OSWireless: Hiding Specification Complexity for Zero-Touch Software-Defined Wireless Networks

Sabarish Krishna Moorthy[a], Nicholas Mastronarde[a],
Elizabeth Serena Bentley[b], Michael Medley[b], and Zhangyu Guan[a*],
[a]Department of Electrical Engineering, University at Buffalo, Buffalo, NY 14260, USA
[b]Air Force Research Laboratory (AFRL), Rome, NY 13440, USA
Email: {sk382, nmastron, guan}@buffalo.edu, {elizabeth.bentley.3, michael.medley}@us.af.mil

Abstract

In the current practice of wireless engineering, to optimize wireless networks engineers usually need to grapple simultaneously with network modeling, algorithm and protocol design as well as their implementation on distributed edge nodes. This process is tedious and error prone. In this article we attempt to address this challenge by designing OSWireless, a new control plane for optimizing software-defined wireless networks. At the core of OSWireless is the virtualization of four control plane functionalities, including intent, mathematical, algorithmic and forwarding specifications, and then provide them as a service to network engineers. To this end, we design two new subplanes for the control plane: Wireless Network Abstraction Specification (WiNAS) Subplane and Optimization-as-a-Service (OaaS) Subplane. The former converts intent specifications defined using high-level Application Programming Interfaces (APIs) to the corresponding mathematical specifications, and the latter generates automatically operational (possibly distributed) algorithmic specifications. We prototype OSWireless and deploy it over NeXT, a newly developed software-defined experimentation testbed, and showcase the automated control program generation capability of OSWireless and the optimality of the resulting programs considering a variety of network control problems. We further test the applicability of OSWireless on *UBSim*, a newly-developed Python based simulator for integrated aerial-ground networks, considering location optimization of mobile nodes as an example. Through developing OSWireless, we hope to accelerate research towards future zero-touch software-defined wireless networks with reduced management complexity.

Keywords: Zero-Touch Networks, Software-Defined Networking, Specification Abstraction, Distributed Control.

1. Introduction

Software-defined networking (SDN) is a key technique to enable next-generation (NextG) *programmable* networks with high scalability and low management complexity, and to accelerate the adoption of new communication techniques and hence hasten the network evolution [2–5]. In the past decade, SDN has captured the attention of both academia and industry [6–18]. Notably, in 2011 the Open Networking Foundation (ONF) released OpenFlow, a standard defining the communication interface between the control and data planes of SDN-based networks [2]. Later,

*Corresponding author

[1]A preliminary shorter version of this paper appeared in the Proceedings of International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, 2022 [1].

[2]ACKNOWLEDGMENT OF SUPPORT AND DISCLAIMER: (a) Contractor acknowledges Government's support in the publication of this paper. This material is based upon work funded by AFRL, under AFRL Contract No. FA8750-20-1-0501 and FA8750-20-C-1021, and in part by the NSF under Grant SWIFT-2229563. (b) Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL.

[3]DISTRIBUTION STATEMENT A: Approved for Public Release; distribution unlimited AFRL-2021-3478 on 12 October 2021. Other requests shall be referred to AFRL/RIT 525 Brooks Rd Rome, NY 13441.

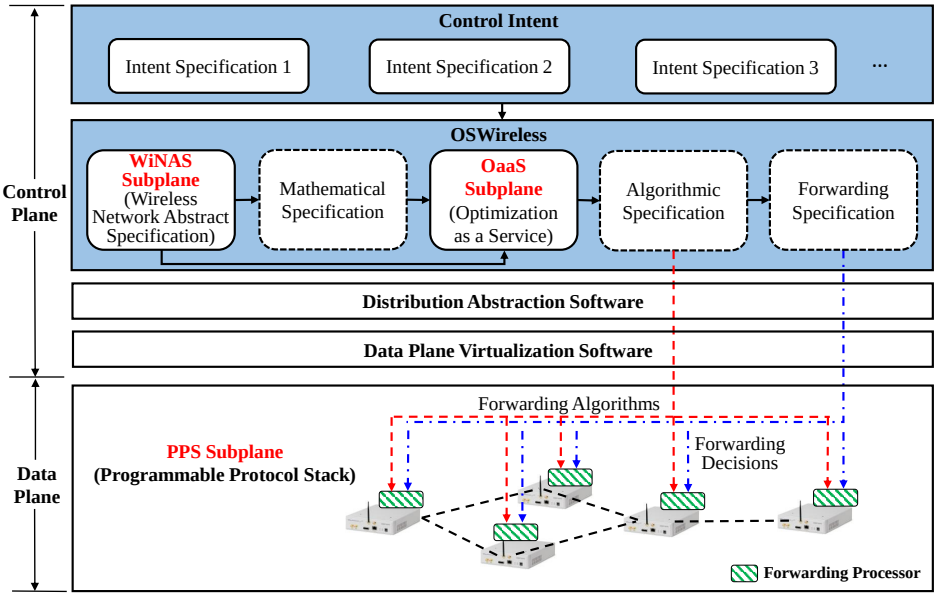


Figure 1: Overall architecture of OSWireless.

Google deployed their first SDN-enabled backbone wide area network called B4 based on OpenFlow for connecting their worldwide data centers [15]. Recently, Google deployed a new-generation distributed SDN controller called Orion in Google's B4 networks and data center Jupiter [17]. Po-Fi [18], is a Protocol-Oblivious Forwarding architecture which follows the SDN consensus and provides a rich and unified programmability to realize novel wireless functionalities with forwarding rules.

