International Journal of Innovative Computing, Information and Control
Volume **16**, Number **2**, April **2020**

ICIC International ©2020 ISSN 1349-4198
pp. **663–679**

AN SDN-BASED WLAN SYSTEM FOR TERMINAL MOBILITY: DESIGN AND IMPLEMENTATION

Yifeng Liu[1,2], Xuewen Zeng[1,2], Rui Han[1,2] and Xu Wang[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ liuyf; zengxw; hanr; wangx }@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received July 2019; revised November 2019

Abstract. *Mobility support is one of the goals of wireless local area networks (WLANs). However, current 802.11-based WLAN structure cannot meet the need for flexible mobility management in high-density deployment. And for delay-sensitive services, mobility handoff in traditional 802.11 protocol will incur unacceptable interruption. In this paper, we propose an SDN-based 802.11 WLAN system and regain the control of handoff from clients in order to achieve faster mobility handoff. On data plane, three modules are designed and implemented working with openvswitch and cfg80211 subsystem in the kernel-space of access point (AP) to include 802.11 devices into SDN domain. On control plane, we implement our controller based on an SDN controller (ONOS), which runs a novel algorithm to make handoff decision. The algorithm takes channel idle evaluation and received signal strength indication (RSSI) into consideration to achieve load balancing. For southbound interface, we use Packet-in and Packet-out messages to carry wireless related information between control and data plane. Moreover, we introduce a unicast channel switch announcement (CSA) frame to speed up terminal's mobility handoff by avoiding handshake overheads in the traditional way. Our proposed system is demonstrated through a real prototype and a test scenario. The experiments results show that terminal mobility handoffs in proposed system are faster than those in the traditional way.*

Keywords: IEEE 802.11, WLAN, Software-defined network, Fast mobility handoff

1. **Introduction.** Mobility and wireless networks are intertwining. It is this advantage that wireless networks are widely used, such as cellular network [1], wireless sensor network [2], and 802.11-based WLAN (WiFi) [3]. As one of the most widely used non-3GPP wireless access technologies, WiFi provides low-cost Internet access services. Many mobile operators deploy WiFi hotspots to offload traffic from base stations in some high-density areas. From IEEE 802.11a/b/g/n to 802.11ac and even 802.11ax, the improvement of performance will enlarge the deployment scale in the near future.

On one hand, flexible management is the prerequisite for supporting mobility. However, current 802.11 WLAN infrastructure cannot meet the need for flexible management especially in high-density deployment. Control plane and data plane being coupled tightly restricts the innovation of intelligent network applications and services, e.g., mobility management. On the other hand, with the emergence of more and more real-time mobile applications or services like radio stream or VoIP, it is significant for user experience to

DOI: 10.24507/ijicic.16.02.663

keep the continuity of these delay-sensitive services. Compared with delay-tolerant ones, these services have higher demands on mobility handoff in that the continuity of wireless link should be guaranteed which involves in 802.11's medium access control (MAC) layer and even physical (PHY) layer. However, mobility handoff in traditional 802.11 protocol will result in unacceptable interruption because of the handshake overheads.

On the aspect of infrastructure, even though CAPWAP is proposed to manage all APs by a global controller [4], it does not give administrators good programming capabilities on various wireless resources. IEEE 802.11v/k provides a distributed way to share more information with terminals for more optimal handoff decision. And 802.11r simplifies authentication to achieve fast handoff. However, these protocols still have some problems in overheads, security and stability [5,6]. Recently, applying software-defined networking (SDN) to wireless domain has been studied. SDN is an innovative paradigm which simplifies network management [7,8] and promotes the innovation of network applications and services. In SDN, the control plane and data plane are decoupled and connected together through southbound interface (SBI). The controller abstracts protocol-specific messages to protocol-agnostic information and exposes it to upper application layer via northbound interface (NBI). In this approach, SDN gives the applications abilities to program underlying networks. Extending SDN to wireless network is an interesting idea. Software-defined wireless network (SDWN) recently gets a lot of attention, because wireless domain is more complicated than wired network in many aspects. The introduction of SDN gives more fine-grained controls to 802.11 wireless network. Up to now, many architectures have been presented, such as OpenRoads [9], CloudMAC [10], Odin [11], and MP-SDWN [12].

Mobility management is an important network application in wireless networks. The key is to provide continuous data transmission with no human intervention when handoff occurs. From the view of protocol layers, this is a cross-layer problem which involves in layers from application layer to physical layer. In IP layer and above, many schemes of mobility have been proposed, such as Mobile IP [13,14], decoupling identifier and locator in information-centric network (ICN) [15,16], connection session IDs in *cookies* [17] or QUIC [18]. However, these solutions focus on the continuity of upper-level sessions and routes and do not care about the underlying MAC and PHY layer. The continuity of MAC and PHY layer means uninterrupted point-to-point data link. As mentioned above, there are more and more delay-sensitive mobile applications today. So maintaining this continuity is important to improve user experience, e.g., user is watching live streaming video when mobility handoff occurs.

In this paper, we design and implement an SDN-based 802.11 wireless system. By regaining the control of handoff from wireless terminals, we get faster wireless handoff which is a key part of mobility support. In summary, the contribution of our work includes:

- On data plane, for higher processing performance, we design three modules in AP's kernel-space working with *cfg80211* subsystem and *openvswitch* to integrate 802.11 wireless into SDN.
- On control plane, we implement the controller based on an SDN controller (*ONOS*) to manage and control all 802.11 devices. And on the controller, we run a mobility management application to speed up wireless handoff.
- For southbound interface, we use *OpenFlow*'s *Packet-in* and *Packet-out* messages to carry out communication between control and data plane. Modules on AP are attaching to *openvswitch* through a virtual network interface.
- We introduce a unicast *CSA* frame to steer terminals' handoff. AP selection is based on received signal strength indication (RSSI) and channel idle estimation.

Furthermore, we abstract the *connection and security context* of each terminal on the network side to support fast mobility handoff.

The rest of the paper is organized as follows. In Section 2, we survey related work. In Section 3, we introduce the system including AP, controller architectures and the southbound interface. In Section 4, we discuss how the terminals execute handoff under the control of the controller. In Section 5, we give a real prototype and experiment scenario. And experiment results are shown as well. Furthermore, we analyze and discuss the handoff. Finally, we conclude the paper with final remarks and discuss future work in Section 6.

2. **Related Work.**

2.1. **SDN architecture in wireless.**

Separating control plane and data plane, SDN makes networks programmable and easy to be managed. Extending SDN to wireless networks is reasonable as wireless network is more complicated than wired network in some aspects like resource management and devices mobility management. Recently, software-defined wireless network has been studied.

OpenRoads [9] is the first work to apply SDN in wireless domain. It separates control plane and data plane and provides a wireless extension *OpenFlow*. Its architecture consists of three layers: flow, slicing, and controller, which provides flexible control, virtualization and abstraction. The prototype of OpenRoads was deployed on campus. SDWN [19] applies SDN approach to LR-WPANs which is an IEEE 802.15.4-based low rate wireless network. It analyzes the advantages of the SDN approach in LR-WPANs, and identifies the difference between 802.3 network and LR-WPANs. And authors deploy a large SDN testbed with a controller. However, it does not target 802.11 wireless network. CloudMAC [10] is an *OpenFlow* based architecture, which consists of virtual access points (VAPs), wireless termination points and OpenFlow switches. It migrates some functions of 802.11 MAC layer, like management frames generation, to VAP and leaves time-constraint functions (ACKs and frame retransmissions) on AP's driver. The VAP is an operating system instance running on the remote virtualization host. It uses *hostapd* to generate management frames and to provide authentication and authorization services. Each VAP has one or several virtual WLAN cards, which are based on *mac80211_hwsim*. However, there is no discussion on implementation and it does not provide programmability of network. Odin [11,20,21] is an SDN framework for enterprise WLANs, with aim to simplify the implementation of high level services. Author introduces the concept of light virtual access point (LVAP) to realize wireless virtualization. Odin uses Floodlight Controller to manage all APs and uses Click Modular Router [22] on APs to boost accelerate packets processing. However, the paper does not give more implementation details on how the Click interacts with wireless driver and does not mention how to control wireless card. Moreover, the Click has the problem of platform dependency. MP-SDWN [12] is based on Odin. By introducing a wireless flow transmission rule table, MP-SDWN can control wireless data flow flexibly. The core of the paper is to use multipath to increase transmission bitrates. It introduces the concept of MVAP (Multi-connection Virtual AP), which is similar to network function virtualization (NFV), to bind a terminal to multi physical APs in order to increase bandwidth. The paper gives the schematic of controller and Linux-based AP daemon. And in its subsequent work [23], authors elaborate the details of APs. However, the paper making all APs on the same channel may be impractical because of the severe co-frequency interference. [24] similarly uses Click and *openvswitch* to build up an SDN-based architecture. Using SDN/NFV technologies, the paper introduces LAP (Logical Access Point) to manage terminals mobility. Authors implement APs

on OpenWRT wireless routers which connect to controller through a switch. However, it does not give many details of implementation of both AP and controller. And for fast handoff, making all APs on the same channel will badly degrade the network capacity.

