UniFlex: A Framework for Simplifying Wireless Network Control

Piotr Gawłowicz, Anatolij Zubow, Mikołaj Chwalisz and Adam Wolisz
{gawlowicz, zubow, chwalisz, wolisz}@tkn.tu-berlin.de
Telecommunication Networks Group (TKN), Technische Universität Berlin (TUB)
Einsteinufer 25, 10587 Berlin, Germany

***Abstract*—Classical control and management plane for computer networks is addressing individual parameters of protocol layers within an individual wireless network device. We argue that this is not sufficient in phase of increasing deployment of highly re-configurable systems, as well as heterogeneous wireless devices co-existing in the same radio spectrum. They demand harmonized, frequently even coordinated adaptation of multiple parameters in different protocol layers (cross-layer) in multiple network devices (cross-node).**

We propose UniFlex, a framework enabling unified and flexible radio and network control. It provides an API enabling coordinated cross-layer control and management operation over multiple wireless network nodes. The controller logic may be implemented either in a centralized or distributed manner. This allows to place time-sensitive control functions close to the controlled device (i.e., local control application), off-load more resource hungry control application to compute servers and make them work together to control entire network.

The UniFlex framework was prototypically implemented and provided to the research community as open-source. We evaluated the framework in a number of use-cases, what proved its usability.

***Index terms*—** Control, Cross-Layer, SDN, Wireless, Het-Net, Distributed Control Plane

I. Introduction

The control plane and the management plane have played a very important role in the classical telecommunication systems, but have been given much less attention in computer networks. As the matter of fact the only widely accepted approach is the usage of SNMP or NETCONF as basis for creating management applications. This is increasingly recognized as not sufficient - especially in case of wireless networks where many parameters have to be frequently tuned in response to changing wireless propagation, interference and traffic conditions. There were already couple of attempts for wireless control protocols including LWAPP and CAPWAP, but those were designed with focus on configuration management and device management and are not suitable for time-sensitive control of devices.

Furthermore, classical control/management actions have been addressing individual parameters of protocol layers within an individual network device. This is not sufficient in phase of increasing deployment of highly re-configurable systems, as well as heterogeneous wireless systems co-existing in the same radio spectrum. They demand harmonized, frequently even coordinated (including simultaneous) change of multiple parameters in different parts of hardware and software in multiple network devices.

Typical example of emerging real scenarios are LTE-U and Wi-Fi in 5 GHz and Wi-Fi, Bluetooth and ZigBee in 2.4 GHz ISM bands. On the other hand even homogeneous deployments are suffering from intra-technology interference [1]. Here, in recent years, we have seen a boom of cross-layer design proposals for wireless networks [2], [3], where additional information from some layers are obtained and used to optimize operation of another layer.

So far control applications had to solve the challenges of harmonized/simultaneous actions on case-by-case basis, which significantly complicated development of such applications - and lead to lack of any compatibility across the individual solutions. We argue that the efficiency of wireless networks can be significantly improved by enabling the management and control of the different co-located wireless technologies and their network protocols stacks in a coordinated way using either centralized or distributed controllers [4].

Contribution: In this paper we propose **UniFlex**, a framework for **Uni**fied and **Flex**ible control in networked systems. The suggested API supports typical functions needed for coordinated cross-layer, cross-technology and cross-node control. Similarly, as in the SDN paradigm, we allow for centralized control, but support equally well also hierarchical control structure and logically centralized but physically distributed control. Network control applications can be either co-located with controlled device (both running on the same network node, e.g. for latency reasons) or separated from each other (running on two nodes, e.g. control application runs on server due to high computing power requirement). We believe that UniFlex will be an enabler for rapid prototyping and deployment of control applications for wireless networks. The UniFlex prototype is provided as open-source to the community: https://github.com/uniflex.

II. System Model

In this section, we define our system model and provide definitions of all terms that we use consistently in this paper.

A **network** is a collection of – possibly heterogeneous – **nodes** under a common management and control authority. A node is collection of equipment and software sharing a common platform and being run under a single instance of operating system. The types of nodes span from small constrained devices to powerful compute servers. A node contains zero or more **controllable units** that fall under two categories: i) **devices** in hardware domain and ii) **protocols** in software domain. A controllable unit is a piece of hardware/software fulfilling a dedicated functionality. Additionally, it may expose a set of operations in the Native Programming Interface (**NPI**) to control its behaviour and parameters. For example, a **wireless network device** provides packet forwarding functions with usage of wireless transmission technology (e.g. 802.11, LTE, ZigBee) and exposes NPI to control its parameters including transmission power, central frequency, bandwidth.