While the immense performance gains have been successfully demonstrated in wired networks (e.g., data center, backbone networks), the extension of SDN to wireless networks is still very challenging. The primary reason is that traditional SDN typically requires high-data-rate reliable links connecting the distributed forwarding substrates to the remote centralized SDN controller, which are typically not available in wireless networks. To address this challenge, significant efforts have been made to push the control plane closer to the edge of wireless networks. Representative examples include software defined radio access networks (SoftRAN) [14], software-defined cellular networks (CellSDN) [10], and mobile cloud computing based software defined wireless networks (MCC-SDWN) [19]. However, these approaches have been focusing on *softwarization* of civilian cellular networks and still require high-data-rate backhaul networks to accommodate the resulting overhead traffic, e.g., around 500 Mbps to support 50 micro cells each with cell range in the order of one kilometer based on SoftRAN [14]. This calls for new SDN architectures for NextG (both civilian and tactical) networks with densely deployed small cells, mobile (even flying) hotspots and heterogeneous network topology.

Meanwhile, existing SDN controllers primarily focus on providing services for *distribution abstraction* and *forwarding abstraction*, while very few efforts have been made for *specification abstraction*, where the objective is to provide simplified network models for mapping abstract specifications to physical operational configurations [20–22]. As a result, to optimize the networks the engineers need to grapple simultaneously with mathematical modeling, design of (possibly distributed) numerical algorithms, protocol design as well as the implementation of the resulting control logic in hardware. *This process is typically tedious and error prone.*

Contributions. In this work we attempt to address the above challenges by designing OSWireless, a new zero-touch SDN controller that can enable intent-driven self-optimizing software-defined wireless networks. A zero-touch network shares similar features with autonomous driving network (ADN) [23] and intent-driven network (IDN) [24], where the former refers to networks that can self-operate and adapt to changing conditions without human intervention thereby achieving self-configuration, self-optimization, self-healing, and self-protection, and the latter refers to networks that can be controlled and managed based on high-level business or operational intent expressed by users or administrators. In a nutshell, OSWireless provides a control plane for optimizing software-defined wireless networks

with automated optimization program generation capabilities, by virtualizing four control plane functionalities, including intent, mathematical, algorithmic and forwarding specifications, and providing them as a service to network engineers. Based on OSWireless, the network engineers are allowed to specify in a centralized manner the control intent (*i.e., what to do*) using the high-level specification abstraction APIs, while the control intent can be translated to (possibly distributed) operational control specifications (*i.e., how to do it*) following a series of carefully designed steps.

Towards this goal, we design two new subplanes for the SDN control plane, i.e., *Wireless Network Abstraction Specification (WiNAS) Subplane* and *Optimization-as-a-Service (OaaS) Subplane*. The former converts intent specifications defined using high-level APIs to the corresponding mathematical specifications, and the latter automatically generates operational algorithmic specifications. The overall architecture of OSWireless is illustrated in Fig. 1, where the red dotted lines represent the *forwarding algorithms*, blue dotted lines represent the *forwarding decisions* and black dotted lines represent the *forwarding routes.* We claim the following three main contributions.

i) WiNAS Subplane Design. We first design the WiNAS Subplane, the network abstraction engine of OSWireless. The objective of this subplane is to provide a set of high-level APIs for specifying various network control intents and to define the programming interfaces between the intent specifications and the OaaS Subplane. To this end, four modules have been designed to integrate those major functionalities, such as network element and topology definition provided by *NetTopo Module*, QoS characterization provided by *QoSPara Module*, parameter modeling provided by *ParaModel Module*, and network control problem construction provided by *MathSpec Module*.

ii) OaaS Subplane Design. Based on the services provided by the WiNAS Subplane, we further design the network optimization engine of OSWireless, i.e., OaaS Subplane. The automated specification translation is accomplished by first converting the intent specifications defined using textual strings to the symbolic domain and then extracting the coupling among the protocol layers and distributed nodes. Then, we prototype the OaaS Subplane by integrating two widely adopted network optimization frameworks as an example: dual method [25] and successive convex optimization [26].

iii) OSWireless Deployment and Evaluation. We demonstrate and evaluate OSWireless by deploying it over NeXT, a newly developed software-defined network emulation and experimentation testbed. OSWireless is deployed over the edge server of the NeXT system to control two programmable networks. We showcase the automated control program generation capability of OSWireless and the optimality of the resulting programs considering a variety of example network control problems. The source code of OSWireless is available in our lab GitHub page [27].

The remainder of the paper is organized as follows. We first discuss the related works in Sec. 2. Then, we describe the overall design objective of OSWireless in Sec. 3. The design of OSWireless kernel (i.e., WiNAS Subplane) and the OaaS Subplane are discussed in Sec. 4 and Sec. 5, respectively. The testbed development and experimental evaluation are respectively presented in Secs. 6 and 7, respectively. Finally, we discuss the learned lessons in Sec. 8 and draw the main conclusions in Sec. 9.[4]

2. Related Work

2.1. Software Defined Networking

There are a few SDN architectures for wireless networks in existing literature. For example, in SoftRAN [14], to alleviate the traffic load pressure of the backhaul network, the centralized controller of SoftRAN makes only those decisions that have network-level influence, while pushing latency-sensitive local decisions into remote radio head controllers. CellSDN [10] and MCC-SDWN [19] share the similar *split-design-control-plane* principles. Recently O-RAN has emerged to support interoperation between vendors' equipment by providing industry-wide standards

[4]The key differences and improvements compared to the conference version [1] are as follows. We have strengthened the motivation of this work by discussing more related works in Sec. 2. Additionally, we have provided the detailed algorithm (Algorithm 1) used for the tree representation of expressions in Sec. 4.2. We have also provided examples of expression and script models for expression definition in Sec. 4.3. In Sec. 6, we introduced a Python-based simulator called UBSim and described its major components. In Sec. 7.1, we have provided an example of network control problem definition and provided a list of key view-behavior elements and APIs available for custom network control problem definition. We have also strengthened the experimental evaluation of OSWireless by discussing three more network control problems. Moreover, in Sec. 8 we have outlined the limitations and future work of OSWireless.

for RAN interfaces and to enhance the RAN performance through virtualized network elements and integrated intelligence in RAN [28, 29]. An automated permission generation and verification system called VOGUE [30] is developed to automatically generate flexible access control mechanisms which can be used across different SDN controllers. The authors of [31] presented an intent-based QoS-aware and SDN-enabled slicing implementation which enables customized specifications and establishment of network slices toward flexible delivery of vertical services. In [32], the authors design an orchestrator that provides high-level interfaces in order to transparently deploy modular control and data plane applications for SDN networks. Readers are referred to [33] for an excellent survey of the main results in this field. Different from these works, which primarily focus on softwarization of cellular networks, in this work we focus on the automated generation of distributed optimization programs in future heterogeneous wireless networks.

