ABSTRACT

Programming wireless networks requires accounting for multiple complex operations, such as monitoring interference and allocating radio resources. Employing the Software-Defined Networking (SDN) paradigm eases the implementation of such tasks when augmented with suitable high-level programming abstractions. In this work, we present a set of programming abstractions modeling three fundamental aspects of a wireless networks, namely state management, resource provisioning, and network state collection. We also describe our proof-of-concept implementation of the proposed abstractions focusing on WiFi networks and show its use for realizing typical control tasks such as mobility management and traffic engineering as Network Apps.

Categories and Subject Descriptors
C.2.3 [Network Operations]: Network management

Keywords
WiFi, Programming Abstractions, Network Management

1. INTRODUCTION

Rapid adoption of smartphones and tablets is fuelling the dramatic rise in mobile data traffic, which has led mobile operators to explore coping mechanisms. Mobile data offloading via public WiFi hotspots is one such key approach. Typically, public WiFi networks are managed via a centralized proprietary controller with limited flexibility for operators to tailor the behavior of their networks. Software-Defined Networking (SDN) offers a more open framework, promising simplified network control and management when enhanced with suitable high-level abstractions while at the same time allowing operators to deploy new services and features as Network Apps.

In this work we present three programming abstractions for WiFi networks, especially keeping in mind the emerging need to flexibly manage large number of public WiFi hotspots. They complement the existing programming abstractions for SDN-enabled networks that are aimed at wired networks (e.g., [2, 4, 1]). Between them, the proposed abstractions address the three fundamental elements that compose a WiFi network control loop, namely: collecting the wireless network state; implementing control and management tasks; and disseminating the new configuration to the network elements.

We realize these abstractions via a proof-of-concept implementation that includes a Python-based Software Development Kit (SDK) and Software-Defined RAN (SD–RAN) Controller. This implementation provides a platform to realize features of WiFi-based WLANs such as mobility management and traffic engineering as Network Apps running on top of the SD–RAN Controller. Moreover, the SDN principles on which the platform is based also allow for advanced network virtualization functionality.

Figure 1 sketches the reference network architecture. One section introduces our three key abstractions for WiFi networks, namely: the Light Virtual Access Point (LVAP); the Resource Pool; and the Interference Map (see Fig. 2). A Resource Block is like in 3GPP mobile networks and represents the minimum chunk of resources that can be assigned to a client, while the LVAP represents the state of a client scheduled on a set of Resource Blocks.
Both the LVAPs and the WTPs support a set of Resource Blocks, named Resource Pool. A relationship exists between each LVAP/Resource Block pair modeling the link quality between the two entities, for example using RSSI; the Interference Map abstraction captures this relationship.

2.1 The Light Virtual Access Point

The LVAP abstraction builds on top of [3] which provides a high-level interface through which the state of a wireless client can be manipulated. The implementation of such an interface is required to handle all the technology-dependent details such as association, authentication, and resource scheduling. A wireless client attempting to join the network will trigger the creation of a new LVAP. Such LVAP is specific to the newly associated client (in a WiFi network the LVAP can be thought as a Virtual AP with its own BSSID). As a result each WTP will host as many LVAPs as the number of clients currently communicating with it. Removing an LVAP from a WTP and instantiating it on another WTP effectively results in a handover.

2.2 The Resource Pool

Programming WiFi networks mandates for a way to expose the programmer with a consistent view of the network resources. The fundamentals of wireless communications basically calls for two main families of strategies to allocate resources in a wireless network: scheduled access and random access. In the former case, re-requests coming from the clients. This allows expressing resource allocation problems as an intersection between the Resource Blocks available in the network and the Resource Blocks supported by a client. Information on the link quality experienced by the requesting LVAP can be used to further filter the set of candidate Resource Blocks, possibly based on application-driven requirements. A non-empty set of Resource Blocks so determined suggests a feasible solution for the resource allocation problem, and an optimal choice (based on performance criteria of interest) from this set can then be made. Note that it may not be meaningful to move tasks, such as resource scheduling and rate adaptation, operating at fast timescales to the remote SD–RAN Controller; such operations would be physically implemented in a distributed manner within the WTPs.

2.3 The Interference Map

As wireless channel and interference characteristics vary over time and space, they need to be taken into account for effective resource allocation. Similarly, application layer requirements also need to be considered. For example, it may make sense for a hotspot operator to optimize system throughput or network utilization whereas in other instances it may be important to ensure allocation of minimum amount of resource for each client. The Interference Map abstraction provides network programmers with a full view of the network state in terms of interference and channel quality between Resource Blocks and LVAPs. The latter essentially consists of a matrix $R$ (typically sparse) where each entry $R(m, l)$ is the channel quality between the Resource Block $m$ and the LVAP $l$. In WiFi networks Interference Map related information can be obtained via passive measurements on the received traffic.

