mtunzi: a programmable approach to software-defined network access control and captive networking for low-resource setting

Innocent Ndubuisi-Obi Jr
innoobi@cs.washington.edu

Nussara ‘Firn’ Tieanklin
nussara@cs.washington.edu

I. Introduction and Background

Small wireless internet service providers and community networks, like any other network providers, are typically reactive: that is, they rely on a number of devices and complex configurations to get their networks up and running and secure. Even more important is the need to support wired and wireless connection, particularly those that allow BYOD policies. This creates headaches to large enterprise networks and can make or break the viability of a small WISP or community network. A widely used approach for managing access to the network is via a captive network portal. Captive networking describes the configuration of a network such that whenever a device is voluntarily attached, network access is limited until some requirements have been fulfilled. When interacting with a captive portal, a user is typically required to use a web browser (captive portal) to fulfill requirements imposed by the network operator, such as reading advertisements, accepting an acceptable use policy or providing some form of credentials.[1] Through the captive portal the network operator is able to allow or block traffic (via HTTP redirection or DNS poisoning) in ways that give tremendous control in how the network is managed and monitored

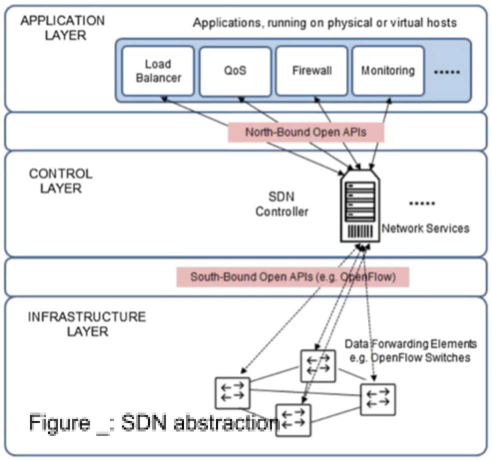


Figure _: SDN abstraction

This activity can be broadly understood as network access control. Network access control emphasises on providing the visibility, capability to control access, and most importantly ability to strengthen the network security infrastructure by enforcing the policies and using profiling. Software-defined network (SDN) unsurprisingly has provided many examples of programmable dynamic network access controller technologies that can further simplify the work of network operators. SDN focuses on physically decoupling the packet forwarding process of the network (the data plane DP) from the routing process (control plane). It centralizes the networking intelligence in the Controller which has a

[1] K Larose, Agilicus, D. Dolson, and H. Liu, “CAPPORT Architecture: draft-ietf-capport-architecture-08.” Internet Engineering Task Force, May 11, 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-capport-architecture-08

comprehensive view of the entire network.[2] The SDN abstraction can be understand as three main components: the application plane (which extends the network's services), the control plane (which houses the controller and other important network services), the infrastructure layer (which includes the data forwarding elements (switches), and the North-bound API which support communication between the control and application layer and the South-bound API which supports communication between the control and the data planes (openFlow).[3]

Openflow is the most well known South-bound API for programming switches in SDN. It provides a uniform abstraction for configuring different network devices. It gives network administrators the possibility to remotely reconfigure the forwarding tables at runtime, collect network statistics, when no match exists in the table for any packet, ro redirect the packet to controller for future processing. The key insight of Openflow and SDN is the separation of the control and forwarding plane with a centralized intelligence in the controller which offers opportunities to reduce network control and management complexity.

II. Literature

Many authors have identified the potential benefits of using SDN to support network access control, or sometimes called port-based access control. Extreme design proposals like Ethane[4] demonstrated the value of access control systems based on reactive control: every flow is intercepted and sent to a central controller which authenticates users and enforces flow-level policies. Building of Ethane[5] and 4D[6], Resonance[7] supports dynamic access control policies by giving each device a "security class" that determines how traffic flows to and from it and a "state"

---

[2] F. Nife and Z. Kotulski, "New SDN-Oriented Authentication and Access Control Mechanism," in *Computer Networks*, vol. 860, P. Gaj, M. Sawicki, G. Suchacka, and A. Kwiecień, Eds. Cham: Springer International Publishing, 2018, pp. 74–88. doi: 10.1007/978-3-319-92459-5_7.

[3] F. Nife and Z. Kotulski, "New SDN-Oriented Authentication and Access Control Mechanism," in *Computer Networks*, vol. 860, P. Gaj, M. Sawicki, G. Suchacka, and A. Kwiecień, Eds. Cham: Springer International Publishing, 2018, pp. 74–88. doi: 10.1007/978-3-319-92459-5_7.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '07*, Kyoto, Japan, 2007, p. 1. doi: 10.1145/1282380.1282382.