2.2. Software-defined Radio Testbeds

There have also been significant efforts made by the wireless community to build open, shared experimental facilities. Towards this goal National Science Foundation (NSF) developed four city-scale shared experimentation platforms for advanced wireless research through Platforms for Advanced Wireless Research (PAWR) program [34]. There are currently three active PAWR platforms namely POWDER [35], a Platform for Open Wireless Data-driven Experimental Research, COSMOS [36], Cloud enhanced Open Software defined Mobile wireless testbed for city-scale deployment (COSMOS), and AERPAW [37], Aerial Experimentation Research Platform for Advanced Wireless. The fourth platform ARA [35], a Living Lab for Smart and Connected Rural Communities was just recently launched.

2.3. Network Automation

Network automation has also attracted significant research attention [38–49]. For example, in [38] the authors develop a human-intervention-in-the-loop quasi-automated model calibration scheme for power budget control and site selection in cellular networks. In [43] Harte et al. design a tool called "THAWS" to speed-up the development and deployment of heterogeneous wireless sensor networks (WSN). The authors of [44] design a toolflow that can help monitor the correctness of the implementation throughout the development of WSN. The authors of [49] propose an autonomous control framework for software-defined swarm UAV networks. *We are not aware of any existing frameworks that can provide open flexible APIs for hiding the specification complexity in software-defined wireless networks.*

The work most related to OSWireless is the open-source project WNOS [50, 51], where we designed an optimization based wireless network operating system for principled software-defined wireless networking [50, 51]. WNOS shares the same design goal with OSWireless, i.e., providing cross-layer distribution optimization as a service in NextG networks. However, the major limitation with WNOS is that the automated decomposition of centralized network control programs is enabled by a technique called Disciplined Instantiation (DI), which can support only limited set of network control applications [49, 52] because of the lack of flexibility in the instantiation of abstract network control problems. In OSWireless we aim to provide a new control plane that is not based on DI and hence can support more sophisticated and practical network control applications.

3. Overall Design Objective

The primary goal of OSWireless is to enable intent-driven wireless networking by hiding the specification complexity from network engineers. Given a control intent (*what to do*, e.g., maximize the network spectral efficiency or minimize the end-to-end delay), OSWireless aims to determine in an automated manner the optimal or at least a desirable operational forwarding strategy (*how to do it*, e.g., which nodes transmit with what power in which frequency bands and in which time slots) that can be executed on the distributed physical forwarding substrates. Denote $\boldsymbol{S}_{\text{int}}$ and $\boldsymbol{S}_{\text{fwd}}$ as the intent specification and forwarding specification, respectively. Then the overall design objective of OSWireless can be expressed as

$$f : C(\boldsymbol{S}_{\text{int}}) \longrightarrow \mathcal{D}(\boldsymbol{S}_{\text{fwd}}), \tag{1}$$

where f denotes the functionalities provided by OSWireless, $C(\cdot)$ indicates that $\boldsymbol{S}_{\text{int}}$ is defined in a centralized manner, and $\mathcal{D}(\cdot)$ represents that $\boldsymbol{S}_{\text{fwd}}$ is deployed on distributed forwarding substrates. Here, $\boldsymbol{S}_{\text{fwd}}$ and $\boldsymbol{S}_{\text{int}}$ represent the two

extremes of network abstraction. By defining $\boldsymbol{S}_{\text{fwd}}$ directly, i.e., with little or no abstraction, network engineers are allowed to control all the forwarding details with the most flexibility, but at the cost of high specification complexity. Alternatively, network applications can be developed independently to accomplish different control intents (i.e., $\boldsymbol{S}_{\text{int}}$) with only the highest-level APIs exposed to the network engineers. This can hide most of the specification complexity but enable limited reconfiguration flexibility. A natural question to ask is: *How to abstract software-defined wireless networks to achieve a good tradeoff between low complexity and high flexibility in defining the control specifications?*

Design Approach. In this work, we attempt to answer this question by designing OSWireless following a three-phase approach. The objective of the first phase is to specify the control intent $\boldsymbol{S}_{\text{int}}$ in a centralized manner using the APIs provided by OSWireless; and then construct the mathematical representation corresponding to intent specification $\boldsymbol{S}_{\text{int}}$. Refer to the output of this phase as *Mathematical Specification* and denote it as $\boldsymbol{S}_{\text{math}}$. In the second phase, OSWireless will generate a set of (possibly distributed) numerical solution algorithms to solve $\boldsymbol{S}_{\text{math}}$. Denote the output of this phase as *Algorithmic Specification* $\boldsymbol{S}_{\text{alg}}$. Finally, in the third phase $\boldsymbol{S}_{\text{alg}}$ is executed on the distributed physical substrates to obtain the *Forwarding Specification* $\boldsymbol{S}_{\text{fwd}}$ at network run time. Then, the design objective (1) can be rewritten as

$$f: \mathcal{C}(\boldsymbol{S}_{\text{int}}) \xrightarrow[(i)]{Phase} \mathcal{C}(\boldsymbol{S}_{\text{math}}) \xrightarrow[(ii)]{Phase} \mathcal{D}(\boldsymbol{S}_{\text{alg}}) \xrightarrow[(iii)]{Phase} \mathcal{D}(\boldsymbol{S}_{\text{fwd}}). \tag{2}$$

It is worth pointing out that in Phase (iii) we consider distributed *Algorithmic Specification* $\boldsymbol{S}_{\text{alg}}$ as an example, while the centralized counterpart can be viewed as a special case. As illustrated in Fig. 1, in OSWireless these three phases are accomplished by designing two new subplanes in the control plane, i.e., *WiNAS Subplane* and *OaaS Subplane*, which will be described in Secs. 4 and 5, respectively.

