Po-Fi: Facilitating Innovations on WiFi Networks with an SDN Approach

Zhou Shi[a], Ye Tian[a,*], Xiaodong Wang[a], Jiangyu Pan[a], Xinming Zhang[a]

[a]*Anhui Key Laboratory on High-Performance Computing*
School of Computer Science and Technology, University of Science and Technology of China, Hefei, 230026, Anhui, China

Abstract

Most of softwareized WiFi networks rely on dedicated softwares for realizing novel wireless functionalities. However, such an approach is not flexible enough, and introduces management complexity. On the other hand, the programmable forwarding pipeline, which models packet processing as multiple stages of forwarding rules, exhibits great flexibility in facilitating network innovations, and has been recognized as a consensus for abstracting SDN data plane. In this paper, we present *Po-Fi*, an architecture for software-defined WiFi networks. Based on Protocol-Oblivious Forwarding (POF), Po-Fi follows the SDN consensus by abstracting WiFi AP as a programmable forwarding pipeline, and provides a rich and unified programmability to enable user to realize novel wireless functionalities with forwarding rules. Comparing with previous approaches, Po-Fi is highly flexible in facilitating WiFi innovations. To show this, we realize a wide range of use cases with Po-Fi, including per-client virtual AP (VAP), seamless mobility, multi-AP MPTCP data transmission, smooth remote AP switching for wireless backhaul, and TDMA MAC scheduling. We implement a Po-Fi prototype with commodity hardware, and evaluate the use cases with real-world experiments. The results show that the Po-Fi facilitated innovations can effectively improve the WiFi network services.

Keywords: WiFi, software-defined networking (SDN), programmable forwarding pipeline, Protocol-Oblivious Forwarding (POF), virtual AP, mobility, MPTCP, WDS, TDMA

1. Introduction

WiFi is everywhere in our life. Cisco predicts that by the year 2022, there will be 549 million WiFi hotspots on the planet [1]. On the other hand, legacy WiFi suffers from inherited deficiencies such as lack of mobility support and inefficient medium access, how to facilitate innovations for overcoming the deficiencies is a critical problem [2].

In the past decade, network innovations are driven by softwareized technologies, including network function virtualization (NFV) and software-defined networking (SDN). The state-of-the-art SDN technologies such as OpenFlow [3], P4 [4], POF [5] are initially targeted at wired networks, and they share a common consensus by abstracting SDN switch as a *programmable forwarding pipeline* that is composed of multiple stages of "match+action" tables. The pipeline abstraction is highly flexible in facilitating a wide range of innovations over the wired SDN (e.g., [6], [7], and [8]), and is supported by emerging programmable hardwares [9][10].

*Corresponding author
Email address: `yetian@ustc.edu.cn` (Ye Tian)

Unfortunately, such a progress does not take place in wireless networks. One reason is that wireless link layer is more complicated than the wired one; another reason is that OpenFlow, the most widely-used SDN southbound interface, does not support 802.11 [3]. Existing softwareized WiFi networks employ dedicated softwares such as `hostapd` [11] and `Click` [12]. For example, CloudMAC [13] runs hostapd for providing 802.11 MAC functionalities on virtual machines (VMs), and connects the VMs to physical thin access points (APs) with an OpenFlow network; Odin [14] implements the split-MAC with a Click modular router agent in AP and a centralized controller, and leverages OpenFlow for Ethernet backhaul. Although some novel functionalities have been realized with these approaches, however, there are limitations.

First, comparing with "white box" SDN switch, dedicated software has limited flexibility, as its built-in functionalities can not be changed by control plane programming. For example, hostapd doesn't allow user to program its low-level behaviors. Although Odin enables some novel use cases [14][15], however, its "Click+OpenFlow" design is based on an implicit assumption that an Odin AP works only under the infrastructure mode for connecting wireless clients to wired Ethernet backhaul. Under this assumption, the Click agent has a built-in functionality to transform the received 802.11 frames to Ethernet frames, and deliver them to the OpenFlow switch. However, a WiFi AP may work under other modes, for example the *wireless distribution system* (*WDS*) mode, in which the AP doesn't connect to Ethernet, but provides wireless backhaul by relaying 802.11 frames between wireless client and a remote AP [16]. In such a scenario, the "Click+OpenFlow" design is no longer applicable.

Second, dedicate software requires a separate control channel besides OpenFlow; moreover, it does not provide the per-flow control as in OpenFlow, but has other control granularities such as the per-client control in Odin. The coexistence of heterogeneous control channels and granularities surely increase management complexity.

Motivated by the observations, in this paper, we present *Po-Fi*, a novel software-defined architecture for WiFi networks. Po-Fi is based on *Protocol-Oblivious Forwarding* (*POF*) [5], a protocol-independent SDN data plane, and it follows the SDN consensus by abstracting WiFi AP as one single programmable forwarding pipeline. Different from dedicated WiFi softwares, a Po-Fi AP doesn't have any built-in functionality. Instead, Po-Fi provides a unified programmability, with which a user can realize wireless functionalities by programming the pipeline with forwarding rules from a centralized controller, thus is more flexible and easier to manage.

To demonstrate Po-Fi's flexibility, we have realized a wide range of network services as use cases: 1) We have realized a *seamless mobility* service that smartly selects new AP for a mobile client to connect to, and maintains the client's throughput during the handover. 2) We also enable a multi-AP *Multipath TCP* (*MPTCP*) data transmission with Po-Fi, which allows a client with only one WiFi interface to connect to multiple APs for carrying its simultaneous MPTCP subflows. 3) To demonstrate Po-Fi's easiness-to-use, we customize a novel service by combining seamless mobility with multi-AP MPTCP, so that a mobile client can place its MPTCP subflows on multiple APs, and seamlessly switch the APs with its movement. 4) We realize a WDS bridge AP for providing *wireless backhaul* with Po-Fi, and enables the mobile bridge AP to switch its remote AP smoothly without harming the throughput. 5) Our last use case is to realize a centralized *time-division* *multiple access* (*TDMA*) MAC scheduling, which dynamically allocates time slots on TDMA APs for improving a client's throughput in a *hidden terminal* environment. Note that some of the use cases are first realized in the literature, and as far as we know, none of the previous softwareized WiFi networks can enable all these use cases.

We have implemented a Po-Fi prototype with commodity hardware. We build a real-world testbed and evaluate the use cases. The results indicate that the Po-Fi facilitated innovations can effectively overcome WiFi's inherited deficiencies and improve the network services. In summary, our contributions are three fold:

- **Architecture**: We present a novel software-defined architecture for WiFi networks named Po-Fi that is highly flexible and easy to program.
- **Use cases**: We realize a wide range of use cases that provide various novel WiFi services with Po-Fi.
- **Prototype and evaluation**: We develop a Po-Fi prototype with commodity hardware, and evaluate the use cases with real-world experiments.

The remainder part of this paper is organized as follows. We discuss related works and introduce POF in Section 2; Section 3 presents the Po-Fi architecture, and elaborates its key concepts and components; We describe the exemplary use cases in Section 4; Section 5 presents the prototype implementation and real-world experiments; We conclude in Section 6.

2. Background

2.1. Related Work

2.1.1. Softwareized WiFi networks

Dedicated softwares such as hostapd and Click are widely adopted for developing a softwareized WiFi networks. CloudMAC [13] runs hostapd for providing 802.11 MAC functionalities on virtual machines (VMs), and employs an OpenFlow network to interconnect the VMs with physical thin APs. ÆtherFlow [17] uses hostapd for WiFi management, and extends OpenFlow for querying and configuring hostapd. However, hostapd doesn't allow users to program its low-level behaviors, which reduces its usage in realizing novel and non-standard functionalities.

Odin [14] follows the one-BSSID-per-client idea to allocate each wireless client a dedicated per-client light virtual AP (LVAP) that is decoupled from the physical AP, and manages the LVAPs from a central controller. Odin realizes LVAP with the Click modular router [12], and uses OpenFlow for Ethernet backhaul. 5G-EmPOWER [18] [19] provides a unified control for heterogeneous RANs including LTE and WiFi, and its WiFi AP is implemented with Click and Open vSwitch (OVS) like Odin. As previously discussed, the "Click+OpenFlow" design is based on an implicit assumption that an AP should always connect wireless clients to wired Ethernet, but in scenarios such as WDS, the design is not be applicable.

2.1.2. Innovative WiFi services

Many enterprise WiFi systems slice multiple virtual APs (VAPs) from a physical AP. As an extreme case, the enterprise WiFi system of Fortinet [20] allow per-station VAP by assigning each station a distinct BSSID, and Odin [14] realizes per-client LVAP and enable it to migrate with client movement. BeHop [21] enables virtual AP with multiple BSSIDs on physical AP, and extends OVS for a centralized controller to remotely configure the APs.

Novel WiFi network architectures usually take fast inter-AP handover as an exemplary use case. OpenRoads [22] employs multiple interfaces to connect to multiple APs. ÆtherFlow [17] reduces packet loss during handover by proactively multicasting downlink traffics to all the APs that the client may connect to. Odin [14] realizes fast and transparent handover by migrating mobile client's LVAP. Zeljkovic *et al.* [23] propose a control plane algorithm over 5G-EmPOWER for proactively migrating mobile client's LVAP, with the global knowledge on locations and mobility of APs and clients.

MPTCP is promising for enhancing reliability and capacity of TCP connections in mobile wireless environment. Croitoru *et al.* [24] argue that a mobile client should connect to multiple APs with MPTCP instead of pursuing fast handover, and propose a client-side AP selection method and sender-side congestion control algorithm. Palash *et al.* [25] propose to suppress a client's additional MPTCP subflows for avoiding an AP to be overwhelmed by too many subflows, while maintain the fairness among clients. Xu *et al.* [15] extend Odin to enable a client to employ multiple APs for data transmission, and present a heuristic algorithm for assigning flows to APs for maximizing the overall throughput [26].

TDMA is an effective way for overcoming the inefficiency of 802.11's distributed coordination function (DCF) MAC in hidden/exposed terminal environments. Soft-TDMAC [27] is a TDMA-based MAC for wireless mesh network. OpenTDMF [28] enables TDMA in WiFi access network, and provides a flow table interface analogous to OpenFlow for user to specify how a flow accesses wireless channel. hMAC [29] is a patch to the `ath9k` software driver for realizing hybrid TDMA/CSMA MAC.

Unlike previous works which provide only configurability or limited programmability, and typically focus on one or a few use cases, Po-Fi provides a rich programmability for controlling low-level behaviors of WiFi APs under the SDN paradigm. Thanks to such a programmability, Po-Fi can facilitate a wide range of use cases. Some of the use cases are first realized in the literature, and as far as we know, none of the previous WiFi networks can enable all these use cases alone.

2.2. Protocol-Oblivious Forwarding

Protocol-Oblivious Forwarding (*POF*) [5][30] is a protocol-independent SDN data plane technology initially proposed by Huawei. Following the SDN consensus, a POF switch is logically a forwarding pipeline consisting of multiple stages of "match+instruction" rules, and is programmed by controller through the POF protocol. Unlike OpenFlow, which can only handle packets of the protocols that it "recognizes", a POF switch doesn't possess any protocol-specific knowledge, instead, it follows the "match+instruction" rules by matching incoming packets with the {*offset*, *length*} tuples as match key, and executes the associated instructions upon the matched packets. In the following, we briefly introduce the key concepts in POF.

