Orchestrating Intelligent Network Operations in Programmable Networks

Hätönen, Seppo

IEEE
2021-12

Hätönen , S , Hafeez , I , Mineraud , J , Rao , A & Tarkoma , S 2021 , Orchestrating Intelligent Network Operations in Programmable Networks . in 2021 IEEE Conference on Standards for Communications and Networking (CSCN) . IEEE , pp. 148-154 , IEEE Conference on Standards for Communications and Networking , 15/12/2021 . https://doi.org/10.1109/CSCN53733.20

http://hdl.handle.net/10138/339520
https://doi.org/10.1109/CSCN53733.2021.9686172

unspecified
acceptedVersion

Orchestrating Intelligent Network Operations in Programmable Networks

Seppo Hätönen, Ibbad Hafeez, *Julien Mineraud, Ashwin Rao, and Sasu Tarkoma
University of Helsinki

***Abstract*—Our communication networks have grown in size and have witnessed an influx of heterogeneous devices and their specialized device controllers. These devices include traditional computers, light-weight Internet-of-Things devices and their controllers, and also programmable switches and routers. Managing and protecting networks used by these devices is non-trivial, especially since the devices either use different standards or no standards at all for control and management. In this article, we discuss PraNA, an SDN-based architecture for managing and orchestrating such heterogeneous networks. At the heart of PraNA is the PraNA Orchestrator which offers a comprehensive network-wide view of the devices by using the services of a wide range of specialized controllers.**

***Index Terms*—SDN, IoT, Network Management, Network Orchestration, Device Classification.**

I. Introduction

We are witnessing an influx of connected devices such as smart lights, smart energy meters, etc. Recent trends indicate that the number of devices in our networks and the density of devices around us will continue to increase [1]. These devices—commonly known as Internet-of-Things (IoT) devices—share the networking resources used by our desktops, laptops, and hand-held smartphones and tablets. However, unlike our smartphones and laptops, IoT devices are typically designed for specific use cases such as sensing, monitoring, and automation; specifically, their computing, storage, and networking capabilities are optimized for the use cases [2].

The term Internet of things implies that these devices require Internet connectivity, and these devices typically prefer to use Wi-Fi and/or cellular to perform their operations. While IoT has been used to encompass devices that do not use IPv4 or IPv6, in this article we limit ourselves to devices that can be managed using IPv4 or IPv6 (§III). In spite of the influx of these devices, networks continue to be managed using legacy solutions that are incapable of scaling at the required pace, and contemporary IoT gateways conflate network connectivity with network services [3]. For instance, most networks are set up using off-the-shelf commodity Wi-Fi access points with minimal setup to enable Internet connectivity. While this can scale to support a small number of laptops and smartphones, the network performance and security of networks that do not use programmable network management tools can quickly deteriorate as the number of devices in the network increase.

We have identified two key challenges in improving the state of networks where a majority of user devices are connected.

1) Some of the devices, primarily IoT devices, have device-specific Wi-Fi controllers and hubs, which act as a relay in connecting user devices to Internet [4]. This results in silos where devices and their controllers work in isolation, and consequently one cannot extract the full benefit of having these devices working towards a common goal.

2) Although IoT devices collect a lot of data about their users or their environment, they are generally not designed with a security-first approach [5]. Consequently, these devices and the networks to which they are connected are susceptible to various types of network attacks [6]. While the traditional network security solutions, such as firewalls, intrusion detection systems (IDS), typically installed at the network perimeter, protect the networks against external threats, these solutions are not effective against threats originating from within the network [7]. These solutions do not monitor and control the device-specific network activity. Therefore, if a vulnerable device joins the network, it can access and potentially infect other devices within the same network boundary.

In this paper, we posit that the availability of specialized off-the-shelf controllers opens a path for building an orchestrator that uses the services of these controllers to programmatically manage and orchestrate the network. Our prior works on IoT controllers [8], [4], securing IoT networks [5], [9], and managing Wi-Fi networks using SDN controllers [10], [11] provide the cornerstone for such an orchestrator, which we name *PraNA*. As detailed in §III, our *PraNA* Orchestrator uses the north-bound APIs of SDN controllers, Wi-Fi controllers, and IoT hubs to compile a comprehensive network-wide view, and uses it to ensure that the network conforms to the desired policies. For instance, our *PraNA* Orchestrator can use REST or other APIs of the Wi-Fi controllers to get up-to-date information on the wireless network. It is also capable of using machine learning based device classification solutions to identify the device, and enforce device-specific network policies. In our prototype, we use our custom random forest based classification engine [12], and we believe that this can be easily swapped with other device classification solutions such as the one used by IoT Sentinel [7]. We believe that our *PraNA* Orchestrator can provide an integration point for third-party services to enhance the security and performance of the network without requiring changes to end-user devices.