The operation of a network is harmonized by single or multiple **controllers**. The controller logic may be implemented either as standalone or multiple cooperating **control applications** that run in node(s). In particular, a control application may be located in the same node as network device that it is controlling. It usually collects information and measurements from the network, make control decisions according to the set policies and perform network reconfigurations.

We assume the existence of a common **Control Channel** enabling the control application(s) to: i) access NPI of all controllable units in network, ii) use it to control their behaviour and iii) exchange control messages between each other for the cooperation purposes. This control channel may be realized over a wireless network itself (in-band) or an additional wired backhaul infrastructure (out-of-band).

III. Design Requirements

The main goal of our work is to facilitate and shorten time required for prototyping of novel control solutions in heterogeneous wireless networks. We argue that novel wireless applications may be realized when following functionality is provided:

- Coordinated **collection** of information from and **execution** of multiple operations on different protocol layers (**cross-layer**), heterogeneous devices (**cross-technology**) and multiple nodes (**cross-node**) within network,
- Support for flexible placement of the control applications, e.g. local control application for frequent interactions and global one for cross-node coordination,
- Support for detecting network changes in proactive and reactive control schemes,
- A high-level API for control of operation of individual controllable units as well as groups of them.

IV. Architecture Overview

The UniFlex framework is a distributed middleware platform running across multiple nodes that interconnects control applications and controllable units. The overview of UniFlex architecture is presented in Fig.1. The control applications perform control tasks by utilizing the provided API in the Northbound Interface, while the Southbound Interface is responsible for translating calls coming from control applications to the NPI. The interfaces are described in the following subsections.

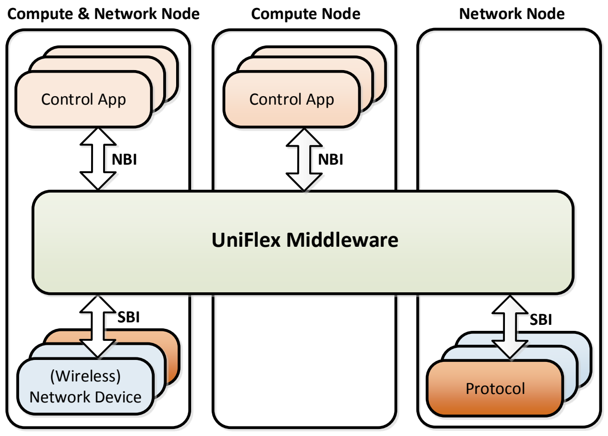

Fig. 1. Overview of UniFlex architecture.

The framework takes care of node management including node discovery and monitoring connections between all nodes. Whenever a new node is discovered or connection to a node is lost, it notifies all control applications about the changes. Moreover, the UniFlex also discovers capabilities of each node (i.e. its controllable modules and supported functions). This way each control application can be supplied with all information needed to create its global view of all nodes. By default, each control application is able to communicate with all other applications and control all units in the entire wireless network, but some access policies restricting this possibility may be applied.

Using UniFlex it is possible to develop distributed control applications, i.e. it is possible to split-up control logic into smaller cooperating control applications running on different nodes. In this way, UniFlex by design support three types of control programs: *i)* local – when controller is running in the same node as controlled device(s), *ii)* non-local – when controller is running on different node then controlled device(s) and *iii)* hybrid or hierarchical – when controller logic is split between multiple nodes.

The communication mechanism in UniFlex is based on exchange of **events**. An event is message that contains some information (e.g. notification, measurement sample, etc.). It can be generated by the framework, a control application, device or protocol – an event contains identifier of node and entity that created it. On the other hand, only control applications can consume events. To this end, they have to subscribe to be notified about events of specific type. A control application may send and subscribe for events in three modes: *i)* unicast, *ii)* node-broadcast and *iii)* global-broadcast. In unicast mode an application sends/receives events to/from a particular entity in network. In node-broadcast mode application subscribes to receive events of specific type generated by all entities running in particular node, and send events to them. Finally, using global-broadcast applications sends/receive events to/from all entities in the entire network. The framework and controllable units may send events only in global-broadcast mode.