[5] Casado, Martin, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. "Ethane: Taking Control of the Enterprise." In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications - SIGCOMM '07*, 1. Kyoto, Japan: ACM Press, 2007. https://doi.org/10.1145/1282380.1282382.

[6] Greenberg, Albert, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. "A Clean Slate 4D Approach to Network Control and Management." *ACM SIGCOMM Computer Communication Review* 35, no. 5 (October 6, 2005): 41–54. https://doi.org/10.1145/1096536.1096541.

[7] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, Barcelona, Spain, Aug. 2009, pp. 11–18. doi: 10.1145/1592681.1592684.

that determines which security class should be applied at a given time. The authors recognized that the use of middleboxes, IDS, and collection of configurations, post-hoc network management led to “a plethora of independent, difficult-to-manage devices that interact in unexpected ways, resulting in weaker security incorrect operation, or both.” Resonance was designed to decrease the complexity of managing networks. Others like AuthFlow[8], FlowNAC[9], FlowIdentity[10], provide authentication and access control using IEEE 802.1x with RaDIUS authentication. Apart from the 802.1x proposal, the previously mentioned solutions use captive portals as their authentication mechanisms which require Layer 3 configurations (DHCP) and ephemeral IPs, as opposed to Layer 2 approaches (based on MAC address) .

Figure _ : comparisons of SDN software access solutions

System	**Authentication Mechanism**	**Layer**	**Behavior**
Ethane	captive portal	L3	reactive
Resonance	captive portal	L3	reactive
AuthFlow	captive portal	L3	reactive
FlowNAC	hostapd	L2	proactive
FlowIdentity	hostapd	L2	reactive
Nife&Koutoulski	hostapd	L2	reactive
SuperFlow	SAVI	L2.5	reactive
SafeFlow	Digital credentials	L3	NA
802.1X Benzekki	hostapd	L2	reactive
Enhanced Campus Network Auth	captive portal	L2.5	reactive

Solutions like AuthFlow and FlowNac associate flows to services and these are uniquely identified and decisions based on user identity. The project highlights some general differences in high-level approaches to the design of SDN-based network access control systems:

[8] D. M. Ferrazani Mattos and O. C. M. B. Duarte, “AuthFlow: authentication and access control mechanism for software defined networking,” *Ann. Telecommun.*, vol. 71, no. 11–12, pp. 607–615, Dec. 2016, doi: 10.1007/s12243-016-0505-z.

[9] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, “FlowNAC: Flow-based Network Access Control,” in *2014 Third European Workshop on Software Defined Networks*, Budapest, Hungary, Sep. 2014, pp. 79–84. doi: 10.1109/EWSDN.2014.39.

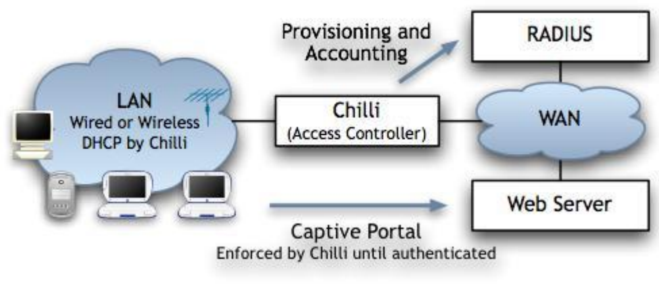
[10] S. T. Yakasai and C. G. Guy, “FlowIdentity: Software-defined network access control,” in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, San Francisco, CA, Nov. 2015, pp. 115–120. doi: 10.1109/NFV-SDN.2015.7387415.

role-based firewall, service-based authentication, and identity-based authentication. Each solution entails different design tradeoffs from the point of view of both the networking administrator and user. Mtunzi builds on these approaches and aims to build a system that can also be used as a testbed to experiment with different AAA mechanisms.

III. Current problems, goals and motivation

Our focus of programmable approaches to network access control is focused on a specific workload: that of the small WISP and community network. Wireless Internet service providers (WISP) are the internet service providers that provide a network based on the wireless networking. WISPs provide a smaller coverage than the commercial networks but affordable broadband to customers at fixed locations such as community centers, schools,or residences. Community networks (CN), or decentralized telecommunications that people build and operate themselves[11]. CNs are built by people for people, often run by volunteers to help under-resourced communities. Today, there are many available SDN tools to support small WISPs and community networks in managing and securing their networks. Current existing solutions range from vendor-supported captive portals and controllers to open-source software access controllers.

Open source software access controller.