The rationale behind the three-phase design approach is to separate those coupled network control processes, including network modeling, objective formulation as well as distributed algorithm design, and provide the functionalities in each process as a service to the other processes. *This is similar to the TCP/IP protocol stack, which separates the network functionalities into five layers and provides the functionalities of each layer as a service to the other layers.* Differently, in OSWireless we focus on separating the functionalities for modeling and optimizing the programmable protocol stack of software-defined wireless networks.

4. Kernel (WiNAS Subplane) Design

The WiNAS subplane is the kernel of OSWireless. Its core functionalities are two-fold: first, construct the corresponding mathematical specification $\boldsymbol{S}_{\text{math}}$ given a user-defined control intent specification $\boldsymbol{S}_{\text{int}}$, i.e., Phase (*i*) in (2); and second, provide various services based on which the other functionalities of OSWireless are implemented, including the OaaS Subplane and the mathematical, algorithmic and forwarding specifications. In OSWireless, these two functionalities are accomplished by four modules of the WiNAS Subplane, i.e., *MathSpec Module*, *NetTopo Module, QoSPara Module* and *ParaModel Module*. The MathSpec Module is the place where the mathematical specification will be constructed, while the other three modules together provide services for the other components of OSWireless. Next, before describing these modules, we first define the notations.

4.1. Notation Definition

In OSWireless, each network component or parameter is referred to as a *Network Element*. We further define two types of network elements: *View Element* and *Behavior Element*.

Definition 1 (View Element)**.** *A view element is a network element that can be used to characterize the global view of the network. Here the global view refers to the graph representation of the network topology based on vertices (i.e., nodes) and edges (i.e., links), with each vertex or edge associated to a set of hardware and radio resources (e.g., antennas, subchannels). Denote $\mathcal{V}$ as the set of view elements involved in a network.*

Definition 2 (Behavior Element)**.** *A behavior element is a network element that can be used to characterize the behaviors of the view elements in $\mathcal{V}$ in terms of various QoS metrics and based on other view and behavior elements, e.g., noise, capacity and delay. Denote the set of all behavior elements in the network as $\mathcal{B}$. Further denote the subset of behavior elements associated to network element $V_i \in \mathcal{V}$ as $\mathcal{B}(V_i)$.*

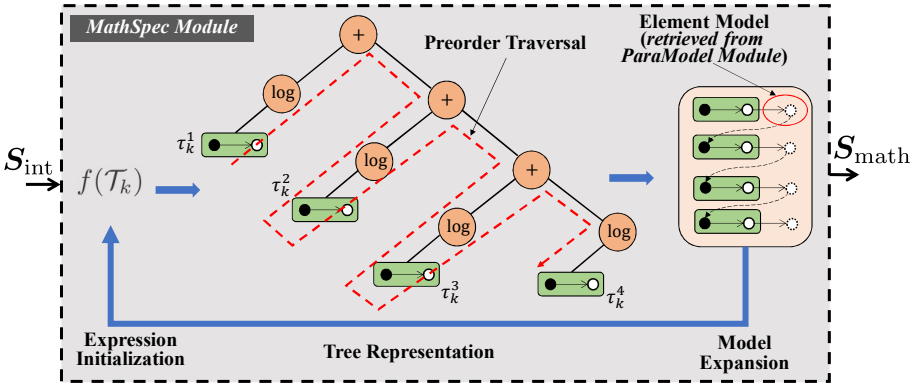


Figure 2: Diagram of *MathSpec Module*. Solid and open dots are respectively, view and behavior elements. Open dots with dotted outlines denote models of behavior elements

A behavior element $B_i \in \mathcal{B}$ is called a *leaf behavior element* if it cannot be represented as a function of other behavior elements in $\mathcal{B}\backslash B_i$, e.g., `noise`, `power`; otherwise, it is called an *intermediate behavior element*, e.g., `capacity`, which is a function of `noise` and `power`. It is worth pointing out that, a behavior element can be either leaf or intermediate but cannot be both simultaneously in an intent specification $\boldsymbol{S}_{\text{int}}$. The type of a specific behavior element depends on the adopted element model. For example, the capacity of a link will be viewed as a leaf element if it is considered as known or can be measured at network run time. The element models are defined in the ParaModel Module such as session delay (`ssdelay`) ; if is defined as an *intermediate behavior element* then the users have an option to choose different models for "`script_str`", "`script_obj`"..

Further define $T_i \triangleq (V_i, B_i)$ as a view-behavior element pair (VBEP) and denote the set of all the possible VBEPs as $\mathcal{T}$. Then, the intent specification $\boldsymbol{S}_{\text{int}}$ and the corresponding mathematical specification $\boldsymbol{S}_{\text{math}}$ can be represented based on a subset of VBEPs in $\mathcal{T}$. The main difference between $\boldsymbol{S}_{\text{int}}$ and $\boldsymbol{S}_{\text{math}}$ is as follows. In $\boldsymbol{S}_{\text{int}}$ the network behaviors are characterized mostly based on intermediate behavior elements without exposing the lower-level details of the coupling among the behavior elements to the network engineers. Differently, $\boldsymbol{S}_{\text{math}}$ characterizes network behaviors by formulating mathematically the corresponding network control problem by exposing all the details of the coupling among the involved behavior elements. The mapping between $\boldsymbol{S}_{\text{int}}$ and $\boldsymbol{S}_{\text{math}}$ is accomplished by the Mathematical Specification (MathSpec) Module as described below.