Finally, as the coordinated control requires all nodes to share the same notion of global clock, the framework provides time synchronization mechanisms that work across different network and on nodes with different capabilities.

A. *Northbound Interface*

The NBI provides a control application with the location-independent API (i.e. the same calling syntax for execution of NPI commands on local and remote controllable entities) that allows for:

- blocking and non-blocking execution of commands,
- delaying execution of commands using relative time,
- scheduling execution of commands in future using absolute time,
- sending, subscribing and receiving events of specific types in three modes – see Section IV,
- creation of group containing multiple similar entities and execution of NPI commands on it,
- transactional execution of NPI commands,
- monitoring delay and time synchronization accuracy between peer nodes,
- controlling live cycle of control applications, including deployment of a new ones at run-time,
- migration of control applications between nodes in network,

In order to execute command, an control application has to specify, so called, *Calling Context*, that contains information of *WHAT* (command), *WHERE* (entity of group of them), *HOW* (blocking, delayed, etc.) and optionally *WHEN* (point in time) is to be executed. Optionally, in case of non-blocking, delayed and time-scheduled execution, it is also possible to register callback function that will be fired upon reception of return value.

B. *Southbound Interface*

The Southbound Interface works in two directions: *i)* control application execute functions and change parameters of controlled unit – *downlink*; and *ii)* the unit sends measurement data, samples, etc. to subscribers – *uplink*. The communication in *downlink* direction is realized with command calls, while communication in *uplink* direction is realized using events.

The Southbound Interface is realized with help of modules, that translate function calls from control applications into NPI – Fig.2A. In other words, a module wraps different APIs and tools used to control devices/protocols and exposes them to framework. In Fig.2B, we present two modules as an example. As shown SBI functions are delivered to modules and translated to proper NPI calls, i.e. NETLINK and XML/RPC respectively.

In many cases, while the semantics of features supported by different devices may be the same, their NPI may differ greatly. We argue that this would be error-prone and prevent portability and re-usability of control applications. We believe that those issues can be solved by a unified abstraction layer that hides specific NPIs of different devices behind common interface. We found Unified Programming Interface defined in WiSHFUL project [5], [6] to be an appropriate option.

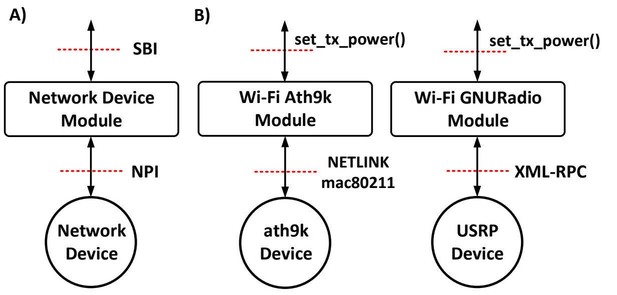

Fig. 2. Device Module is a Python wrapper for Native Programming Interface.

V. Implementation Details

The UniFlex framework was prototypically implemented. The particular attention was paid to enable code re-usability and support for different programming languages as well as the possibility to use specialized external software libraries.

Our prototype is implemented in Python language, what makes it possible to run on multiple different host types and allows for rapid prototyping of control applications. As we used only standard and common Python libraries, we are able to run and test our implementation on multiple platforms, including x86, ARM and MIPS. An overview of framework implementation is given in Fig.3. An Agent exposes *Northbound Interface* to Control Applications and connects to Device/Protocol Modules using *Southbound Interface*. The Agents communicate with each other over Broker. All agents together with the broker constitute the distributed UniFlex control framework middleware. Each depicted entity is described in detail in the following subsections.

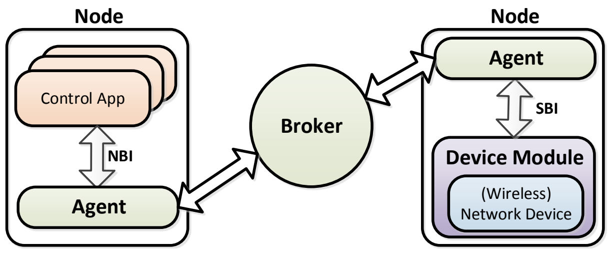

Fig. 3. Overview of UniFlex framework implementation.

