Interference Management in Software–Defined Mobile Networks

Roberto Riggio*, Mahesh K. Marina†, Tinku Rasheed*
*CREATE-NET, Trento, Italy; Email: rriggio@create-net.org, trasheed@create-net.org
†The University of Edinburgh, Edinburgh, UK; Email: mahesh@ed.ac.uk

***Abstract*—Software–Defined Networking promises to deliver more flexible and manageable networks by providing a clear decoupling between control plane and data plane and by implementing the latter in a logically centralized controller. However, if such principles are to be applied also to wireless networks, new primitives and abstractions capable of providing programmers with a global view of the network capturing channel quality and interference must be devised. Moreover, the dynamic radio environment necessitates fast adaptation of physical parameters such as power, modulation and coding schemes. So the wireless SDN abstractions should allow for such adaptations to happen closer to the air interface. In this paper, we present high level abstractions for channel quality, interference and network reconfiguration; the latter permits operations differing in timescales to be carried out at different controller entities. The proposed concepts have been implemented and evaluated over a WiFi–based WLAN. Empirical measurements show that the proposed platform can be used to implement typical WiFi network management tasks such as channel assignment and interference monitoring.**

I. Introduction

Mobile networks are currently faced with a steep increase in data traffic generated by modern mobile applications. Operators are coping with this trend by deploying denser and heterogeneous radio access network (RAN) and by utilizing WiFi as a traffic offloading technology. Until a few years ago WiFi deployments used to be unplanned, however lower equipments costs and ease of setup has led to a vast number of uncoordinated and mutually interfering deployments. As a result, novel WLAN network planning tools have emerged while fully distributed WLANs are being replaced, especially in large enterprises, with centralized setups where a network controller is in charge of managing the network. Recently, another trend has emerged in the network management scenario calling for more flexibility in the way networks are managed. Such trend, named Software–Defined Networking (SDN) aims at clearly separating policies from mechanisms and at putting the former in the hands of network developers through a set of high–level and possibly open APIs.

However, despite several examples of SDN concepts applied to wireless networks, the API support for monitoring and controlling interference is still very limited. Nevertheless, such knowledge is extremely valuable for implementing network management solutions capable of adapting in response to time-varying interference conditions. Moreover, there is the need for a high–level network reconfiguration model that clearly distinguishes the latency–bound control policies executed at the edges of the network from monitoring and reconfiguration tasks implemented at the (possibly) centralized controller.

This work was partially supported by European Institute of Technology (EIT) ICT Labs under the activity SDN@Edges (15270/A1502).

In our previous work [1] we presented a set of SDN abstractions specifically tailored for the WiFi networking domain. These abstractions have been implemented in a prototype *SD–RAN Controller* and are exposed trough a Python–based SDK. However, that work did not investigate how network interference shall be abstracted and presented to the network programmer, and also it did not consider how network programmers can reconfigure or replace autonomic network control policies. In this work we take a first step in that direction by: (i) extending the APIs with new interference modeling primitives; (ii) proposing a network reconfiguration model clearly separating network control from network management; and (iii) testing the new primitives over a small scale testbed.

The next section briefly summarizes our original work [1] and introduces the new abstractions. Section III provides the *SD–RAN Controller* implementation details together with an overview of the new SDK. Section IV reports on the evaluation campaign. Finally, we discuss the related work in Sec. V and then we draw our conclusions in Sec. VI.

II. Revisiting Network Control and Management

In this work then we draw a clear line between network control and network management. The former (control) deals with fast timescale operations executed by the elements at the edges of the network, such as scheduling in LTE networks or transmission rate selection in WiFi networks. The latter (management) is in charge of checking whether the operating conditions for a certain policy are still met, and, if this is not the case, of reconfiguring or replacing the policies.

Figure 1 sketches the reference network architecture and introduces the terminology used throughout the paper. We name Wireless Termination Points (*WTP*s), the physical devices that form the RAN providing clients with wireless connectivity. *WTP*s basically coincide with Access Points (APs) in a WiFi network or eNodeBs (eNBs) in a LTE network. The *WTP* are connected to the *SD–RAN Controller* trought a secure channel. *Network App* run in their own slice of resources on top of the *SD–RAN Controller*. The RAN exploits a (possibly) programmable backhaul in order to reach the public Internet. Finally, although OpenFlow is a candidate backhaul technology, the abstractions proposed in our work do not rely on it and are effectively backhaul agnostic.