2.2. **Handoff in 802.11.** Traditional 802.11 handoff includes channels scanning, authentication, (re)association and authorization [25], which takes lots of time. Standards (IEEE 802.11k/v/r) for access points allow terminals to make better performance-impacting decisions such as choosing to roam between AP [24] and to reduce interrupt delay of handoff. However, it still has some problems [4]. Some researchers migrate handoff decision to network side by introducing the concept of "LVAP", "VAP" or "LAP" [10,11,20,24,26,27], which represents a virtual access point. Each terminal will be given a BSSID-different VAP, and the VAP will be migrated to the corresponding physical AP as terminal moving. So, a physical AP could have a list of VAPs, and the migration of VAP maintains the connection relationship after handoff. However, this has a limit that all APs should be set to the same channel, which will result in great interference. [28,29] introduce channel switch announcement (*CSA*) element to solve this problem, but one-terminal-one-VAP means that AP should send different beacon frames periodically to maintain the user connections. This will result in significant overheads of communication especially in high-density deployment, though [12] slightly increases the interval of beacon frames. So, in this paper, we use a unicast action frame to steer terminal's handoff and abstract *connection and security context* rather than VAP to support fast mobility handoff.

3. **Proposed System.** Our system is organized into a centralized architecture mainly including APs (Access Point) and a controller, as shown in Figure 1. APs connect to controller through an SDN network which has faster path convergence. In application layer of controller, we implement mobility management application. More details will be discussed in this section.

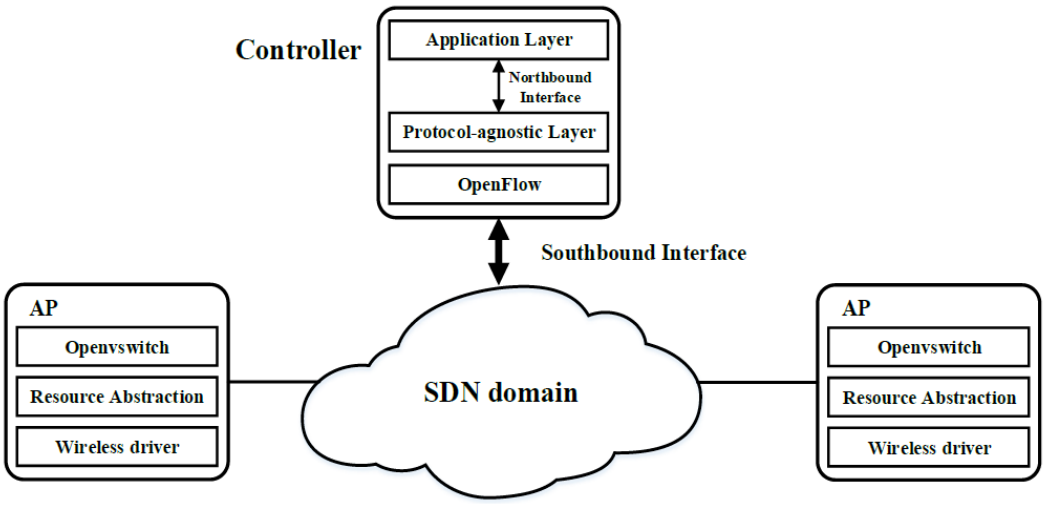

FIGURE 1. Overview of system (simplistic view)

3.1. **Wireless structure in Linux kernel-space.** To avoid great memory copy between kernel and user space, our modules are implemented in the Linux kernel-space. There are two kinds of wireless cards (802.11) in Linux, FullMAC and SoftMAC. The main difference between them is that for FullMAC card, media access control sublayer management entity (MLME) is managed in hardware while for SoftMAC card, MLME is in software. We use

the SoftMAC wireless card in order to get more fine-grained control abilities, though the FullMAC executes faster.

Linux kernel uses *cfg80211* subsystem and *mac80211* subsystem to manage two kinds of wireless cards, as shown in Figure 2. *Mac80211* is a driver frame for SoftMAC wireless driver including data path and management path. On data path, *mac80211* bridges 802.11 to 802.3 network through a network interface. And every physical wireless card can have several interfaces each of which can work in different modes (e.g., one AP mode, one monitor mode). On management path, *mac80211* exposes application program interfaces (APIs) to *cfg80211*. These APIs are implemented by wireless driver and invoked by upper level users.

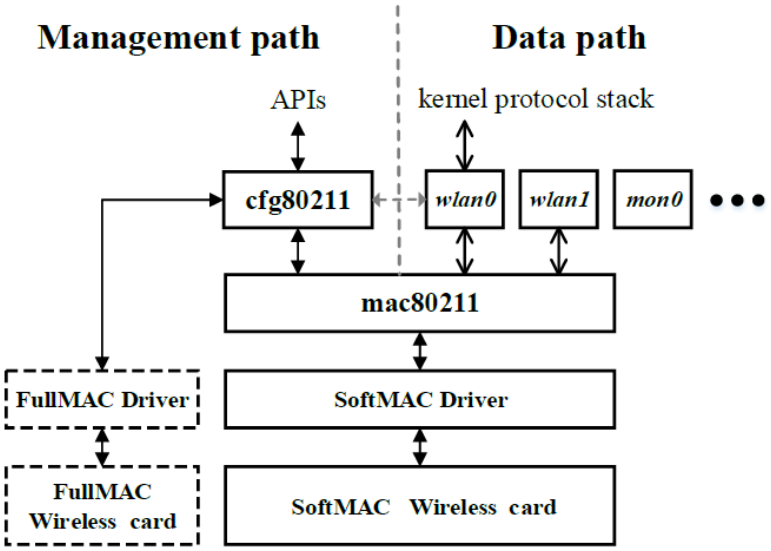

FIGURE 2. Overview of wireless structure in Linux kernel-space

Cfg80211 subsystem, used by all modern wireless drivers in Linux, is the configuration API for 802.11 devices in Linux. It bridges user-space and drivers, and offers some utility functionality associated with 802.11. Through it, user can easily control the behavior of wireless cards.

Table 1 gives the description of some important structures and functions in kernel-space. Through these, we get the capabilities and status of wireless card or driver and hook our processing into kernel protocol stack working with *openvswitch* and *cfg80211*. In later sections, we will implement our AP based on these.

3.2. **AP architecture.** The high-level architecture of AP is depicted in Figure 3. There are three modules written in C: *Perception* module, *MLME* module and *Configuration* module. Each of them is in a separate kernel thread working with *openvswitch* and *cfg80211*. On the data forwarding path, we bridge wired (*eth0*) and wireless interface (*wlan0*) together through *datapath*, which is actually a special bridge.

