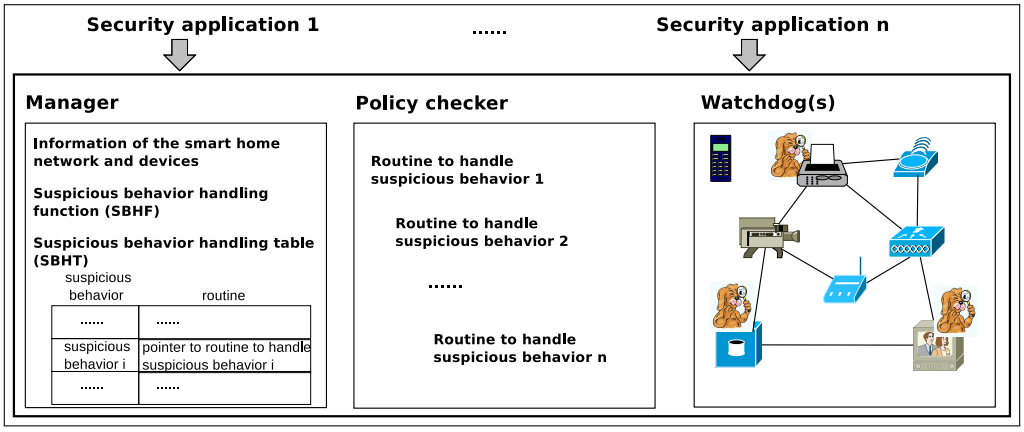


Fig. 3: The basic architecture of TWINKLE

The manager maintains the information of the smart home network, such as the network topology, routing information, or allowed bandwidth of each out bound connection. More importantly, it supports a function to handle suspicious behavior, or the **suspicious behavior handling function (SBHF)**. It maintains a **suspicious behavior handling table (SBHT)**, in which for each suspicious behavior it points to a specific routine for handling that suspicious behavior.

The policy checker maintains routines for handling suspicious behavior. Such routines are usually heavyweight and should only be running in vigilant mode when invoked on demand.

The watchdog is a lightweight running process that monitors the smart home for suspicious behavior. Multiple watchdogs can also be running at multiple devices. Whenever a watchdog detects a suspicious behavior, it invokes the function above to process the suspicious behavior. As soon as the function begins its execution, the system will enter vigilant mode.

When the TWINKLE framework supports a security application, it will instantiate the manager, the policy checker, and the watchdog according to the security application. The security application must define the attack it targets and the suspicious behavior that its watchdog should monitor. Furthermore, it needs to develop routines to handle each suspicious behavior, plug these routines into the policy checker, and populate the suspicious behavior handling table with every suspicious behavior that the security application is concerned about and the routine that handles the suspicious behavior. Additionally, the security application needs to provide the manager with the necessary information so that the suspicious behavior handling routines can refer to as a basis for their operations.

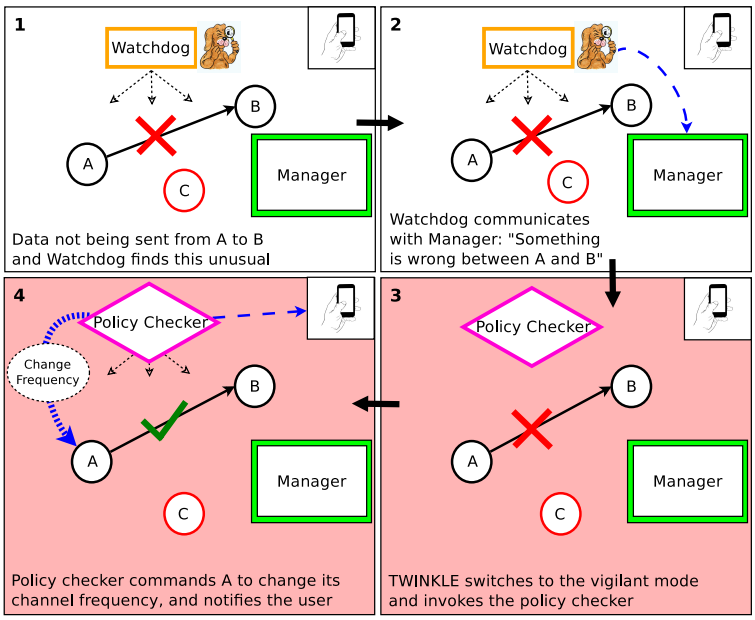


Fig. 4: An example of using TWINKLE to handle a jammed link

TWINKLE also provides a dynamic mechanism for a security application to install its watchdog at any device needed. Unlike the manager or policy checker that can run at the central node, depending on the security application in question, the watchdog may need to run on arbitrary devices in the smart home. To cater to this need, TWINKLE deploys a lightweight process called elf at each device that may be a candidate for running a watchdog of a security application. When the TWINKLE framework deploys a new security application and needs to run the watchdog code of the application at a device, TWINKLE can communicate with the elf on the device to ship, install, and eventually run the watchdog code on the device.

C. Examples

As stated previously, the TWINKLE framework can support various security applications which can be used to handle different types of attacks. Through two examples, we describe how TWINKLE can be used to support different security applications.

In Figure 4, TWINKLE is being used to handle a jamming attack where the link between devices A and B is being jammed by an unknown attacker. In the first two steps (box 1 and box 2), TWINKLE is running in regular mode. In the first step, the watchdog detects suspicious behavior, which has been defined by the security application. In this case, device B should be receiving traffic from device A, but is not which is abnormal. Therefore, the watchdog notifies the