With this holistic view of the network, the *PraNA* Orchestrator can support additional services, such as, device classification, threat monitoring, quality-of-service (QoS) guarantees. In this paper, we use device classification service as an

*This work was done by Julien Mineraud when he was working at the University of Helsinki.

example use case. With access to inbound and outbound traffic at device-level granularity, the *PraNA* Orchestrator can use a third-party classification service to detect what devices are connected to the network. Using this information, it can also assign network profiles to specific devices which can be used to restrict device communications to devices and services within and across the network.

Our key contributions are as follows.

1) *Design and architecture of the modular PraNA framework.* As discussed in §II, prior works have focused on controlling individual network elements. For instance, SDN controllers are designed and optimized for managing forwarding elements such as switches and routers, Wi-Fi controllers are aimed at managing Wi-Fi networks, and IoT Hubs are aimed at managing the IoT devices that are configured to be managed by the hub. In §III, we show that by leveraging the north-bound APIs of these controllers it is possible to compile a comprehensive view of the network and enforce policies that encompass a wide range of devices including SDN switches, Wi-Fi APs, and IoT hubs.
2) *Empirical analysis of network performance with PraNA Orchestrator to identify performance pain-points.* Not all controllers expose well-defined north-bound APIs, and these APIs can evolve over time. For the state-of-the-art practice, we introduce a SHIM layer that enables our *PraNA* Orchestrator to exchange information with the controllers, and this SHIM layer uses MQTT for message exchange. In §IV, we present the results of our evaluation. We present the latency incurred by using this message bus, and also identify the latency incurred for device classification and for enforcing the policy. We observe that the time required to capture packets required for device classification is significant compared to other operations in the network orchestration.
3) *Design guidelines for developing flexible network management system which can scale from small office home offices (SOHO) networks to enterprise networks.* Our prototype *PraNA* Orchestrator was primarily designed to explore the issues that can arise when leveraging the north-bound APIs of off-the-shelf network controllers for orchestrating networks that serve devices with heterogeneous capabilities and requirements. We believe that given the siloed nature of IoT devices, orchestrating future networks would require a similar approach, and in §IV and §V we discuss the performance impact of the interactions between these controllers. Specifically, we observe that the packet capture required for device classification takes the most amount of time, while other operations can be performed in a fraction of time compared to the capture.

In summary, we believe that our modular open-source *PraNA* Orchestrator provides a vantage point to explore the design space of combining various network controllers to orchestrate the network. As discussed in the next section, the need for combining network controllers arises because of siloes, and our *PraNA* controller builds on the insights of modular tools like Home Assistant [13] and IFTTT [14] that have been designed to work across the siloes.

II. Background and Related Work

In this section, we take a look into the background of key aspects of *PraNA*. These aspects include network management and orchestration, detecting and classifying the devices inside the network, and securing them.

TABLE I
Control interfaces of Wi-Fi Controllers.

Vendor	SNMP	Proprietary	REST	Command Line
Cisco Wireless LAN Controller	✓	✓		✓
Aruba Mobility Controller	✓	✓	✓	✓
Ubiquiti UniFI	✓	✓	✓	✓

Pre*All controllers support both SNMP and command line interfaces in addition to proprietary API. Some controllers also support REST APIs.*

a) Network Orchestration: Our wireless networks have grown in size over the last few years. Larger networks require provisioning and channel management, which can no longer be managed by hand. Many commercial vendors offer managed wireless networks, which are managed by their controllers. These include offerings from Cisco, Ubiquiti, and Aruba to name a few. All these vendors provide their wireless controllers either in physical or virtual form. While these are proprietary systems, all of them offer at least Simple Network Management Protocol (SNMP) [15] and command line interfaces for monitoring and managing the wireless network events as shown in Table I.

For controlling these controllers remotely, a REST API is typically the preferred way to control and configure the controllers, as long as there is a reference available. The command line interface can also be utilized in automation by using tools like Ansible [16] to configure the controllers.

However, if the cloud based networks are brought into the picture, orchestrating the network efficiently is far from easy. In [17], a survey of different network orchestration systems provides possible methods for orchestration.

b) Behavioral Fingerprinting and Categorization: In recent years, there have been several proposals for device fingerprinting by capturing network traffic from the device and using Machine Learning (ML) to classify and identify the device [18]. The data and fingerprints collected from the devices can be used to train the ML models. Using automatic device classification allows the network users to be aware of what devices are inside the network, and pairing the information with security services, classifying allows detection and mitigation of possibly malicious devices inside the network.