4.2. Mathematical Specification Module

Given intent specification $\boldsymbol{S}_{\text{int}}$, the MathSpec Module constructs the corresponding mathematical specification $\boldsymbol{S}_{\text{math}}$ following three steps: *Expression Initialization*, *Tree Representation* and *Model Expansion*, as illustrated in Fig. 2.

Each intent specification $\mathcal{S}_{\text{int}}$ consists of a set $\mathcal{K}$ of textual expressions as expressed in(3). Each expression defines either the utility or a constraint of the target network control problem, by referencing to a subset of VBEPs using the APIs provided by the NetTopo Module such as `getCompExpr` which is used to retrieve operators and operands in $\mathcal{K}$; `checkVBEP` determines whether a textual operand is a VBEP or not. Denote the set of VBEPs referenced in expression $k \in \mathcal{K}$ as $\mathcal{T}_k \triangleq \{(T_k^i)_{i=1}^{I_k}\}$ with $T_k^i \triangleq (V_k^i, B_k^i)$ and I_k being the number of VBEPs in $\mathcal{T}_k$. Then the objective of the first step (i.e., expression initialization) is to construct an initial mathematical representation (denoted as $f(\mathcal{T}_k)$) of the intent specification $\boldsymbol{S}_{\text{int}}$. Figure 2 shows a toy example of $f(\mathcal{T}_k)$ with four VBEPs. If we consider VBEP (`link`, `rate`) for $T_k^i \in \mathcal{T}_k$, where `link` and `rate` are respectively view and behavior elements, then the initial expression $f(\mathcal{T}_k)$ can be represented in this toy example as

$$\begin{aligned} f(\mathcal{T}_k) &= \log(T_k^1) + \log(T_k^2) + \log(T_k^3) + \log(T_k^4) \\ &= \log(\texttt{link1.rate}) + \log(\texttt{link2.rate}) \\ &\quad + \log(\texttt{link3.rate}) + \log(\texttt{link4.rate}). \end{aligned} \tag{3}$$

Algorithm 1: Tree representation of expression f

Data: The initial set of expressions $\mathcal{F}' = \emptyset$, $\mathcal{F} = \{f\}$, the initial VBEP set $\mathcal{T} = \emptyset$
Result: Updated set $\mathcal{T}$ of VBEPs contained in expression f

```
1  while F' ≠ F do
2      Update F': F' ← F
3      for Each expression g ∈ F do
4          Update F : F ← F\{g}
5          (Operator g_0, operands g_1, g_2) = getCompExpr(g)
6          Update F : F ← F ∪ {g_1, g_2}
7          for Each operand g' ∈ {g_1, g_2} do
8              if checkVBEP(g') is True then
9                  Update VBEP set T : T ← T ∪ {g'}
10             end
11         end
12     end
13 end
```

This defines the sum-log-rate of the four links, a widely adopted utility function with `log` introducing proportional fairness among the links. In OSWireless, this step is accomplished by replacing the textual API expressions with the corresponding VBEPs. This is accomplished by using the `get_expr_new` API. For example, `get_expr_new("link", "capacity")` returns the corresponding VBEP `"link_capacity"`.

With the initial expression $f(\mathcal{T}_k)$, the MathSpec Module will then construct iteratively an expanded expression to characterize the mathematical coupling among the involved behavior elements. To this end, the MathSpec Module needs to identify the set of VBEPs contained in $f(\mathcal{T}_k)$ in each iteration as the model expansion progresses. In OSWireless, this is accomplished by converting the textual expression $f(\mathcal{T}_k)$ into a binary tree in each iteration. As illustrated in Fig. 2, each leaf node of the resulting tree is a VBEP and the other nodes are mathematical operators. To this end, preorder traversal is adopted to identify VBEPs. For example, given the initial expression of $f(\mathcal{T}_k)$ in (3), the MathSpec Module will construct in the first iteration a binary tree with three nodes: operator " + ", leaf nodes $\log(T_1)$ and $\log(T_2) + \log(T_3) + \log(T_4)$. Since $\log(T_i)$ is not a VBEP, the MathSpec Module further represents $\log(T_i)$ as a tree with operator log and leaf node T_1 (which is a VBEP). The tree representation stops when all the leaf nodes are VBEPs. The process of tree representation is summarized in Algorithm 1, where the job of `getCompExpr(·)` in line 5 is to retrieve the operator and operands contained in a textual expression; `checkVBEP(·)` in line 8 determines if a textual operand is a VBEP. Both of the two functionalities are accomplished based on the kernel services provided by the other three modules of the WiNAS Subplane, as described below.

Finally, each of the identified VBEPs is substituted with its corresponding model defined in the ParaModel Module. The updated $f(\mathcal{T}_k)$ will then be converted to a new binary tree. This procedure will be repeated until all the behavior elements (see Definition 2) in the expression are leaf behavior elements.

4.3. Kernel Service Provision

Four types of services are provided by the OSWireless kernel. These are i) definitions of the view and behavior elements, ii) definitions of the association of behavior elements to view elements, iii) modeling of the behavior elements, and iv) operations of the view and behavior elements, such as `getCompExpr(·)` and `checkVBEP(·)` in Algorithm 1. These services are accomplished in three modules, i.e., *NetTopo Module*, *QoSPara Module* and *ParaModel Module*, as illustrated in Fig. 3.