A. *Northbound Interface*

The overview of Northbound Interface is depicted as UML diagram in Fig. 4. In order to control a node, an control application has to first obtain a `NodeProxy` object. To this end, it subscribes to `NewNodeEvent` events. On discovery of a new node Agent notifies the application by sending event containing a node proxy object. Thereafter, the application can retrieve the `DeviceProxy` and/or `ApplicationProxy` objects from the received `NodeProxy` and execute commands on them – see Listing 2.

An control application sends events using *send_event(event)* function and subscribe for events using *subscribe_for_events(eventType, callback)*. Once subscribed the framework will deliver events of proper type to the control application and fire the bounded callback passing

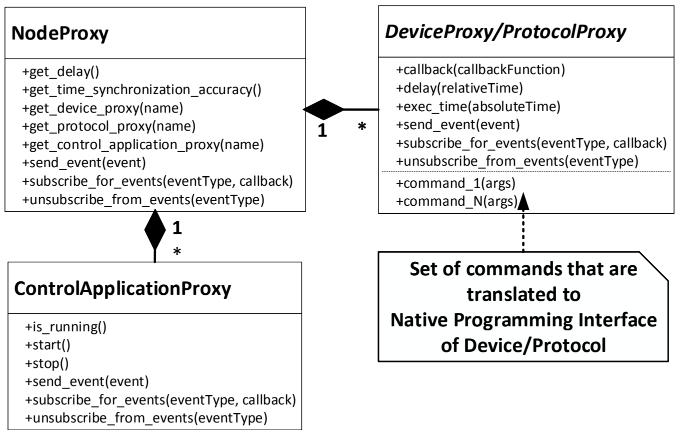
NodeProxy
+get_delay()
+get_time_synchronization_accuracy()
+get_device_proxy(name)
+get_protocol_proxy(name)
+get_control_application_proxy(name)
+send_event(event)
+subscribe_for_events(eventType, callback)
+unsubscribe_from_events(eventType)
DeviceProxy/ProtocolProxy
+callback(callbackFunction)
+delay(relativeTime)
+exec_time(absoluteTime)
+send_event(event)
+subscribe_for_events(eventType, callback)
+unsubscribe_from_events(eventType)
+command_1(args)
+command_N(args)
1
*
1
*
Set of commands that are translated to Native Programming Interface of Device/Protocol
ControlApplicationProxy
+is_running()
+start()
+stop()
+send_event(event)
+subscribe_for_events(eventType, callback)
+unsubscribe_from_events(eventType)

Fig. 4. Overview of Northbound Interface

event as an argument. Sending and subscription of events are implemented around PUB and SUB sockets available in the *ØMQ* communication library [7]. Note that, for security reasons an authentication and encryption can be applied on *ØMQ* level.

The execution of commands was implemented on top of unicast event mechanism. For this purpose, we introduced two events, namely CommandEvent and ResponseEvent that are handled internally by the Agent. The CommandEvent event contains *Calling Context*. The creation of *Calling Context* and sending of CommandEvent is hidden behind execution of command call on proxy objects. To facilitate building of the *Calling Context*, we introduce following functions: *delay(relative_time)*, *exec_time(absolute_time)* and *callback(cb)* that may be executed on proxy object. They are optional and may be chained together. The examples of the supported calling semantics are presented in Listing 1.

In order to support delayed and time-scheduled function execution, the Agent class is equipped with scheduler. Note, when coordinating multiple nodes by means of time scheduled execution, nodes in network must have common notion of global clock. Currently, the synchronization mechanism is based on Precision Time Protocol (PTP) and can be enabled in the framework.

Listing 1. Calling examples.

```
# definition of callback function
def my_get_power_cb(data):
    print(data)
# get device proxy from node proxy
device = node.get_device(0)
# execution of blocking call
result = device.get_tx_power()
# execution of non-blocking call
device.callback(my_get_power_cb) \
      .get_tx_power()
# delay execution of call by 3 seconds
device.delay(3).callback(my_get_power_cb) \
      .get_tx_power()
# schedule execution of non-blocking call
t = datetime.now() + timedelta(seconds=3)
device.exec_time(t) \
      .callback(my_get_power_cb) \
      .get_tx_power()
```

B. Agent

The agent is an entity that runs control applications and device/protocol modules. It connects with other agents present in network and provides information about them and their capabilities to its local control applications. For this purpose, we implemented node discovery and heartbeat mechanisms. Moreover, the agent is responsible for transferring events from applications and modules to the broker.