To our best knowledge, [9] was the first system to detect and classify devices based on the network traces using cloud services. However, this brings privacy and performance problems as the classification is done in a remote location. More advanced systems such as IoT Sentinel [7] and IoT Keeper [5] solve this by performing the classification at the edge gateways.

In addition to detecting devices when they connect to the network, it is also needed to detect attacks and changes in device behavior during their normal operations. For this, approaches such as Kitsune [19] or AuDI [20] can be used to detect changes in the network traffic. The output of these can then be disseminated to the network orchestration systems to automatically isolate devices and react to attacks.

c) Controlling and Securing the Networks Using SDN: SDN can be used to secure networks, provided that the network is SDN capable. One large survey [21] presents an overview of different methods to secure the network.

In general, the SDN controllers can be used to segment parts of the networks, even with the granularity of a single device [22]. For example, IoT-Keeper [5] is capable of segmenting the network where malicious IoT devices are detected.

One key problem here is that the network should be SDN capable. However, currently there are almost no Wi-Fi access points that support SDN. To alleviate this, Swift [10] can be used to bring SDN capabilities to the Wi-Fi access points that do not otherwise support SDN.

d) Managing Devices in the Network: The IoT and similar devices have a multitude of different standards and other methods for accessing them. While some of the devices have their own controllers, not all of them support standardized methods [8]. In addition to Wi-Fi, many of the devices support standards like ZigBee [23] or Z-Wave [24], which allows them to communicate with their respective controllers and other devices using these standards. For example, the ZigBee standard includes device categories such as lights and switches, which can allow device controllers from different vendors to communicate with them.

One way for controlling and managing the IoT devices is to use software controllers like Home Assistant (HA) [13]. Home Assistant provides a platform, which supports the integration of different devices either directly using their APIs, or using a custom shim layer between the HA and the device. For example, if the HA does not have direct access to a device, an agent that can communicate with the device can expose the device's data over the MQTT bus for HA.

III. Architecture

The goal of our *PraNA* framework is to provide network orchestration for networks that contain different kinds of heterogeneous devices, hubs, and controllers. Furthermore, security is not an after-thought and our orchestrator is capable of orchestrating network security because these networks are inherently insecure, even when the network borders are protected from outside threats. Our *PraNA* Orchestrator, that is at the heart of our *PraNA* framework, is designed to scale to networks that are large enough to require management and orchestration. Such networks require different controllers, for example large Wi-Fi deployments require their own wireless controllers.

An example network is shown in Figure 1, which is split into control and data planes. Each of the components shown in the figure manages their own parts of the network with *PraNA* Orchestrator orchestrating the efforts. Each component

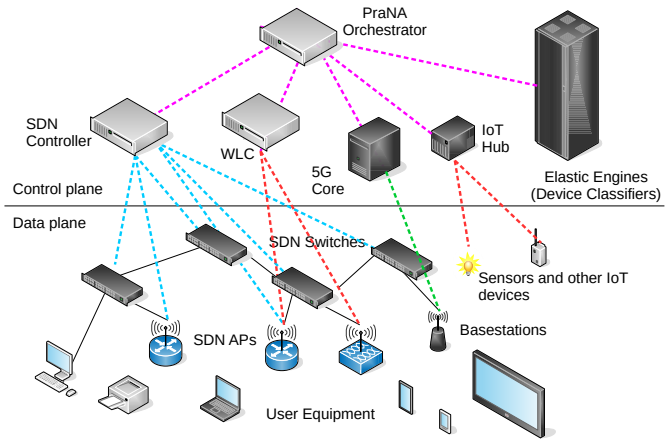

Fig. 1. **PraNA architecture**. *Flows traversing the network are controlled by our SDN Controller, while Wi-Fi APs are managed by the Wi-Fi Controllers and IoT devices are managed by their respective controllers.*

supports APIs that can be leveraged to gather information about their operations, and to control them. The architecture is designed to support both modularity and decentralization, allowing the control plane to be customized and deploying control plane components in different locations, both inside the network and outside for example in the cloud.

In this section, we describe components of the control and data plane and provide an example of how *PraNA* operates.