3.2.1. *Wconf.* We create a virtual network interface (*wconf*) in kernel-space. It is a *net_device* structure which can own IP, MAC and be attached to bridge or *openvswitch*. As shown in Figure 3, *wconf* is a link between modules and Ethernet. Modules will set packets' entrances to *wconf* (*skb's dev* = *wconf*) before sending them out so that *openvswitch* can identify modules' traffic through network interface. For those packets sent to modules, *wconf* is a messages dispatcher. In its *ndo_start_xmit* function, we parse packets and send them to the corresponding queue waiting for further processing by

TABLE 1. Some structures and functions in Linux kernel-space

net_device	An abstraction of device and it presents a network interface, e.g., *wlan0, wlan1, eth0.*
wiphy	The fundamental structure for each 802.11 device connecting to Linux. It describes wireless hardware.
wireless_dev	Wireless part of a *net_device.* It presents a virtual interface in 802.11 part.
cfg80211_ops	APIs of wireless exposed to users. It can be gotten through *wiphy.*
sk_buff	Represent a packet in kernel. *Sk_buff's dev* means the input device of this packet.
ptype_base	Aglobal chain including all registered IP layer protocols, e.g., *ipv4, ipv6.* We can register our own processing function for specific protocol.
ndo_start_xmit()	*Net_device*'s TX function. It will be invoked by Linux kernel protocol stack to send package to driver.
netif_receive_skb()	The entrance of Linux kernel protocol stack on receiving path. It will iterate through *ptype_base* and invoke corrected processing function to process packet according to its characteristics.

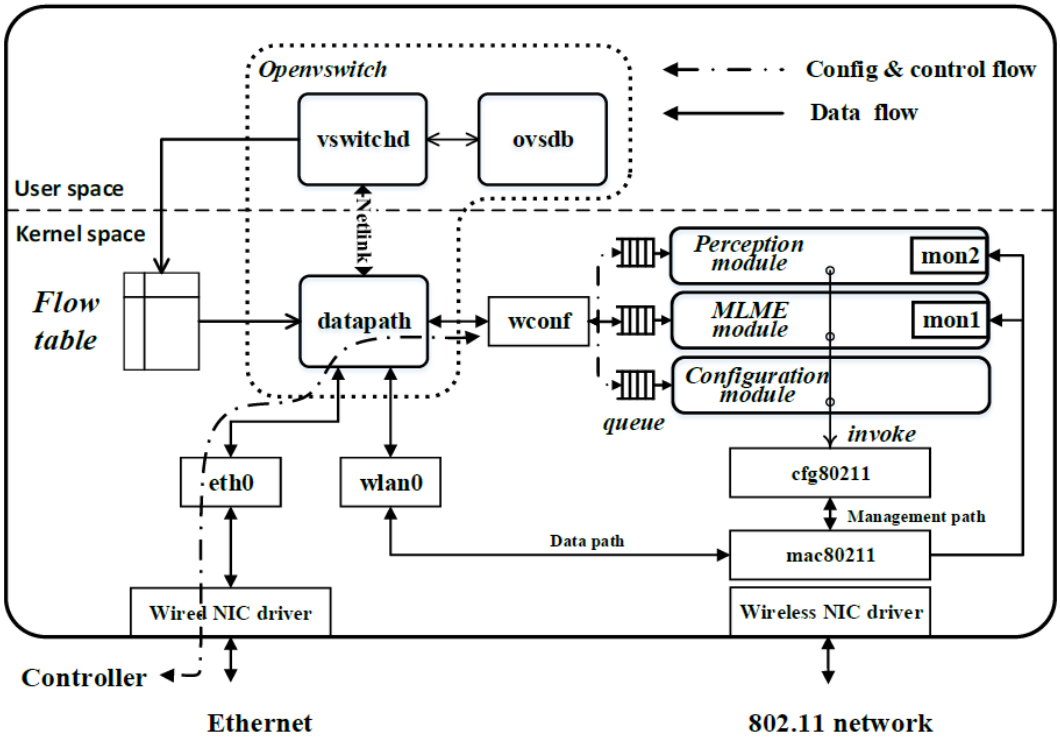

FIGURE 3. AP implementation

relevant module. Modules are in different threads from *ndo_start_xmit*, so that modules' asynchronous time-consuming operations will not take up time slices of protocol stack.

3.2.2. *Configuration module. Configuration* module, which interacts with *cfg80211* directly, is responsible for telling the controller what capabilities the wireless card enables. And through it, controller can configure and manage wireless card's behaviors, like *BSS*, channel and bandwidth setting, beacon frame and probe reply frame, beacon interval,

security-related parameters. Moreover, this module will report runtime parameters of wireless card and associated terminals periodically or when asked by controller.

3.2.3. *MLME module.* *MLME* module is responsible for terminal access, including *Authentication*, *Deauthentication*, *Association*, *Disassociation*, and *Reassociation*. The module sends these management frames through *cfg80211*, and receives them via a wireless interface working in monitor mode (*mon1* in Figure 3). In Linux kernel, *mac80211* subsystem will duplicate each 802.11 frame and send the copy with a *radiotap* header to every monitor interface by invoking *netif_receive_skb* (Table 1). To capture these copies of management frame for processing, we register a processing function in *ptype_base* chain. And Ethernet protocol type of these packets is *ETH_P_802_2 (LLC)*.

The key of one terminal's fast mobility handoff is its *connection and security context*, which is built when it accesses to the wireless system, including terminal's *mac address*, *supported rates*, *listen interval*, *supported channels*, *VH & VHT capabilities (e.g., HT short GI for 20/40 MHz, A-MPDU parameters, and HT support channel width), (extern) capabilities (e.g., short preamble, short slot time, and automatic power save), AID (associate ID), IV/PN for TKIP and CCMP keys, cipher suite selector* (defined in IEEE 802.11i) *and association state machine*. These parameters are essential for AP's wireless driver to communicate with the terminal. *MLME* module will store these and sync them to the controller.

3.2.4. *Perception module.* APs are deployed on different channels. *Perception* module is to detect terminals around on different channels. As most wireless drivers do not support driver-level off channel detection, using a cheap auxiliary card to do this without interrupt communication of master wireless card is a common way. The auxiliary card works in monitor mode (*mon2* in Figure 3). It continually switches channel and stays for a dwelling time to capture all 802.11 frame on current channel. It can get *RSSI*, *Noise* information from *radiotap* header of each frame, and report to the controller periodically.

3.3. **Controller.** The controller has a global perspective, and it can get network information and control all 802.11 devices through southbound interface.

3.3.1. *Southbound interface.* As shown in Figure 4, communication between APs and controller relies on *Packet-in* and *Packet-out* messages (defined in *Openflow*). Packets between *openvswitch* and modules are proprietary messages, which are simple MAC layer frames created by us and used to carry 802.11-related information. For uplink, *openvswitch* has a flow table entry so that proprietary message from modules (entrance is *wconf*) will be sent to the controller by *Packet-in* message. For downlink, the controller uses *Packet-out* messages to carry proprietary messages to *openvswitch* on the AP. The "output" field in *Packet-out*'s header is set to *wconf* so that *openvswitch* will extract proprietary message from *Packet-out* message and send it to kernel modules via *wconf* interface.

3.3.2. *Architecture.* Our controller is based on *ONOS* (Open Network Operating System), which is a commercial-grade high-performance SDN controller. Thanks to its modularization and high abstraction of network elements, our modules can not only own a global view of network but use various useful built-in services. There are three layers in *ONOS* named Provider, Core and Apps. Modules in Provider layer are protocol-aware network-facing that interact with network directly. They send protocol-specific descriptions to registered modules in Core layer. And the latter abstracts these protocol-specifics into protocol-agnostic *Model Objects* which are what *ONOS* exposes to its applications in Apps layer. In short, *Model Objects* are *ONOS*'s protocol-agnostic representations of various network elements and properties.