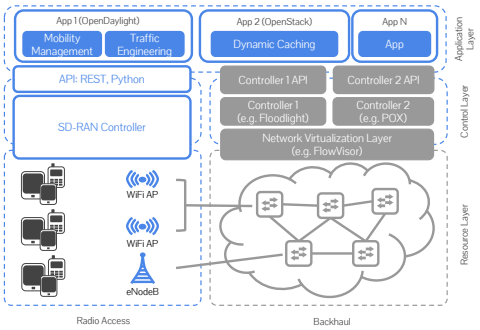


Fig. 1: The reference network architecture.

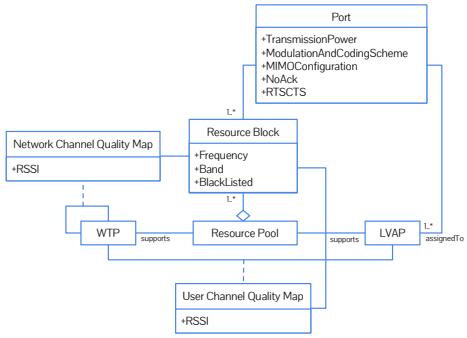


Fig. 2: The abstractions class–diagram.

In this section we briefly summarize our previous work [1] (II-A and II-B) before describing the new abstractions proposed in this paper (II-C and II-D). Figure 2 depicts the relationship between the four abstractions (*LVAP*, *Resource Pool*, *Channel Quality Map*, and *Port*) using an UML class–diagram. Here, a *Resource Block* represents the minimum chunk of wireless resources that can be assigned to a wireless client while the *LVAP* represents the state of a UE scheduled on a set of *Resource Block*s. Both *LVAP*s and *WTP*s support a set of *Resource Block*s, named *Resource Pool*. The *Port* abstraction models the dynamic and reconfigurable characteristics of the link between *WTP* and *LVAP* on a set of *Resource Block*s. A relationship exists between *LVAP*s and *WTP*s and between *WTP*s modeling the link quality between the two entities. This latter relationship constitute the *Channel Quality Map*.

A. Light Virtual Access Point

The *LVAP* abstraction [1], [2], [3] provides a high–level interface for wireless clients state management. The implementation of such an interface handles all the technology–dependent details such as association, authentication, handover, and resource scheduling. A client attempting to join the network, will trigger the creation of a new *LVAP*. Conversely each *WTP* will host as many *LVAP*s as the number of wireless clients that are currently under its control. Such *LVAP* has an ID that is specific to the newly associated UE (in a WiFi network the *LVAP* can be thought as a Virtual AP with its own BSSID). Removing an *LVAP* from a *WTP* and instantiating it on another *WTP* effectively results in a handover.

B. Resource Pool

The *Resource Pool* abstraction is tightly coupled with the *LVAP* abstraction and goes into the direction of abandoning the concept of cell and exposing the network programmer with the collective resources in time, frequency, and space that are available in the network. The minimum allocation unit in the *Resource Pool* is the *Resource Block* and is identified by a frequency band, a time interval, and the *WTP* at which it is available. The *Resource Pool* is exposed to the programmer through a set P where each *Resource Block* $i \in P$ is a 2–tuple $\langle f, t \rangle$, where f is the frequency band, and t is the time slot. A frequency band is a 2–tuple $\langle c, b \rangle$ where c and b are, respectively, the center frequency and the bandwidth. For example, the *Resource Pool* made available by a legacy 802.11g AP tuned on channel 1 would be represented by the tuple $((1, 20), \infty)$. Here 1 is the channel, and 20 is the bandwidth (in MHz). Notice that no time dimension is provided or, more precisely, the *Resource Block*s are allocated for all the time modeling WiFi random channel access scheme. Finally, *Resource Block*s can also be *blacklisted* preventing applications from using them.

C. Channel Quality and Interference Maps

The *Channel Quality Map* abstraction provides network programmers with a full view of the network state in terms of channel quality between *LVAP*s and *WTP*s over the available *Resource Block*s. Let $G = (V, E)$ be an directed graph, where $V = V_{WTP} \cup V_{LVAP}$ is the set of $v_1 = |V_{WTP}|$ *WTP*s and $v_2 = |V_{LVAP}|$ *LVAP*s in the network, and E is the set of edges or links. An edge $e_{n,m,i} \in E$ with $n, m \in V$ exists if m is within communication range of n over the *Resource Block* $i \in P$. A weight $q(e_{n,m,i})$ is assigned to each link $e_{n,m,i} \in E : q(e_{n,m,i}) \in \mathbb{N}^+$ represents the channel quality of the link between the two nodes.