The objective of the NetTopo Module is to enable flexible definition of different types of view and behavior elements, characterize the coupling among the defined network elements and further associate them to the corresponding mathematical models. This module also provides two types of control interfaces, i.e., internal interfaces among the different modules of the WiNAS Subplane and external interfaces with the OaaS Subplane. To this end, the NetTopo Module manages a set of view records each corresponding to a view element in $\mathcal{V}$ referenced by intent specification $\boldsymbol{S}_{\text{int}}$. Denote the set of the records as $\mathcal{R}_{\text{v}} = \{\text{R}_{\text{v}}^n \mid n = 1, 2, \cdots, N_{\text{v}}\}$, where R_{v}^n represents the nth view record and N_{v} is the total number of records. As illustrated in the top block of Fig. 3, each record R_{v}^n is a variable-length

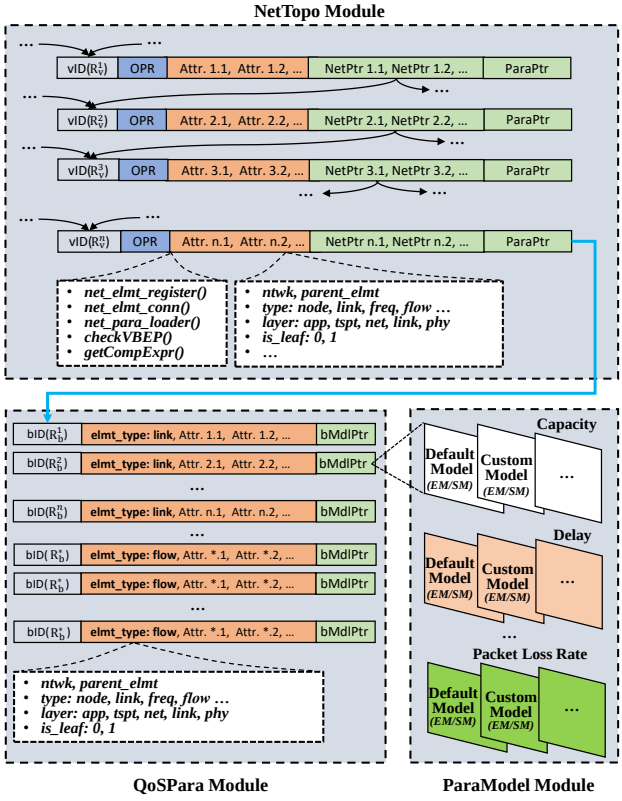


Figure 3: Diagram of NetTopo, QoSPara and ParaModel modules. These three modules together provide various kernel services for the other modules of OSWireless.

tuple, consisting of five categories of fields. These are $\texttt{vID}(\mathrm{R}_\mathrm{v}^n)$, which is the OSWireless-wide unique ID of view element R_v^n; $\texttt{vAttrList}(\mathrm{R}_\mathrm{v}^n)$, which comprises the list of attributes defined for view element R_v^n, such as the name of the element (`vElmtName`), the parent view element that R_v^n is associated with (`vPrntElmt`), whether R_v^n is a set or an individual element (`vElmtType`), among others; field $\texttt{vNetPtr}(\mathrm{R}_\mathrm{v}^n)$ consists of a set of pointers, each pointing to another associated view element $\mathcal{R}_\mathrm{v}' \in \mathcal{V}$, e.g., the individual elements that form a set element; Finally, $\texttt{vBPtr}(\mathrm{R}_\mathrm{v}^n)$, i.e., field ParaPtr in Fig. 3, maintains the set of behavior elements attached to R_v^n and $\texttt{vOprPtr}(\mathrm{R}_\mathrm{v}^n)$ defines the set of the operations supported by R_v^n. It is worth mentioning that these operations are designed to enable automated expression initialization and model expansion in the MathSpec Module (e.g., `checkVBEP(.)`) and associate the behavior elements defined in the QoSPara Module to the corresponding behavior element (e.g., `vBElmtLoader(.)`).

Similar to NetTopo Module, the QoSPara Module maintains the set of behavior elements $\mathcal{B}$. Similar to the view records, as illustrated in Fig. 3, each behavior record also consists of an OSWireless-wide unique ID $\texttt{bID}(R_\mathrm{b}^n)$ and an attribute list $\texttt{bAttrList}(R_\mathrm{b}^n)$. Examples of these behavior attributes include $\texttt{bLayer}(R_\mathrm{b}^n)$, which provides the protocol layer information of the behavior elements; and $\texttt{bHID}(R_\mathrm{b}^n)$, which defines the *node* view element that behavior element R_b^n is associated to. This information will be provided to the OaaS Subplane for automated distributed problem decomposition in Sec. 5.

Finally, each behavior record $R_\mathrm{b}^n \in \mathcal{B}$ also has a `bMdlPtr` field, i.e., MdlPtr in Fig. 3, which is a pointer pointing to the mathematical model of the behavior element. All the models available to a behavior element are defined in the ParaModel Module. For each behavior element, two types of models have been defined in OSWireless: i) *Expression Model (EM)*, which models a behavior element using a closed-form mathematical expression; and ii) *Script Model (SM)*, which models a behavior element using a script. Examples for expression model and script model are given in Fig. 4. Compared to expression models, script models can provide more flexibility in characterizing the coupling among different behavior elements and hence are more suitable for defining more sophisticated network control problems. In Fig. 4, `list_str_parameter` denotes the list of dependent behavior elements to define a new behavior

```
##################################
## Expression Model - Link Capacity
##################################
lkcap_default = 'log(1+lksinr)/log(2)'

##################################
## Script Model  - Session Delay
## Defined using BHV element name
##################################
def __script__str__(obj, list_str_parameter):
    str_parameter1 = list_str_parameter[0]
    str_parameter2 = list_str_parameter[1]
    expr = '1/('+str_parameter1+'-('+str_parameter2+'))'
    return expr

##################################
## Script Model  - Session Delay
## Defined using BHV element object
##################################
def __script__obj__(obj, list_obj_parameter):
    obj_name_parameter1 = list_obj_parameter[0].name
    obj_name_parameter2 = list_obj_parameter[1].name
    expr = '1/('+obj_name_parameter1+'-('+obj_name_parameter2+'))'
    return expr
```

Link Capacity

Session Rate

Figure 4: Examples of Expression Model and Script Model.

element. Let us consider the following example, `nt.install_model(net_name_g2.ssdelay, 'script__str', ['lkcap', 'lkinrate'])`. The format for all the built-in models will be made available to users by OSWireless developers and the format for all third-party models integrated into OSWireless will be made available from the corresponding model developers. Additionally, from the figure it can be seen that script models are more general and therefore can be easily used for defining new hierarchical script models by combining multiple low-level script models. It is worthy pointing out that a set of script models have been developed for OSWireless, based on which users can develop their own models to characterize the behavior element for custom designed network control problems.