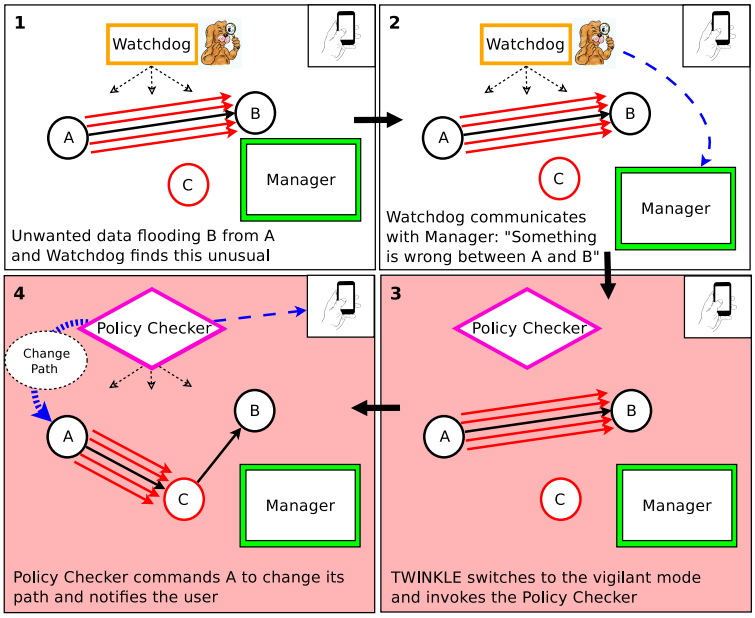


Fig. 5: An example of using TWINKLE to handle a flooding attack

manager of the suspicious behavior by invoking the manager's SBHF. As the function begins its execution, the manager switches the security application to vigilant mode (box 3). The manager's SBHT will match the suspicious behavior detected with the routine to handle that suspicious behavior, and the manager will then invoke the policy checker to run that routine. By running the routine, the policy checker will first command A to change its frequency to possibly alleviate the jammed link. The policy checker will also notify the user of the jamming attack (box 4). After mitigating the attack, the policy checker would notify the manager, which returns the security application to regular mode.

Figure 5, shows an example of how TWINKLE can be used to handle a flooding attack where device A is flooding device B with unwanted traffic. Similarly to the jamming example, the first two boxes show TWINKLE's operation in regular mode. First, the watchdog detects an unusually high amount of traffic being sent to device B by device A, which it considers as suspicious behavior. The watchdog then invokes the manager's SBHF and the manager switches the security application to vigilant mode (boxes 2 and 3). The SBHT at the manager will point to the policy checker's routine for handling this suspicious behavior and, as a result, the manager will invoke the policy checker. The policy checker will then command A to change its path to route its traffic to device C that can filter the unwanted traffic and route only the wanted traffic

to B (box 4). Like before, the policy checker will also notify the user of the attack and once the attack has been mitigated, the policy checker will notify the manager and the security application will return to regular mode.

III. Case Study 1: Sinkhole Attack Detection By Transforming SVELTE

In this case study, we transform SVELTE, an IDS for detecting sinkhole attacks in 6LoWPAN networks, into a more resource-efficient security application on TWINKLE.

A. Sinkhole Attack in 6LoWPAN Networks

6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) is a wireless technology that combines IPv6 and Low-power Wireless Personal Area Networks (LoWPAN) to enable low-powered devices to communicate using an Internet protocol. A 6LoWPAN network uses RPL (Routing Protocol over Low Powered and Lossy Networks) as its routing protocol [9]. For each destination in a 6LoWPAN network to reach, RPL creates a graph called *Destination Oriented Directed Acyclic Graph* (DODAG) where every node is a device in the network and the destination is the root. Each node in a DODAG has a set of parents, including a preferred parent, where every parent is a potential next hop to reach the root. Moreover, every node in a DODAG has a *rank* to represent the distance between the device and the root (the distance can be calculated in a number of ways, the simplest being hop-count).

Each device periodically sends out a *DODAG Information Object* (DIO) message to advertise its rank. A device can also solicit DIO messages from its neighbors by sending them *DODAG Information Solicitation* (DIS) messages. An entering device, upon the receipt of DIO messages from its neighboring devices, will create its set of parents, choose the preferred parent, and calculate its own rank (which is greater than the rank of each of its parents).

The 6LoWPAN network is subject to the sinkhole attack. In such an attack, a compromised device announces a short path toward a destination node to attract traffic from other nodes to the destination, therefore intercepting or dropping the traffic and creating a sinkhole. A sinkhole attack via RPL can happen when a device sends to its neighbors a DIO message to lie that the device has a low rank. It has been shown that RPL's self-healing and repair mechanisms are not resilient against the sinkhole attack [10].

B. Prior Art: SVELTE Against the Sinkhole Attack in 6LoWPAN

SVELTE detects sinkhole attacks in 6LoWPAN networks that occur through RPL rank manipulation. It has three main modules running on the border router (6BR) of a 6LoWPAN network: 6LoWPAN Mapper (6Mapper) that gathers information about the network and determine the DODAG rooted at 6BR, an intrusion detection module that checks the rank inconsistency in data obtained by 6Mapper to detect sinkhole attacks, and a distributed mini-firewall that filters unwanted traffic before it enters the network. 6Mapper sends *probing* messages to nodes in the entire network at regular intervals (e.g., 2 minutes). Each node then sends a *response* message to 6Mapper, which includes its node ID, node rank, parent ID, and all of its neighbors' IDs and ranks.

Unfortunately, SVELTE's probing mechanism can increase the network overhead, device battery consumption, and the latency of detecting sinkhole attacks. Every probe from the 6BR will increase the network overhead. Every response from a device will consume more battery of the device. Worst of all, SVELTE has a dilemma in choosing the probing interval: a short interval will lead to a low latency in detecting sinkhole attacks, but a large overhead due to frequent probing and responding; a long interval will result in a low overhead, but a high latency in detecting sinkhole attacks.

C. SVELTE+: A Two-Mode Approach Against 6LowPAN Sinkhole Attacks

To be more resource-efficient, we transform SVELTE to SVELTE+ that runs on TWINKLE. The essential difference between SVELTE+ and SVELTE is that the 6Mapper in SVELTE+ will not probe the entire network periodically and correspondingly, the intrusion detection component will not run periodically, either.

When SVELTE+ is plugged into TWINKLE, its manager, watchdog, and policy checker are as follows. The manager will consist of the 6Mapper module from SVELTE which runs on the central node and the distributed mini-firewall which may run on devices. The policy checker, which also runs on the central node, will include two suspicious behavior handling routines: (1) a sinkhole detection routine (i.e., the intrusion detection module from SVELTE) that inspects the ranks of nodes in the DODAG graph to determine if a sinkhole attack is occurring; and (2) a sinkhole mitigation routine that SVELTE+ newly introduced to mitigate a detected sinkhole attack. Finally, for each device that originally runs a 6Mapper client, we instead equip it with a

SVELTE+ watchdog through TWINKLE; it monitors the RPL ranks of its neighbors and alerts the manager of a suspicious behavior when it receives a new rank advertisement; the manager in turn determines how to handle the suspicious behavior, such as invoking the sinkhole detection routine.