A. Control and Management Plane

In Figure 1 we present an example of the *PraNA* control and management plane. This plane consists of multiple different controllers and other network functions that control and manage their respective siloes. Some of these network functions may also be capable of providing additional services; the hubs interacting with the Home Assistant [13] exemplify some of the popular services. Regardless, these controllers exchange control plane messages with the *PraNA* Orchestrator via a management bus that uses MQTT [25]. This bus is also designed to allow individual controllers to communicate with the other network elements and controllers. Next, we detail the functions of the shown control elements.

a) The Wireless LAN Controller (WLC): This controller is responsible for managing the Wi-Fi access points (APs) in the network. In large Wi-Fi deployments, individual APs cannot be managed manually and require a dedicated controller that manages the AP configuration and deployment including that Wi-Fi channels used by the APs.

b) IoT hubs and gateways: These hubs manage their own IoT devices. Many IoT devices are proprietary, and use different communication technologies such as ZigBee. Furthermore, some of these devices may not be part of the IP network and only interact with their hubs and gateways [3]. Managing such IoT devices requires gateways that expose open APIs that provide access to the IoT devices and their data. Mineraud *et al.* [8] provide several examples of such hubs and gateways.

c) The Cellular Core: 5G networks and their successors are expected to open and slice their cellular infrastructure to enable enterprises and factories to run their own cellular networks [1], [26], [27]. The base stations that offer cellular connectivity are also expected to support both cellular network and Wi-Fi connectivity [28]. The cellular base stations require a cellular core for managing the connectivity, and OpenAir exemplifies of open cellular cores that provide a north-bound API for integrating with network orchestrators [29].

d) The SDN controller: In *PraNA* we envision a programmable network in which the network traffic flows are managed using SDN controllers. In our prototype we use our custom SDN controller based on Ryu [30] and its design and implementation are presented in our previous work on managing commodity Wi-Fi APs using an SDN controller [10]. This controller implements the features required by the *PraNA* Orchestrator including message exchange via management bus. This message exchange enables our orchestrator to a) receive the network-wide view of the devices in the network, and b) provide network policies that are implemented by our SDN controller.

e) Elastic Engines: We abstract out machine learning based activities such as device classification as activities performed by elastic engines. In our prototype, this engine provides our *PraNA* Orchestrator with device classification services. Specifically, it consumes the packet traces generated by forwarding elements such as APs to classify the devices connected to the network. In our prototype, we use a random forest based classifier, the details of which are presented by Aluthge[12]. Note that machine-learning based device classification is not the primary focus of our work. Instead, our system is designed such that this engine can be easily swapped by other device classification solutions such as the one used by IoT Sentinel [7].

f) Management bus: The management bus ties different elements, including our *PraNA* Orchestrator, together. We implemented the management bus using MQTT, which is a publish-subscribe protocol [25]. We chose the publish-subscribe model because it allows the system to scale by decoupling the message production/generation by its consumption. Specifically, multiple control plane elements may subscribe to the same message topic, and we present examples of the message topics in Table II. Each of the network elements registers to the bus, and subscribes to the relevant topics. After an element has received a command encapsulated in an MQTT message it can act upon it and publish its response via an MQTT message. Furthermore, it can also publish an MQTT message in other situations, for instance, after it has acted upon information received through its APIs.

g) PraNA Orchestrator: Its main functions include receiving information from different sources such as network events from the SDN controller, classification results from the elastic engines, and creating network policies such as isolating or allowing a client to communicate. In addition to these, the *PraNA* Orchestrator also provides a user interface that shows the network state and allows administrators to manually set policies. For implementing the policies, the *PraNA* orchestrator provides the other controllers with the policy along with the required information. We provide three example network policies in §III-C.

Each control plane element is autonomous, and it can perform its duties independently using preset default policies if the *PraNA* Orchestrator is not online. This means that if an element is offline, the network still functions, although not as efficiently. For example, if elastic engines are offline, the network still detects new devices and assigns them default classification. However, as the elastic engines are offline, the default values for the classification and the corresponding network policies need to be set manually. Similarly, if the *PraNA* Orchestrator is offline, the network can still continue to be operational using these default presets. In our future work, we envision to extend our *PraNA* Orchestrator using past work on the IoT Sentinel [7] for building *Elastic Data Analytics Engines* which offer network analytic services.

B. Data Plane

The *PraNA* data plane consists of programmable network forwarding elements and the devices connected to the network. These devices communicate either over wireless links such as cellular, Wi-Fi, or ZigBee, or over wired links such as Ethernet.

In our testbed, we use our past work [10] to allow SDN controllers to manage the flows traversing existing enterprise and commodity access points. This is done to simplify the integration of our system into existing networks. Our current implementation is restricted to devices that communicate using IP protocols over Wi-Fi or Ethernet. Devices using protocols other than IP are currently not managed directly by our *PraNA* controller; instead, our *PraNA* controller can manage them via their IoT hubs.