The most popular open-source option for network access control is the Coova-chilli project[12]. Coova-chilli is a software access control that provides a captive portal and walled garden with RADIUS support to access provisioning (authentication and authorization) and accounting. One of the authors has spent considerable time interrogating the architecture and chill after facing significant limitations in a live production version of the system. A key aspect of Coova-chilli's design is the use of tunneling and encapsulation using TUN/TUP interfaces and the shuffling of packets from user to kernel space to the main chilli controller. This is a main source of cpu load[13] and throughput[14] performance problems that plagues the system.

[11] J Bidwell, Nicola. "Wireless in the Weather-World and Community Networks Made to Last." In Proceedings of the 16th Participatory Design Conference 2020 - Participation(s) Otherwise - Volume 1, 126–36. PDC '20. Manizales, Colombia: Association for Computing Machinery, 2020. https://doi.org/10.1145/3385010.3385014.

[12] https://coova.github.io/CoovaChilli/

[13] https://github.com/coova/coova-chilli/issues/466

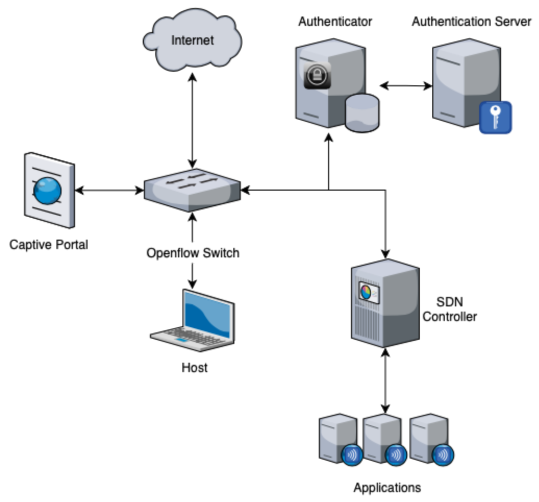
[14] https://github.com/coova/coova-chilli/issues/61

Coova-chilli is also a monolithic system that is brittle and non-performant, but demonstrates a powerful abstraction that is a main pillar of the SDN paradigm: the software access controller. We propose to continue the insight and complete decoupling the control and data plane therefore introduce a fully software-defined network access controller. (this would require an open-flow enabled switch or an OVS).

Captive networking standards. Recent RFCs captive portal standards (RFC8908, RFC8910, RFC8952) have proposed enhancements to the way captive portals are designed and implemented from both the client (captive portal) to the vendor-side (laptop, phones, etc). The RFCs aims to provide a more user-friendly captive portal experience but does not explore security innovation or potential approaches to support more secure and convenient authentication. Although they target the usability of the portal, they don't do much to support dynamic network access control that are in tandem with captive networking.

IV. The mtunzi system

Figure _: Authentication Setup



Mtunzi is a system for software-defined network access control and captive networking for low-resource settings. It is aimed toward the networking needs of small WISPs and community networking operators. With mtunzi we are simply trying to use a software-defined approach to (1) registering and authenticating switches, (2) authenticating hosts and users, and (3) dynamically managing hosts and users. The goal is to design and develop a network access control and authorization mechanism for SDN networks.

Feature 1: Dynamic and stateful policy specification.

Similar to abstractions provided in Resonance and Ethane, Mtunzi used the flow abstractions to represent the high-level dynamic policies and rules: state, matches, actions, and transitions. In our implementation a host's State is represented as a Session object. A Session can be thought of as a product type of a record type containing the following elements: mac address, ipv4 address, output port on the switch, datapath for the switch. Matches and Actions are coupled together to execute openflow operations. A Match is an Openflow struct that takes a number of arguments - from the switch input port and ethernet destination address to the VLAN id and MPLS label. Actions as operations to be completed on the subset of flows that match the arguments of a Match structure. An Action could modify, delete, or output a flow rule. A Transition is not Openflow structures but represents operations in the controller application that must be carried out whenever the state of a host, flow, or port is changed.

The current implementation of Mtunzi provides two possible states for a connected device: *Authenticated* and *Unauthenticated*. Figure _ demonstrates the matches and actions relevant to each state. Transitions are intended to maintain the correctness of the flow table and flow rules. When a new port is registered on the switch a *PACKET_IN* message is sent to the controller. On start_up, the switch is configured with lower priority flow rules that will flood DHCP and ARP traffic while forwarding all new traffic to the controller. On the *PACKET_IN* message, the controller learns the FIB and specially processes TCP packets. For each TCP connection (in_port and out_port), the controller will mangle the packet in the following two ways: forward action and backward action. In the forward action, the controller will write a flow rule that will change the destination mac address and ip of the packet to that of the captive portal. In the reverse direction, it will rewrite any packet from the captive portal to the entail source such that the source mac address and ip is changed from the captive portal to that of the original destination. This rewriting provides the ability to redirect all TCP traffic to the captive portal. Once a user transitions to the authenticated state, the default rules are removed and the user is given access and authorization to use all of the network's resources. The controller application that facilitates the redirection also forks a long-running green thread that listens for authentication status messages from the Authenticator.