5. OaaS Subplane Design

With the mathematical specification $\boldsymbol{S}_{\text{math}}$ obtained by the WiNAS Subplane in Sec. 4, the OaaS Subplane will construct in an automated manner a set of numerical solution algorithms (i.e., the algorithmic specification as illustrated in Fig. 1) that can be executed on distributed forwarding substrates to solve $\boldsymbol{S}_{\text{math}}$. Since the mathematical specification $\boldsymbol{S}_{\text{math}}$ is defined centrally and possibly across multiple protocol layers, we need to convert it into distributed algorithm specification $\boldsymbol{S}_{\text{alg}}$. This is done following a two-step approach. First, we decompose $\boldsymbol{S}_{\text{math}}$ into distributed specifications using principles of dual and indirect decomposition. Second, we design custom algorithm templates with placeholders that are automatically updated based on the algorithm type and the variables in the decomposed $\boldsymbol{S}_{\text{math}}$. To this end, we design the OaaS Subplane for decomposition leveraging the kernel services provided by the WiNAS Subplane. OaaS Subplane's architecture is illustrated in Fig. 5, which consists of two major components, i.e., *Decomposition Engine* and *AlgoGen Engine*.

5.1. Decomposition Engine

Simply speaking, a given textual expression f can be decomposed in two steps: i) break down f into a set of component expressions connected with operator "+"[5]; and ii) assign the component expressions into different groups corresponding to different subproblems. The first step can be accomplished based on the tree representation service provided by the WiNAS Subplane as described in Sec. 4.2. Denote the resulting set of component expressions as

[5]Operator "−" is considered as part of the component expression. For example, expression $x - y$ will be reformulated as $x + (-1) * y$ before the tree representation.

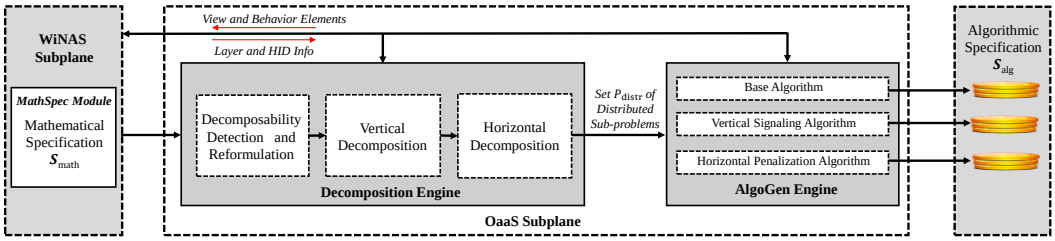


Figure 5: Overall architecture of Optimization-as-a-Service (OaaS) Subplane.

$\mathcal{K}_{\text{comp}}(f)$. In the second step, the component assignment is accomplished through two types of decomposition: vertical and horizontal decomposition, based on the protocol layer and horizontal index (HID) information. These information can also be obtained based on the kernel services provided by the WiNAS Subplane. The overall decomposition architecture is illustrated in Fig. 5.

The objective of vertical decomposition is to uncouple the coupling among the protocol layers involved in mathematical specification $\boldsymbol{S}_{\text{math}}$. To this end, for each component expression $k_{\text{comp}} \in \mathcal{K}_{\text{comp}}(f)$ its associated layer is obtained using WiNAS Subplane API $\texttt{bLayer}(R_{\text{b}}(k_{\text{comp}}))$, where $R_{\text{b}}(k_{\text{comp}})$ represents the VBEPs contained in component expression k_{comp}. For example, for VBEP `(ses, rate)`, i.e., the transport-layer transmission rate of a session, the corresponding component expression will be assigned to the transport-layer subproblem. Please refer to Sec. 4.1 for the definition of VBEP and Sec. 4.3 for the field structure of element records. Denote $\mathcal{L}$ as the set of subproblems obtained through vertical decomposition, with each subproblem involving one protocol layer.

Based on horizontal decomposition, each subproblem in $\mathcal{L}$ is further decomposed into a set of subproblems each associated to a single node. To this end, we first introduce the notation of HID in the following definition.

Definition 3 (Horizontal Index (HID))**.** *An HID is an identifier that can be used to identify the view network element where another behavior or view element should be registered and managed at network run time.*

Each behavior element is associated with a network view element as its HID when the behavior element is invoked for the first time during the construction of the mathematical specification $\boldsymbol{S}_{\text{math}}$. By default, the HID will be the parent view element (i.e., `parent_elmt` in QoSPara Module of Fig. 3) of the behavior element. For example, the HID of behavior element `link_capacity` will be its parent view element `link`; the HID of a `link` will be the `source_node` of the link. In horizontal decomposition, to assign a component expression $k_{\text{comp}} \in \mathcal{K}_{\text{comp}}(f)$ to a distributed problem, we only need to get its HID using WiNAS Subplane API $\texttt{bHID}(k_{\text{comp}})$ and further get the HID of $\texttt{bHID}(k_{\text{comp}})$. This process is repeated until the HID of the view element is itself and then component expression k_{comp} will be assigned to the corresponding subproblem.

It is worth pointing out that in the above decomposition process it has been assumed that expression f is decomposable. However, this is not always the case in wireless network modeling and optimization because of the coupled control variables at different protocol layers and different nodes. To address this challenge, as illustrated in Fig. 5, a *Decomposability Detection and Reformulation* module has been designed in the OaaS Subplane.