A link interference (conflict) graph or *Interference Map* $G^I = (V^I, E^I)$ can be also constructed in such a way that we have a vertex $v^I \in V^I$ for each communication link in E. A directed edge $e^I_{n,m,i} \in E^I$ if the transmitter of the link $n^I \in V^I$ is within the interference range of the receiver of the link $m^I \in V^I$ over the *Resource Block* $i \in P$. A weight $q^I(e^I_{n,m,i})$ is assigned to each link $e^I_{n,m,i} \in E^I : q^I(e^I_{n,m,i}) \in \mathbb{N}^+$ represents the interference level between the two links.

The *Channel Quality Map* is exposed to the network programmer by means of two data structures: the *User Channel Quality Map* (UCQM) and the *Network Channel Quality Map* (NCQM). Both are 3-dimensional matrices where each entry is the channel quality (in dBm) over one *Resource Block* between: an *LVAP* and a *WTP* in the case of the UCQM; and between two *WTP*s in the case of the NCQM.

These channel quality and interference map abstractions can be used to select the *Resource Block*s that can satisfy the QoS requirements of an *LVAP* by intersecting the set of available *Resource Block*s in the network (P_N) and the requested *Resource Block*s (P_L). The set of available *Resource Block*s is obtained as union of the *Resource Block*s supported by all the *WTP*s: $P_N = W_1 \cup W_2 \cup \cdots \cup W_N$. The matching *Resource Block*s M are then given by: $M = P_N \cap P_L$. The list

of *Resource Block*s M' that satisfy a certain interference level condition, such as the signal to interference plus noise ratio (SINR) between the *LVAP* n and the *WTP* m on the *Resource Block* i being greater than a certain threshold t, is given by: $M' = \{i \in M : SINR(e_{n,m,i}) > t\}$ where $SINR(e_{n,m,i})$ can be estimated via edge weights in *Channel Quality Map* and in the *Interference Map*. M' is the empty set if a valid resource allocation is not found. Allocating the valid *Resource Block*s is simply a matter of assigning one or more *Resource Block*s from M' to the *LVAP*, which may result in an *LVAP* handover if the new *Resource Block*(s) are handled by a different WTP.

D. Port

Links in a wired network, e.g. a switched Ethernet LAN, are essentially deterministic and the status of a port in a switch is binary, i.e. active or not active. While some Ethernet switches can select the transmission rate (10, 100, 1000 Mb/s), this feature is aimed at reducing power consumption when the traffic load is low and not a mechanism for coping with fluctuations in the channel quality. In contrast, links in a wireless network are stochastic and, as a result, the physical layer parameters that characterize the radio link between an *LVAP* and a *WTP*, such as transmission power, modulation and coding schemes, and MIMO configuration must be adapted according to the actual channel conditions.

Such level of adaptation requires real–time coordination between *LVAP*s and *WTP*s and can only be implemented near the air interface. The *Port* abstraction allows the *SD–RAN Controller* to reconfigure or replace a certain control policy if its optimal operating conditions are not met. A port is defined by a 3–tuple $\langle p, m, a \rangle$ where p is the transmission power, m is the set of available Modulation and Coding Schemes (MCS), and a is the MIMO configuration (number of spatial streams). For example, in the case of an 802.11n network, assigning the port configuration: $l \xleftarrow{\langle 30,(0:7),1 \rangle} M'$ to an *LVAP* means that the *WTP* will use a fixed transmission power of 30 dBm, the set of MCS between 0 and 7, and single antenna configuration for its communication toward the *LVAP*. This port abstraction thus allows fast timescale adaptations (MCS adaptation in this example) to be delegated to a local controller located near (from the latency standpoint) the *WTP* [4], [5] or to the *WTP* itself. In this work we assume the latter. Finally, since a *Port* specifies the configuration of the link between a *WTP* and an *LVAP* over a certain *Resource Block*, a *WTP* will have as many *Port* configurations as the number of *LVAP*s it is currently managing.