C. Broker

The broker takes care for delivering events between agents. Events of specific type are delivered only to those agents that subscribed for them (i.e. at least one control application running on agent subscribed for them). The broker is switching events between XPUB and XSUB sockets available in *ØMQ* library [7] library. The ZMQ itself implements mechanisms for topic agreement and message routing.

D. Control Application

A Control Application is an entity that implements the entire or part of the network control logic. It can be as simple as a signal filter or as complicated as a mobility management unit. A control application has to inherit from ControlApplication base class, provided in UniFlex framework package, to get access to NBI.

While it is possible to subscribe at run-time to events using *subscribe_for_event()* as described in Sec. IV-A, we expect that mostly permanent (i.e. lasting for life-cycle of an application) subscriptions will be used. For this purpose, we provide *on_event* decorator that binds event notification with desired function. For example in Listing 2, the control application subscribes for NewNodeEvent events and gets notification whenever a new node is discovered.

Listing 2. Example of blocking command call.

```
@on_event(NewNodeEvent)
def add_node(self, event):
    node = event.node
    device = node.get_device(0)
    txPower = device.radio.get_tx_power()
```

E. Southbound Interface

As already mentioned the SBI is realized with help of modules. In order to be used within the UniFlex framework, a device's (protocol's) NPI has to be wrapped with module that inherits from the provided Module base class. This class connects to agent, receives commands from control applications and executes corresponding function implementation according to requested commands.

An example of function implementation is presented in Listing 3. Here, a *wifi_set_channel* function takes channel as an argument and uses the NETLINK interface to communicate with the Linux 802.11 subsystem to configure the network device. We provide *bind_function* decorator to mask function names which can also be used to implement an unified abstraction layer. In the example, the function is hidden behind proper operation from UPI definition. Note that the `Module` is a Python object and it may keep state – recent return values may be cached to reduce delay. Finally, the `Module` class takes care of serialization and parsing of events as well as function arguments and return values.

The agent creates and presents a local proxy object for each discovered module to all control applications running on top of it. Such a proxy object contains all functions (i.e. the same signatures) of remote module and corresponding functions are bound together, i.e. calling function on proxy object translates in execution of function in remote module object. This way, we achieve location-transparency.

Listing 3. Example implementation of a device module function.

```
@bind_function(upi.radio.set_channel)
def wifi_set_channel(self, channel):
    self.channel = channel
    # set channel in wireless interface using NETLINK
    return reponse
```

VI. Applications

In this section, we present the selected control applications, which we have implemented using the UniFlex framework.

A. Mobility Management

Novel applications, e.g. mobile HD video, and devices, e.g. smartphones and tablets, require much better mobility support and higher QoS/QoE. Therefore, in [8] we presented BIGAP, a seamless handover scheme for high performance enterprise IEEE 802.11 networks. We implemented the mobility management function of BIGAP in UniFlex. The Fig. 5 shows the hierarchical controller architecture consisting of two central control applications and two local ones running in each AP. The local control applications are collecting information about: *i)* quality of the active wireless links as well as potential links to client stations in communication range which are currently being served by another co-located AP; and *ii)* the current network load at each AP. This data is reported as events (*CQIReportEvent* and *LoadReportEvent*, respectively) to the *Central Mobility Manager*, which decides on handover by sending out a *HORequestEvent*. This event is processed by the *Handover Control Application* which performs the actual handover operation as described in [8].