Figure _: Match and Action sets			
Unauthenticated		*Authenticated*	
Match	**Action**	**Match**	**Action**
ARP (0x806)	FLOOD	ARP (0x806)	FLOOD
UDP src=68 dst=67	FLOOD	UDP src=68 dst=67	FLOOD
UDP dst=53	FLOOD	UDP dst=53	FLOOD
TCP dst 80/443	REDIRECT	TCP dst 80/443	FLOOD
*	DROP	*	DROP

Feature 2: IP source and MAC source address binding and validation.

Mtunzi uses a mixture of IP source and mac address validation to identify devices and to tie every packet to an authenticated source. Similar to the methods used in Superflow inspired by the SAVI proposal, Mtunzi establishes a triangle relationship of IP address, MAC address, and uplink-port for each host. This binding can be used as an anti-spoofing mechanism to deter IP address protocol sniffing and support ingress filtering, source address encryption, and many other features. The assumption is that there is a registration process in place that associates a user to a MAC address. It is not necessary that the IP address be statically assigned to that user. Some future work will explore other authentication mechanisms beyond the common Layer 3 captive portal[15].

Feature 3: Event-based, composable and reactive controller applications

Default flows for each switch will redirect all host traffic to the captive portal. The controller maintains local state keeping track of active hosts and sessions. In order for a device to gain access they require authentication from the central authentication server. Figure _ demonstrate these steps. We assume that the host device has already received a dynamic IP address from a DHCP server. First, a host device's MAC address, port and IP is learned by the learning switch application on the controller. The default flow rules sent by the controller to the switch will redirect the device to the captive portal. On the captive portal page, the user enters their credentials and the form is submitted to the Authenticator. The Authenticator initiates a RADIUS session and confirms the user's credentials. On success it informs the Controller which then initiates the transition operations from the *Unauthenticated* to the *Authenticated* state. If the rules are successfully added the Controller communicates this event back to the Authenticator which then informs the client of the Captive portal of their captive status. There are two main features that make this process possible. First, the controller is running two applications: a layer 2 learning switch, and tcp redirection application. The second hooks into the Ryu openflow event loops and is able to respond to PACKET_IN messages that contain TCP packets. The second is the tcp redirection application's ability to spawn a lightweight monitoring thread that creates a connection to the Authenticator and listens for user authentication updates. The mechanism of latching on to openflow event hook's support the creation of a number of features on top of the Ryu controller.

[15] …..

Figure _: Authentication Setup

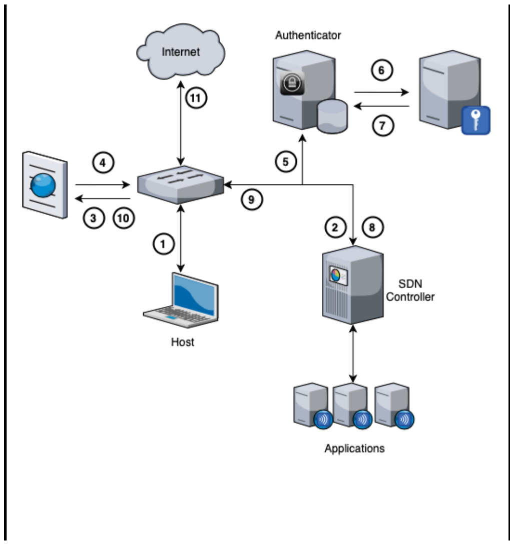
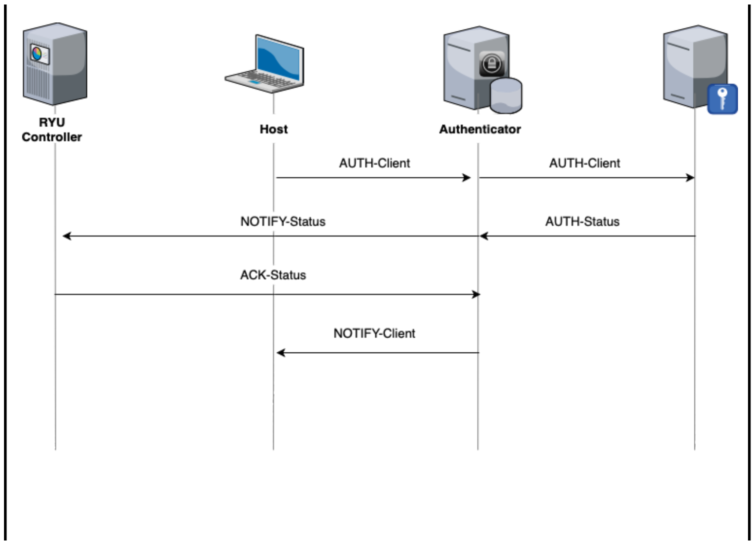
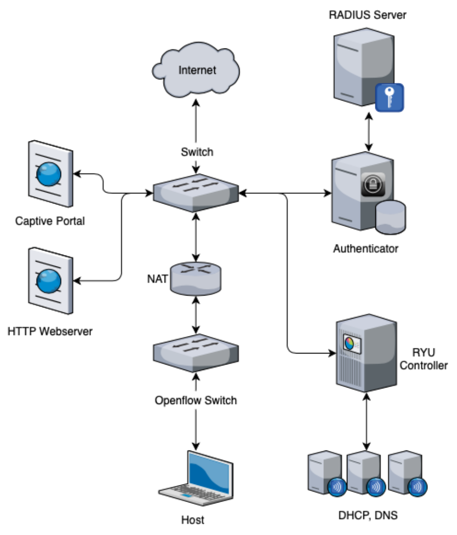