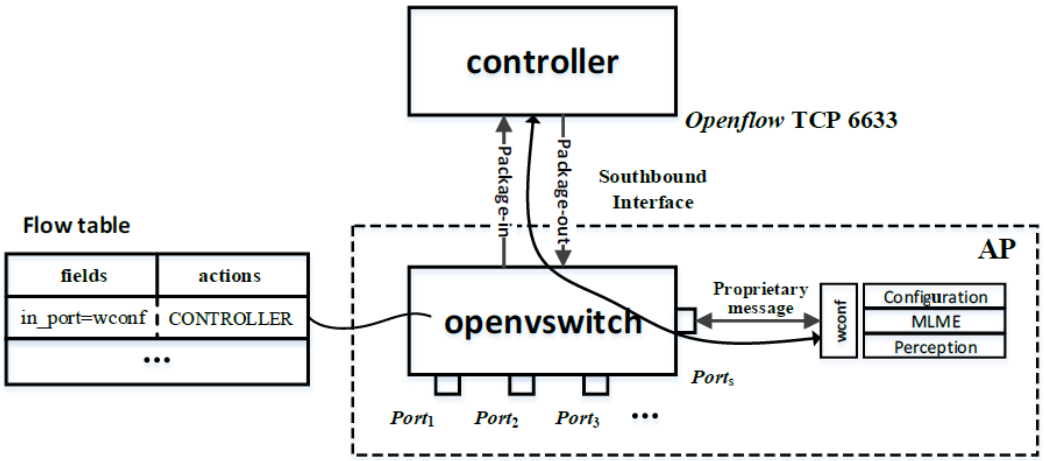

FIGURE 4. Southbound interface

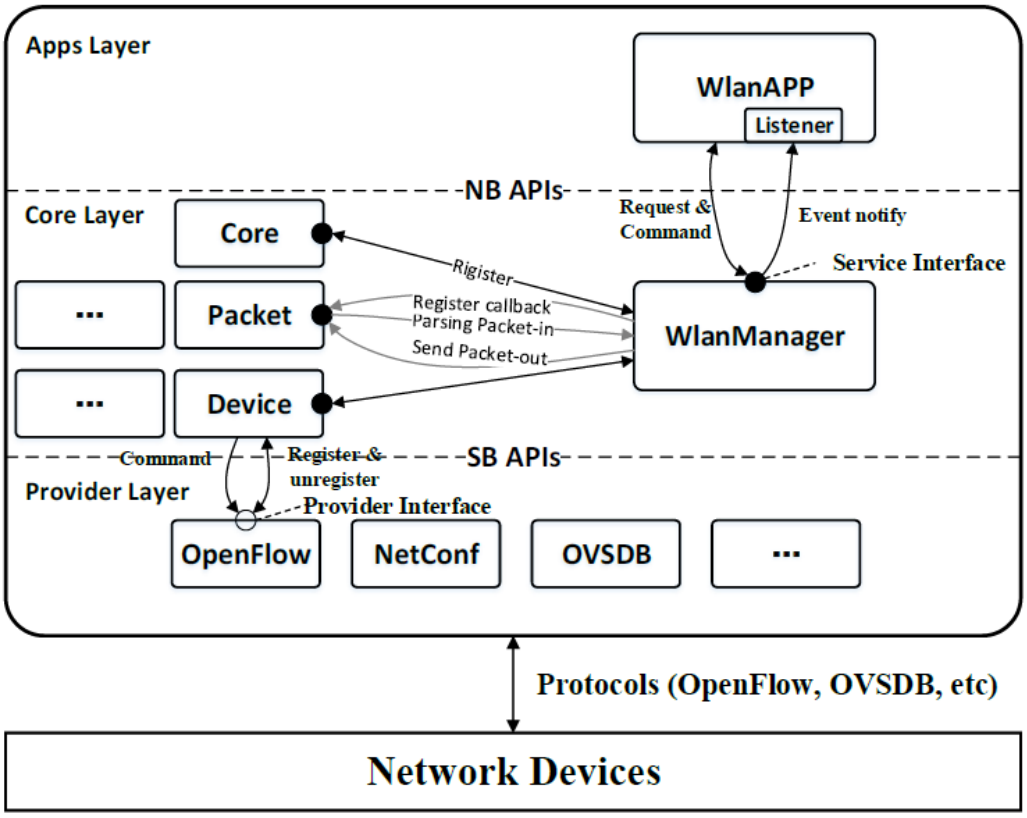

FIGURE 5. Architecture of controller

In Figure 5, we give the overview of our controller. In it, *WlanManager* and *WlanAP-P* modules are implemented. The *WlanManager*, located in Core layer, is based on *Core* subsystem, *Packet* subsystem and *Device* subsystem. By registering a callback into *Packet* system, every *Packet-in* message will be processed by this callback. And then *WlanManager* gets 802.11-related proprietary messages and converts them into protocol-agnostic representations (*Model Objects*) for storing. Given that each *datapath* on AP will be allocated a globally unique *dpid*, we use $<dpid,mac>$ two-tuples to exactly retrieve information of each AP or terminal.

To expose 802.11 *Model Objects* to *WlanAPP* module, *WlanManager* imports a "Service" interface, which is a northbound interface. The "Service" interface contains synchronous and asynchronous operations. Synchronous operations refer to those through which applications or other modules can learn about a particular aspect of the network state. These network states or properties are stored locally and maintained by *WlanManager*. Synchronous operations will get results soon, because they do not involve packets interactions with APs. Instead, asynchronous operations are for taking administrative or configuration commands and applying them onto the network devices. These operations will make *WlanManager* send *Packet-out* messages with a proprietary part to APs. The reply messages returned from APs in *Packet-in* messages are asynchronous. So we implement an event dispatch mechanism in *WlanManager* to notify interested listeners the results of asynchronous operations and their interested events, as shown in Figure 5. Moreover, *WlanManager* will maintain the consistency between physical network status and 802.11 *Model Objects* stored locally.

Process of terminal mobility handoff is running in the *WlanAPP* module. As shown in Figure 5, through synchronous operations, *WlanAPP* can obtain all information of 802.11 devices and network. And *WlanAPP* implements an event listener to get asynchronous operations' results or interested events, e.g., changes of network states. Moreover, *WlanAPP* is able to manage and control all devices. And these are the basis of our mobility handoff application. More details will be shown in the next section.

4. **Mobility Support.**

In this paper, we focus on handoff in 802.11 MAC layer and physical layer, which is significant to delay-sensitive applications and services. It consists of two parts: handoff decision and handoff execution.

4.1. **Handoff decision.**

Handoff decision refers to when to handoff and how to select a new AP for the terminal to achieve load balancing. Traditional user-based handoff decision may be not optimal because terminal lacks of a holistic view. In our proposed system, handoff decision is made by controller, which takes received signal strength indication (RSSI) and channel idle condition into consideration.

In 802.11, since the energy of WiFi signal is usually at milliwat level, people used to convert received signal power P (mW) to the RSSI (dBm): $RSSI = 10 \log \left(\frac{P}{1\,\mathrm{mW}}\right)$. *RSSI* represents the strength of a received signal relative to 1 mW and is an important indicator to estimate wireless link quality. When environment noise is constant, it directly affects physical layer bitrates, because low RSSI (high signal-noise ratio) means high packet error rate (PER) which results in a lower TX bitrates. Due to various path losses in practice, P is usually less than 1 mW even though the transmitter is very close to the receiver. So RSSI is usually less than 0 dBm. Normally, we think it is not suitable to communicate when RSSI is less than -70 dBm [30]. Let average RSSI be $\overline{RSSI}$, which can be obtained from *cfg80211*. We set a threshold value θ ($\theta = -70$ dBm) to decide when to handoff. If the following inequality is true:

$$\overline{RSSI} < \theta \tag{1}$$

the controller will start to select a new AP for the terminal. Selecting AP takes the channel idle condition into account. The primary medium access control (MAC) technique of 802.11 is called distribution coordination function (DCF), which is a carrier sense multiple access with collision avoidance (CSMA/CA) scheme with binary slotted exponential backoff. The key is listen-before-send (LBS) that 802.11 devices can only send after the channel has been idle for time of a *DIFS* and some slot time. For cost sake, most 802.11 devices use virtual carrier sense mechanism to listen the channel. A terminal will set a NAV value (μs) in *Duration* field of each unicast frame's header before sending it out.

The value, including time of transmitting the frame, waiting for one *SIFS* interval and replying an ACK by receiver, is used to tell other terminals how long the channel will be occupied by this communication. Other terminals will hang up (stop back off) for the time before competing channel access again. In some way, *Duration* represents the channel busy condition. Besides basic access, 802.11 provides a request-to-send/clear-to-send (RTS/CTS) access mechanism. Transmitter can reserve a period for large packets transmission by sending an RTS frame to AP, the latter responds with a CTS frame to keep other terminals hanging up for a while just like *Duration* field does. So, our channel idle estimation is based on the proportion of unicast data frames and RTS/CTS frames.