SVELTE+ detects sinkhole attacks by switching between the two modes. It begins in regular mode. Each time a node advertises a new rank, the watchdogs that are within the range of the advertisement will treat the node as a suspect and detect a suspicious behavior. Each watchdog then invokes the manager's function for handling suspicious behavior, including passing to the function a suspicious behavior description block (SDB). The SDB will include the rank of the suspect and the rank of the watchdog itself. More importantly, as soon as the function begins its execution, SVELTE+ enters vigilant mode, allowing it to invoke the corresponding routine to handle the suspicious behavior. Based on the SDB, the function will inspect the behavior and further decides to invoke the sinkhole detection routine inside the policy checker to handle the behavior. The sinkhole detection routine first queries the 6Mapper in the manager for an up-to-date DODAG; then, if the watchdog is a parent (child) of the suspect and its rank is lower (greater) than the rank of the suspect as expected, it then has verified the consistency between this watchdog and the suspect. If it has verified the rank consistency with all of the parents and children (or a threshold number of each) of the suspect, it will treat the suspect as a benign node and invoke the 6Mapper to add the node to the DODAG, or simply update its rank if it is already in the DODAG. In case the sinkhole detection routine cannot establish the rank consistency between the suspect and its parents and children, it will detect a sinkhole attack, label the suspect as a sinkhole attacker, and further invoke the sinkhole mitigation routine as described below. The sinkhole detection routine then finishes its execution, followed by the function for handling suspicious behavior, and SVELTE+ returns to regular mode.

The sinkhole mitigation routine's main purpose is to remove a sinkhole node from not only the DODAG, but also the records of any device. Specifically, every parent of the attacker will remove it as their child. Every child of the attacker will remove it as its parent; it may also add a new parent as well as choose a new preferred parent. As a result, the attacker is isolated and can no longer successfully reach any other node.

SVELTE+ outperforms SVELTE in multiple ways. SVELTE+ can reduce the latency in detecting sinkhole attacks to a negligible amount because the watchdog immediately invokes

the suspicious behavior handler whenever a new rank is advertised, without having to wait for the next probing interval, as in SVELTE. SVELTE+ also decreases the network overhead and device battery consumption as compared to SVELTE; SVELTE+ may incur more overhead in the beginning as nodes join the network, but as the network stabilizes, the amount of times SVELTE+ switches to vigilant mode will be low. An exception here is that a malicious node may frequently advertise a new, legitimate rank, causing SVELTE+ to repeatedly process the suspicious behavior; SVELTE+ sets up an upper bound at which a benign node would advertise a new rank and labels a node as malicious if it advertises a new rank too frequently (it can further remove the node using the sinkhole mitigation routine).

IV. Case Study 2: DDoS Attack Detection By Transforming D-WARD

In this case study, we transform D-WARD, a classic security system for detecting and mitigating DDoS attacks at the source end of the DDoS traffic, into D-WARD+, a new DDoS defense solution as a security application on TWINKLE.

A. DDoS Attacks with IoT Devices

In a DDoS attack, an attacker sends a victim, such as a web server, an overwhelming amount of traffic to make it unavailable. The attacker usually employs a botnet, or a network of compromised devices, to send the traffic. Due to their abundance and the ease to be compromised, IoT devices are easy targets to be recruited by a botnet. As shown in the Mirai attack [3], recent DDoS attacks have been launched from compromised IoT devices and networks.

B. Prior Art: D-WARD Against DDoS Attacks

While trying to mitigate a DDoS attack at the victim end may face the challenge of dealing with a high volume of DDoS traffic, researchers have found that links closer to the sources of DDoS traffic are less likely to be overwhelmed, making filtering at the source end more feasible. One source-end solution example is D-WARD [8]. Deployed at the border router of a policed network, D-WARD consists of an observation module, a rate-limiting module, and a traffic-policing module. The observation module classifies each aggregated flow, or **agflow**, from all devices in the policed network to an entity outside, **receiver**, as good, suspicious, or attack, based on the ratio of sent packets to received packets of each agflow. Also, each agflow consists of multiple connections where each connection is the traffic from a specific device to

the receiver, and for each attack agflow, D-WARD classifies each individual connection as good, transient, or bad, also based on the ratio of sent packets to received packets of the connection. The rate-limiting module applies to each bad and transient connection in an agflow, and it cuts the allowed sending rate of each of these connections to a fraction, f_{dec}, of its current amount. If the device complies with the rate-limit, the rate-limit is increased linearly and eventually removed. The traffic-policing module drops all traffic that surpasses the rate-limit.

While D-WARD is primarily designed for DDoS attacks launched from traditional end-hosts on the Internet, when deployed in a smart home environment, it could hurt benign devices if their connections are labeled as transient connections since their traffic, if over the rate limit, will then be dropped. While a traditional benign end-host can recover from the accidental loss of their packets, in a IoT environment such as a smart home, a benign device could instead suffer significantly from such a loss.

C. D-WARD+: A Two-Mode Approach Against DDoS Attacks

We therefore transform D-WARD to D-WARD+ that runs on TWINKLE. To overcome the aforementioned drawback of D-WARD, when detecting a DDoS attack from a policed network, D-WARD+ leverages the *fast retransmit* mechanism in TCP congestion control to reduce the sending rate of transient connections, rather than literally dropping their packets as done in D-WARD. Since these connections could be from benign devices, doing so will not cause their packets to be dropped, while still lowering the amount of DDoS traffic departing from the network.

All running at the border router, the manager, watchdog, and policy checker of D-WARD+ are designed as follows. The manager keeps track of the rate-limit of every connection in every attack agflow. The policy checker consists of an agflow monitoring routine. The watchdog monitors the suspicious behavior of each agflow and has the agflow monitoring routine invoked if it detects an attack agflow.