C. Network Policies

A key aspect of orchestrating a network is the network policies that govern how devices can communicate. We base our policies based on a *Z*ero Trust approach, *i.e.* we do not trust any devices more than we need to [31]. Each device or device type can either have a predefined default policy, or a fine-tuned policy. The default policies include *Restricted*, *Allowed*, and *Blocked*.

Restricted. Restricted policy is granted to those devices that are unknown to the orchestrator, *i.e.* not in the device registry, but do not have known vulnerabilities. This policy allows the devices to have similar access to the network as typical guest network devices, *i.e.* access to only a limited set of relevant services such as DNS, and limited access to the Internet. For instance, *Restricted* devices cannot access services such as printing, or communicating with other devices in the network.

Allowed. Allowed policy grants the device full access to the network, including services that are otherwise restricted. To get this privilege, the device must be registered and contain no known vulnerabilities.

Blocked. A device is blocked when it is known to be vulnerable, it is known to be malicious or due to other reasons set by the administrators. A blocked device cannot communicate

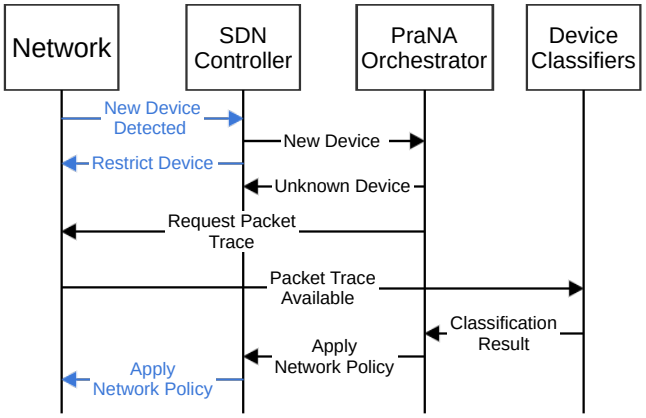

Fig. 2. Sequence of messages over the management bus. OpenFlow control messages between the SDN controller and the network are denoted in blue.

TABLE II
EXAMPLE *PraNA* MQTT TOPICS.

Topic	Description
prana/status	Status of network elements.
prana/network/device/switch/new	New switch added.
prana/network/device/switch/update	Update switch information.
prana/network/device/switch/remove	Switch removed from network.
prana/network/topology/links/new	New link between switches or between a switch and a device.
prana/network/topology/links/update	Update link information.
prana/network/topology/links/remove	Removed link from the network.
prana/network/device/host/new	New host added to the network.
prana/network/device/host/update	Update host information.
prana/network/device/hosts/remove	A host is removed from the network.
prana/classification/result	Result of the classification.
prana/classification/update	Update classification information.
prana/classification/pcap/request	Request a capture from a host.
prana/classification/pcap/response	Capture available for classification.
prana/policy/allow	Change device policy to Allow.
prana/policy/block	Change device policy to Block.
prana/policy/restrict	Change device policy to Restrict.
prana/policy/implemented	A policy has been implemented.

The MQTT topics shown in this table are currently implemented in our PraNA framework. These topics allow network orchestration and management.

with other devices, and all traffic either to or from it is dropped by the network.

We chose these three policies in our prototype because they lay the cornerstone to create and implement complex custom policies that are beyond the scope of this work.

D. Example PraNA Orchestration

Our *PraNA* Orchestrator implements the policies described in §III-C by leveraging the north-bound APIs of the control plane elements described in §III-A. These control plane elements can send and receive messages using either the management bus, or using their specific protocols and APIs. For example, the SDN controller communicates with the programmable forwarding elements such as switches and APs using OpenFlow, while using the management bus to communicate with other control plane elements.

In the Figure 2, we show an example sequence of messages traversing the management bus when a new device joins the network. The relevant management bus MQTT message topics are shown in Table II.

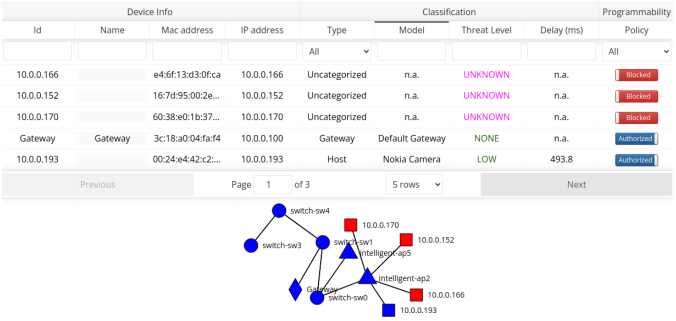

Fig. 3. **The network view created by the *PraNA* Orchestrator.** *In this example, the PraNA Orchestrator shows the network topology with known information of the devices.*

TABLE III
DEVICES USED FOR PROTOTYPE EVALUATION.