III. Implementation Details

This section briefly summarize the *SD–RAN Controller* main features [1]. More information on the software/hardware platform, named *EmPOWER* can be found in online[1]. Notice how, since in its current implementation the *SD–RAN Controller* supports only WiFi–based *WTP*s, *Resource Block*s are identified by the 2–tuple $\langle c, b \rangle$ (i.e.: no temporal dimension).

The *SD–RAN Controller* is built using the Tornado Web Server framework [6]. The main reason for choosing Tornado is its non–blocking network I/O which allows to continue serving incoming requests while the others are being processed. The *SD–RAN Controller* can run multiple virtual networks, or slices, on top of the same physical infrastructure. A network slice is a virtual network with a specific SSID and its own set of *WTP*s. Clients can *opt–in* a certain slice by associating to its SSID. *Network App*s run on top of the *SD–RAN Controller* in their own slice of resources and exploit the programming primitives trough either a RESTful interface or a native Python API (bindings for other programming languages can be easily added). The *SD–RAN Controller* ensures that an *Network App* is only presented a view of the network corresponding to its slice. Notice that in this architecture the term *Network App* is used to address any consumer of the *SD–RAN Controller* API such as OpenDaylight and Openstack.

*WTP*s are built around the PCEngines ALIX platform and run the latest version of the OpenWRT operating system (Chaos Calmer r42609). Each *WTP* runs an instance of the Click modular router [7] implementing the WiFi datapath. Communications between Click and the *SD–RAN Controller* takes place over a persistent TCP connection. *WTP*s are equipped with two WiFi interfaces both leveraging a patched *ath9k* driver for their operations. Such modifications include the *LVAP* logic and the per–packet configuration of parameters such as transmission rate and power delegated via the *Port* abstraction.

A. Port

The *LVAP* and the *Port* abstractions are exposed to the programmers through a Python dictionary[2] mapping *Resource Block*s to *Port*s. The programmers can fetch the *Resource Block*(s) on which an *LVAP* is currently scheduled together with its *Port* configuration by accessing the *assigned_to* property of an *LVAP* object. For example:

```
>>>lvap.assigned_to
{(04:F0:21:09:F9:96, 36, L20):
 (0C:3E:9F:57:1A:B6, <6,12,24,36,48,54>, 27 dBm, 1)}
```

As it can be seen, the dictionary above contains a single entry mapping a *Resource Block* with a *Port* configuration. In this example, the *LVAP 0C:3E:9F:57:1A:B6* has been assigned to the *Resource Block* $\langle 36, L20 \rangle$ scheduled at the *WTP 04:F0:21:09:F9:96*. The *Port* configuration specifies which range of parameters the *WTP* can use for its communication with the *LVAP*, in this case: single spatial stream, fixed transmission power, and adaptive transmission rates selection (with a constraint on the possible MCS). The current implementation of the *Port* abstraction supports the following parameters:

- *TX Power*. Fixed transmission power (in dB).
- *Modulation and Coding Scheme (MCS)*. List of MCS values that can be used by the rate selection algorithm.
- *MIMO Configuration*. Number of spatial streams.
- *RTS/CTS Threshold*. Frame length value above which the RTS/CTS handshake must be used.
- *No ACK*. The *WTP* will not wait for ACKs.

[1] Available at: http://empower.create-net.org

[2] Python dictionaries are associative arrays mapping key to values.

To/From DS	*Type*	*Sub–type*	*STA/AP*
0 / 0	Data	Any	STA
0 / 0	Mngt	Beacon, Probe Response	AP
0 / 0	Mngt	Probe Request, Disassociation (Re) Authentication Request (Re) Association Request Authentication, De–Authentication	STA
1 / 0	Any	Any	STA
0 / 1	Any	Any	AP
1 / 1	Any	Any	AP

TABLE I: Station/Access Point classification criteria.

Notice that, although semantically the model allows manipulating both the downlink and the uplink connections, in the current implementation only the downlink, i.e. from the *WTP* to the client, parameters can be configured.

B. Channel Quality and Interference Maps

In our implementation of the *Channel Quality Map* we use the RSSI measured at each *WTP* as an approximation of the channel quality. A monitor interface is created on top of each physical radio available at each *WTP*. The monitor interface extracts the signal strength field present in the radiotap header for every decoded WiFi frame. In order to separately build the UCQM and the NCQM, the *To–DS* and the *From–DS* bits present in the 802.11 header are used. The frame type and sub–type are also used for determining the transmitter type. Table I summarizes the classification criteria.