Suppose that there are N APs. Controller has a vector $\boldsymbol{CIC}$ representing channel idle conditions:

$$\boldsymbol{CIC} = [B_1, B_2, \ldots, B_N] \tag{2}$$

where B_i is the estimation of idle condition of i-th AP's master card's channel:

$$B_i = 1 - \frac{\sum D_{data} + \sum D_{CTS}}{T} \tag{3}$$

Here T is the dwell time of auxiliary card in master card's channel, D_{data} and D_{CTS} represent the time specified by *Duration* field in unicast data frame and clear-to-send (CTS) frame, respectively. Moreover, for each terminal, there is a vector in controller:

$$\boldsymbol{R} = \left[\overline{RSSI_1}, \overline{RSSI_2}, \ldots, \overline{RSSI_N} \right] \tag{4}$$

where $\overline{RSSI_i}$ $\left(\overline{RSSI_i} < 0\right)$ is average signal of the terminal measured by i-th AP. We use weighted average method with the recent 3 samples values to get $\overline{RSSI_i}$ at time t:

$$\overline{RSSI_i} = 0.6 \cdot RSSI_i^t + 0.3 \cdot RSSI_i^{t-1} + 0.1 \cdot RSSI_i^{t-2} \tag{5}$$

Note that the sampling interval is 0.5 s and the weights are determined experimentally, which are likely more efficient than those derived from complex calculations. The purpose of using this model is to alleviate the jitters of *RSSI* caused by interference, which may result in superfluous handoffs. Finally, we introduce a weighting model to rank N APs:

$$\boldsymbol{INDEX} = [idx_1, idx_2, \ldots, idx_N] = \alpha \cdot \boldsymbol{CIC} + \beta \cdot \boldsymbol{R} \cdot \boldsymbol{X}$$

Here α and β are weighting factors; we let $\alpha = 0.3$, $\beta = 0.7$. Similarly, they are determined experimentally as well, with aim to avoid wrong choices in extreme situations, e.g., AP's $\overline{RSSI}$ is good but actually its channel is very busy. The $\boldsymbol{INDEX}$ is the rank of N APs, idx_i is evaluation index of the i-th AP, $\boldsymbol{X}$ is an $N \times N$ matrix to normalized $\boldsymbol{R}$:

$$\boldsymbol{X} = \begin{bmatrix} \mu_1 & 0 & \cdots & 0 \\ 0 & \mu_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mu_N \end{bmatrix} \tag{6}$$

And:

$$\mu_i = \begin{cases} 0, & \overline{RSSI_i} \le \theta < 0 \\ \dfrac{1}{\overline{RSSI_i}} - \dfrac{1}{\theta}, & \theta < \overline{RSSI_i} < 0, \end{cases} \qquad i = 1, 2, 3, \ldots, N \tag{7}$$

where θ is the threshold value (-70 dBm) mentioned before. And finally, we tend to select the m-th AP whose idx_m satisfies the following equation:

$$idx_m = \max\{idx_i | i = 1, 2, 3, \ldots, N\} \tag{8}$$

In other words, the AP we chose has more idle channel and better $\overline{RSSI}$.

4.2. **Handoff execution.** To speed up wireless link handoff, we introduce a unicast *action* frame with channel switch announcement (*CSA*) element defined in IEEE 802.11h to steer wireless terminal handoff with high performance. In *CSA* depicted in Figure 6, *channel switch mode* represents whether packet sending is interrupted during the handoff. *New channel* represents the channel number terminal should switch to. *Channel switch count* means the number of beacon frames that the terminal should wait for before handoff. There is a premise that all APs' BSSIDs should be the same. In Figure 7, when controller makes handoff decision for a terminal (step 1), it will send terminal's *connection and security context* (Section 3.3.2) to the new AP through control link (step 2), and then send a unicast *action* frame with *CSA* to the terminal (step 3). The terminal receiving *CSA* will switch to the channel of the new AP (step 4) and be able to communicate with new AP directly (step 5) because the BSSIDs of old and new AP are the same. From the view of terminal, it does not think it makes handoff but just only switches to another channel. This will not incur the handshakes existing in traditional 802.11 handoff, which quite wastes time.

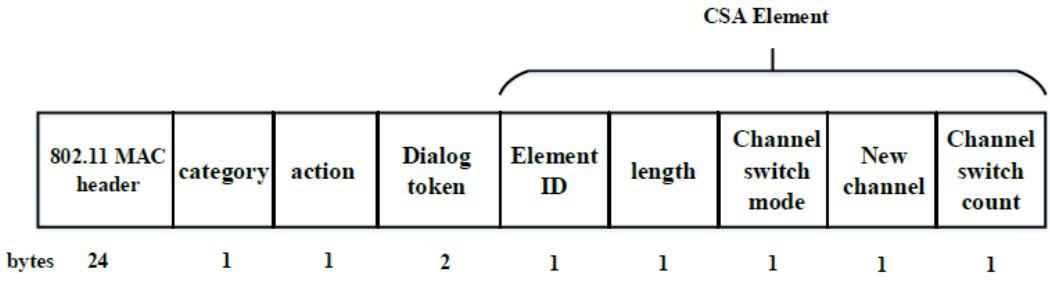

FIGURE 6. *Action* frame with *CAS*

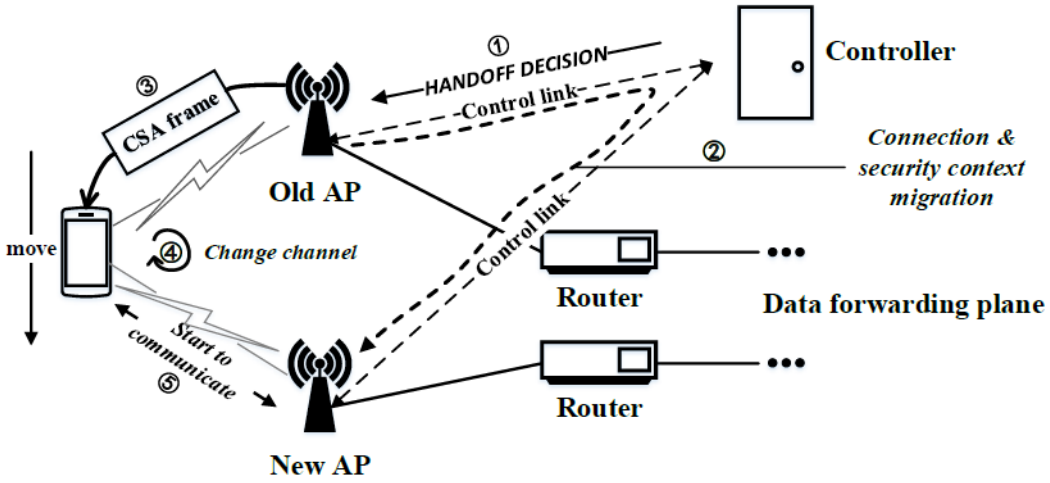

FIGURE 7. Handoff execution

5. **Performance Evaluation.** In this section, we implement a prototype of proposed system and carry out several experiments. The topology and some details of experimental environment will be described. Performance of mobility handoff is measured quantitatively when terminals keep moving, including handoff delay, jitters of delay, and jitters of bandwidth (TCP/UDP). Finally, some comparative data are given to illustrate the advantages of our system.

TABLE 2. Hardware and software configuration of prototype

AP	Raspberry 3B with Raspbian *2018-06-27-raspbian-stretch-lite* (Linux 4.14.50-v7+)
Master wireless card	Atheros AR9271 (ath9k_htc driver)
Auxiliary wireless card	MediaTek RT3070
Openvswitch	Version 2.11.90
Controller	Based on *ONOS* 1.14
Wireless terminals	Raspberry 3B with brcmfmac wireless card

5.1. **Prototype.** We implement a prototype system, which includes three APs, and a controller. The hardware and software configurations are shown in Table 2.