Model	Purpose
D-Link DCS-935L	IP camera
D-Link Siren	Wi-Fi Siren
Google Chromecast	Video streaming
Lenovo X280	Laptop with Ubuntu
Nokia 9	Android smart phone
Nokia WBP02	IP camera and air quality sensor
Wemo Switch	Smart power plug

The devices shown here are used in the evaluation and chosen to represent different categories of devices. The categories range from general-purpose devices (laptop and phone) to single-purpose connected devices with a small amount of traffic (smart power plug).

First, the SDN switches in the network detect a new device and inform the SDN controller using OpenFlow messages. The SDN controller then applies the default *Restrict* network policy to the detected device, and informs the *PraNA* Orchestrator of the new device. The Orchestrator sets the device default classification to unknown if the device does not match any previously known devices. Note that Orchestrator does not rely only on the device MAC address and instead uses the behavioral fingerprinting offered by our Elastic Engines for device classification. Consequently, the Orchestrator requests a packet trace for device classification. The switch/AP publishes the availability of a packet trace, and the classifier receives this message because it subscribes to the `prana/classification/pcap/response` topic. Finally, when the classification is ready, the classifiers inform the *PraNA* Orchestrator of the result, which in turn tells the SDN controller to apply specific network policy to the device. Similarly other operations such as manually changing the device policy or if the device roams into another location in the network, are handled through the management bus using relevant topics.

IV. EVALUATION

For our evaluation we consider a network orchestrated by our *PraNA* Orchestrator where each device is classified upon joining the network and is then assigned a network policy based on the classification.

A. Testbed and Methodology

We demonstrate our *PraNA* Orchestrator using our testbed whose key network functions are shown in Figure 1. The control plane of the testbed consists of the *PraNA* Orchestrator, an SDN network controller, and a device classifier. The control plane elements communicate over an MQTT bus, where each element subscribes to the relevant topics and publishes events. For instance, the SDN controller compiles the network topology which is exported to the *PraNA* Orchestrator using the messages summarized in Table II. For device classification, each AP spawns our *PraNA* packet capture agent which captures the first 20 packets of any new devices joining the network, copies it to a location specified in its configuration, and publishes the availability of the packet trace. The data plane consists of five LinkSys WRT-3200ACM access points and a FIT-PC3, each of them running *OVS* [32], and are configured as Intelligent APs as described in our prior work [10]. Our *PraNA* Orchestrator monitors the state of the network, and we present an example network view in Figure 3. This holistic network view allows the *PraNA* Orchestrator to take decisions for implementing policies detailed in §III-C. The *PraNA* Orchestrator sets the network-wide default policies to *Restricted*, *i.e.*, the client devices in our network will be allowed to communicate with other hosts on the Internet, but they will not be able to communicate with other clients unless otherwise specified. An exception to this are flows such as DHCP, DNS, and other flows required to manage the network and assign IP addresses.

We connect devices listed in Table III to the network and measure the duration of each step shown in Figure 2 over the MQTT bus. These devices were chosen to represent different categories of devices that are connected in modern networks, from laptops to IoT devices. Each of the devices will generate different amounts of traffic, which affects the time required for device detection and classification.

Each device has its own way of generating traffic and using services such as DHCP and DNS, and its network traffic characteristics determine its fingerprint that is used by our classifier. For example, a device might only request an IP address and then start publishing sensor readings, causing the capture to take a long time. Another device on the other hand would check for updates and other services on connecting the network, generating a larger stream of traffic.

B. Results

In Figure 4, we present the time taken for each step—packet capture, device classification, and policy implementation—for each device and also the total time, *i.e.* the time from device detection to policy implementation. The policy implementation duration is the time from device classification to the time when the SDN switches have acknowledged that the rules have been updated in their forwarding table.