The sniffer computes the average of the received signal over windows of 500ms, moreover, an exponential weighted moving average (EWMA) and a smoothing moving average (SMA) are also maintained for each neighbor. Both filters have been selected in that they can reduce noise while keeping the sharpest step response, i.e. they are fast to react to changes in the input signal. Such property is particularly useful when dealing with RSSI signals in that they are severely affected by white noise due to fast–fading. At the same time a fast response is required in order to react promptly to changes in the interference conditions.

The *Channel Quality Map* can be leveraged by the programmers using either a reactive or a proactive programming model. The RSSI triggers allow programmers to generate a callback when a certain condition is verified at any *WTP* in the network. Consider for example the following statement:

```
rssi(lvaps='11:22:33:44:55:66',
     relation='LT',
     value=-70,
     ssid='Guests',
     callback=rssi_callback)
```

Listing 1: Create an RSSI trigger.

The statement above generates a callback the first time the RSSI of the specified *LVAP* goes below -70 dB at any *WTP* in network. After the trigger has fired the first time and as long as the RSSI remains below -70 dB, the callback method is not called again by the same *WTP*, however the same callback may be triggered by other *WTP*s. In order to detect RSSIs that are going above -70 dBm another trigger must be created. Specifying `FF:FF:FF:FF:FF:FF` as *LVAP* will trigger the callback when the RSSI of any *LVAP* at any *WTP* is below -70 dBm.

Moreover, the *Channel Quality Map* allows developers to track the RSSI levels of any WiFi device within decoding range of a *WTP*. For example, the code below periodically queries the specified *WTP* for its neighboring stations.

```
ucqm(addrs='ff:ff:ff:ff:ff:ff',
     block=('04:F0:21:09:F9:96', 36, L20)
     every=5000,
     ssid='Guests',
     callback=ucqm_callback)
```

Listing 2: UCQM query creation.

The query is executed periodically with the period set by the `every` parameter (in ms)[3]. Similarly, the RSSI from neighboring WiFi Access Points can be tracked using the *ncqm* primitive. In the above example, as before, specifying `FF:FF:FF:FF:FF:FF` will return the RSSI of any station within decoding range of *WTP 04:F0:21:09:F9:96* on the legacy channel 36 (i.e., an 802.11g channel). It is worth noticing that, we are using the general term *stations* and *access points* instead of, respectively, *LVAP*s and *WTP*s, in that the *Channel Quality Map* tracks the RSSI level of any active WiFi device including the ones belonging to networks that are not under the administrative domain of our *SD–RAN Controller*. This includes wireless clients that are not associated to *any* network but have their wireless interface active. This is due to the fact that such clients periodically broadcast *Probe Requests* messages in order to discover available APs. As all queries are non–blocking and it is possible to specify an optional *callback* method to be executed when the query response is available at the controller.

IV. Evaluation

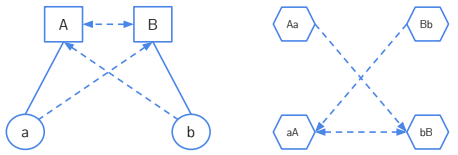
In this section, we present three case studies of our proposed WiFi SDN abstractions. In the first case study, we show how *Interference Map* and *Channel Quality Map* abstractions can be used to perform interference–aware channel assignment and thereby balance load across cells. Second case study demonstrates how time-varying channel quality can be tracked via the *Channel Quality Map* abstraction. Finally, we show how to leverage the *Channel Quality Map* to implement a simple proximity detection *Network App*.

The system has been evaluated over a simple testbed composed of three *WTP*s and two clients (Dell D630 notebooks). Each *WTP* is equipped with two Wirelss NIC tuned on different channels, namely 6 (2.4 GHz band), and 36 (5 GHz band). Notice that channel 36 is not shared with any other network, while channel 6 is used by several other access points (test are carried out in a typical office environment). Iperf [8] is used in order to generate synthetic traffic.

A. Interference–Aware Channel Assignment

An efficient channel assignment can dramatically improve network performance in dense WLANs. In this section we

[3]Specifying $every = -1$ will result in a single query being issued



(a) *Channel Quality Map*. (b) *Interference Map*.

Fig. 3: Interference–Aware Channel Assignment. *WTP*s are represented as squares while *LVAP*s are represented as circles.