- **Matching**. POF employs an {*offset*, *length*} tuple to match a packet field, where *offset* indicates the position from which the match begins, and *length* is the bits included in the key. For example, to match the *address1* to *address3* fields in the 802.11 header, the match keys should be {32b, 48b}, {80b, 48b}, and {128b, 48b}

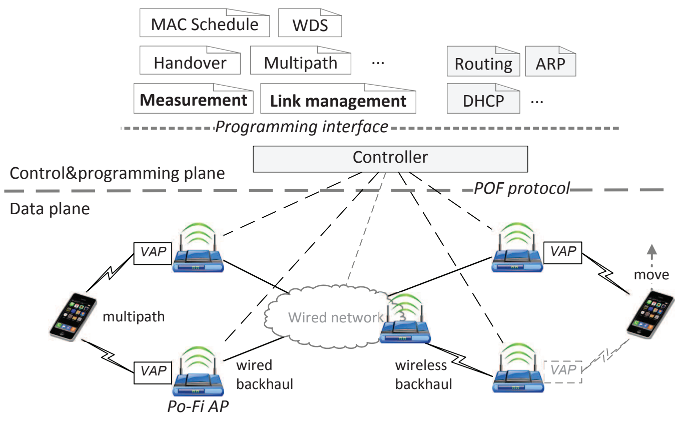


Figure 1: Architectural overview of Po-Fi.

respectively.

- **Instruction**. POF defines a concise set of protocol-agnostic *instructions*. For example, by executing the `PacketIn` instruction, a POF switch reports a matched packet with some auxiliary information to controller; the `Output` instruction forwards a packet out through a specified port; the `AddField`/`ModifyField`/`DeleteField` instruction adds, modifies, or deletes a packet field specified by its {*offset*, *length*} tuple; POF also defines arithmetic and logical instructions, the `CalculateField` instruction performs in-network computing such as checksum with given algorithm and arguments, and the `ConditionalJump` instruction executes a subsequent instruction based on the condition in its argument; in addition, POF allows multiple instructions to be grouped into an *instruction block* for batch execution.

- **Command**. POF defines a set of *commands* that enable the controller to directly instruct a POF switch to perform certain operations. For example, the `PacketOut` command instructs a POF switch to actively send a given packet out through a specified port.

- **Table and pipeline**. After being programmed by controller, a POF switch can be logically viewed as a *forwarding pipeline* composed of multiple *flow tables*, where each table is a collection of "match+instruction" rules. POF defines the `TableMod` command to add or remove flow table to/from the pipeline, and the `FlowMod` command to install or revoke rule table entries. For directing a packet from one table to a next table, POF defines the `GotoTable` instruction.

- **Metadata**. Metadata is a piece of memory in POF switch for state tracing. POF defines two types of metadata, namely *packet metadata* and *flow metadata*. Packet metadata is associated with each packet in pipeline for tracing packet-level states such as ingress port and timestamp; while the flow metadata is associated with a forwarding rule, and is used to trace the flow-level statistics. Data pieces in metadata are also addressed with {*offset*, *length*} tuples, and POF defines the `WriteMetadata` to write a value to a data piece in metadata.

2.2.1. Comparison with P4

P4 [4] is programming language for configuring clean-slate P4 switches. A P4 program includes header formats, parser specification, match-action table specification, and control flow that describes the pipeline structure. The P4 compiler compiles the program into configurations for different target switches (e.g., software switch, Tofino switch). After the configuration, a P4 switch still needs to connect to a controller, and populates its tables with rules for processing packets at run time.

Comparing with P4, POF doesn't have a configuration phase, and all capabilities of a POF switch is determined by the control plane program at run time. Moreover, POF allows an on-demand parsing scheme, where a packet field will be parsed in a POF switch only when it is necessary; on the other hand, in P4, the frontend parser comprehensively parses all the fields regardless of whether or not they are of interest.

Despite these differences, we believe that the basic idea in this paper, that is, realizing innovative WiFi services by programming a forwarding pipeline, can also be enable with P4 in principle. We select POF over P4 for the reason that currently P4 has only Ethernet chips, while the P4 software switch is considerably slower that the POF one [31].

3. Architecture

We present an overview of the Po-Fi architecture in Fig. 1. Po-Fi has a POF-compatible wireless data plane consisting of POF-compatible WiFi APs (called *Po-Fi APs*), and a centralized control and programming plane. Logically, a programmed Po-Fi AP is a forwarding pipeline containing multiple stages of "match+instruction" rules, which dictate how incoming packets should be handled in the AP.

Po-Fi's control and programming plane contains a POF SDN controller (e.g., [31] and [32]), which allows a user to arbitrarily define packet formats, construct pipeline in Po-Fi AP, and implement network functionalities with forwarding rules. More specifically, the controller program handles `PacketIn` event reported from a Po-Fi AP, install, modify, and revoke forwarding rules on the AP as in conventional SDN like OpenFlow. More details on POF SDN programming can be found in [31]. A number of program modules that realize various network services run over the controller. In particular, the *Link management* module manages all the wireless links, and the *Measurement* module collects measurements from the network. The two modules provide supports to other modules for realizing specific services.

3.1. Per-Client Virtual AP

Some novel WiFi services are based on the concept of *per-client virtual AP* (*VAP*). Logically, a VAP is a dedicate AP for an individual wireless client. For facilitating per-client VAP, when a client joins the network, Po-Fi assigns a distinct *Basic Service Set Identifier* (*BSSID*) for it to associate with, and places rules in Po-Fi AP for handling its uplink/downlink traffics. With one-BSSID-per-client, a VAP is completely decoupled from the physical AP, and can be placed at any Po-Fi AP(s). Note that per-client VAP is not mandatary, and a Po-Fi AP may or may not host VAPs depending on application contexts.

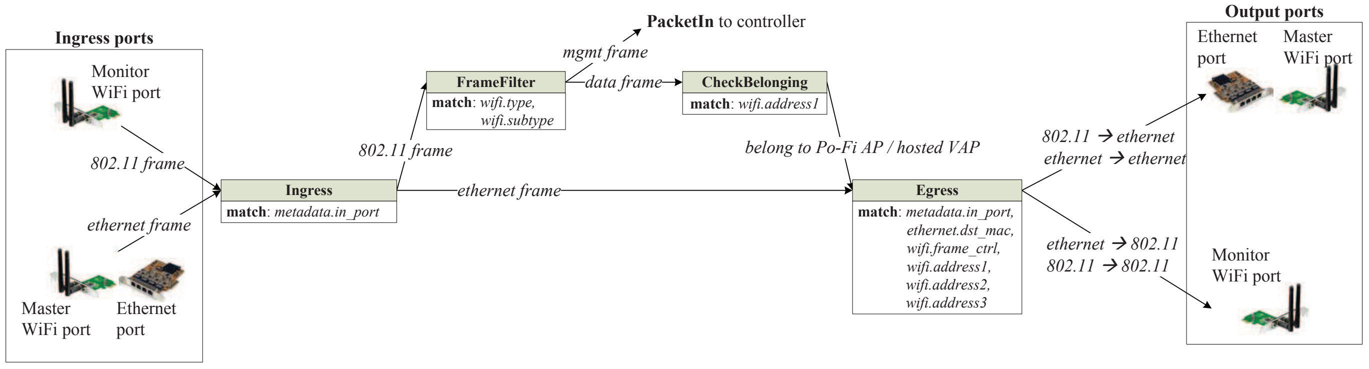


Figure 2: Layout of programmable forwarding pipeline in Po-Fi AP. Note that an Ethernet frame enters into the pipeline from Ethernet or master WiFi port, and a raw 802.11 frame enters the pipeline from Monitor WiFi interface; when outputting a packet through monitor WiFi interface, the packet must be a raw 802.11 frame, similarly, the packet must be an Ethernet frame if it is outputted via Ethernet or master WiFi interface.

3.2. Po-Fi AP

3.2.1. Forwarding pipeline

Roughly speaking, a Po-Fi AP is developed by interconnecting commodity wired and wireless network interfaces with a POF switch. A Po-Fi AP has at least one WiFi interface in the monitor mode, and may also have an Ethernet interface and a WiFi interface under the master mode, depending on its application context. When a Po-Fi AP first connects to the controller, it is initialized as a forwarding pipeline. As shown in Fig. 2, the pipeline is composed of four "match+instruction" tables, which we describe as the following.

a. The **Ingress** table at the head of the pipeline receives incoming packets from both wired and wireless ports. The table employs rules to match a packet's ingress port *in_port* in its associated metadata[1]. If the packet is an Ethernet frame from an Ethernet or a master WiFi port, a `GotoTable` instruction is applied to direct it to the Egress table at the tail of the pipeline; for an 802.11 frame from the monitor WiFi port, the Ingress table strips its RadioTap header and directs it to the FrameFilter table.

b. The **FrameFilter** table differentiates 802.11 frames by matching the *type* and *sub_type* fields in the 802.11 header. All management frames are reported to the controller with the `PacketIn` instruction, while data frames are directed to the CheckBelonging table.

c. The **CheckBelonging** table examines whether an incoming 802.11 data frame should be handled by the Po-Fi AP. More specifically, rules are applied to match the frame's *address1* field against the BSSIDs of the hosted VAPs and the Po-Fi AP's MAC address, and a matched frame, which either belongs to a hosted VAP or directly belongs to the AP, is directed to the Egress table.

d. The **Egress** table receives both Ethernet frames from the Ingress table and 802.11 data frames from the CheckBelonging table, and outputs them according to their destination MAC addresses. Before being sent

[1]Unless otherwise specified, we refer to metadata as packet metadata in this paper.

out, frames are transformed to fit their output ports: for a frame to be sent out through a monitor WiFi port, `DeleteField` and `AddField` instructions are applied to encapsulate its payload with 802.11 and RadioTap headers; similarly, a frame sent out through an Ethernet or the master WiFi port must be transformed into an Ethernet frame.

Note that here we only describe the minimum set of rules. For realizing a specific functionality, the tables will be augmented with additional rules.

3.2.2. Handling uplink / downlink traffics

A Po-Fi AP has different network interfaces, and its pipeline can be programmed to handle uplink/downlink traffics differently. Following we describe some modes that a Po-Fi AP can be programmed into.

- **Infrastructure mode**: A Po-Fi AP in this mode has a monitor WiFi interface and an Ethernet interface for connecting wireless clients to Ethernet backhaul. A client can either connect to the Po-Fi AP directly by using the AP's MAC address as BSSID, or connect to its VAP hosted on the AP with the assigned BSSID. For an uplink 802.11 data frame from a wireless client, if it belongs to a hosted VAP or the Po-Fi AP, it traverses the four tables, and at the Egress table, it is modified into an Ethernet frame and sent out to Ethernet. For a downlink Ethernet frame destined to a client, it is checked at the Egress table to find out the client's BSSID, if matched, the frame is modified into an 802.11 data frame for carrying the BSSID, and is sent out via the monitor WiFi port. Use cases for this mode can be found in Section 4.1 and 4.2.