We observe that the majority of the total time is taken by the packet capture. The duration of the packet capture depends on the amount of data required for packet classification and the network traffic characteristics of the device. In our prototype, our classification engine requires 30 packets, and this number

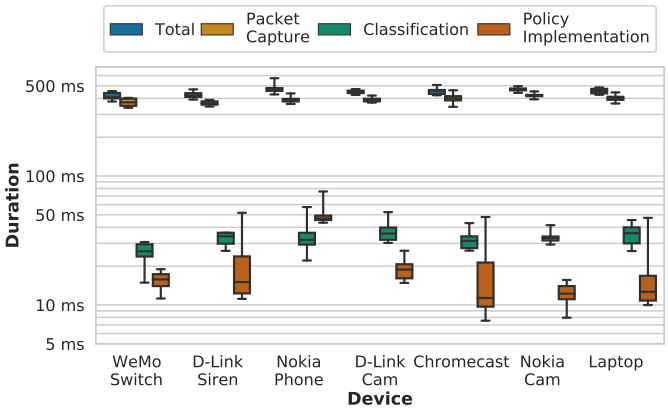

Fig. 4. **Total time.** *We present a) total time taken by the PraNA Orchestrator from the moment a device is detected to the time a network policy has been implemented, b) time for the packet capture, and c) time for classification, and d) time to implement the policy.*

is similar to prior works [7]. This packet capture time can be tuned by adjusting how much traffic is captured when the device joins the network. The rest of the events, *i.e.* detection, classification, and implementing the policy, take on average about 80 ms in our testbed.

We acknowledge that our testbed is only designed to demonstrate the *PraNA* orchestration in a small-scale test environment, where network delays are small. Similarly, even if the classification algorithm is optimized to reduce the time required for classification, the proportional amount of time taken still would continue to remain small compared to the packet capture. In contrast, our results highlight that classifiers requiring fewer packets for classification would have a significantly larger impact on the total time.

Evaluating the performance of our *PraNA* Orchestrator in large-scale enterprise networks is difficult primarily because our solution is not enterprise-grade. However, a dataset containing Wi-Fi device associations to *eduroam* from 2014 to 2015 gives a picture of how many network events we could expect from a large network [33]. In September 2014, the authors of the eduroam dataset observed 7657792 associations from 36045 unique clients, and there were 9.6 events per second during the peak hour. While this only accounts for Wi-Fi devices, it gives us an estimate on how often devices join a network. We believe that even our system could scale to support 10s of events per second because the computational load of our *PraNA* Orchestrator is negligible. This is primarily because control plane services like classification and SDN controllers can be performed on devices fine-tuned to support the computational and network load required for those services.

V. Concluding Remarks and Future Work

In this paper, we have presented our *PraNA* framework for network orchestration. *PraNA* is a modular system, that allows the usage of different off-the-shelf components to be tied together over a common management bus. The decentralized nature allows *PraNA* components to be deployed in different locations or in a single server depending on the network size.

The key benefit of the *PraNA* is the network orchestration of different components and controllers. In the example presented

here, the *PraNA* brings together network management, device classification, and network security. These components can be outsourced from third-party vendors as long as there is a way to use their available APIs to connect them to the network management bus either directly or using a SHIM layer. For example, the device classifiers could be provided from a security vendor with a suitable shim layer between the classifiers and *PraNA*.

Another example is different device controllers. Depending on the controller, the high-end controllers such as Wi-Fi controllers typically expose SNMP or REST APIs, while lower-end IoT controllers may expose similar APIs, or use their own custom APIs. Regardless of the controller, a shim layer between the controller and *PraNA* Orchestrator can be built to allow *PraNA* to communicate with the controller. A good example of this is the Home Assistant, which uses custom integrations to build and manage home automation using off-the-shelf IoT devices.

Similarly, as the traffic is controlled through programmatic network elements, instead of restricting or blocking the devices, the traffic of the device can be redirected to different network service chains. These chains can then be used for more thorough traffic analysis and security.

If we take this even farther, the fundamentals of *PraNA* can be also applied to containers. By using the *PraNA*, the traffic originating from the containers can be redirected to for example NIDS systems that could determine if the containers are behaving as designed or are malicious. Similar principles could also be applied to 6G networks, that will contain a large variety of heterogeneous devices.

The *PraNA* framework is also scalable. In our test environment, each component except the packet capturers in the APs are single-threaded. However, our evaluation and the insights from the *eduroam* dataset show that *PraNA* can cope with larger networks with only small tweaks such as parallelizing the classification engines.

As the *PraNA* framework is modularized, it can be deployed without all components. For example, if the network does not support SDN, the device detection can be gleaned from other network services such as DHCP or Wi-Fi controllers. While the current protections in *PraNA* are implemented using SDN and OpenFlow, other methods may be available to achieve similar results such as quarantining a Wi-Fi client through the Wi-Fi controller.