B. Interference Management

Another known problem experienced in 802.11 networks is performance degradation due to co-channel interference caused by hidden nodes. The impact can be mitigated by preventing overlapping transmissions (in time) of affected nodes, e.g. APs, by efficient airtime management through interference avoidance techniques at the MAC layer. We implemented interference management in UniFlex. In particular the following two features have been implemented: *i)* detection of wireless links suffering from hidden nodes and *ii)* execution of airtime management in which two wireless links suffering from the hidden node problem are getting exclusive time slots assigned. The Fig. 6 illustrates the developed hierarchical

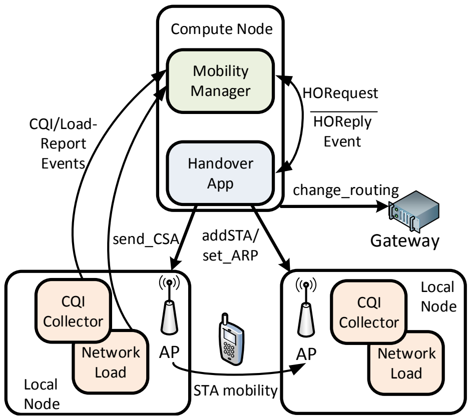

Fig. 5. Mobility management for enterprise 802.11 networks.

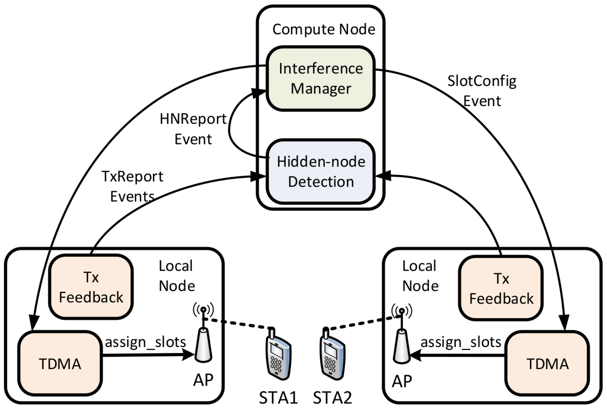

Fig. 6. Interference management through airtime management in 802.11 networks.

controller architecture. Here, we have two control applications running on each AP locally due to timing constraints and efficiency reasons. The *TxFeedback* control application provides transmission feedback information like number of ARQ retries to the central *Hidden-node Detection* control application which is using this information for discovery of hidden-nodes. Each pair of wireless links suffering from hidden-node is reported using *HNReportEvent* and consumed by another central control application, *Interference Manager*, which in turn decides on the time slot configuration to be used. The actual assignment of time slots to nodes is performed by the local *TDMA* scheduler.

VII. Evaluation

In this section we analyze the performance of our prototypical implementation with respect to two categories: i) basic network operation and ii) scalability with respect to the number of controlled network nodes.

A. Basic Network Operation

Since observing and modifying the network state by means of executing API functions is a basic building block of UniFlex operations, its performance is of great importance on the overall system's performance. We identified latency for network state monitoring and API function execution as an important performance metric.

For this measurement, the experiments were conducted using three different network nodes: i) high performance Intel i7-4790, ii) small-form-factor-PC based on Intel NUC and iii) low-power single-board ARM Cortex-A8 machines (BeagleBone). All three nodes were equipped with a single 802.11 network device. For the evaluation of the performance of local calls we implemented a local control application whereas for remote calls a global controller running on a different node connected by Gigabit-Ethernet was used. We measured the latency of executing API functions, both locally and remotely.

Table I shows the mean and 99th percentile of the latency when executing a single blocking local API function call, *get_interfaces()* which returns the available wireless interfaces of a wireless node.

Latency	*Median*	*99 %ile*
Intel (i7-4790, 3.6 GHz)	0.4017 ms	0.5009 ms
Intel NUC (i5-4250U, 1.3 GHz)	0.7627 ms	1.3986 ms
BeagleBone (ARM armv7l, 1 GHz)	10.0138 ms	11.4258 ms

TABLE I
LATENCY FOR EXECUTING SINGLE BLOCKING LOCAL API FUNCTION.

Further, Table II shows the results when executing the same function remotely. Note that the network overhead for the execution of this API call is around 1600 Bytes per call.

From the results we can conclude that the latency of performing an API call, locally or remotely, is sufficiently low to be used for real-world control applications. However, when using slow ARM SoCs the latency is $11 - 25\times$ larger as compared to i7-4790 which might be insufficient. We argue that the UniFlex agent can be easily implemented in a low-level programming language like C, what will definitely shorten execution time of the API functions.