- **WDS mode**: In this mode, a Po-Fi AP works as a bridge AP in WDS, and it relays 802.11 frames between wireless clients and a remote AP with a monitor WiFi interface. Both uplink and downlink frames traverse the four tables: an uplink frame from a wireless client is modified at the Egress table to carry the remote AP's MAC address in its BSSID field (or Receiver Address (RA) field if a 4-addressed WDS scheme is adopted); and a downlink frame is checked at the CheckBelonging table to see if it belongs to this AP by matching the frame's BSSID or RA with the AP's MAC address, if matched, the frame is modified at the Egress table to carry its destination client's BSSID, and sent out to the air. Section 4.3 describes a use case for this mode.

- **Regular mode**: In this mode, a Po-Fi AP has a master WiFi interface, an Ethernet interface, and a monitor WiFi interface. The Po-Fi AP in this mode doesn't host VAPs, instead, a client connects to the master WiFi interface as connecting to a regular AP, and Ethernet frames are switched between the master WiFi interface and the Ethernet ports at the Egress table as in wired SDN. Meanwhile, the monitor WiFi interface overhears the 802.11 frames transmitted and received by the master WiFi interface, and collects wireless measurements by inserting rules in the Ingress table. Section 4.4 presents an example use case for this mode.

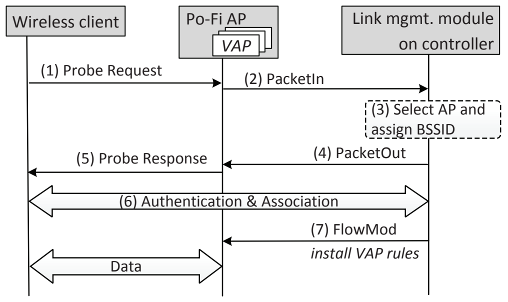


Figure 3: Client association and VAP deployment.

3.2.3. Acknowledging data frame

The pipeline in a Po-Fi AP is responsible for non time-critical tasks, however, a WiFi AP is required to acknowledge a data frame after an interval known as Short Inter-frame Spacing (SIFS). In particular, when hosting multiple VAPs, a Po-Fi AP must acknowledge all the data frames that carry the BSSIDs of the VAPs. Here we employ Wi-5's approach [33] by patching the `ath9k` driver for acknowledging the data frames. On receiving an 802.11 data frame, the patched driver matches the BSSID field with its locally stored BSSID mask, and acknowledges the matched frames.

We extend the software POF switch to add a new command for updating the BSSID mask. Each time the controller inserts or removes the rules of a VAP on a Po-Fi AP, it also executes the command to instruct the POF switch to update the AP's BSSID mask as well.

3.3. Link Management

The controller Link management module is responsible for managing wireless links. In particular, since a client may connect to VAP instead of physical AP, the module also manages the rules that realize VAPs in Po-Fi APs.

A VAP is created on client association. As demonstrated in Fig. 3, when a wireless client connects to the Po-Fi network, it broadcasts an 802.11 Probe Request to all the APs (step 1). As a management frame, the frame is reported to the controller (step 2). On receiving the client's Probe Request, the Link management module selects an AP among the Probe-reporting APs for hosting the VAP, and assigns a distinct BSSID to it (step 3). The module further instructs the selected AP to reply a Probe Response with the assigned BSSID to the client, using the `PacketOut` command (step 4-5). The client and the selected AP then proceed to exchange the authentication and association-related frames (step 6). Note that during the association, all the management frames from the client are indeed handled by the Link management module and replied through the selected AP. Finally, the module installs rules to the CheckBelonging and Egress tables in the selected AP, and completes the client association (step 7). In step 3-4, the controller can also instruct the selected Po-Fi AP to reply with its own MAC address as BSSID without hosting any VAP.

For setting up a WDS wireless backhaul link, the Link management module instructs a Po-Fi AP to serve as a bridge AP by actively connecting to a remote AP (which can be a regular AP) by playing the client side in Fig. 3. In this case, the controller module sends out all the request frames and handles all the responses

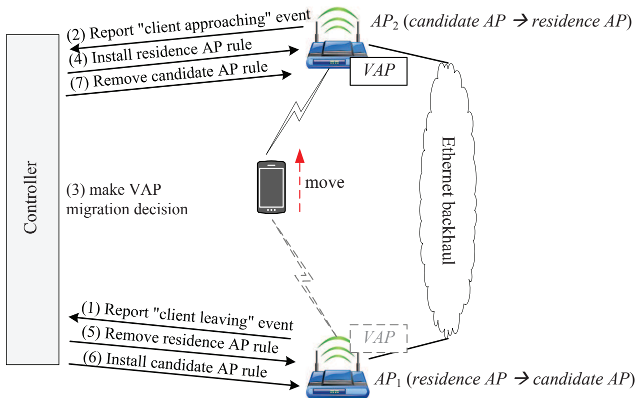

(2) Report "client approaching" event
(4) Install residence AP rule
(7) Remove candidate AP rule
AP2 (candidate AP → residence AP)
VAP
Controller
(3) make VAP migration decision
move
Ethernet backhaul
(1) Report "client leaving" event
(5) Remove residence AP rule
(6) Install candidate AP rule
VAP
AP1 (residence AP → candidate AP)

Figure 4: VAP migration-based seamless handover facilitated by Po-Fi.

through the bridge AP.

4. Use Case

4.1. Seamless Mobility

In regular WiFi, when a mobile client moves away from its connected AP and approaches to a new AP, it needs to disconnect from its current AP before connecting to the new one. During such a "break-before-make" AP handover, considerable packets are lost. In the following, we show how we can realize a *seamless mobility* service with Po-Fi.

4.1.1. VAP migration-based AP handover

For realizing seamless mobility, the Po-Fi APs are programmed into the infrastructure mode and host VAPs for clients. Since a client's per-client VAP is a set of forwarding rules, and installing/revoking the rules is light-weighted, we can realize a seamless AP handover by migrating the client's VAP to a new physical AP rather than forcing the client to reconnect to the new AP. For enabling a smart VAP migration, we program the Po-Fi APs within the mobile client's communication range to monitor their perceived signal strengths, and migrate the VAP accordingly. Hence, for a mobile client, we refer to the AP that hosts its VAP as its *residence AP*, and the other AP(s) that measure the client's signal strength as the *candidate AP(s).*

To avoid immature migration, VAP migration decisions should be made based on predicted signal strengths rather than the currently measured ones. For signal strength predicting, we employ the Holt-Winters double exponential smoothing [34] as

$$\begin{aligned} L_i &= \alpha R_i + (1-\alpha)(L_{i-1} + T_{i-1}) \\ T_i &= \beta(L_i - L_{i-1}) + (1-\beta)T_{i-1} \\ F_{i+k} &= L_i + k \times T_i \end{aligned} \tag{1}$$

where α and β are weight parameters, R_i is the i^{th} measured signal strength, L_i and T_i are the smoothed signal strength and change rate up to the i^{th} measurement. F_{i+k} is the predicted signal strength in a near future after k measurements, which is calculated by taking L_i and T_i into consideration.

Ingress	
match: *metadata.in_port*	*metadata.rssi* ← *radiotap.rssi*

CheckBelonging	
match: *wifi.address1* #rule for residence AP	#instruction block #1 **CalculateField** (*metadata.rssi*, L_i, T_i, F_{i+k}) **ConditionalJump** ($F_{i+k} < th_1$, **PacketIn** (*pkt*, "client_leaving", F_{i+k})) **GotoTable** (*Egress*)
otherwise	**PacketIn** (*pkt*, "new_client")

(a) AP_1 (residence AP)

Ingress	
match: *metadata.in_port*	*metadata.rssi* ← *radiotap.rssi*

CheckBelonging	
match: *wifi.address1* #rule for candidate AP	#instruction block #2 **CalculateField** (*metadata.rssi*, L_i, T_i, F_{i+k}) **ConditionalJump** ($F_{i+k} > th_2$, **PacketIn** (*pkt*, "client_approaching", F_{i+k})) **Drop**
otherwise	**PacketIn** (*pkt*, "new_client")

(b) AP_2 (candidate AP)

Figure 5: Rules in the pipeline for (a) residence AP and (b) candidate AP.

As demonstrated in Fig. 4, a client's residence and candidate APs continuously monitor and predict their perceived signal strengths. If the predicted signal strength ss_1 at the residence AP is lower than a threshold th_1, the AP reports a "client leaving" `PacketIn` event with ss_1 to the controller (step 1); if the predicted signal strength ss_2 at a candidate AP is above a threshold th_2, the AP reports a "client approaching" event with ss_2 (step 2). On receiving the two reports, the controller checks if $ss_1 - ss_2 > \Delta$, and migrates the VAP (step 3-7).

4.1.2. Realization

Next, we present how to realize the VAP migration-based seamless handover with Po-Fi. We use Fig. 4 as example to describe the realization details, where AP_1 is the client VAP's residence AP, and AP_2 is its candidate AP.

Signal strength measurement and predicting

We exploit the RadioTap header's *Received Signal Strength Indicator* (*RSSI*) field for measuring the signal strength. More specifically, as shown in Fig. 5, for both residence and candidate APs, a rule is inserted in the Ingress table to extract each incoming 802.11 frame's RSSI, and write the value to metadata using the `WriteMetadata` instruction.

Frames from a mobile client are processed differently by the CheckBelonging tables in the residence and candidate APs. Fig. 5(a) presents the rules in a residence AP (i.e., AP_1 in Fig. 4). The rule in the Check-Belonging table matches a frame's *address1* field with the VAP's BSSID, and applies an instruction block for

Algorithm 1: VAP migration

```
Input    : $event$

Algorithm VAP Migration
    if $event.type$ = $client\_leaving$ then
        OnClientLeaving($event$);
    if $event.type$ = $client\_approaching$ then
        OnClientApproaching($event$);
    if $event.type$ = $new\_client$ then
        OnNewClient($event$);

Procedure OnClientLeaving($event$)
    Add $event.client$ to $client\_set$;

Procedure OnClientApproaching($event$)
    $client \leftarrow event.client$;
    Add $event.\{AP, ss\}$ to $client.AP\_set$;
    if $client \in client\_set$ then
        $\{AP_1, ss_1\} \leftarrow$ $client$'s current residence AP;
        Select $\{AP_2, ss_2\}$ from $client.AP\_set$;
        if $ss_2 - ss_1 > \Delta$ then
            Install residence AP rule for $client$ to $AP_2$;
            Remove residence AP rule for $client$ from $AP_1$;
            Install candidate AP rule for $client$ to $AP_1$;
            Remove candidate AP rule for $client$ from $AP_2$;

Procedure OnNewClient($event$)
    if $event.client$ is legitimate then
        Install candidate AP rule for $event.client$ to $event.AP$;
```

signal strength predicting and event reporting.

We further elaborate the instruction block, which applies Equation (1) to predict the signal strength, in details. We keep L_i, T_i, and the predicted signal strength F_{i+k} as three data pieces in the flow metadata associated with the rule, and use the `CalculateField` instruction to update their values on each matched frame. Afterwards, a `ConditionalJump` instruction is executed: if $F_{i+k} < th_1$, the `PackedIn` instruction is employed to report a "client leaving" event with F_{i+k}. Afterwards, the frame is directed to the Egress table.