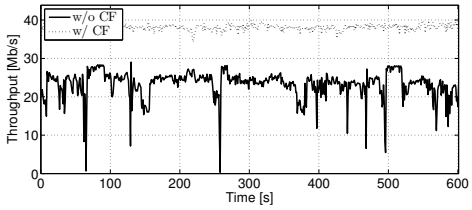


Fig. 4: Instantaneous aggregated client throughput before $(w/o\ CF)$ and after $(w/\ CF)$ channel assignment.

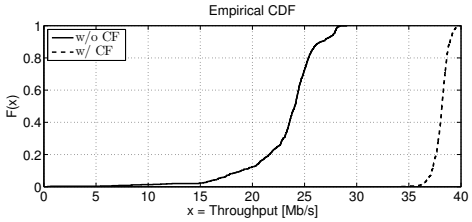


Fig. 5: Distribution of the throughput before $(w/o\ CF)$ and after $(w/\ CF)$ channel assignment. re–assignment.

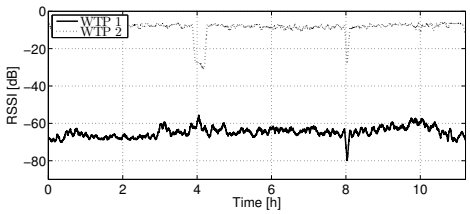


Fig. 6: Wireless client RSSI at two different *WTP*s over a 10–hours lonk measurement campaign.

shall demonstrate how the *Channel Quality Map* and *Interference Map* abstractions can be leveraged for this purpose. Note that the definition of *Interference Map* from section II-C can be extended to result in a more realistic interference map including *external* but nearby WiFi APs and stations.

Figure 3a sketches the *Channel Quality Map* for the setup used in this experiment whereas a graphical representation of the *Interference Map* is shown in Fig. 3b. The solid lines in Fig. 3a represent network connectivity while the dashed lines are the weighted edges in the UCQM for a given *Resource Block*. As it can be seen given the fact that all the nodes are within decoding range the resulting conflict graph is almost fully connected. No edge exists between the nodes Aa and Bb in Fig. 3b because due to limitations in the current implementation it is not possible to gather the list of interfering devices *at* the client.

The knowledge contained in the conflict graph can be effectively leveraged to improve network performance by implementing suitable load–balancing and channel assignment algorithms. A simple implementation of the DSATUR [9] algorithm has been implemented as proof–of–concept (due to space constraints the code is not reported). The system performance have been tested before and after the channel assignment. Traffic consists of two saturated TCP connections generated at the two clients toward a node which shares the backhaul with the two *WTP*s. Figure 4 reports the instantaneous aggregated client throughput before $(w/o\ CF)$ and after $(w/\ CF)$ channel assignment. As expected given the very simple topology, when channel assignment is performed the aggregated throughout improves significantly as well as its stability. The latter effect can be better seen in Fig. 5 where the empirical CDF of the throughput samples is plotted.

B. Channel Quality Monitoring

In this section we discuss the evaluation of the interference tracking mechanism. RSSI values are monitored at different *WTP*s. A drop in signal strength measured at a single *WTP* can be due to a deterioration in channel performances due to for example fading. On the other hand, a concurrent drop of signal strength at different *WTP*s could be traced back to hardware failures and/or mis–configuration.

Figure 6 shows the signal strength of a wireless client at two different *WTP*s here named *Sensor A* and *Sensor B* during a 10–hours measurement campaign. As it can be seen a drop in RSSI can be observed at hour 4 for *WTP* A. This event is due to localized channel fading simulated by removing the antenna of one *WTP*. Conversely a concurrent drop of RSSI at both *WTP*s can be observed at hour 8. This event is due to a simulated hardware failure at the wireless client simulated by reducing the transmission power to 0 dB.

C. Proximity Detection

Modern location–based applications and services rely on the possibility to know in real–time the geographical position of customers. While GPS–based localization can provide precise and real–time geo–localization, its reliability drops dramatically in indoor settings. Several indoor localization solutions leveraging various technologies (WiFi, Bluetooth, acoustic, etc.) are currently commercially available. While some of them are characterized by sub–m precision, their cost could be prohibitive for many deployments. Moreover, for several use cases proximity based localization is sufficient instead of precise indoor geo–localization. By *proximity detection*, we refer to the capability of knowing if a certain wireless client is within a few meters from an anchor point (a *WTP* in this case). Notice that the assumption here is that anchor points are deployed in close proximity of points of interests in a certain venue, such as check–in desks or shops in an airport.