Latency	*Median*	*99 %ile*
Intel (i7-4790, 3.6 GHz)	1.2896 ms	1.5042 ms
Intel NUC (i5-4250U, 1.3 GHz)	2.6748 ms	3.1662 ms
BeagleBone (ARM armv7l, 1 GHz)	14.5829 ms	16.4588 ms

TABLE II
LATENCY FOR EXECUTING SINGLE BLOCKING REMOTE API FUNCTION.

B. Scalability

Another important performance metric is scalability. A key feature of our framework is its distributed architecture for scale-out performance. As the number of network nodes to be controlled grows the demand on the control plane increases.

For this measurement, the experiments were conducted in the ORBIT testbed [9] consisting of i7-4790 x86 machines. The number of controlled network nodes was varied from 1 to 87 nodes. A single central control program was executing API calls, *get_interfaces()*, on each node using non-blocking calling semantic. We measured the latency to get the results from all nodes.

The results are shown in Fig. 7. It takes less than 25 ms to execute a non-blocking API call on all 87 network nodes. Note, that the latency per API call decreases with the number of nodes, i.e. 2.37 ms vs. 0.24 ms for 1 and 87 nodes respectively. This is because non-blocking calls are executed in parallel.

Note, that with 87 nodes and a API calling rate of 10 Hz the control plane workload at the central controller is already high, i.e. 16 Mbit/s. In order to reduce it, the use of hierarchical or local controllers is advisable.

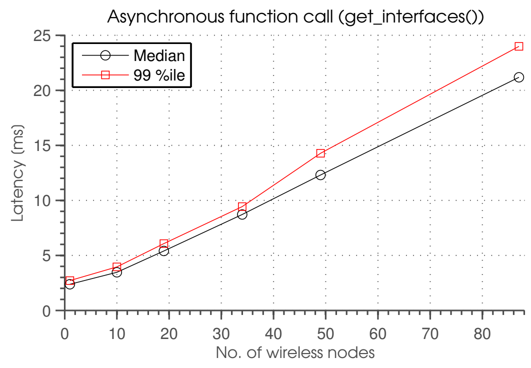

Fig. 7. Latency for executing single non-blocking API function call on a set of nodes.

VIII. Related Work

Related work falls into three categories:

Cross-layer Control: CRAWLER [10], [11] is experimentation architecture for centralized network monitoring and cross-layer coordination over different devices. Click-Watch [12] aims for simplification of experimentation of wireless cross-layer solutions implemented using the Click Modular Router [13]. Both frameworks aim to facilitate experimentation and offer possibility to control all nodes in the network from a single centralized controller. In contrast, UniFlex is more flexible as it allows distributing controller logic over multiple nodes so that the time sensitive control logic can be executed directly on the network node.

Software-defined Networking: There are already several distributed control frameworks, but they are mostly focused on control of wired switches using open protocols (e.g. OpenFlow). Some of them, like ONOS [14] and ONIX [15] are focused on scalability and performance. As they are already in very advanced state, it is hard to use them for resource constrained devices or to adjust them to wireless networking. Ryuo [4] and Kandoo [16] provide the possibility for offloading of control applications to local controllers as a way to reduce the control plane load. The local controllers handle frequent events, while a logically centralized root controller handles rare events. In contrast UniFlex is not restricted to two levels of controllers as it allows direct communication between any control applications. Beehive [17][18] provides interesting features like automatic distribution of control applications over network nodes. While having similar concepts, Beehive does not differentiate between control applications and device modules which are of great importance when targeting the control of heterogeneous wireless networks.

CoAP [19] proposes a vendor neutral centralized framework for configuration, coordination and management of residential 802.11 APs using an open API implemented over OpenFlow [20]. In contrast to UniFlex, the CoAP API is restricted to control of 802.11 networks. Moreover, only centralized control programs are possible. OpenRF [3] provides programming abstractions tailored for wireless networks, i.e. MIMO interference management techniques that impact the physical layer. OpenRF is restricted to centralized control of 802.11 infrastructure networks. Finally, in [21] SDN architecture for centralized spectrum brokerage in residential infrastructure Cognitive Radio networks was proposed.

General Distributed Control Platforms: ROS [22] is an open source robot operating system for rapid prototyping. ROS is focused on providing control for a single robot, trying to achieve one goal, and having all devices working towards that goal. In UniFlex, we are trying to achieve harmonization of multiple devices. Moreover, we also provide time scheduled execution of operations on multiple devices.