The rules in the candidate AP (i.e., AP_2 in Fig. 4) is presented in Fig. 5(b). The rule in the CheckBelonging table checks BSSID as in Fig. 5(a), but applies a different instruction block that predicts the signal strength F_{i+k}, and reports a "client approaching" event if $F_{i+k} > th_2$. Afterwards, the frame is dropped, as a replica is forwarded to its destination by AP_1, the residence AP of the client.

Finally, if a frame's *address1* field doesn't match any BSSIDs of the VAPs that the Po-Fi AP serves as a residence or a candidate AP, it is reported to the controller as a "new client" event, as such an incident indicates that the AP "sees" a new client, and is a potential candidate AP for the client.

VAP migration algorithm

We present the controller VAP migration algorithm, which handles the three event reports, in Algorithm 1. On receiving a "client leaving" event, the algorithm adds the corresponding client to *client_set*, which contains the clients that are unsatisfied with their current residence APs. When a "client approaching" event is reported from an AP, the AP (together with the predicted signal strength) is added to *AP_set* associated with the client, which keeps the candidate APs that are good enough to migrate to. The algorithm then examines if the client is in *client_set*: if yes, the AP with the strongest signal is selected from the client's *AP_set* as its new residence AP, and the VAP is migrated if the difference between the signal strength predictions exceeds Δ. Finally, on receiving a "new client" event from an AP, the algorithm installs the rule to make the AP a new candidate AP for the client.

The Po-Fi facilitated realization for seamless handover is highly scalable, as all the computational-intensive tasks of signal strength smoothing and predicting are performed at the Po-Fi APs, while the controller algorithm only handles event reports. We examine the use case on real-world testbed in Section 5.3.

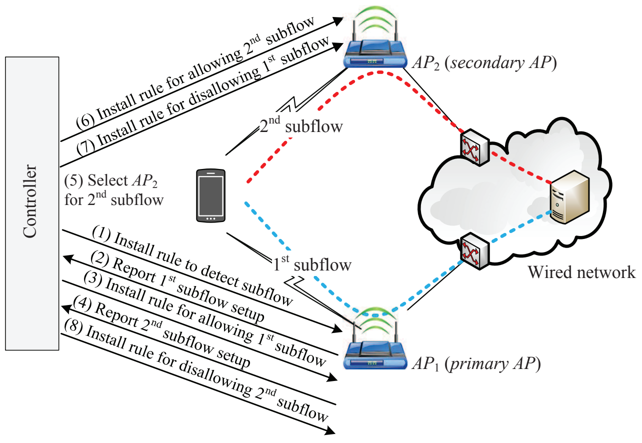
4.2. Multi-AP MPTCP Data Transmission

4.2.1. Multi-AP MPTCP

Over 75% of the wireless bottlenecks are closely related to AP [35], and simultaneously employing multiple APs is promising for enhancing the communication reliability in WiFi networks [24]. On one hand, client devices are ready for multipathing, as the *Multipath TCP* (*MPTCP*) protocol, which is the de facto standard for multipath data transmission [36][37], has been implemented in Linux kernel and iOS. But on the other hand, in regular WiFi, a client connects to only one AP at a time. In this use case, we show that with Po-Fi, we can realize a *multi-AP MPTCP data transmission* that enables a client with only one WiFi interface to employ multiple APs to carry its MPTCP subflows.

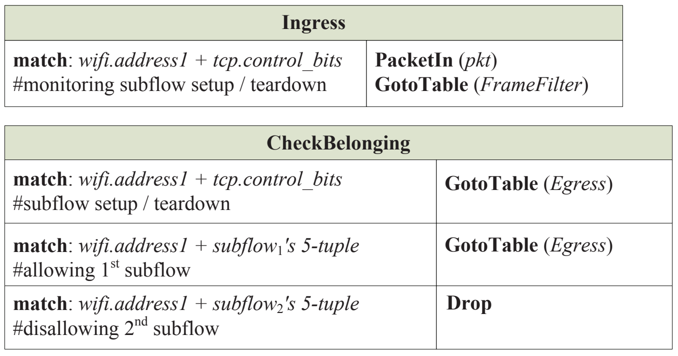
MPTCP is an extension of TCP for allowing multiple subflows within a connection. Being backward-compatible, in MPTCP, two ends follow the three-way handshake to establish a subflow as in regular TCP. During the handshake, if both ends contain the `MP_CAP` (multipath capable) option in the TCP header's option field, they set up the first subflow; if the `MP_JOIN` (join connection) option is contained, a subsequent subflow is established and joins the existing MPTCP connection. Each MPTCP subflow has a unique {*proto*, *src_ ip*, *dst_ ip*, *src_port*, *dst_port*} tuple for distinguishing it from other subflows and regular TCP flows, and subflows belonging to a same MPTCP connection share a same *token* [36].

The multi-AP MPTCP use case is demonstrated in Fig. 6, where Po-Fi APs are programmed into the infrastructure mode and host VAPs. Here we refer to the AP on which a client's VAP is initially placed and carries its first subflow as its *primary AP*. When a subsequent subflow (from a same WiFi interface but a different TCP port) is added, the controller selects a different AP as its *secondary AP* for carrying the subflow. As a consequence, a wireless client may have its VAP simultaneously placed at one primary AP and multiple secondary APs, thus greatly enhances its data communication reliability.



(6) Install rule for allowing 2nd subflow
(7) Install rule for disallowing 1st subflow
AP_2 (*secondary AP*)
2nd subflow
Controller
(5) Select AP_2 for 2nd subflow
1st subflow
Wired network
(1) Install rule to detect subflow
(2) Report 1st subflow setup
(3) Install rule for allowing 1st subflow
(4) Report 2nd subflow setup
(8) Install rule for disallowing 2nd subflow
AP_1 (*primary AP*)

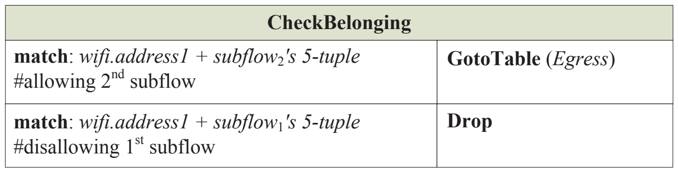
Figure 6: Po-Fi facilitated Multi-AP MPTCP data transmission.



Ingress	
match: *wifi.address1* + *tcp.control_bits* #monitoring subflow setup / teardown	**PacketIn** (*pkt*) **GotoTable** (*FrameFilter*)

CheckBelonging	
match: *wifi.address1* + *tcp.control_bits* #subflow setup / teardown	**GotoTable** (*Egress*)
match: *wifi.address1* + *subflow*$_1$*'s 5-tuple* #allowing 1st subflow	**GotoTable** (*Egress*)
match: *wifi.address1* + *subflow*$_2$*'s 5-tuple* #disallowing 2nd subflow	**Drop**

(a) AP_1 (primary AP)



CheckBelonging	
match: *wifi.address1* + *subflow*$_2$*'s 5-tuple* #allowing 2nd subflow	**GotoTable** (*Egress*)
match: *wifi.address1* + *subflow*$_1$*'s 5-tuple* #disallowing 1st subflow	**Drop**

(b) AP_2 (secondary AP)

Figure 7: Rules in the pipeline for (a) primary AP and (b) secondary AP.

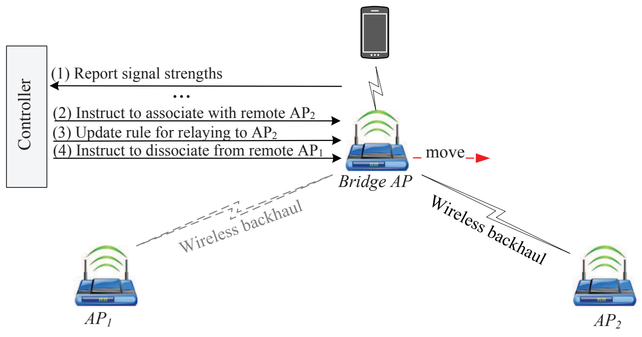

Controller
(1) Report signal strengths
...
(2) Instruct to associate with remote AP₂
(3) Update rule for relaying to AP₂
(4) Instruct to dissociate from remote AP₁
move
Bridge AP
Wireless backhaul
Wireless backhaul
AP₁
AP₂

Figure 8: Po-Fi facilitated wireless backhaul and smooth remote AP switching for mobile bridge AP.

4.2.2. Realization

We describe how to realize the Po-Fi facilitated multi-AP MPTCP data transmission with Fig. 6 as example, where AP_1 is the client's primary AP, and AP_2 is a secondary AP.

In multi-AP MPTCP, we require the controller to be aware of a wireless client's MPTCP connection status. To enable this, we insert rules in the Ingress table of the primary AP (i.e., AP_1 in Fig. 6) to monitor all the TCP control segments (step 1), and report them to the controller before directing them to the next table.

The controller examines the `PacketIn` TCP control segments and detects setup/teardown of each MPTCP subflow. If an MPTCP subflow is set up with a new token at AP_1, the controller infers that it is the first subflow of a new MPTCP connection, and inserts a rule in AP_1's CheckBelonging table to allow the subflow (step 2-3). As shown in Fig. 7(a), the rule matches the 5-tuple of the subflow as well as the client VAP's BSSID, and directs the matched frames to the Egress table for transmission.

If setup of a second subflow is detected, the controller selects a different Po-Fi AP as the client's secondary AP (i.e., AP_2 in Fig. 6) (step 4-5), inserts a rule in its CheckBelonging table for allowing the second subflow, and disallowing the first subflow (step 6-7). Meanwhile, as shown in Fig. 7(a), a rule is inserted in AP_1's CheckBelonging table to disallow the second subflow (step 8). Similarly, each time a subsequent subflow is added, the controller selects a new secondary AP, inserts the rules for allowing the subflow and disallowing the previous subflows on that AP, and installs rules to disallow the new subflow in the primary and all the other secondary APs. For tearing down a subflow, the controller removes the corresponding rules for allowing and disallowing the subflow from the primary and secondary APs. Algorithm 2 presents the complete algorithm, and we valuate the use case in Section 5.4.

4.3. Smooth Remote AP Switching

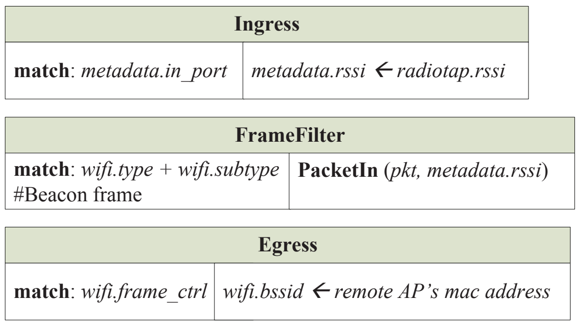
802.11's wireless distribution system (WDS) defines wireless backhaul in WiFi networks [16]. In WDS, a bridge AP plays the client side in the association procedure to set up a wireless backhaul link with a remote WiFi AP, and relays 802.11 frames between wireless clients and the remote AP. WDS can be used for quickly providing WiFi access to areas that lack infrastructures.