As a security application of TWINKLE, D-WARD+ handles DDoS attacks by switching between the two modes. Beginning with regular mode, if the watchdog of D-WARD+ detects an attack agflow, it will invoke the manager's function for handling suspicious behavior, including passing the handler a suspicious behavior description block (SDB). The SDB will include the identifier of the attack agflow and other meta-data of the agflow. D-WARD+ then executes this function and enters vigilant mode. In doing so, based on the SDB, the function will determine

to invoke the agflow monitoring routine to handle the agflow in question. The routine will then monitor each transient connection of the attack agflow; it will send three duplicate TCP acknowledgments to the device of the connection, which, by following the TCP congestion control design, will reduce its congestion window by half, thus halving its sending rate and mitigating the ongoing DDoS attack. Here, we call the three duplicate TCP acknowledgments a *signal*. In case the device ignores the signal and continues to send its traffic at the original rate, the routine will detect it and label the connection as a bad connection. (Note that if a DDoS device follows the signal in the same way as a benign device, it lowers its sending rate and effectively mitigates the DDoS attack.) Furthermore, if the traffic volume of the connection is still above certain threshold after sending a signal, the routine can send another signal and observe the volume change of the connection, and it can repeat this procedure until the connection is no longer overwhelming its receiver.

Based on the two-mode design above, D-WARD+ is more suitable to a smart home environment than D-WARD. By not literally dropping packets as in D-WARD, D-WARD+ instead informs devices to transmit more slowly. Doing so avoids retransmissions of packets from benign devices, thus lowering network overhead and battery consumption.

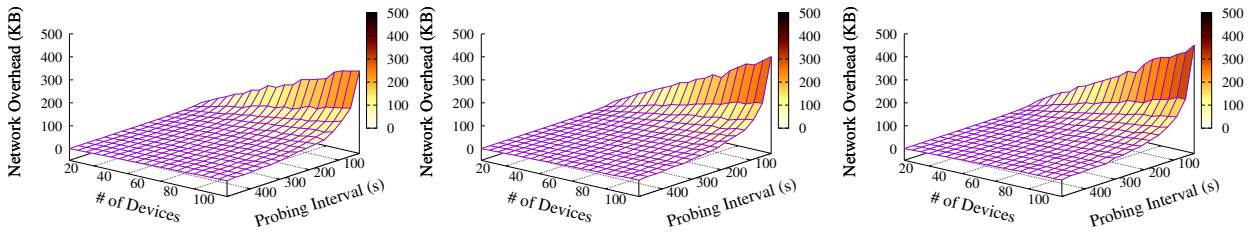
V. Evaluation

We evaluated TWINKLE's two-mode design by showing how SVELTE+ outperformed SVELTE in sinkhole attack detection and how D-WARD+ outperformed D-WARD in source-end DDoS defense. The metrics we focused on were network overhead and detection latency for the sinkhole case study, and retransmissions and connection duration for the source-end DDoS defense case study. For the source-end DDoS defense case study, we additionally compared the effects of D-WARD+ and D-WARD on a simple TCP flooding attack versus a smart TCP flooding attack. [1] We simulated SVELTE+, SVELTE, D-WARD+, and D-WARD in Java and further analyzed TCP flooding attacks using MATLAB.

A. SVELTE+ vs. SVELTE

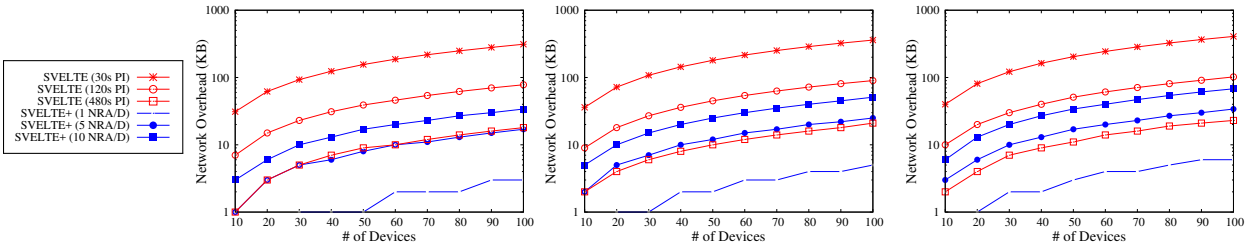
The main difference between SVELTE+ and SVELTE is that SVELTE+ utilizes on-demand probing while SVELTE utilizes periodic probing. In this section, we explored how this difference

[1]Note that we did not compare SVELTE+ to SVELTE, and D-WARD+ to D-WARD in terms of detection accuracy because SVELTE+ and D-WARD+ use the same detection modules as SVELTE and D-WARD, respectively.



(a) Average degree of 2 **(b) Average degree of 3** **(c) Average degree of 4**

Fig. 6: Network overhead for SVELTE with devices having an average degree of 2, 3, and 4



(a) Average degree of 2 **(b) Average degree of 3** **(c) Average degree of 4**

Fig. 7: Difference in network overhead between SVELTE and SVELTE+ based on probing intervals (PI) and number of new rank advertisements per device (NRA/D)

affected network overhead and detection latency for both security applications.

1) Network Overhead: This metric measures the number of extra bytes that are sent using SVELTE as compared to SVELTE+. Sending traffic requires battery consumption and therefore, a security system that sends less extra traffic by power-constrained devices is highly desired.

We simulated SVELTE and SVELTE+ by running them for one hour in multiple different conditions. For SVELTE, the probing interval, number of devices, and average number of neighbors for each device (referred to as the average degree) affects network overhead. For SVELTE+, only the number of new rank updates affects network overhead. [2] Therefore we varied the number of devices, the average degree, probing interval for SVELTE, and additionally varied the number of rank updates that occur during the life of the simulation for SVELTE+.

Figure 6 shows the network overhead of SVELTE with respect to probing interval in seconds

[2]While network overhead in SVELTE+ is not directly affected by the number of devices in the network, the probability that new rank advertisements will occur increases as the number of devices in the network increases.

and number of devices in the network. The three subfigures, 6a, 6b, and 6c, show that as the probing interval decreases and the number of devices increase, the network overhead increases. In fact, network overhead increases linearly with the number of devices and decreases exponentially with the probing interval. The difference between the three subfigures also shows how the average degree per device affects the network overhead. As the average degree per device increases, so does the network overhead. Specifically, for a network of 100 devices and a 30-second probing interval, SVELTE adds about 309 KB, 371 KB, and 421 KB of network overhead when the average number of neighbors for each device is 2, 3, and 4, respectively.