The RSSI tracking capabilities allowed by the *Channel Quality Map* can be effectively leveraged to implement such a proximity detection system. A simple RSSI tracking *Network*

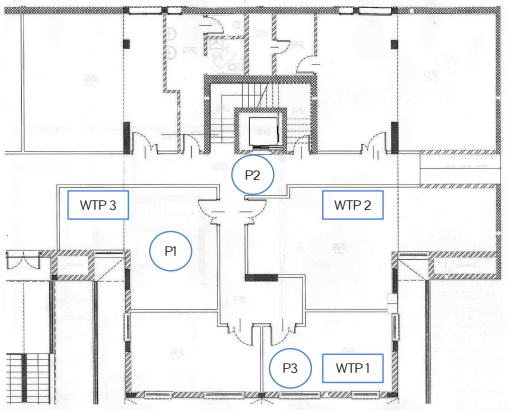


Fig. 7: Proximity detection scenario floor layout.

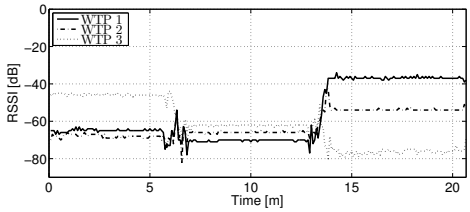


Fig. 8: RSSI values of a mobile device at different *WTP*s.

App has been implemented as proof–of–concept. The *Network App* tracks in real–time the RSSI of wireless clients at different *WTP*s in the network.

Figure 7 depicts the floor layout and the position of the *WTP*s used in order to evaluate the proximity detection *Network App*. The scenario consists of three *WTP* deployed across single floor in a typical office building.

RSSI samples are collected for a stationary wireless client performing standard internet browsing tasks. Three reference positions were considered, labeled as $P1$, $P2$, and $P3$ in Fig. 7. The client was stationary at each position for about 5 mins and then moved to the position. Figure 8 show the instantaneous RSSI of the wireless client at the three *WTP*s. As it can be seen the RSSI values measured by the *WTP*s can provide a reliable proximity information of the wireless client.

V. Related Work

Due to the vast literature on interference monitoring and management, in this section we will focus only on empirical approaches with a real–world evaluation. For a broad survey on interference modeling techniques in 802.11 networks we refer the reader to [10].

Passive and/or active measurements are leveraged by several authors to derive either the conflict graph or the interference graph of a wireless network. In [11], [12] active measurements are exploited in order to study how mutual link interference affects packet delivery ratio and throughput. In [13] micro–probing (i.e. active measurements lasting few milliseconds) is used in order to detect conflicts between links. Passive interference graph construction techniques are presented in [14], [15]. A framework capable of performing root cause analysis in WiFi networks is presented in [16]. WIT [17] and Jigsaw [18] another two examples of passive interference monitoring techniques aiming at modeling cross–link interference.

Conflict graphs are leveraged in [19] to manage the effect of interference when multiple transmitters employ variable channel widths. An architecture using micro–probing to jointly address channel assignment and transmission power control is presented in [20]. Centralized scheduling is exploited by the authors of [21] in order to mitigate hidden and exposed terminals issues in WiFi–based networks. Interference modeling plays a key role in the client scheduling problem. Finally, a distributed anomaly detection system for WiFI networks is presented in [22].

The argument for handling fast timescale events as close as possible to the place where they are originated is made in [4], [5]. However, both works do not address the way global channel quality information shall be exposed to the nertwork programmer (the *Channel Quality Map*abstraction) nor they propose a viable network reconfiguration model (the *Port* abstraction).

The works above show how interference modeling and network reconfiguration are receiving continuing interest from the research community. Nevertheless none of them has the goal of providing programmers with an high–level interface to access network interference information, nor they define a scalable network reconfiguration model that clearly separates autonomic control policies from other network management tasks. Our work is instead aimed at defining high–level programming primitives for both representing and manipulating the network state [23], [24], [25], [26], [27], [28], [29]. Unlike them however, our work focuses on modeling the most critical aspects of a WiFi–based RAN (gathering network state, interference-aware resource allocation and network reconfiguration and adaptation) and exposing them to the network programmers through a set of technology agnostic programming primitives.