We consider a use case as demonstrated in Fig. 8, where Po-Fi controls only a mobile AP and programs it into a bridge AP, while all the remote APs are regular APs. Since the mobile bridge AP moves around, when

Algorithm 2: Multi-AP MPTCP

```
Input    : $AP_1$ the primary AP, $Secondary\_AP\_Set$, $Sflow\_Set$

Algorithm Multi-AP MPTCP
    if $sflow$ setup detected then
        if first subflow then
            Install rule for allowing $sflow$ to $AP_1$;
        else
            Select $AP_2$ as a new secondary AP;
            Install rule for allowing $sflow$ to $AP_2$;
            for $sflow_p \in Sflow\_Set$ do
                Install rule for disallowing $sflow_p$ to $AP_2$;
            Install rule for disallowing $sflow$ to $AP_1$;
            for $AP \in Secondary\_AP\_Set$ do
                Install rule for disallowing $sflow$ to $AP$;
            Add $AP_2$ in $Secondary\_AP\_Set$;
        Add $sflow$ in $Sflow\_Set$;
    if $sflow$ teardown detected then
        if first subflow then
            Remove rule for allowing $sflow$ from $AP_1$;
        else
            $AP_2 = sflow$'s secondary AP;
            Remove all MPTCP rules from $AP_2$;
            Remove $AP_2$ from $Secondary\_AP\_Set$;
            Remove rule for disallowing $sflow$ from $AP_1$;
            for $AP \in Secondary\_AP\_Set$ do
                Remove rule for disallowing $sflow$ from $AP$;
        Remove $sflow$ from $Sflow\_Set$;
```



Ingress	
match: *metadata.in_port*	*metadata.rssi* ← *radiotap.rssi*

FrameFilter	
match: *wifi.type* + *wifi.subtype* #Beacon frame	**PacketIn** (*pkt, metadata.rssi*)

Egress	
match: *wifi.frame_ctrl*	*wifi.bssid* ← *remote AP's mac address*

Figure 9: Rules in the pipeline for bridge AP.

it departs from its currently connected remote AP (i.e., AP_1 in Fig. 8) and approaches to a new one (i.e., AP_2 in Fig. 8), its wireless backhaul link with AP_1 deteriorates, and the bridge AP should switch its remote AP from AP_1 to AP_2 for maintaining its wireless backhaul link.

However, in regular WiFi's "break-before-make" approach, the bridge AP must first disconnect from AP_1 before connecting to AP_2, and considerable packets are lost during the switching. In this use case, we realize a smooth remote AP switching with Po-Fi, which enables the bridge AP to switch its remote AP in a "make-before-break" manner.

In our use case, the bridge AP is responsible for collecting wireless measurements and initiating the switching. In particular, signal strengths of AP_1 and AP_2 are collected from their Beacon frames. As shown in Fig. 9, we insert a rule at the bridge AP's Ingress table to extract RSSI from a Beacon frame, keep it in metadata, and report the frame with RSSI to the controller at the FrameFilter table (step 1). The controller compares the RSSIs from AP_1 and AP_2, and decides to switch to AP_2 only when the signal from AP_2 is δ dBm stronger than the ones from the other APs for consecutive r Beacons.

Once the controller decides to switch to AP_2, it first instructs the bridge AP to associate with AP_2 by following the procedure described in Section 3.3 (step 2). As shown in Fig. 9, the controller also updates the rule in the Egress table to modify each out-going uplink data frame's BSSID (or RA if a 4-addressed WDS scheme is adopted) as AP_2's MAC address (step 3). Finally, the controller instructs the bridge AP to disconnect from AP_1 (step 4).

We can see that in the Po-Fi facilitated AP switching, the bridge AP disconnects from AP_1 only after it has successfully connected to AP_2 and updated the rule for relaying the frames. Obviously with such a "make-before-break" approach, packet losses can be avoided. We evaluate the use case in Section 5.6.

4.4. TDMA MAC Scheduling

802.11's distributed coordination function (DCF) is inefficient in handling interferences in hidden/exposed terminal environments, while TDMA, which schedules medium access in a synchronized and centralized manner, is effective for overcoming such deficiency [27][28]. Although by exploiting 802.11's power saving mechanism, medium-competing flows can be allocated to pre-configured non-overlapping time slots, however, how to smartly allocate time slots to different flows for improving the wireless transmission is still unknown.

In this use case, we integrate a TDMA MAC layer named hMAC [29] on Po-Fi AP, moreover, with Po-Fi's

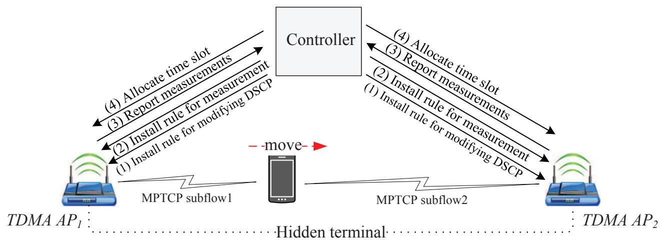


Figure 10: Po-Fi facilitated TDMA MAC scheduling.

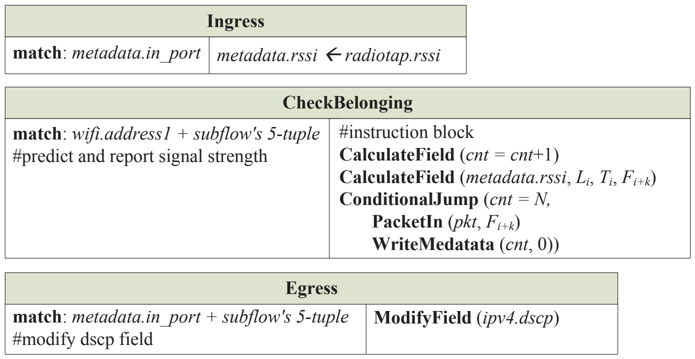
programmability, we realize a centralized TDMA MAC scheduling that allows user to dynamically allocate TDMA time slots on medium-competing network flows based on wireless measurements, for maximizing the overall transmission throughput.

A TDMA AP is developed by programming a Po-Fi AP into the regular mode, and applying the hMAC driver patch to the AP's master WiFi interface. hMAC exploits 802.11 standard power saving functionality to pause and unpause software queues within the `ath9k` wireless driver, and each software queue is identified with destination MAC address and IP header's DSCP field. hMAC also provides APIs for user to divide time slots, and configure each time slot with an access rule based on destination MAC and DSCP. Note that hMAC does not replace the standard CSMA/CA, as within a time slot, the AP still follows CSMA/CA to compete for air time with other non-TDMA transmitters.

To enable the centralized TDMA MAC scheduling, we expand the software POF switch to add a new command, which allows the controller to instruct a POF switch to configure a given time slot's access rule using hMAC's API. Each time the controller allows a flow on an AP and allocates some time slots for it, it assigns the flow with a DSCP value that is unique among the flows destined to the same client on that AP, and inserts a rule in the Egress table as shown in Fig. 11 to modify each outgoing packet's DSCP field. The controller then calls the command to update the access rules of the allocated time slots on the AP.

We demonstrate the use case for the Po-Fi facilitated TDMA MAC scheduling in Fig. 10: Two TDMA APs have their master WiFi interfaces working on a same channel. A client employs two virtual interfaces to connect to two TDMA APs, and sets up an MPTCP connection containing two subflows, with each AP carrying a subflow for downloading from a wired host. For each subflow, rules are placed in the Egress table to modify DSCP field for downlink packets as previously described (step 1). The client moves from one AP towards the other. However, although the client can communicate with both APs, the APs are out of each other's carrier sense range, thus are hidden terminals to each other. As a consequence, the two APs will contend for medium access without any coordination, and the client will suffer low throughput due to the interferences between its two subflows.

One naive approach is to configure the two subflows with equal number of none-overlapping time slots for avoiding the interference. However, such a static allocation ignores the fact that the AP closer to the client has a stronger signal, and should be allocated with more time slots. Motivated by this observation, we employ a monitor WiFi interface on the AP, collect signal strengths, and allocate time slots to the subflow on the AP accordingly. More specifically, we insert rules as in Fig. 11 to measure and predict RSSIs from the overheard



Ingress	
match: *metadata.in_port*	*metadata.rssi* ← *radiotap.rssi*

CheckBelonging	
match: *wifi.address1 + subflow's 5-tuple* #predict and report signal strength	#instruction block **CalculateField** (*cnt = cnt*+1) **CalculateField** (*metadata.rssi*, L_i, T_i, F_{i+k}) **ConditionalJump** (*cnt = N,* **PacketIn** (*pkt*, F_{i+k}) **WriteMedatata** (*cnt*, 0))

Egress	
match: *metadata.in_port + subflow's 5-tuple* #modify dscp field	**ModifyField** (*ipv4.dscp*)

Figure 11: Rules in the pipeline for TDMA AP.

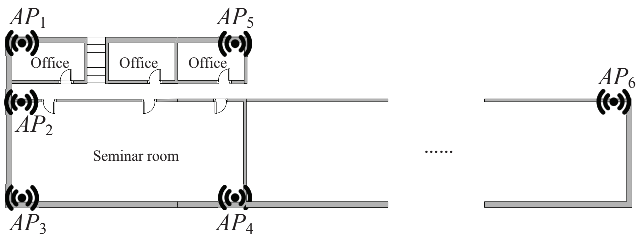


Figure 12: Testbed with six APs, where AP_2 and AP_6 are out of each other's carrier sense range.

802.11 frames using Equation (1) (step 2), and report the results to the controller every N frames (step 3). Note that here we employ an instruction block in the CheckBelonging table, in which we use metadata *cnt* as the frame counter, and clear it every N frames.

On receiving the predicted RSSIs from the APs, the controller computes the time slots that should be allocated to the i^{th} subflow as

$$n_i = [\frac{f(RSSI_i)}{\sum_{j=1}^{2} f(RSSI_j)} \times n] \tag{2}$$

where $RSSI_i$ is the signal strength predicted for AP_i, $f(x) = 0.3382 \times x + 26.5$ is the liner regression between RSSI and throughput [38], and n is the total number of time slots that can be allocated. Finally, to avoid frequent adjustment, the controller updates a subflow's time slot allocation only when the time slot difference before and after the update is above a threshold d (step 4). We evaluate the use case with real-world testbed in Section 5.7.

4.5. Discussion

From the use cases, we can see that Po-Fi can facilitate a wide range of WiFi innovations. Furthermore, with the unified control plane programming, different functionalities implemented as forwarding rules can be easily combined. For example, as we will see in Section 5.5, the seamless mobility service can be easily combined with the multi-AP multipath data transmission for enhancing a mobile client's QoE. Finally, being protocol-agnostic in nature, Po-Fi is capable to support new and proprietary protocols over WiFi, therefore is “future-proof”.

5. Implementation and Evaluation

5.1. Prototype and Testbed

We develop two kinds of Po-Fi APs based on different hardwares. One kind of Po-Fi AP is developed on the Netgear R6100 WiFi router with an Atheros AR9344 WiFi SoC, and we also develop Po-Fi AP by attaching the Atheros AR9485 PCI-E WiFi NIC(s) to a PC equipped with an Intel Core i5 CPU and 4G RAM. We flash both Router and PC with `OpenWrt Chaos Calmer 15.05` [39] and the `ath9k` software driver [40]. We expand the `POFSwith 1.4.015` [30] software switch and run it in user space.

We develop a Po-Fi controller based on `Floodlight` [41], and run it on a server equipped with an Intel Xeon E3 CPU and 8G RAM. To compare Po-Fi with regular WiFi, we also configure the router and PC into regular APs with OpenWrt, and employ OVS for packet switching.