Figure 7 depicts the difference between SVELTE and SVELTE+ in network overhead. We compare SVELTE with three different probing intervals (30, 120, and 480 seconds) to SVELTE+ with three different new rank advertisement frequencies (1, 5, and 10 new rank advertisements per device). SVELTE+ incurs less network overhead than SVELTE with probing intervals of 120s and 30s. At an average degree of 3 and 4, SVELTE+ with 5 and 10 new rank advertisements per device incur more network overhead than SVELTE with a probing interval of 480s. However, if a network contains 10 devices and each device advertises a new rank 10 times in a one-hour period, there would be more than 1 new rank advertised each minute. This would be highly unlikely in a stable and stationary environment, such as a smart home. Also, note that while a 480s probing interval for SVELTE incurs a relatively small amount of overhead, the detection latency will be relatively high.

2) Detection Latency: The metric measures the difference in the amount of time it takes SVELTE to detect an attack as compared to SVELTE+. When considering detection latency in our simulations, we did not take into account the negligible round-trip time (RTT) between the border router and each device or processing time at the border router.

Figure 8 shows the detection latency of SVELTE with respect to the probing interval and number of devices in the network with an average degree of 3. Detection latency increases linearly with the probing interval and is not affected by the number of devices and the average degree per device.

Figure 9 illustrates the relationship between network overhead and detection latency for SVELTE. As the probing interval increases, network overhead decreases exponentially and detection latency increases linearly. Unlike SVELTE+, SVELTE must strike a balance between network overhead and detection latency.

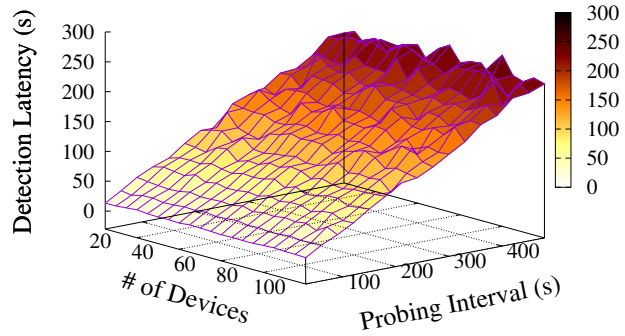


Fig. 8: Detection Latency for SVELTE with devices having an average degree of 3.

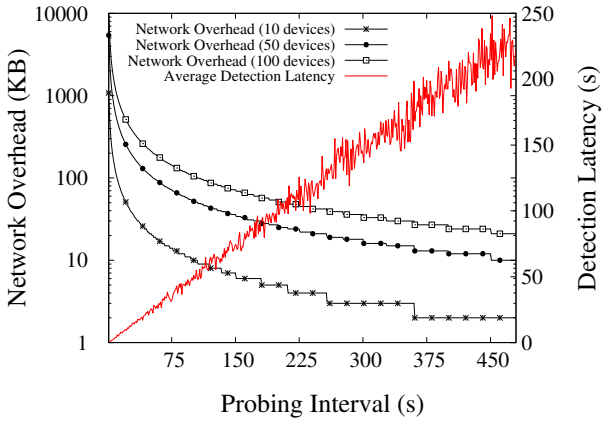


Fig. 9: The effect of probing interval on network overhead and detection latency for SVELTE

The detection latency with SVELTE+ instead is negligible since it immediately responds to a sinkhole attack on demand. It incurs only some communication delay and processing time for the watchdog to report the manipulation of rank by an attacker and for the policy checker to process the report. Here, the on-demand probing method employed by SVELTE+ is a clear advantage over the periodic probing of SVELTE.

B. D-WARD+ vs. D-WARD

The main difference between D-WARD+ and D-WARD is that D-WARD+ utilizes the fast retransmit mechanism instead of dropping packets from transient connections. The fast retransmit mechanism allows D-WARD+ to throttle DDoS traffic that leaves the source network it polices

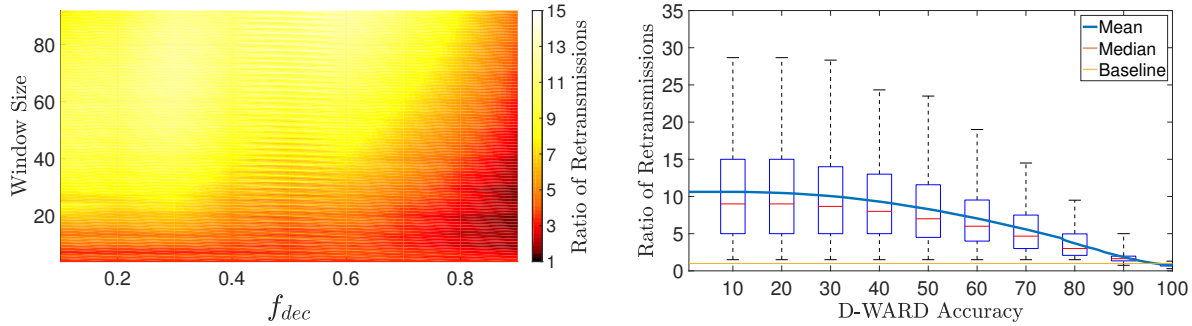
and avoid resource penalties on benign traffic. In this section, we analyzed the achievability of these goals in a smart-home network that utilizes D-WARD+. Specifically, we analyzed the following:

1) the ratio of retransmissions D-WARD requires of a benign transient connection over the amount required by D-WARD+;
2) the difference in connection duration of a benign transient connection under D-WARD compared to that of D-WARD+;
3) the maximum length of time that D-WARD+ allows a malicious transient connection to perform a TCP flooding attack; and
4) the maximum length of time that D-WARD+ allows a malicious transient connection to perform a "smart" TCP flooding attack by following TCP congestion control.

1) Retransmissions: In order to calculate the ratio of retransmissions D-WARD requires of a benign transient connection over the amount required by D-WARD+, we examined the number of retransmissions required of a benign transient connection that attempts to send 100 KB of data outside of the policed network under both D-WARD and D-WARD+.

Figure 10a presents the average number of retransmissions that D-WARD+ and D-WARD requires of benign transient connection over two main parameters: the sender's congestion window size, W, at the time D-WARD or D-WARD+ detects an attack agflow, and the pre-set fraction of traffic, f_{dec}, that D-WARD or D-WARD+ allows to leave the source network during a suspected DDoS attack. Upon detection of an attack agflow, D-WARD only allows $W * f_{dec}$ segments to the sender each RTT to mitigate any DDoS attacks. Therefore, when the benign transient devices follow TCP congestion control, D-WARD drops $W - W * f_{dec}$ segments every two RTTs. Thus, as W increases or f_{dec} decreases, D-WARD drops more segments which causes more retransmissions. When W is greater than 50 and f_{dec} is less than 0.5, D-WARD can require over 15 times the number of retransmissions than D-WARD+. Even when W less than 10 and f_{dec} is greater than 0.9, D-WARD still requires more retransmissions than D-WARD+, but to lesser degree.