Figure _: Authentication Diagram





V. Implementation and Testbed

Mininet environment: Testing the application

Our testbed environment foruses two Ubuntu 18.06 VMs running on VirtualBox. Mininet version 2.3 is installed on two machines while the . The first VM runs the RYU controller which includes two applications: a L2 Learning switch and a custom captive control application. The second is our implementation of an event-based North-bound application that statefully injects and removes flows and policies. They use the Mininet CLI to create a topology of 5 hosts that run in a network namespace that is natted to that of the host VM. IN order to test the redirection, two python applications are run on the host network: the first a stub captive portal page and the other a simple web server. The README documents how to run and interact with this system. Given that the two components (the frontend and the Mininet testbed) are not integrated, the README includes instructions on how to use Mininet to test the captivity of the testbed network.

Capport application and authentication setup: Testing the authentication process

We implemented the captive portal using the Flask[16] web application framework written in Python version 3.6.1. The application runs on the virtual environment. One of the main implementation details is that mtunzi uses the Eventlet networking library[17] to support the non-blocking event-based sockets to listen for messages. The reason that we chose to use the non-blocking sockets is that we need to be listening to the response/result of the authentication process from the radius server and whether the flow rules are successfully adjusted. We use FreeRADIUS[18]as a RADIUS client running on the virtual machine (VM) on the local machine. The VM runs on Ubuntu 16.04.7 LTS. Due to the time constraint, the current implementation for mtunzi does not have the database supported. But the future work will be using a SQL database for back-ends. Insteads we verify login credentials against the *users* text file via the LDAP integration that comes with FreeRADIUS. The *clients.conf* file is used to define which client to be the FreeRADIUS server. Authenticator will expect a notification from the controller via a non-blocking socket whether or not to allow the user to access the internet. The flask application is expected to redirect to the appropriate page based on the authentication result messages from the Authenticator.

Reflections

In the future work, we would like to provide a unified higher-level abstraction for specification and configuration that allows users to adjust the OpenFlow rules easily. This includes but is not limited to using the NETCONF protocol, YANG integration, etc. Regardless of the benefits that the software-defined network access controller has, it is still not the simplest solution yet. Particular for the small WISPs and community network volunteers who may have less experience in networking and have fewer skilled network engineers. Though we try to simplify our implementation as much as we can, extracting simplicity is much harder which comes with the design tradeoffs. Not to mention that OpenFlow itself also has its limitations of what you can do. The current design presents multiple potential points of failure that may occur. And the redirect implementation is not ideal from a usability perspective and lacks compatibility with existing CAPPORT infrastructure. The next iteration of work we plan to use stub-proxy for HTTP Redirect instead. Because___________. Lastly, the full programmable control requires implementing DHCP additionally as application as well as other potentially services (NAT, firewall, etc) on our own.

[16] https://flask.palletsprojects.com/en/2.0.x/
[17] https://eventlet.net/doc/
[18] Walt, Dirk Van der. Freeradius: Beginner's Guide: Manage Your Network Resources with Freeradius. Birmingham: Pack Publ., 2011.