AR9271 card is a single-band one working on 2.4 GHz with MCS 1-7 (described in 802.11n D5.0 20.3.5/20.6, means just one spatial stream) and 20/40 MHz bandwidth. Its maximum transmission rate in theory is 150 Mbps. On AP, *openvswitch* version is 2.11.90 which can support *OpenFlow* 1.3. For the controller, we use the *ONOS* (version 1.14) SDN controller running on an Ubuntu 64 server to manage APs and terminals and update some data forwarding paths necessarily (Ethernet).

5.2. **Experiment topology.** The topology consists of three AP daemons, an *ONOS*-based controller running on an Ubuntu server and a server for communication with terminals. The terminal we use in the experiment is a Raspberry 3B with brcmfmac wireless card. As depicted in Figure 8, we adopt the *bound-in* networking way in order to deploy conveniently. And APs connect to the controller through *OpenFlow* switch, which has better path convergence.

In experiment, we not only make wireless handoff but switch wired path under the control of the controller. We place DHCP server on the controller. And through *openvswitch*es, we redirect DHCP requests and ARP requests for their respective gateways to the controller for unified replies. We set all devices in the same network segment so that we can use destination MAC address to forward packets simply in our experiment. Suppose that the i-th AP is on the $Port_i$ of Router A (Figure 8) and the Server, whose mac is mac_s, is on the $Port_s$. Mac address of the j-th terminal on the i-th AP is mac_j^i. We set the followed flow table entry to Router A for forwarding:

$$\text{in\_port} = Port_s,\ \text{dl\_dst} = mac_j^i,\ \text{actions} = \text{output: } Port_i$$
$$\text{in\_port} = Port_i,\ \text{dl\_dst} = mac_s,\ \text{actions} = \text{output: } Port_s$$

And on i-th AP, we set:

$$\text{in\_port} = \text{eth0},\ \text{dl\_dst} = mac_j^i,\ \text{actions} = \text{output: wlan0}$$
$$\text{in\_port} = \text{wlan0},\ \text{dl\_dst} = mac_s,\ \text{actions} = \text{output: eth0}$$

Note that when the terminal switches to k-th AP, controller changes the flow table entries on Router A ($Port_i \to Port_k$) and copies above flow table entries from i-th AP to k-th AP.

5.3. **Performance measurement.** We use traditional handoff in 802.11 as the baseline, and measure handoff delay, jitters of delay, and jitters of bandwidth (TCP/UDP) when terminals making round trips in normal walking speed.

In detail, terminal uses *ping* to send an ICMP request every 10 ms to the Server in Figure 8, the handoff delay is estimated by counting the number of lost ICMP replies. And RTT of ICMP is used to measure the jitters of delay when handoff occurs. For jitters of bandwidth, we use the *iperf* to measure the change of TCP and UDP bandwidth. The downlink and uplink measurements will be given separately as follows.

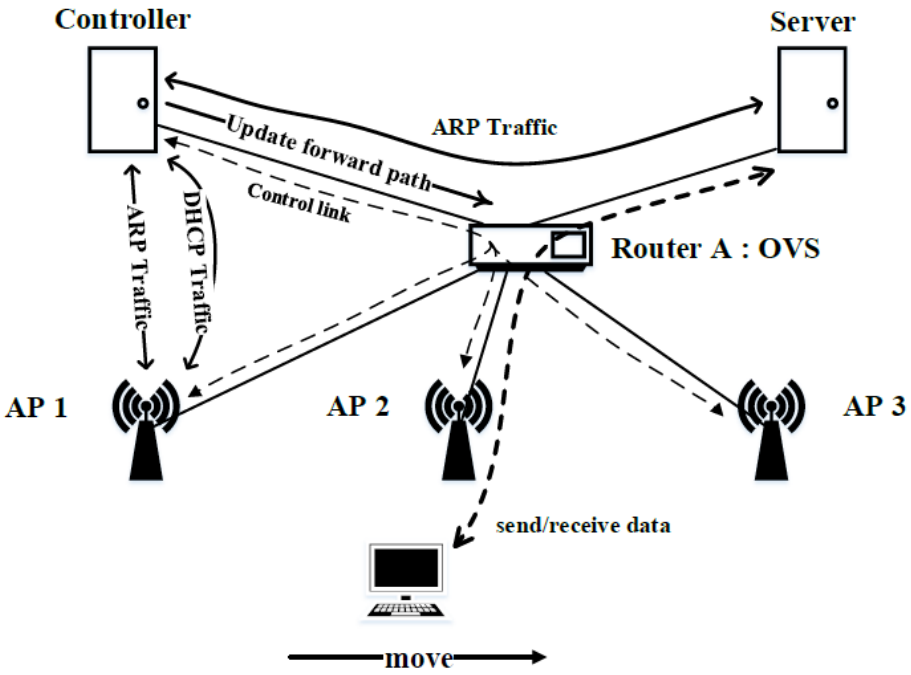

FIGURE 8. Network topology of experiment

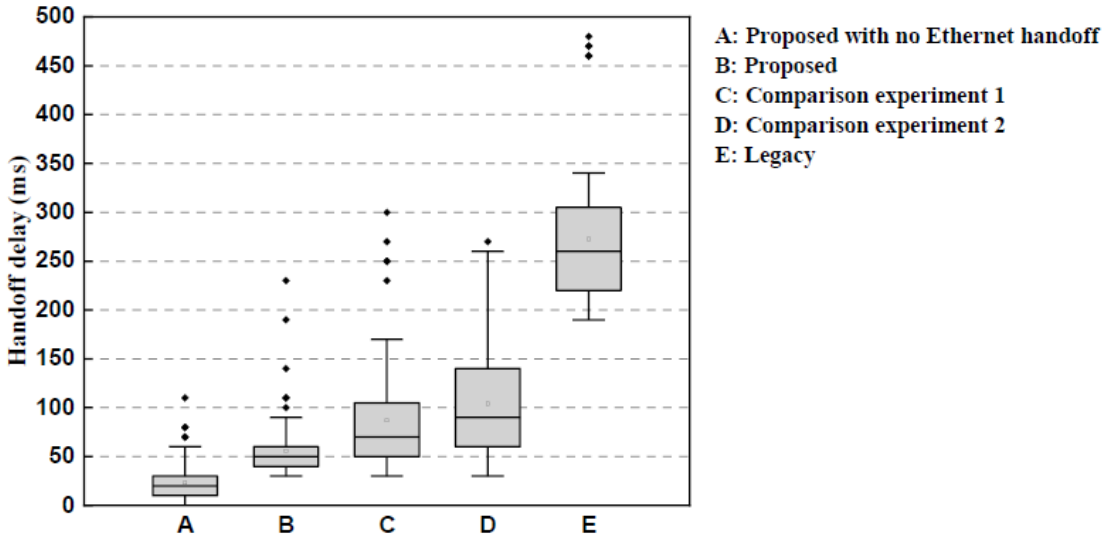

FIGURE 9. The boxplots of handoff delay

5.3.1. *Handoff delay.* In Figure 9, we draw four boxplots of handoff delay, where boxplot A represents proposed system's delay with no Ethernet handoff and B, C, D, E include both wireless and wired handoffs. C and D are the best two experiments in [20], and E represents handoff delay of traditional Wi-Fi handoff. We use ICMP packets (request/reply) to probe the connectivity of communication. The interval between two ICMP packets is set to be 10 ms, in order to get more accurate measurement results. So we get *Handoff delay* (ms) = *number of lost ICMP replies*× 10 ms. We did 100 times experiments in various situations, and got similar results no matter ICMP packet's size was 64 bytes or 800 bytes and no matter downlink or uplink. The wireless handoff delay (boxplot A) in our system is less than 50 ms. It is about 50% less than comparison experiments 1

(boxplot C) and 2 (boxplot D) and much less than traditional handoff delay (boxplot E). Boxplot B is slightly larger than boxplot A because it considers the delay of wired link handoff in our experiment. However, it still has good performance.