Figure 10b describes how D-WARD's prediction of the amount of traffic that the receiver can handle, $RECW$, affects the ratio of retransmissions required of benign transient devices. When D-WARD fails to accurately predict $RECW$, D-WARD causes more than ten times the amount of retransmissions than D-WARD+. Further, D-WARD needs to maintain better than $\sim 97\%$



(a) Effect of W and f_{dec} on the ratio of retransmissions **(b) Effect of D-WARD's accuracy on the ratio of retransmissions**

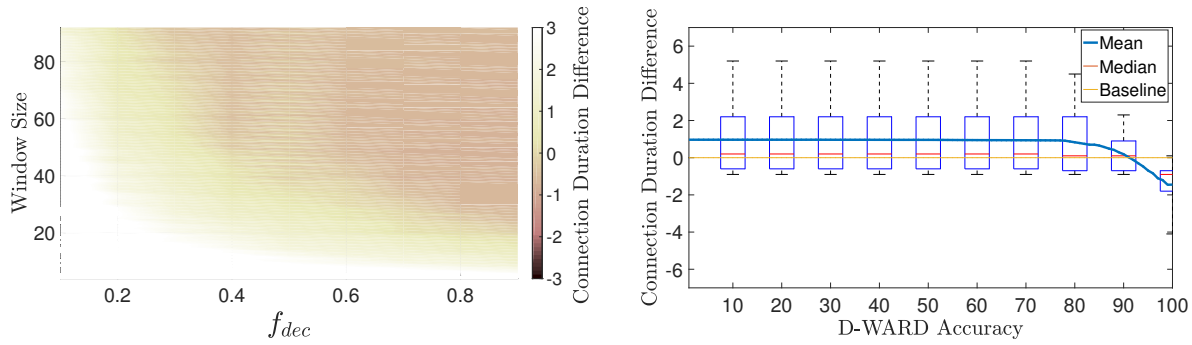
Fig. 10: Comparison of retransmissions under D-WARD and D-WARD+

accuracy in order to cause a similar amount of retransmissions as D-WARD+. Moreover, even at 100% accuracy, D-WARD still always requires more retransmissions than D-WARD+.

2) Connection Duration: We further compared how long a benign transient connection may last under D-WARD and D-WARD+. Clearly, when transmitting the same amount of data, a shorter duration is desired. We examined the duration of a benign transient connection that attempts to send 100 KB of data outside of the policed network at a maximum bandwidth of 250 Kb/s under both D-WARD and D-WARD+.

Figure 11a shows the average difference in connection duration under between D-WARD and D-WARD+ over the two main parameters W and f_{dec}. When f_{dec} is set low, D-WARD may punish a transient connection too heavily which leads to long connection durations. However, in cases where f_{dec} is set high and W is large, a transient connection's duration under D-WARD can be, at most, three seconds faster than if it were under D-WARD+.

Figure 11b depicts how D-WARD's prediction of $RECW$ affects a transient connection's duration. When D-WARD poorly predicts $RECW$, a transient connection will on average need more time to complete its TCP connection than D-WARD+. When D-WARD can predict $RECW$ with higher than 90% accuracy, it will on average allow a transient connection to complete its TCP connection one to two seconds faster than D-WARD+. However, even though D-WARD sometimes achieves faster connections, note a malicious connection can follow TCP congestion control to evade D-WARD's detection, while D-WARD+ can successfully mitigate it.



(a) Effect of W and f_{dec} on connection duration **(b) Effect of D-WARD's accuracy on connection duration**

Fig. 11: Comparison of connection duration under D-WARD and D-WARD+

3) Simple TCP Flooding Attack: A "simple" TCP flooding attack is one in which the attacker ignores TCP congestion and flow control. We formulate the maximum length of time that D-WARD+ allows a malicious transient device to perform a simple TCP flooding attack as follows. Upon detection of an attack agflow, D-WARD+ provides all transient connections belonging to the attack agflow a window of D seconds to prove the benevolence of their traffic. If a transient device performs a simple TCP flooding attack, it will not follow TCP congestion control which D-WARD+ notices within two RTTs. At this point, D-WARD+ now has high confidence the transient connection is malicious, and begins to drop this connection's traffic, thus ending the DDoS attack.

4) Smart TCP Flooding Attack: A "smart" TCP flooding attack is one in which the attacker follows TCP congestion control. We formulate the maximum length of time that D-WARD+ allows a malicious device to perform a "smart" TCP flooding attack by following TCP congestion control. Upon detection of an attack agflow, D-WARD+ sends s signals every RTT for the next D seconds to each transient connection belonging to the attack agflow so that after D seconds, each transient connection's congestion window will be below $RECW$.

Figure 12 compares how D-WARD and D-WARD+ handle a smart attacker in TCP Reno. This figure shows two transient connections that belong to an attack agflow, which was detected at 0 RTT. For simplicity, the victim has a constant $RECW$ of 10. Any traffic that surpasses the $RECW$ of 10 is considered DDoS traffic. Lastly, the circle-dotted (blue) line represents the amount of traffic sent by the attacker and the square-dotted (green) line represents the amount

of traffic that is successfully received by the victim.