We build a real-world testbed composed of six APs at the fourth floor of our academic building as in Fig. 12. In our real-world experiments, we use a laptop equipped with a Realtek BGN WiFi NIC as the mobile wireless client. We also test Po-Fi with iPhone 6 plus and Huawei P10 smartphones, and find that they are compatible with Po-Fi without the need of any client-side modifications.

5.2. Basic Performance and Overhead

5.2.1. AP throughput

We first examine Po-Fi AP's throughput. We program a Po-Fi AP into the infrastructure mode connected by a client, and forwards traffic between the laptop client and a wired host. As a benchmark test, the wireless client is close to the AP and doesn't move. We initiate a TCP flow with `iPerf`, and measure the throughputs for both the router- and the PC-based APs. We find that the PC-based AP has a throughput about 20 Mbps, while the router-based AP, which has limited computing power with its embedded processor, also achieves a throughput over 10 Mbps. Both APs have decent throughputs that suffice for most applications. In the following experiments, we use router-based APs if not otherwise specified.

5.2.2. Client association time

We then examine the time required for a client to connect to the Po-Fi network, and compare with the client association time in regular WiFi. We execute the `nmcli` command on the laptop client to instruct it to connect to a Po-Fi AP and a regular WiFi AP, and record the client association time. For simplicity, no authentication is involved. We repeat the experiment 10 times, and find that on average, it takes 4.84 s for the client to connect to the Po-Fi AP, which is slightly longer than the mean association time of 4.06 s in regular WiFi.

Po-Fi's longer association time can be explained with the fact that the centralized WiFi management incurs additional delay between the Po-Fi AP and the controller. However, the slightly longer time is worthwhile, as a client can associate only once, then keeps stayed in the network by migrating its per-client VAP afterwards.

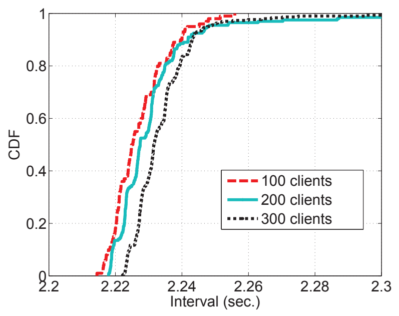


Figure 13: Distributions of intervals between first Probe Request and VAP installation with 100, 200, and 300 clients connecting to Po-Fi in one second.

5.2.3. Scalability

We also consider the case that a large number of wireless clients simultaneously connect to a Po-Fi network. More specifically, we generate the Probe and Associate Request frames from 100, 200, and 300 emulated clients at a Po-Fi AP within one second, and send them to the controller. For each emulated client, we record the interval between the time when the first Probe Request is sent out and the time that the rules of the corresponding VAP are installed.

Fig. 13 presents the distributions of the intervals. From the figure, we can see that the intervals only slightly vary, with the standard deviations of 8.5, 14.8, and 11.0 ms for 100, 200, and 300 simultaneous clients respectively. Moreover, when more clients connect to the Po-Fi network, the overall intervals only slightly increase. The observation suggests that Po-Fi is scalable enough for handling many simultaneous connecting requests.

5.3. Seamless Mobility

5.3.1. Experiment setup

We realize the seamless mobility use case as described in Section 4.1 by programming Po-Fi APs into the infrastructure mode and implementing Algorithm 1 on the controller. For evaluating the use case, we move a laptop mobile client in a counterclockwise order from AP_1 to AP_3, then to AP_4 and AP_5, and back to AP_1 in 60 seconds. We set $\alpha = \beta = 0.5$, $ss_1 = ss_2 = -38$ dBm, $\Delta = 5$ dBm, and $k = 20$ in the experiment.

As discussed in Section 4.1, the client movement causes the Po-Fi APs to detect persistent WiFi signal strength changes, and report the "client leaving"/"client approaching" events. On receiving the event reports, the controller migrates the client's VAP from AP_1 to AP_3, AP_3 to AP_4, AP_4 to AP_5, and AP_5 to AP_1 sequentially.

For comparison, we repeat the experiment with four regular WiFi APs at the same positions. For enabling a naive AP handover, we run a script on the laptop client, which repeatedly scans the APs' signal strengths, and switches to the AP that has the strongest signal. We also compare with Odin by using Odin's AP source code [33], and follows the strategy in [14] to make the AP handover. We use iPerf to initiate TCP or UDP flows from the mobile client to a wired host, and perform measurements between the two ends.

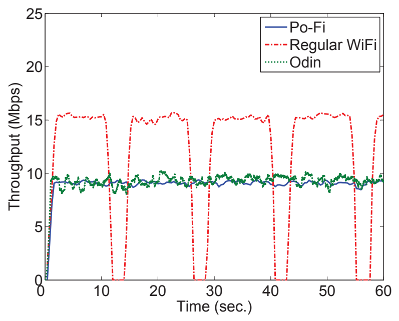


Figure 14: Comparison of mobile client's TCP throughputs in Po-Fi, regular WiFi and Odin networks.

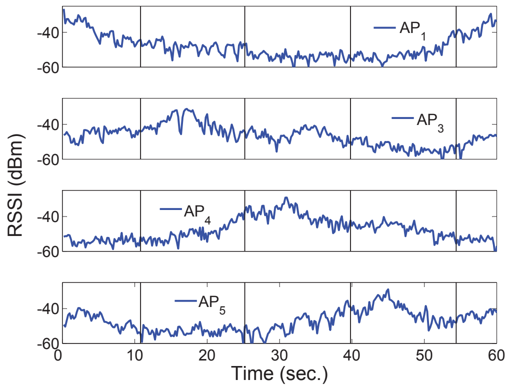


Figure 15: RSSIs perceived by the four Po-Fi APs.

5.3.2. Result

Fig. 14 presents the TCP throughputs achieved in Po-Fi, regular WiFi, and Odin networks, and in Fig. 15, we present the RSSIs perceived by the four Po-Fi APs in the experiment. For reference, timings of the VAP migrations are labeled on the figure. The figures suggest that RSSI is a good indication for making the VAP migration decisions. For example, the RSSI perceived by AP_1 has its peaks at the beginning and end of the experiment, when the client is close to the AP; and the RSSIs perceived by the other three APs also reach to their peaks when the client is approaching them, and decline when it departs away.

From Fig. 14 one can see that when the client switches its associated APs, the TCP throughput in regular WiFi drops to zero. On the other hand, the TCP throughputs in both Po-Fi and Odin networks remain stable all the time. This is easy to understand, as in regular WiFi, the mobile client must disconnect from its current AP before connecting to a new one, and during the handover, all the TCP segments are lost. But in Po-Fi and Odin networks, the client's VAP (or LVAP in Odin) migrates to a new physical AP in a "make-before-break" manner, and as long as the client's VAP is functional on at least one AP, no segment will be lost. From Fig. 14, we can see that Po-Fi and Odin have throughputs that are very close to each other, and are relatively lower comparing with a regular AP. This is because both Po-Fi and Odin process packets in user space, thus share a same performance bottleneck, while the regular AP processes packets in kernel. However, with the PC-based Po-Fi AP, we can achieve a much higher throughput at about 20 Mbps.

To testify our point, we send a UDP flow at a constant rate of 1 Mbps from the mobile client, in which

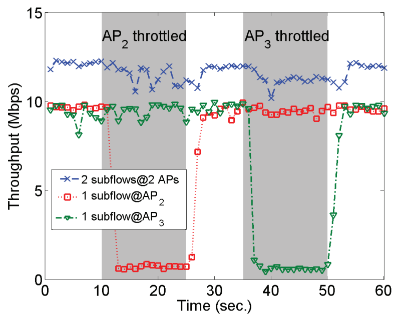


Figure 16: Comparison of client's MPTCP throughputs with the case that the connection contains two subflows carried by two APs, and the cases the connection contains one subflow on AP_2 and AP_3 respectively. Shaded areas indicate the intervals that specific APs are throttled.

each packet is 1200 B, and perform the VAP migration-based handover in Po-Fi and Odin as well as the naive handover. We infer the lost packets by comparing the traces captured at the sender and receiver sides. We repeat the experiment five times, and find that no packet is lost during a VAP migration in Po-Fi, but as many as 385 packets are lost in a naive handover on average.

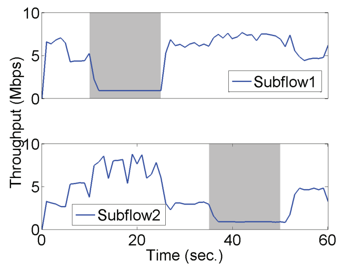
We also test how frequently a client can handoff by repeatedly migrating its VAP between two Po-Fi APs. We find that it is possible to make the client to handoff up to 15 times in one second, suggesting that Po-Fi is capable to handle highly dynamic clients. The experiment results indicate that, the seamless mobility service facilitated by Po-Fi can greatly enhance a wireless client's QoE under mobile circumstances.

5.4. Multi-AP MPTCP Data Transmission

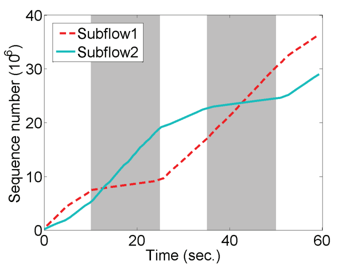
5.4.1. Experiment setup

We program Po-Fi APs into the infrastructure mode and implement Algorithm 2 on the controller for realizing the multi-AP MPTCP data transmission as described in Section 4.2. For enabling MPTCP on end hosts, we compile the `MultiPath TCP v0.92` protocol stack [37] in their Linux kernels, and use the stack's "default" scheduler that prefers subflows with shorter RTTs. In our experiment, we employ two APs, AP_2 and AP_3, that work on a same channel within each other's carrier sense range, and place a wireless client at a same distance from each AP. Note that with the Po-Fi facilitated multi-AP MPTCP, the client only needs to have one WiFi interface, but can employ multiple Po-Fi APs to transmit data. For emulating AP bottlenecks, we employ `tc-netem` to restrict network flows from a particular AP within a specified bandwidth.

We use iPerf to initiate an MPTCP connection containing two subflows from the wireless client to a wired host. As discussed in Section 4.2, on detecting the subflows, the controller installs the rules for the client's primary and secondary VAPs on AP_2 and AP_3 respectively, each carrying a subflow. The experiment lasts 60 seconds, during which we emulate AP bottlenecks by throttling AP_2's bandwidth to 0.5 Mbps from 10 to 25 s, and throttle AP_3 from 35 to 50 s. For comparison, we disable the multi-AP MPTCP service, and repeat the experiment twice, each time the client associates with only one AP (AP_2 or AP_3), and transfers data with an MPTCP connection containing only one subflow.



(a)



(b)

Figure 17: (a) Throughputs and (b) sequence numbers of the two MPTCP subflows under multi-AP MPTCP. Shaded areas indicate the intervals that specific APs are throttled.

5.4.2. Result

Fig. 16 presents client's MPTCP throughputs with and without the multi-AP MPTCP service facilitated by Po-Fi. The shaded areas indicate the intervals during which the specific APs are throttled. From the figure we can see that with the service, bandwidth throttling only slightly influences the client's overall MPTCP throughput; but without the service, the throughput drops to less than 1 Mbps when the client's associated AP gets throttled.