IX. Conclusions

This paper introduces UniFlex, a framework that uses SDN concepts to simplify prototyping of novel wireless networking solutions requiring cross-layer control coordinated among multiple heterogeneous wireless network nodes. It provides a rich API for management and control of operation of network entities and allows for implementation of local, central and distributed network controllers. For future work, we plan to further develop and optimize UniFlex. In current implementation events between agents are sent over broker, what might be a performance bottleneck and single point of failure. We are working on direct communication between agents (while keeping the broker for discovery purposes) and on support for transactional execution of commands on multiple nodes.

X. Acknowledgment

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 645274 (WiSHFUL project).

References

[1] J. Mvulla, E.-C. Park, M. Adnan, and J.-H. Son, "Analysis of asymmetric hidden node problem in IEEE 802.11 ax heterogeneous WLANs," in *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*. IEEE, 2015, pp. 539–544.

[2] V. Pejovic and E. M. Belding, "Whiterate: A context-aware approach to wireless rate adaptation," *IEEE Transactions on Mobile Computing*, vol. 13, no. 4, pp. 921–934, 2014.

[3] S. Kumar, D. Cifuentes, S. Gollakota, and D. Katabi, "Bringing cross-layer MIMO to today's wireless LANs," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 387–398.

[4] S. Zhang, Y. Shen, M. Herlich, K. Nguyen, Y. Ji, and S. Yamada, in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific,*.

[5] P. Ruckebusch, S. Giannoulis, E. De Poorter, I. Moerman, I. Tinnirello, D. Garlisi, P. Gallo, N. Kaminski, L. DaSilva, P. Gawlowicz *et al.*, "A unified radio control architecture for prototyping adaptive wireless protocols," in *Networks and Communications (EuCNC), 2016 European Conference on*. IEEE, 2016, pp. 58–63.

[6] C. Fortuna, P. Ruckebusch, C. Van Praet, I. Moerman, N. Kaminski, L. DaSilva, I. Tinirello, G. Bianchi, F. Gringoli, A. Zubow *et al.*, "Wireless software and hardware platforms for flexible and unified radio and network control," in *European Conference on Networks and Communications (Eu-CNC)*, 2015.

[7] iMatix Corporation, "ZMQ - Code Connected," *http://zeromq.org/*, January 2014, accessed: 2015-08-04.

[8] A. Zubow, S. Zehl, and A. Wolisz, "BIG AP – Seamless Handover in High Performance Enterprise IEEE 802.11 Networks," in *Network Operations and Management Symposium (NOMS), 2016 IEEE*, April 2016.

[9] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 3. IEEE, 2005, pp. 1664–1669.

[10] I. Aktaş, O. Punñal, F. Schmidt, T. Drüner, and K. Wehrle, "A framework for remote automation, configuration, and monitoring of real-world experiments," in *Proceedings of the 9th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*. ACM, 2014, pp. 9–16.

[11] I. Aktas, F. Schmidt, M. H. Alizai, T. Drüner, and K. Wehrle, "CRAWLER: An experimentation platform for system monitoring and cross-layer-coordination," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*. IEEE, 2012, pp. 1–9.

[12] M. Scheidgen, A. Zubow, and R. Sombrutzki, "ClickWatch—An experimentation framework for communication network test-beds," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2012, pp. 3296–3301.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[15] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 351–364.

[16] 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*. ACM, 2012, pp. 19–24.

[17] S. H. Yeganeh and Y. Ganjali, "Beehive: Towards a Simple Abstraction for Scalable Software-Defined Networking," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 13:1–13:7.

[18] ——, "Beehive: Simple Distributed Programming in Software-Defined Networks," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 4:1–4:12.

[19] A. Patro and S. Banerjee, "COAP: A software-defined approach for home WLAN management through an open API," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 18, no. 3, pp. 32–40, 2015.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[21] A. Zubow, M. Döring, M. Chwalisz, and A. Wolisz, "A SDN approach to spectrum brokerage in infrastructure-based Cognitive Radio networks," in *Dynamic Spectrum Access Networks (DySPAN), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 375–384.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.