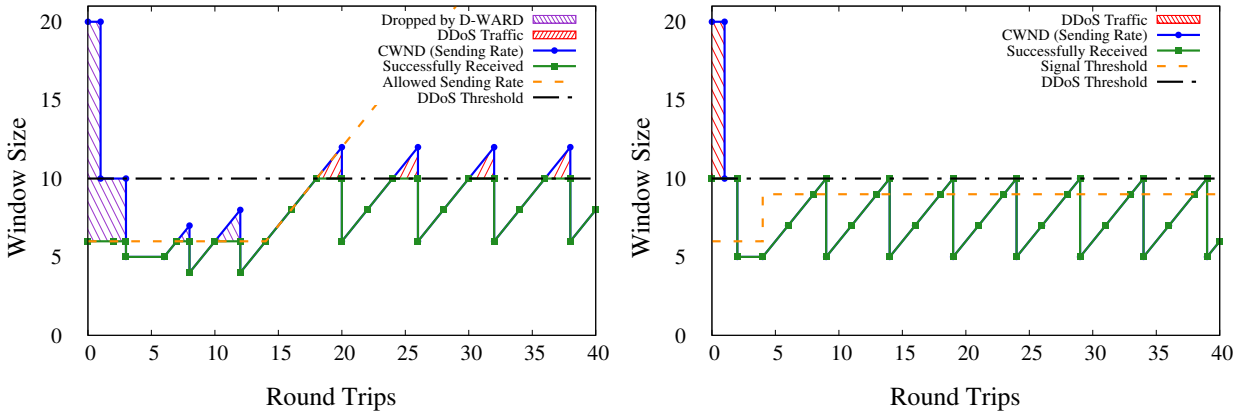
In 12a, D-WARD drops all traffic that surpasses the allowed window size of 6, which in this case means D-WARD set f_{dec} to 0.3. At 3 RTT, the smart attacker is sending less than or equal to the allowed amount and will begin to linearly increase its sending rate, following TCP congestion control. If the smart attacker follows TCP congestion control for a certain period of time, D-WARD will treat the connection as good and begin to linearly increase the allowed amount. In this case, the period of time D-WARD observes the connection before increasing the sending rate is from 0 RTT to 14 RTT. From 17 RTT on, the smart attacker can send DDoS traffic to the victim while still following congestion control. In summary, D-WARD initially throttles the smart attacker which forces the attacker to send at the allowed rate, but once the congestion window settles below this initial allowed rate, and because the smart attacker follows TCP congestion control, D-WARD continues to linearly increase the allowed amount even past an amount the receiver can manage. This provides the smart attacker an opportunity to successfully send DDoS traffic in the future.

In 12b, D-WARD+ allows all traffic to be sent to the victim, but sends signals to the smart attacker to force it to send less than or equal to the allowed rate. In this case, after one signal, the victim is no longer under DDoS attack (the victim was under attack for only 1 RTT). After two signals, the attacker will be below the allowed sending rate of 6, or signal threshold (threshold at which a signal will be sent to the attacker). After 4 RTTs, D-WARD+ will have gained enough information about the victim's $RECW$ and increase the signal threshold to just below $RECW$. Now the attacker will linearly increase its sending rate. However, when it passes the signal threshold, D-WARD+ will send it a signal and force it to cut its sending rate, preventing it from causing a DDoS attack. The signal threshold will only increase if $RECW$ increases. In summary, while D-WARD+ allows DDoS traffic for a short period initially, because the smart attacker follows TCP congestion control, D-WARD+ can continue to preemptively restrict the smart attacker's congestion window preventing any further DDoS attacks.

VI. Discussion

In this section we discuss the feasibility of deploying TWINKLE in a smart home environment, possible extensions to TWINKLE, and open issues that will be addressed in future work.

One issue is the feasibility of deploying TWINKLE components on a smart home's border router. We assume that the border router has enough resources to run TWINKLE's manager and



(a) D-WARD **(b) D-WARD+**

Fig. 12: Behavior of a smart attacker under D-WARD and D-WARD+

policy checker components. While this may be a safe assumption to make for many commercial home routers, we have yet to do sufficient evaluation to make a strong claim that TWINKLE can run on a majority of routers. Also, the feasibility of running TWINKLE on routers is highly dependent on the security application. We can estimate the memory consumption of SVELTE+ and D-WARD+ from evaluation done on SVELTE and D-WARD. For example, SVELTE consumes at most 4.724 KB of RAM (49.924 KB of ROM) at the router in a smart home containing 16 devices, while D-WARD consumes at most 37.581 KB of RAM at the router in a smart home with 100 outbound connections.

Another issue is the feasibility of running watchdog code on devices in the smart home. We also assume that each watchdog device can run an elf process which can be used to transfer watchdog code onto the device. However, some devices, such as legacy and very low-powered devices, do not have the ability to install even lightweight processes like elf. Therefore, in some cases, additional devices may need to be added to the network to act as watchdog devices. Furthermore, watchdog devices are required to have enough resources to run the lightweight algorithms of regular mode. Again, the feasibility of running these algorithms is dependent on the security application. SVELTE consumes at most 0.350 KB of RAM (1.414 KB of ROM) at each device while D-WARD does not run any code on the devices.

TWINKLE is a security framework that addresses the resource limitations of IoT devices by giving security applications the ability to run in two modes, regular and vigilant mode. In regular

mode which will run more frequently than vigilant mode, security applications will invoke less resource-consuming functions than will be invoked in vigilant mode. However, TWINKLE can be easily extended to include more than two modes of operation. Certain security applications need to invoke a wide range of functions in order to mitigate an attack. Therefore, allowing for multiple modes of operation in which applications can invoke functions of varying resource-consumption intensity may, in some cases, further reduce resource consumption. For simplicity, we only present two modes in this paper and from the evaluation results of the SVELTE+ and D-WARD+, we show that TWINKLE's two mode design reduces resource consumption.

In future work, we plan on studying, and eventually addressing, the aforementioned issues. Specifically, we will evaluate multiple different security applications on TWINKLE and implement TWINKLE on a real IoT testbed to present a more comprehensive study on the feasibility of deploying components on border routers and devices. Also, we plan on evaluating the extent to which multiple modes reduce network and device resource consumption by plugging various security applications into TWINKLE.

VII. Background & Related Work

We organize the related work into three sections. The first is on work that provides analysis on smart home security and goals for securing smart home environments. The second is on work that introduces security frameworks and systems targeted towards IoT environments. The last section is on work that motivates different components of our two-mode framework.

A. Smart Home Security Analysis

Recent work explores the current state of smart home security and provide suggestions on improvements in this environment [11], [12], [13] Denning et al. [11] group security and privacy goals into three categories: device goals such as device privacy, device availability, command authenticity, and execution integrity, data goals such as data privacy, data integrity, and data availability, and environment goals such as environment integrity, activity pattern privacy, sensed data privacy, sensor validity, and sensor availability. Notra et al. report vulnerabilities in various household devices, such as the Phillips Hue light-bulb, the Belkin WeMo power switch, and the Nest smoke-alarm. The main contribution of [13] is the discovery of security-critical design flaws in the SmartThings capability model and event subsystem. These papers give insight into the missing gaps that need to be addressed by smart home security frameworks and systems.