VI. Conclusions

In this paper, we have examined the set of high–level programming abstractions and primitives needed for effective interference management in Software–Defined Wireless Networks. The proposed primitives are designed around the consideration that a clear line must be drawn between network control and network management. The former involves autonomic control policies operating at network edges while the latter include slower timescale network management tasks. A preliminary implementation of the proposed abstractions has been evaluated over a small scale WiFi testbed. Results show that the programming primitives can actually be used to realize practical resource allocation algorithms. As a future work we plan to validate the programming primitives over a wider deployment as well as to open–source the entire software stack, from the *WTP* firmware to the *SD–RAN Controller* and the Python SDK, making it available to the research community under a permissive license.

REFERENCES

[1] R. Riggio, K. M. Gomez, T. Rasheed, J. Schulz-Zander, S. Kuklinski, and M. K. Marina, "Programming Software–Defined Wireless Networks," in *Proc. of IEEE CNSM*, Rio de Janeiro, Brasil, 2014.

[2] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *Proc. of ACM HotSDN*, Helsinki, Finland, 2012.

[3] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz, "Programmatic orchestration of wifi networks," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 347–358. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/schulz-zandery

[4] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342446

[5] J. Schulz-Zander, N. Sarrar, and S. Schmid, "Towards a scalable and near-sighted control plane architecture for wifi sdns," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 217–218. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620772

[6] "Tornado Web Server." [Online]. Available: http://www.tornadoweb.org/

[7] 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, Aug. 2000.

[8] "Iperf." [Online]. Available: http://iperf.sourceforge.net/

[9] P. San Segundo, "A new dsatur-based algorithm for exact vertex coloring," *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1724–1733, Jul. 2012.

[10] P. Cardieri, "Modeling interference in wireless ad hoc networks," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 4, pp. 551–572, April 2010.

[11] D. Niculescu, "Interference map for 802.11 networks," in *Proc. of ACM IMC*, San Diego, California, USA, 2007.

[12] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Measurement-based models of delivery and interference in static wireless networks," in *Proc. of ACM SigComm*, Pisa, Italy, 2006.

[13] N. Ahmed, U. Ismail, S. Keshav, and K. Papagiannaki, "Online estimation of rf interference," in *Proc. of ACM CoNEXT*, Madrid, Spain, 2008.

[14] V. Shrivastava, S. Rayanchu, S. Banerjee, and K. Papagiannaki, "Pie in the sky: Online passive interference estimation for enterprise wlans," in *Proc. of USENIX NSDI*, Boston, MA, 2011.

[15] M. Vutukuru, K. Jamieson, and H. Balakrishnan, "Harnessing exposed terminals in wireless networks," in *Proc. of USENIX NSDI*, San Francisco, California, 2008.

[16] D. Giustiniano, D. Malone, D. Leith, and K. Papagiannaki, "Measuring transmission opportunities in 802.11 links," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 5, pp. 1516–1529, Oct 2010.

[17] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Analyzing the mac-level behavior of wireless networks in the wild," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 75–86, Aug. 2006.

[18] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and S. Savage, "Jigsaw: Solving the puzzle of enterprise 802.11 analysis," in *Proc of SigComm*, Pisa, Italy, 2006.

[19] S. Rayanchu, V. Shrivastava, S. Banerjee, and R. Chandra, "Fluid: Improving throughputs in enterprise wireless lans through flexible channelization," in *Proc. ACM MobiCom*, Las Vegas, Nevada, USA, 2011.

[20] N. Ahmed and S. Keshav, "Smarta: A self-managing architecture for thin access points," in *Proc. of ACM CoNEXT*, Lisboa, Portugal, 2006.

[21] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra, "Centaur: Realizing the full potential of centralized wlans through a hybrid data path," in *Proc. of ACM MobiCom*, Beijing, China, 2009.

[22] A. Sheth, C. Doerr, D. Grunwald, R. Han, and D. Sicker, "Mojo: A distributed physical layer anomaly detection system for 802.11 wlans," in *Proc. of ACM MobiSys*, Uppsala, Sweden, 2006.

[23] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. of ACM WREN*, 2009.

[24] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proc. of USENIX NSDI*, 2013.

[25] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Proc. of ACM PADL*, 2011.

[26] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *SIGPLAN Not.*, vol. 46, no. 9, pp. 279–291, Sep. 2011.

[27] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," in *Proc. of ACM HotSDN*, 2012.

[28] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. of ACM WREN*, 2009.

[29] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," in *Proc. of ACM POPL*, 2012.