Acknowledgments

The authors would like to thank Business Finland PraNA project and the Academy of Finland IDEA-MILL Project.

References

[1] A. Pouttu, F. Burkhardt, C. Patachia, L. Mendes, G. R. Brazil, S. Pirttikangas, E. Jou, P. Kuvaja, F. T. Finland, M. Heikkilä *et al.*, "6g white paper on validation and trials for verticals towards 2030's," *6G Research Visions*, no. 4, 2020.

[2] M. Bauer, M. Boussard, N. Bui, J. De Loof, C. Magerkurth, S. Meissner, A. Nettsträter, J. Stefa, M. Thoma, and J. W. Walewski, "Iot reference architecture," in *Enabling Things to Talk*, 2013, pp. 163–211.

[3] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta, "The Internet of Things Has a Gateway Problem," in *Proceedings of HotMobile'15*, 2015, p. 27–32.

[4] J. Mineraud and S. Tarkoma, "Toward interoperability for the internet of things with meta-hubs," *arXiv preprint arXiv:1511.08063*, 2015.

[5] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 45–59, 2020.

[6] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE Internet of things Journal*, vol. 1, no. 3, pp. 265–275, 2014.

[7] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2177–2184.

[8] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of internet-of-things platforms," *Computer Communications*, vol. 89, pp. 5–16, 2016.

[9] I. Hafeez, A. Y. Ding, L. Suomalainen, A. Kirichenko, and S. Tarkoma, "Securebox: Toward safer and smarter iot networks," in *Proceedings of the ACM Workshop on Cloud-Assisted Networking*, 2016.

[10] S. Hätönen, P. Savolainen, A. Rao, H. Flinck, and S. Tarkoma, "SWIFT: bringing SDN-Based flow management to commodity Wi-Fi access points," in *IFIP Networking Conference*, May 2018, pp. 172–180.

[11] S. Hätönen, J. Mineraud, A. Rao, H. Flinck, and S. Tarkoma, "The 5d approach to control and manage smart spaces," in *Proceedings of IEEE EuCNC*, 2017, pp. 1–6.

[12] N. Aluthge, "Iot device fingerprinting with sequence-based features," 2018.

[13] "Home Assistant," https://www.homeassistant.io/, visited 2021-10-25.

[14] "IFTTT," https://ifttt.com/, visited 2021-10-25.

[15] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case, "Simple Network Management Protocol (SNMP)," RFC 1157, May 1990. [Online]. Available: https://rfc-editor.org/rfc/rfc1157.txt

[16] "Ansible," https://www.ansible.com/, visited 2021-10-25.

[17] N. F. Saraiva de Sousa, D. A. Lachos Perez, R. V. Rosa, M. A. Santos, and C. Esteve Rothenberg, "Network service orchestration: A survey," *Computer Communications*, vol. 142-143, pp. 69–94, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366418309502

[18] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, and G. M. Pérez, "A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets," *IEEE Communications Surveys Tutorials*, vol. 23, no. 2, pp. 1048–1077, 2021.

[19] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018.

[20] S. Marchal, M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan, "Audi: Toward autonomous iot device-type identification using periodic communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, 2019.

[21] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A survey of securing networks using software defined networking," *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 1086–1097, 2015.

[22] F. Restuccia, S. D'Oro, and T. Melodia, "Securing the internet of things in the age of machine learning and software-defined networking," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4829–4842, 2018.

[23] "Zigbee," https://csa-iot.org/, visited 2021-10-25.

[24] "Z-Wave," https://z-wavealliance.org/, visited 2021-10-25.

[25] "MQTT," https://www.mqtt.org/, visited 2021-10-25.

[26] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, "Network slicing for 5g: Challenges and opportunities," *IEEE Internet Computing*, vol. 21, no. 5, pp. 20–27, 2017.

[27] S. Zhang, "An overview of network slicing for 5g," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 111–117, 2019.

[28] "Nokia - flexi zone," https://www.nokia.com/networks/mobile-networks/flexi-zone/, visited 2021-10-25.

[29] "Openair cloud ran (c-ran)," https://openairinterface.org/use-cases/cloud-ran-c-ran/, visited 2021-10-25.

[30] "Ryu SDN Controller," https://osrg.github.io/ryu/, visited 2021-10-25.

[31] S. W. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," 2020.

[32] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *In Proc. of USENIX NSDI*, 2015.

[33] L. Pajevic, V. Fodor, and G. Karlsson, "Revisiting the modeling of user association patterns in a university wireless network," in *IEEE WCNC*. IEEE, 2018, pp. 1–6.