5.3.2. *Jitters of delay.* We tested 200 times handoff both in uplink and downlink case and all results were similar. Here we give one experiment data in two cases, respectively, as shown in Figure 10. Statistically, handoff in our proposed system does not introduce significant delay jitters before and after handoff. The terminal will quickly converge to the state of the new path.

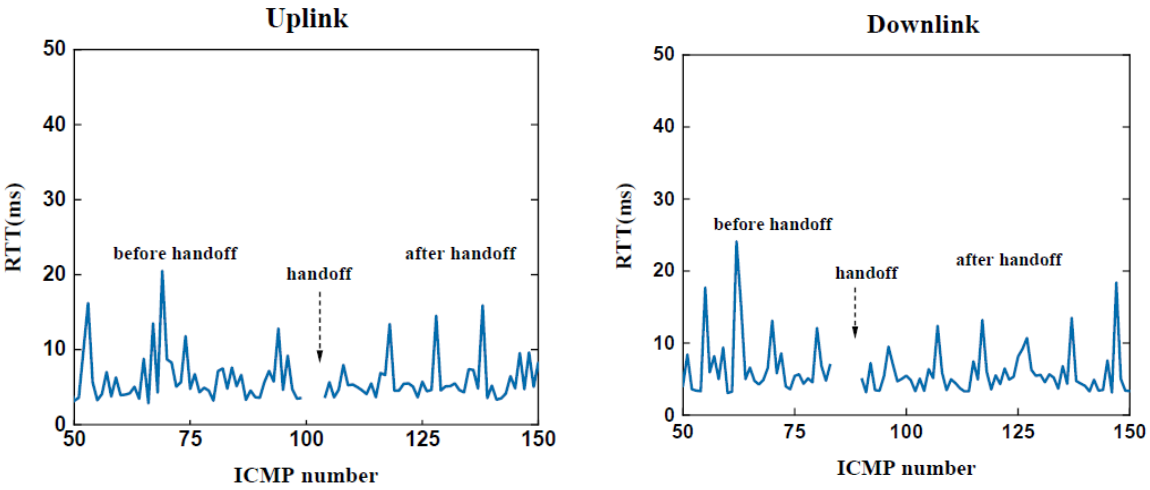

FIGURE 10. Jitters of delay (uplink and downlink)

5.3.3. *Jitters of bandwidth.* We measure the jitters of UDP/TCP bandwidth both in uplink and downlink cases, which are shown in Figure 11 and Figure 12, respectively. Compared to traditional handoff, it is easy to see that our proposed approach has smaller jitters of bandwidth when handoff occurs. On the other hand, jitters are greater in downlink case than in uplink case, and greater bandwidth is more likely to cause greater jitters.

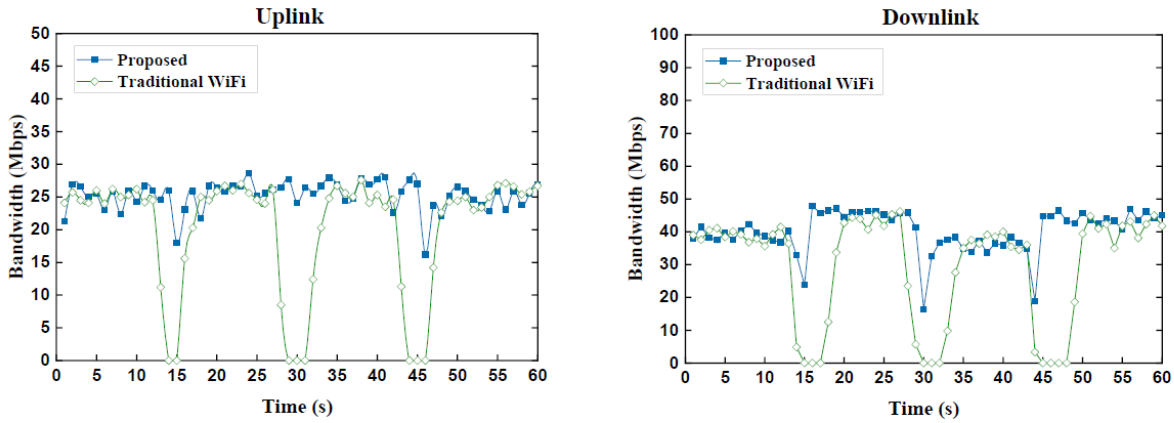

FIGURE 11. Jitters of UDP bandwidth (uplink and downlink)

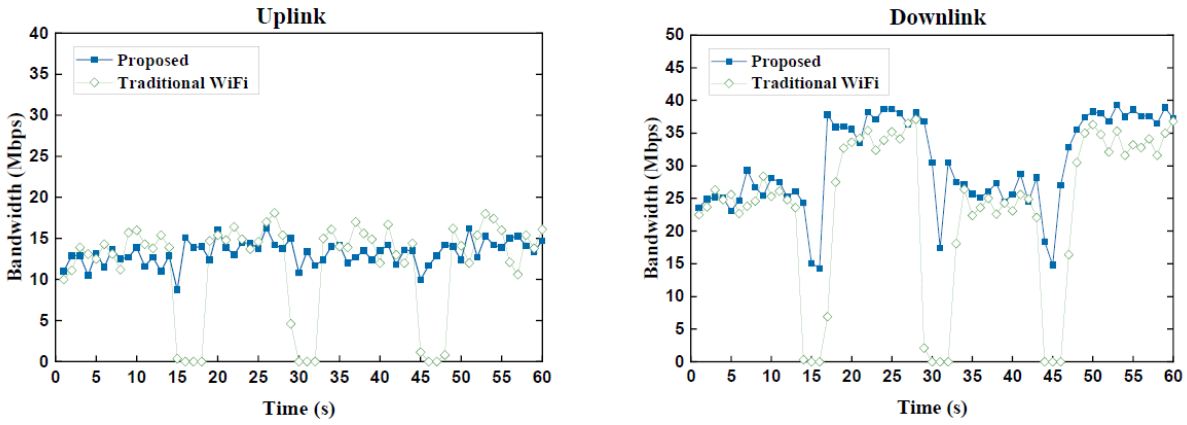

FIGURE 12. Jitters of TCP bandwidth (uplink and downlink)

5.4. **Analysis and discussion.** Here, we analyze and discuss our system prototype and experiment results.

Traditional 802.11 handoff is actually accessing system again. The reason our system greatly reduces handoff delay is that we migrate *connection and security context* and steer terminal's handoff to reduce handshake overheads. Therefore, the physical channel switch delay is the main part of wireless handoff delay. Furthermore, we test the performance of communication under this very fast wireless handoff. It means that we must consider wired link handoff. We use an *openvswitch* to simulate wired link because *openvswitch* has faster path handoff speed that matches wireless handoff.

Next, we analyze the packet loss and delay of experiments. In the case of uplink, packet loss and delay rely on implementation of terminal-side driver. Instead, our work is focused on the access side, so we mainly analyze and discuss the case of downlink here. In the case of downlink, there are two kinds of handoff, which are wireless handoff and wired handoff. Delay depends on the speed of two kinds of handoff. In this paper, we introduce *CSA* frame to reduce wireless handoff's delay and use *OpenFlow* to update wired forwarding paths under the control of controller. In this SDN-based system, though there is a global controller, these two kinds of handoff are in physical because of the distributed nature of the system. Moreover, packet propagation needs time and communication link has capability to hold packets. And it is this asynchronization and link holding packets that are the main reason of packets loss.