We explain the observation as the following. Without multi-AP MPTCP, when the client's associated AP is throttled, the single-flow MPTCP connection has a bottleneck. But with the service, the two subflows of the client's MPTCP connection are carried by two Po-Fi APs; when one AP is throttled, the MPTCP protocol stack detects the bottleneck, and schedules the other subflow to transfer more data.

To support our point, in Fig. 17(a) we plot the throughputs of the two MPTCP subflows under the multi-AP MPTCP service. We also present how the two subflows' TCP sequence numbers change over time in Fig. 17(b). From the figures we can see that the two subflows make roughly equal contributions, but when one Po-Fi AP is throttled, the subflow carried by the other unthrottled AP grabs more transmitting capacity, and achieves a higher throughput by increasing its sequence number more rapidly. From Fig. 16–17, we can conclude that the multi-AP MPTCP service facilitated by Po-Fi can greatly enhance the communication reliability of a wireless client.

5.5. Customized Service

5.5.1. Experiment setup

In this experiment, we show how we can customize a novel service by combining the use cases of seamless mobility with multi-AP MPTCP data transmission. We compose a controller program to enable the "mobility+multipath" service, which has two key features: 1) The service places a client's MPTCP subflows on different Po-Fi APs, with each AP carrying at most one subflow at a time; 2) It instructs the Po-Fi APs to monitor and predict the client's signal strengths, and migrates the client's primary and secondary VAPs to the Po-Fi APs that have better signal receptions. Obviously, under the customized service, a mobile client's QoE and reliability can be greatly enhanced.

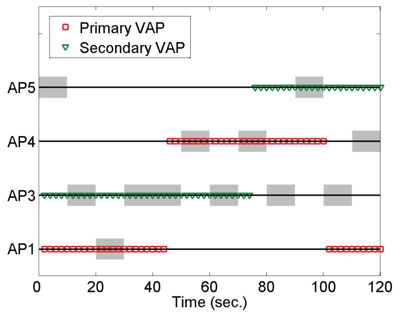


Figure 18: AP bandwidth throttling and VAP migrations. The shaded area on each AP's timeline indicates a 10-second interval during which the Po-Fi AP is throttled, and the red squares and green triangles indicate the times when the primary and secondary VAPs are placed at the APs respectively.

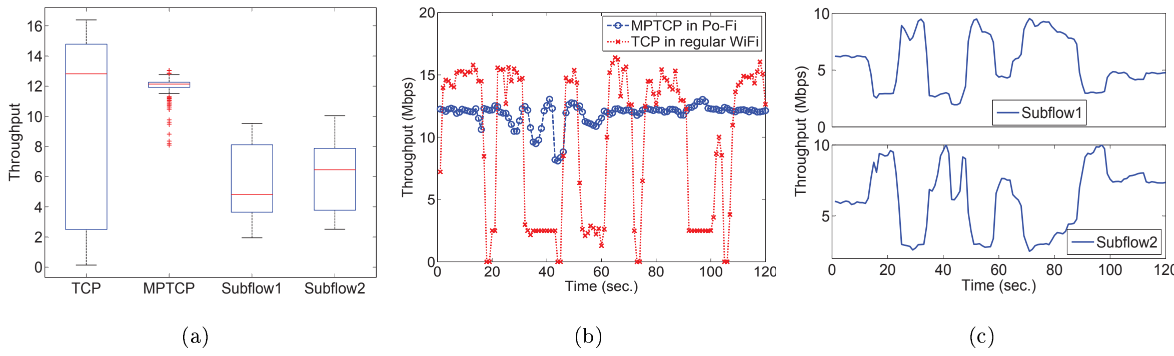


Figure 19: (a) Throughputs of single TCP flow, MPTCP connection, and two MPTCP subflows in boxplots; (b) comparison of client's MPTCP throughput in Po-Fi and TCP throughput in regular WiFi network; (c) throughputs of two MPTCP subflows.

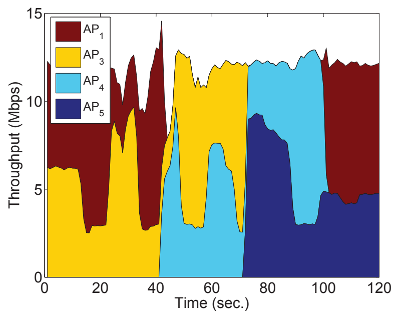


Figure 20: Throughputs contributed from the four Po-Fi APs over time.

For evaluating the "mobility+multipath" service, we program Po-Fi APs into the infrastructure mode, and place them at AP_1, AP_3, AP_4, and AP_5 of the testbed. We move the laptop mobile client in a counterclockwise order as in Section 5.3 in 120 seconds, and set up an iPerf MPTCP connection containing two subflows from the client to a wired host. For emulating AP bottlenecks, we randomly throttle the APs' available bandwidths to 3 Mbps. Timing of the bandwidth throttling is presented in Fig. 18, where a shaded area on an AP's timeline indicates a 10-second interval during which the AP is throttled. For comparison, we replace Po-Fi APs with regular WiFi APs and throttle their bandwidths as in Fig. 18, and let the mobile client to always connect to the AP that has the strongest signal, while transmitting to the wired host with one TCP flow during its movement.

5.5.2. Result

We repeat each experiment 5 times. We present the TCP throughput in regular WiFi, and the throughputs of the MPTCP connection in Po-Fi as well as its two subflows in boxplots in Fig. 19(a) . We can see that the throughputs of the TCP flow and the MPTCP subflows have large variances, but the MPTCP's overall throughput is quite stable.

We select one representative experiment execution, and present how the client's throughputs in the Po-Fi and regular WiFi networks change over time in Fig. 19(b). From the figure, we can see that in regular WiFi, the TCP throughput fluctuates greatly. The throughput drops are caused by two factors: 1) packet losses during the naive AP handovers initiated by the client, and 2) bandwidth throttling on the AP that the client connects to.

On the other hand, in the Po-Fi network, the client's throughput remains stable. Since the MPTCP connection is composed of two subflows, we also present the throughputs of the two MPTCP subflow in Fig. 19(c), and plot in Fig. 18 how the mobile client's primary and secondary VAPs migrate among the four Po-Fi APs as a result of client movement. By examining the figures, we can see that although the bandwidth throttling on the Po-Fi AP that carries a subflow can greatly reduce its throughput, however, since at most one subflow is throttled, the MPTCP protocol stack can schedule the other subflow, which is carried by an unthrottled AP, to transfer more data for compensation. Moreover, comparing with the naive handover, the VAP migrations cause few packet losses. Combining these factors, the client's overall MPTCP throughput is maintained at about 12 Mbps under the Po-Fi facilitated "mobility+multipath" service, and is much more stable than the TCP throughput in the regular WiFi network.

In Fig. 20, we present the traffics transmitted by each of the four Po-Fi APs. We can see that the client's traffic is jointly dispatched among the four APs by our controller program and the end-end MPTCP protocol, so that an unthrottled AP close to the mobile client is always scheduled to transmit more data than the other APs, which is ideal for mobile wireless networks.

5.6. Smooth Remote AP Switching

5.6.1. Experiment setup

In this section, we realize and evaluate the smooth remote AP switching use case as described in Section 4.3. In particular, we use a Po-Fi AP with two monitor WiFi interfaces on different channels, program it into the WDS mode for serving as a bridge AP, and place four regular WiFi APs as candidate remote APs at AP_1, AP_3, AP_4, and AP_5 of the testbed. We develop a controller program that instructs the bridge AP to set up a wireless backhaul link with a remote AP. In addition, the controller consistently monitors the signal strengths of the Beacon frames from the remote APs, and instructs the bridge AP to switch to the AP with the strongest signal by following the steps in Section 4.3.

In our experiment, we move the mobile bridge AP in a counterclockwise order as in Section 4.1 in 60 seconds, and during the movement, the bridge AP is connected by the laptop wireless client that moves along with it. We let $r = 3$ and $\delta = 5$ dBm in the experiment. For comparison, we use a regular AP as the bridge AP, configure its WiFi interface into the 4-address mode for setting up a WDS wireless backhaul link, and run a script to switch its remote AP to the closest AP during its movement. In both the Po-Fi and regular WiFi networks, we initialize a UDP flow at a constant rate of 20 Mbps from the client to a wired host for measuring the end-end throughput.

5.6.2. Result

In Fig. 21, we present the throughputs of the UDP flows that traverse the wireless backhaul links in both the Po-Fi and regular WiFi networks. From the figure we can see that in regular WiFi, the throughput drops to zero when the mobile bridge AP naively switches its remote AP, as it must first disconnect from its current AP before connecting to a new one, and considerable packets are lost during the switching.

On the other hand, in the Po-Fi facilitated smooth AP switching, the throughput remains stable at about 10 Mbps all the time. We explain the stable throughput with the fact that during the AP switching, the controller instructs the mobile bridge AP to connect to a new AP before it dissociates from its current remote AP, and with such a "make-before-break" approach, no packet shall be lost. To verify our point, we use the method as in Section 5.3 to examine the packet losses, and find that on average, 196 packets are lost during a naive AP switching, but we do not observe any packet loss in the Po-Fi network.

Note that although both seamless mobility and this use case achieve smooth AP switching, they are very different. In the seamless mobility use case, the switching is accomplished by migrating VAP between Po-Fi APs, and is transparent to client. But in this use case, the switching is initiated from the Po-Fi programmed bridge AP that plays the client side on the wireless backhaul link, and is compatible with legacy remote APs.

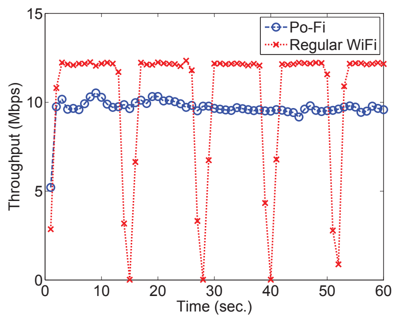


Figure 21: Comparison of throughputs on wireless backhaul links in Po-Fi and regular WiFi networks.

5.7. TDMA MAC Scheduling

5.7.1. Experiment setup

We realize and evaluate the TDMA MAC scheduling use case as described in Section 4.4 in this experiment. In particular, we develop TDMA APs, and realize a Po-Fi facilitated per-flow TDMA MAC scheduling, in which the controller can allocate time slots to specific network flows. All the APs in our experiment are developed on PC, and we employ the Precision Time Protocol (PTP)[42] for clock synchronizing.

We place two TDMA APs at AP_2 and AP_6 of the testbed that are about 50 m apart. The two APs have their WiFi interfaces working on a same channel, but they are out of each other's carrier sense range. We configure two virtual WiFi interfaces on the laptop mobile client, each connecting to one TDMA AP. The client sets up an MPTCP connection containing two subflows for downloading from a wired host, with each AP carrying one subflow. The wireless client moves as the following: Initially, the client is 10 m away from AP_2 and moves towards AP_6, when it is 10 m away from AP_6, it returns back towards AP_2 and eventually stops at the start position. The movement lasts 150 seconds, during which the client downloads through AP_2 and AP_6 all the time. Clearly in such an experiment setting, AP_2 and AP_6 are hidden terminals to each other when they are transmitting to the mobile client.