Of the three papers, only [12] provides a security solution. However, this solution only provides protection via access control rules deployed at the gateway router to prevent unauthorized in-bound and out-bound traffic. Our framework, not only monitors traffic leaving and entering the network, but also monitors device to device communication from within the network. This allows our framework to potentially detect and prevent attacks that cannot be detected or prevented solely at the gateway router.

B. Frameworks and Systems

In this section, we survey select papers which introduce security frameworks for IoT environments [14], [15], [16], [17]. In [14], the authors present a security framework based on the Architecture Reference Model (ARM) of the IoT-A EU project. The work in [15], uses game theory and context-aware techniques to create a risk-based adaptive security framework for IoT in an eHealth environment. Both [14] and [15] are proof-of-concept papers that do not provide evidence that the presented frameworks are viable in resource constrained environments. Similar to our framework, the frameworks presented in [16] and [17] are both targeted towards smart home environments. Also, the authors of [16] present a modular security manager which is similar to the Manager component in our framework. However, like [14] and [15], the authors of both papers do not address the limitations of IoT devices nor provide evaluation results for the resource costs of deploying their solutions. In contrast, our framework's primary focus is to reduce resource consumption while maintaining a secure environment. Furthermore, we show that our framework can reduce resource consumption through the evaluation of two concrete case studies.

C. Motivation for Certain Components and Policies

Papers [18], [19], [20], and [21] motivate the need of certain components in a security framework for the smart home environment. Instead of introducing new security countermeasures, the authors of [18] attempt to strengthen security for smart home networks by making it easier for non-expert home owners to set up secure networks and intuitively manage trust and access to their devices. The research in [19] attempts to provide adequate mechanisms to control the flow of data and enforce policies based on users' preferences. In [21], the authors utilize special nodes that monitor traffic within the network to detect certain routing attacks. The work in [18], [19], and [21] show the need of user interaction, adjustable policies set by users, and dedicated

watchdog nodes for inspection of in-network communication, respectively. Also, the work in [20] provides motivation for allowing security policies, such as using efficient authentication and key agreement methods. Work presented in this section can supplement and extend our framework.

VIII. Conclusion

The staggering growth of Internet of Things (IoT) brings serious security concerns. However, due to the constrained resources of IoT devices and their networks, many traditional attack detection methods become less effective or even not applicable in an IoT environment. Using the smart home as the battleground, this paper proposes a security framework called TWINKLE that endeavors to address a fundamental dilemma facing any security solution for IoT: the solution has to consume as little resources as possible while still aspiring to achieve the same level of performance as if the resources needed are abundant. It introduces a two-mode design to enable any security application plugged into the framework to handle their targeted attacks in an on-demand fashion. Every security application can simply run lightweight operations in regular mode most time, and only invoke heavyweight security routines when it needs to cope with suspicious behaviors. By applying TWINKLE toward the sinkhole attacks and the distributed denial-of-service (DDoS) attacks, we can successfully convert prior solutions to more resource-efficient versions, as demonstrated by our evaluations.

References

[1] R. van der Meulen, "Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015," http://www.gartner.com/newsroom/id/3165317, 2015, [Online; accessed 7-May-2017].

[2] A. Nordrum, "Popular internet of things forecast of 50 billion devices by 2020 is outdated," http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated, 2016, [Online; accessed 7-May-2017].

[3] S. Hilton, "Dyn analysis summary of friday october 21 attack," https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, 2016, [Online; accessed 22-May-2017].

[4] B. Krebs, "Did the mirai botnet really take liberia offline?" https://krebsonsecurity.com/2016/11/did-the-mirai-botnet-really-take-liberia-offline/, 2016, [Online; accessed 22-May-2017].

[5] O. P. Team, "Ossec: Open source hids security," https://ossec.github.io/index.html, 2010–2017, [Online; accessed 22-May-2017].

[6] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the internet of things," *IEEE Communications Magazine*, vol. 50, no. 12, 2012.

[7] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[8] J. Mirkovic and P. Reiher, "D-ward: a source-end defense against flooding denial-of-service attacks," *IEEE transactions on Dependable and Secure Computing*, vol. 2, no. 3, pp. 216–232, 2005.

[9] R. IETF, "Routing over low power and lossy networks."

[10] L. Wallgren, S. Raza, and T. Voigt, "Routing attacks and countermeasures in the rpl-based internet of things," *International Journal of Distributed Sensor Networks*, 2013.

[11] T. Denning, T. Kohno, and H. M. Levy, "Computer security and the modern home," *ACM Communications*, vol. 56, no. 1, pp. 94–103, 2013.

[12] S. Notra, M. Siddiqi, H. H. Gharakheili, V. Sivaraman, and R. Boreli, "An experimental study of security and privacy risks with emerging household appliances," in *IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 79–84.

[13] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 636–654.

[14] J. B. Bernabe, J. L. Hernandez, M. V. Moreno, and A. F. S. Gomez, "Privacy-preserving security framework for a social-aware internet of things," in *International Conference on Ubiquitous Computing and Ambient Intelligence*. Springer, 2014, pp. 408–415.

[15] H. Abie and I. Balasingham, "Risk-based adaptive security for smart iot in ehealth," in *Proceedings of the 7th International Conference on Body Area Networks*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, pp. 269–275.

[16] A. K. Simpson, F. Roesner, and T. Kohno, "Securing vulnerable home iot devices with an in-hub security manager," in *IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE, 2017, pp. 551–556.

[17] W. M. Kang, S. Y. Moon, and J. H. Park, "An enhanced security framework for home appliances in smart home," *Human-centric Computing and Information Sciences*, vol. 7, no. 1, p. 6, 2017.

[18] D. N. Kalofonos and S. Shakhshir, "Intuisec: a framework for intuitive user interaction with smart home security using mobile devices," in *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 2007, pp. 1–5.

[19] R. Neisse, G. Steri, and G. Baldini, "Enforcement of security policy rules for the internet of things," in *IEEE 10th International Conference on Wireless and Mobile Computing*. IEEE, 2014, pp. 165–172.

[20] P. Kumar, A. Braeken, A. Gurtov, J. Iinatti, and P. Ha, "Anonymous secure framework in connected smart home environments," *IEEE Transactions on Information Forensics and Security*, 2017.

[21] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos, "Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things," in *IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, 2015, pp. 606–611.