In detail, there are two situations that cause packet loss in the downlink case, one is that wired forwarding path changes before wireless handoff, and the other is that wireless handoff happens before wired forwarding path changing. In the former situation, packets are lost on the old path, and in the latter situation, packets are lost on the new path. Suppose that wired handoff and wireless handoff happen at T_{wired} and $T_{wireless}$, respectively. And at time T, both wireless and wired links succeed in switching. So the lost bytes by handoff are approximately:

$$Bytes_{lost} \approx \begin{cases} \displaystyle\int_{T_{wired}}^{T} B(t) \cdot dt, & \text{when } T_{wired} < T_{wireless} \\ \displaystyle\int_{T_{wireless}}^{T} B(t) \cdot dt, & \text{when } T_{wired} > T_{wireless} \end{cases}$$

Here $B(t)$ is the real-time bandwidth. Actually, $Bytes_{lost}$ is larger in deployment because wired path will consist of many hops instead of just one in our experiment. Besides

between wired handoff and wireless handoff, the problem exists between wired links hops as well. On the other hand, when the new wired path is established prior to the new wireless path, many packets will reach wireless card waiting for sending before the new wireless path establishes. This will cause congestion, because packets will go through multiple retransmission failures and be eventually discarded by wireless driver, so it will influence driver's TX bitrates selecting for this terminal. The more packets fail to send or wait for retransmission, the longer it takes to converge to the new path's bandwidth. What is worse, these unusual packet impacts can even make wireless card crash because of some drivers' protective mechanisms. Similarly, this problem can also occur with wireless driver on the old path. All of these rely on the implementation of wireless driver. So, it is important to design a cache mechanism on both new path and old path for better reliability and stability. And this will be done in the future.

6. **Conclusion.** In this paper, we proposed an SDN-based WLAN system to manage wireless resource and support terminal mobility. Our work focused on the wireless handoff and involved both control plane and data plane. Firstly, based on Linux kernel, we designed the AP daemon working with *cfg80211* subsystem and *openvswitch*. Secondly, we designed a controller based on *ONOS*. The introduction of *openvswitch* simplified the communication between APs and controller. And we used *Packet-in* and *Packet-out* messages to carry 802.11-related information between control and data plane. Thirdly, we ran an algorithm on the controller to make handoff decision. It took channel idle and *RSSI* into consideration. Moreover, we used the *CSA* frame to steer terminals to execute handoff with high performance. Finally, we implemented a prototype system with an experiment scenario, and experiment results showed that our proposed system has better performance in delay and bandwidth jitters than the traditional way.

In the future, there are several studies that need to be done. Firstly, a cache mechanism to AP should be designed to reduce packets loss and protect wireless drivers. Secondly, we will implement more useful network application on our controller. And by customizing and extending GUI functions of *ONOS*, we hope to exhibit out system through webpages. Thirdly, we will incorporate more 802.11 devices such as IEEE 802.11ac and even IEEE 802.11ax (WiFi 6) into our system and test compatibility.

Acknowledgment. This work is partially supported by Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100).

REFERENCES

[1] *LTE-Advanced Technology Introduction*, White paper, https://cdn.rohde-schwarz.com.cn/pws/dl_downloads/dl_application/application_notes/1ma232/1MA232_1e_LTE_Rel11_technology.pdf.

[2] M. A. Sayeed and R. Shree, Optimizing unmanned aerial vehicle assisted data collection in cluster based wireless sensor network, *ICIC Express Letters*, vol.13, no.5, pp.367-374, 2019.

[3] E. Perahia and R. Stacey, *Next Generation Wireless LANs*, Cambridge University Press, Cambridge, 2013.

[4] M. Bernaschi, F. Cacace, G. Iannello, M. Vellucci and L. Vollero, OpenCAPWAP: An open-source CAPWAP implementation for management and QoS support, *The 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks (IT-NEWS)*, Venezia, Italy, pp.72-77, 2008.

[5] A. Bhartia, B. Chen, D. Pallas and W. Stone, ClientMarsha: Regaining control from wireless clients for better experience, *The 25th Annual International Conference on Mobile Computing and Networking*, 2019.

[6] M. I. Sanchez and A. Boukerche, On IEEE 802.11K/R/V amendments: Do they have a real impact?, *IEEE Wireless Commun.*, vol.23, no.1, pp.48-55, 2016.

[7] D. Kreutz et al., Software-defined networking: A comprehensive survey, *Proc. of IEEE*, vol.103, no.1, pp.14-76, 2015.
[8] X. Wang and Y. Tian, Research on network virtualization for software defined networking, *Journal of Network New Media*, no.4, 2017.
[9] K.-K. Yap et al., OpenRoads: Empowering research in mobile networks, *SIGCOMM Comput. Commun. Rev.*, vol.40, no.1, p.125, 2010.
[10] P. Dely et al., CloudMAC – An OpenFlow based architecture for 802.11 MAC layer processing in the cloud, *2012 IEEE Globecom Workshops (GC Wkshps)*, Anaheim, CA, USA, pp.186-191, 2012.
[11] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann and T. Vazao, Towards programmable enterprise WLANS with Odin, *The First Workshop*, Helsinki, Finland, p.115, 2012.
[12] C. Xu et al., A novel multipath-transmission supported software defined wireless network architecture, *IEEE Access*, vol.5, pp.2111-2125, 2017.
[13] C. E. Perkins, Mobile networking through Mobile IP, *IEEE Internet Comput.*, vol.2, no.1, pp.58-69, 1998.
[14] C. E. Perkins and D. B. Johnson, Route optimization for mobile IP, *Cluster Computing*, vol.1, no.2, pp.161-176, 1998.
[15] M. Bari, S. Chowdhury, R. Ahmed, R. Boutaba and B. Mathieu, A survey of naming and routing in information-centric networks, *IEEE Commun. Mag.*, vol.50, no.12, pp.44-53, 2012.
[16] D. Raychaudhuri, K. Nagaraja and A. Venkataramani, MobilityFirst: A robust and trustworthy mobility-centric architecture for the future Internet, *SIGMOBILE Mob. Comput. Commun. Rev.*, vol.16, no.3, p.2, 2012.
[17] A. Barth, *TTP State Management Mechanism*, IETF RFC 6265, https://datatracker.ietf.org/doc/rfc6265/, 2011.
[18] J. Roskind, *QUIC – Multiplexed Stream Transport over UDP – Design Document*, 2013.
[19] S. Costanzo, L. Galluccio, G. Morabito and S. Palazzo, Software defined wireless networks: Unbridling SDNs, *2012 European Workshop on Software Defined Networking (EWSDN)*, Darmstadt, Germany, pp.1-6, 2012.
[20] J. Saldana et al., Unsticking the Wi-Fi client: Smarter decisions using a software defined wireless solution, *IEEE Access*, vol.6, pp.30917-30931, 2018.
[21] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hhn and R. Merz, Programmatic orchestration of WiFi networks, *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Philadelphia, USA, 2014.
[22] E. Kohler, R. Morris, B. Chen, J. Jannotti and M. F. Kaashoek, The click modular router, *ACM Trans. Comput. Syst.*, vol.18, no.3, pp.263-297, 2000.
[23] X. Wang, C. Xu, G. Zhao and S. Yu, Tuna: An efficient and practical scheme for wireless access point in 5G networks virtualization, *IEEE Commun. Lett.*, vol.22, no.4, pp.748-751, 2018.
[24] S. M. M. Gilani et al., Mobility management in IEEE 802.11 WLAN using SDN/NFV technologies, *J. Wireless Com. Network*, vol.2017, no.1, p.217, 2017.
[25] M. Gast, *802.11 Wireless Networks: The Definitive Guide*, 2nd Edition, O'Reilly Media, 2009.
[26] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski and T. Rasheed, Programming abstractions for software-defined wireless networks, *IEEE Trans. Netw. Serv. Manage.*, vol.12, no.2, pp.146-162, 2015.
[27] J. Schultz et al., OpenGUFI: An extensible graphical user flow interface for an SDN-enabled wireless testbed, *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*, Liverpool, United Kingdom, pp.770-776, 2015.
[28] A. K. Rangisetti, H. B. Baldaniya, P. K. B and B. R. Tamma, Load-aware hand-offs in software defined wireless LANs, *2014 IEEE the 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Larnaca, Cyprus, pp.685-690, 2014.
[29] M. E. Berezin, F. Rousseau and A. Duda, Multichannel virtual access points for seamless handoffs in IEEE 802.11 wireless networks, *2011 IEEE Vehicular Technology Conference (VTC 2011-Spring)*, Budapest, Hungary, pp.1-5, 2011.
[30] Metageek, *Understanding RSSI*, https://www.metageek.com/training/resources/understanding-rssi.html, 2018.