As described in Section 4.4, in the Po-Fi facilitated TDMA MAC scheduling, the TDMA APs are programmed to measure and predict RSSIs from the overheard 802.11 frames, and report to controller every $N = 50$ frames. The controller uses Equation (2) to allocate time slots to the two subflows. We divide time into 12 time slots on the TDMA APs, with each slot lasting 20 ms, and set the threshold $d = 2$ for updating the time slot allocations.

For comparison, we also consider two alternative cases: In the first case, we replace TDMA APs with regular WiFi APs, so that the two subflows contend for medium access without coordination; and in the second case, we statically allocate 6 time slots on each TDMA AP all the time.

5.7.2. Result

We present the client's MPTCP throughputs under 1) no TDMA, 2) static time slot allocation, and 3) Po-Fi facilitated dynamic time slot allocation in boxplots in Fig. 22(a). From the figure we can see that without TDMA, the throughput is very low, due to the hidden terminal problem. When TDMA is enabled,

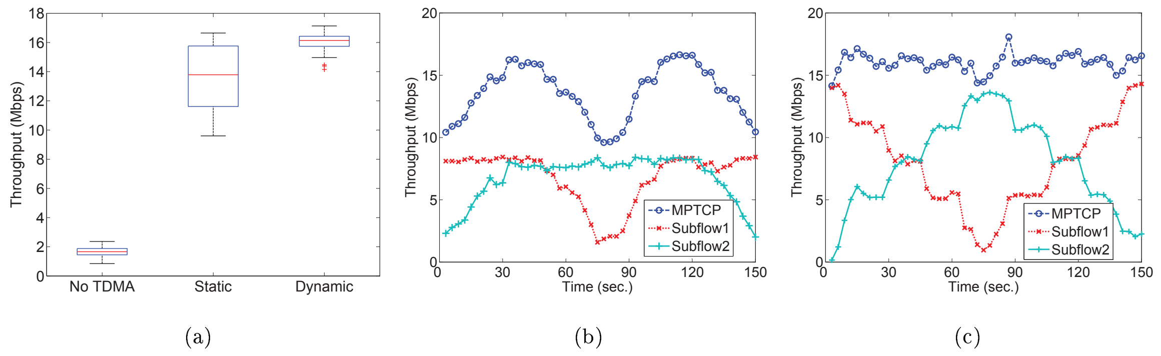


Figure 22: (a) Boxplots of MPTCP throughputs in various cases; (b) MPTCP and subflow throughputs under static time slot allocation; (c) MPTCP and subflow throughputs under dynamic time slot allocation.

the throughput is much higher, as the two subflows no longer contend for medium access, but transmit in their own time slots. Moreover, with the Po-Fi facilitated dynamic allocation, the client's throughput is higher and more stable than the one under the static allocation.

In Fig. 22(b) and (c), we present how the throughputs of the MPTCP connections, as well as their subflows, change over time under the static and dynamic time slot allocations respectively. From the figures we can see that, with the static allocation, the overall MPTCP throughput drops when the client is close to one AP but far away from the other. On the other hand, with the Po-Fi facilitated TDMA MAC scheduling, the controller dynamically revokes time slots from the AP that has a poor signal reception, and allocates them to the AP that can better transmit the data, thus achieves a higher and more stable MPTCP throughput for the mobile client. The experiment suggests that Po-Fi is highly flexible to provide programming interface for novel functionalities at lower layers in WiFi.

6. Conclusion

In this paper, we presented Po-Fi, a software-defined WiFi network architecture. Different from the previous NFV-based solutions, Po-Fi follows the SDN consensus by abstracting WiFi AP as a programmable forwarding pipeline, thus is more flexible and provides a rich and unified programmability for realizing innovations on WiFi networks. We discussed a wide range of use cases facilitated by Po-Fi, and described their realization details. We implemented a Po-Fi prototype with commodity hardware, and evaluated the use cases with real-world testbed. Experiment results showed that the Po-Fi facilitated innovations can effectively overcome WiFi's inherited deficiencies and improve the wireless services.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grants 61672486 and 61672485, and in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDC02070300.

References

[1] Cisco visual networking index: Forecast and trends, 2017-2022, White paper, Cisco (Feb. 2019).

[2] B. Dezfouli, V. Esmaeelzadeh, J. Sheth, M. Radi, A review of software-defined WLANs: Architectures and central control mechanisms, IEEE Commun. Surveys Tuts. 21 (1) (2019) 431–462.

[3] OpenFlow switch specification version 1.5.1, Standard ONF TS-025, Open Networking Foundation (Mar. 2015).

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming protocol-independent packet processors, ACM SIGCOMM Computer Communication Review 44 (3) (2014) 87–95.

[5] H. Song, Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane, in: Proc. of ACM SIGCOMM HotSDN'13, Hong Kong, China, 2013.

[6] P. Lu, L. Zhang, X. Liu, J. Yao, Z. Zhu, Highly-efficient data migration and backup for big data applications in elastic optical inter-datacenter networks, IEEE Netw. 29 (2015) 36–42.

[7] L. Gong, Z. Zhu, Virtual optical network embedding (VONE) over elastic optical networks, J. Lightw. Technol. 32 (2014) 450–460.

[8] S. Tang, B. Hua, Increasing multicast transmission rate with localized multipath in software-defined networks, Front. Comput. Sci. 13 (2019) 413–425.

[9] The world's fastest & most programmable networks, White paper, Barefoot.

[10] Programming Netronome Agilio SmartNICs, White Paper WP-NFP-Prog-Model-7/18, Netronome Systems (2018).

[11] hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS authenticator, `https://w1.fi/hostapd/`, accessed: May. 25, 2020.

[12] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, The click modular router, ACM Trans. on Computer Systems 18 (3) (2000) 263–297.

[13] J. Vestin, P. Dely, A. Kassler, N. Bayer, H. Einsiedler, C. Peylo, CloudMAC: Towards software defined WLANs, ACM SIGMOBILE Mobile Computing & Communication Review 16 (4) (2012) 42–45.

[14] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, R. Merz, Programmatic orchestration of WiFi networks, in: Proc. of USENIX ATC'14, Philadelphia, PA, USA, 2014.

[15] C. Xu, W. Jin, G. Zhao, H. Tianfield, S. Yu, Y. Qu, A novel multipath-transmission supported software defined wireless network architecture, IEEE Access 5 (2017) 2111–2125.

[16] IEEE 802.11 part 11: Wireless LAN MAC and PHY specifications, Ieee standard, IEEE (Dec. 2016).

[17] M. Yan, J. Casey, P. Shome, A. Sprintson, A. Sutton, Ætherflow: Principled wireless support in SDN, in: Proc. of IEEE ICNP'16, Nov., 2016.

[18] E. Coronado, S. N. Khan, R. Riggio, 5G-EmPOWER: A software-defined networking platform for 5g radio access networks, IEEE Trans. Netw. and Service Manag. 16 (2) (2019) 715–728.

[19] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, T. Rasheed, Programming abstractions for software-defined wireless networks, IEEE Trans. Netw. and Service Manag. 12 (2) (2015) 146–162.

[20] Meru technical note - what is shared BSSID and per-station BSSID?, `https://kb.fortinet.com/kb/documentLink.do?externalID=FD37783`, accessed: May. 25, 2020.

[21] Y. Yiakoumis, M. Bansal, A. Covington, J. van Reijendam, S. Katti, N. McKeown, BeHop: A testbed for dense WiFi networks, in: Proc. of WiNTECH'14, Maui, HI, USA, 2014.

[22] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, G. Parulkar, Blueprint for introducing innovation into wireless mobile networks, in: Proc. of ACM SIGCOMM VISA'10, New Delhi, India, 2010.

[23] E. Zeljkovic, J. M. Marquez-Barja, A. Kassler, R. Riggio, S. Latré, Proactive access point driven handovers in IEEE 802.11 networks, in: Proc. of CNSM'18, Rome, Italy, 2018.

[24] A. Croitoru, D. Niculescu, C. Raiciu, Towards WiFi mobility without fast handover, in: Proc. of USENIX NSDI'15, Oakland, CA, USA, 2015.

[25] M. R. Palash, K. Chen, MPWiFi: Synergizing MPTCP based simultaneous multipath access and WiFi network performance, IEEE Trans. Mobile Comput. 19 (1) (2020) 142–158.

[26] C. Xu, W. Jin, X. Wang, G. Zhao, S. Yu, MC-VAP: A multi-connection virtual access point for high performance software-defined wireless networks, J. Netw. Comput. Appl 122 (2018) 88–98.

[27] P. Djukic, P. Mohapatra, Soft-TDMAC: A software-based 802.11 overlay TDMA MAC with microsecond synchronization, IEEE Trans. Mobile Comput. 11 (3) (2012) 478–491.

[28] Z. Yang, J. Zhang, K. Tan, Q. Zhang, Y. Zhang, Enabling TDMA for today's wireless lans, in: Proc. of IEEE INFOCOM'15, Hong Kong, China, 2015.

[29] S. Zehl, A. Zubow, A. Wolisz, hMAC: Enabling hybrid TDMA/CSMA on IEEE 802.11 hardware, Technical Report TKN-16-004, Technische Universität Berlin (Nov. 2016).

[30] Protocol Oblivious Forwarding projects, `https://github.com/ProtocolObliviousForwarding`, accessed: May. 25, 2020.

[31] X. Wang, Y. Tian, M. Zhao, M. Li, L. Mei, X. Zhang, PNPL: Simplifying programming for protocol-oblivious SDN networks, Computer Networks 147 (2018) 64–80.

[32] M. Li, X. Wang, H. Tong, T. Liu, Y. Tian, SPARC: Towards a scalable distributed control plane architecture for protocol-oblivious SDN networks, in: Proc. of ICCCN'19, Valencia, Spain, 2019.

[33] Wi-5 project, `https://github.com/Wi5`, accessed: May. 25, 2020.

[34] P. Goodwin, The Holt-Winters approach to exponential smoothing: 50 years old and going strong, FORESIGHT 19 (2010) 30–33.

[35] S. Sundaresan, N. Feamster, R. Teixeira, Locating throughput bottlenecks in home networks, ACM SIGCOMM Comp. Comm. Rev. 44 (4) (2014) 351–352.

[36] C. Paasch, O. Bonaventure, Multipath tcp, Communications of the ACM 57 (4) (2014) 51–57.

[37] C. Paasch, S. Barre, MultiPath TCP - Linux Kernel implementation, `https://multipath-tcp.org`, accessed: May. 25, 2020.

[38] V. Mhatre, K. Papagiannaki, Using smart triggers for improved user performance in 802.11 wireless networks, in: Proc. of MobiSys'06, Uppsala, Sweden, 2006.

[39] OpenWrt 15.05.1, `https://openwrt.org/releases/15.05/start`, accessed: May. 25, 2020.

[40] ath9k wireless driver, `https://wireless.wiki.kernel.org/en/users/drivers/ath9k`, accessed: May. 25, 2020.

[41] Floodlight OpenFlow controller, `http://www.projectfloodlight.org/floodlight/`, accessed: May. 25, 2020.

[42] IEEE standard for a precision clock synchronization protocol for networked measurement and control systems, Ieee standard, IEEE (Jul. 2008).