3. SOFTWARE-DEVELOPMENT KIT

In this section we briefly describe the salient features of our Python-based SDK\(^2\) using some practical examples. The platform supports multiple logical virtual networks each characterized by its own SSID and a set of WTPs. Network App can be instantiated within one or multiple virtual networks.

**Resource Management.** The following Python function assigns an LVAP to a random Resource Block whose RSSI to the LVAP is greater than $-65$ dB:

```
def handover(lvap, wtps):
    # Initialize the Resource Pool
    pool = ResourcePool()

    # Add all available Resource Blocks
    for wt in wtps.values() :
        pool = wt.supports | pool

    # Select matching Resource Blocks
    matches = pool & lvap.supports

    # Filter matching Resource Blocks by RSSI
    valid = {block for block in matches if block.rssi_to[lvap] >= -65}

    # Perform handover
    lvap.assigned_to = valid.pop()  
```

The method above accepts two parameters as input: an LVAP and a list of WTPs. The method initializes the network Resource Pool with the Resource Blocks available at every WTP. Then, an intersection between the network Resource Pool and the LVAP Resource Pool is computed. The resulting set is then traversed in order

\(^1\)Notice that, for spread spectrum-based technologies, such as UMTS, also the code space shall be considered.

\(^2\)Available at http://empower.create-net.org
to filter–out the Resource Block whose RSSI to the LVAP is below a certain threshold (−65dB in this case). Finally, one random Resource Block matching the above condition is assigned to the LVAP.

For the sake of simplicity, error handling code has been omitted.

**Querying** Packets counters allow programmers to track the traffic exchanged by a certain LVAP. For example:

```python
```

The statement above tracks the packets transmitted and received by a certain LVAP and aggregates the information into the specified bins. The counters’ current state can be accessed with:

```python
>>> s.tx_samples
[60, 10, 0]
```

Meaning that the LVAP transmitted 60, 10, and 0 packets smaller or equal to respectively 500, 1460, and 8192 bytes. Similarly, bytes_count allows to track the bytes transmitted and received by the LVAP.

**Interference Tracking** The Interference Map allows the developers to meet LVAPs’ traffic requirements. Moreover, Network App can react to changes in the network conditions by setting callbacks that are triggered when the measured RSSI for a given set of LVAPs verifies a specified condition. For example:

```python
t.callback = handle_rssi_trigger
```

The callback is triggered the first time the RSSI of the LVAP goes above −50 dB at any WTP in network. After the trigger has fired the first time and as long as the RSSI remains greater than −50 dB, the callback method is not called again by the same WTP, however the same callback can be triggered by multiple WTPs. In order to detect RSSI values that are going below the −50 dB threshold another trigger must be created.

4. **USE CASE: MOBILITY MANAGEMENT**

In this section we sketch basic Mobility Manager as a sample Network App that makes use of the proposed abstractions. The Mobility Manager leverages RSSI tracking to detect when a wireless client’s link quality is deteriorating. Moreover, the Mobility Manager periodically checks if a better handover opportunity exists. The Mobility Manager code is reported below.

```python
class MobilityMngr(BPW):
    # Initialize the object
    def __init__(self, pool, period=5):
        BPW.__init__(self, pool, period)
        r = self.rssi(relation='LT', value=-50, lvap='FF:FF:FF:FF:FF:FF')
        r.callback = self.callback

    # Check if a better handover plan can be found
    def loop(self):
        for lvap in self.lvaps().values():
            handover(lvap, self.wtps())

    # Callback on low RSSI
    def callback(self, lvap, wtp):
        handover(lvap, self.wtps())

    # Initialize the App
    def launch(pool, period=5):
        return MobilityMngr(pool, int(period))
```

A Network App in our implementation is essentially a Python module that can be loaded/unloaded at run-time. Each module is required to implement a launch method called when the module is initialized. The above Mobility Manager implementation consists of a single class implementing three methods: (i) the constructor which creates an RSSI trigger matching all LVAPs in the network; (ii) loop, which periodically checks if an LVAP shall be handed–over to another WTP; and (iii) callback which is invoked when the RSSI is going below a certain threshold in which case the handover routine is invoked.

5. **CONCLUSIONS**

In this work we have proposed a set of high–level programming abstractions for WiFi networks. The proposed primitives allow new features and services to be implemented as software modules hiding away the implementation details of the underlying technology. We also developed a proof-of-concept implementation including an SDK and a SD–RAN Controller. As future work we plan extend the proposed abstractions to mobile (LTE/LTE–Advanced) networks.

6. **REFERENCES**