Decomposability Detection and Reformulation. In OSWireless, we determine the decomposability of an expression f by determining the decomposability of its component tree. As mentioned above, expression f can be represented as a tree with a set $\mathcal{K}_{\text{comp}}(f)$ of component expressions connected with operator "+". Then the decomposability of f depends on the decomposability of each branch of the tree. This can be simply accomplished based on the layer checking service provided by $\texttt{bLayer}(R_{\text{b}}(k_{\text{comp}}))$ and HID checking service by $\texttt{bHID}(R_{\text{b}}(k_{\text{comp}}))$. Take vertical decomposition as an example. $\texttt{bLayer}(\cdot)$ first detects the behavior elements in k_{comp} and then retrieves all the corresponding protocol layers. If different protocol layers are identified, it means the component expression is nondecomposable and cannot be assigned to a subproblem corresponding to any specific protocol layer. *How can we still decompose the mathematical specification $\boldsymbol{S}_{\text{math}}$ in this case and in an automated manner?*

Notice that different decomposition theories can be used to tackle the decomposability problem, such as dual decomposition, primal decomposition, hybrid decomposition and hierarchical decomposition as well as indirect decomposition. *However, it is by no means easy to automate the decomposition based on these theories because they all require rather complicated expertise-based analysis and reformulation of the specification expressions.* In this work,

by designing the OaaS Subplane we hope to provide the first-of-its-kind framework for automating the decomposition of centralized network control problems based on certain decomposition theories.

Next we describe the design logic of the OaaS Subplane considering indirect decomposition and dual decomposition as examples, while the design can also be extended to other decomposition theories. Each specification (mathematical) expression can be decomposed using different techniques such as dual decomposition, primal decomposition, hybrid decomposition, among others, following their own principles. In addition to these techniques, a specification expression can also be decomposed based on indirect decomposition by introducing auxiliary variables to i) uncouple the coupling among the variables in a nondecomposable expression and ii) coordinate the optimization of the resulting subproblems. Consider a toy example of the queuing delay model. If the M/M/1 model is considered, the average response time can be expressed as a function of the link capacity and the source rate, i.e., $\frac{1}{\texttt{link\_capacity}-\texttt{src\_rate}}$. Here, the model is defined for `link_capacity` > `src_rate`, otherwise the average delay hence the response time will become infinity. This expression is not directly decomposable since `link_capacity` and `src_rate` operate at different protocol layers (physical and transport, respectively), and the expression will be viewed as a single branch in the tree representation process. Based on indirect decomposition, the auxiliary variable $\texttt{v}_{\text{aux}}$ will be introduced along with a decomposable equality constraint $\texttt{v}_{\text{aux}}$ == `link_capacity` − `src_rate`. Another example of an auxiliary variable is the Lagrangian coefficient, which can be introduced in dual decomposition to absorb the constraints into the utility function. Technically speaking, an auxiliary variable can be assigned arbitrarily to one of the involved protocol layers in the case of a tie. In OSWireless, the rules for auxiliary variable introduction and their assignment to different protocol layers are defined in the Decomposition Detection and Reformulation Module (i.e., this module) and in the behavior element models in ParaModel Module (Fig. 3).

5.2. AlgoGen Engine

Recall from (2) that the output of Phase (ii) is the distributed algorithmic specification $\mathbf{S}_{\text{alg}}$. In the OaaS Subplane, this is accomplished by further generating automatically numerical solution algorithms to solve each of the subproblems obtained above. As shown in Fig. 5, three types of algorithms will be generated, namely the *base optimization algorithm*, the *vertical signaling algorithm* and the *horizontal penalization algorithm*.

A set of numerical algorithm templates have been designed in the AlgoGen Engine for base algorithm construction. Each template defines the formats for the optimization variables, the signaling parameters and the penalization terms. To this end, we define the algorithm templates based on CogApp, an open-source content generator for executing Python snippets in source files [53]. To construct the base algorithm, we first detect the optimization variables present in each of the distributed problems obtained in Sec. 5.1 and substitute the detected variables into the algorithm template. The optimization variables are specified in intent specification $\mathbf{S}_{\text{int}}$, and the corresponding information is stored in the behavior element in the QoSPara Module of the WiNAS Subplane and can be retrieved using WiNAS API `bVar`$(\cdot)$ when constructing the base algorithm. Both scalar (i.e., $|\mathcal{V}_{\text{opt}}| = 1$) and vector (i.e., $|\mathcal{V}_{\text{opt}}| > 1$) variables can be supported by the AlgoGen Engine, where $\mathcal{V}_{\text{opt}}$ denotes the variables detected in the subproblem. The vertical signaling parameters can be detected and substituted into the algorithm template similarly. For example, if dual decomposition is considered, the vertical signaling parameters are the Lagrangian coefficients introduced when constructing the dual expression of the original expression. These parameters can be updated at network run time by a projected linear operation derived based on the corresponding constraint expression and then exchanged among the corresponding subproblems.

The base algorithms are designed to optimize a modified version of the original utility in each of the generated distributed subproblems. To this end, a penalization term is incorporated to the utility function to prevent a distributed node from hurting other nodes too much. The automated generation of the penalization terms is accomplished by converting the textual expression into the symbolic domain.

6. OSWireless Prototyping

Recall from Sec. 3 that the Algorithmic Specification $\mathbf{S}_{\text{alg}}$ generated by the OaaS Subplane will be executed either in OSWireless in the case of centralized control or the distributed forwarding substrates $\mathbf{S}_{\text{fwd}}$ in the data plane. In this work we consider distributed control as an example while viewing centralized control as a special case with preconfigured routing between different nodes. To this end, we implement OSWireless and deploy it over NeXT [54], a newly designed software-defined testbed for wireless network modeling, optimization and deployment.

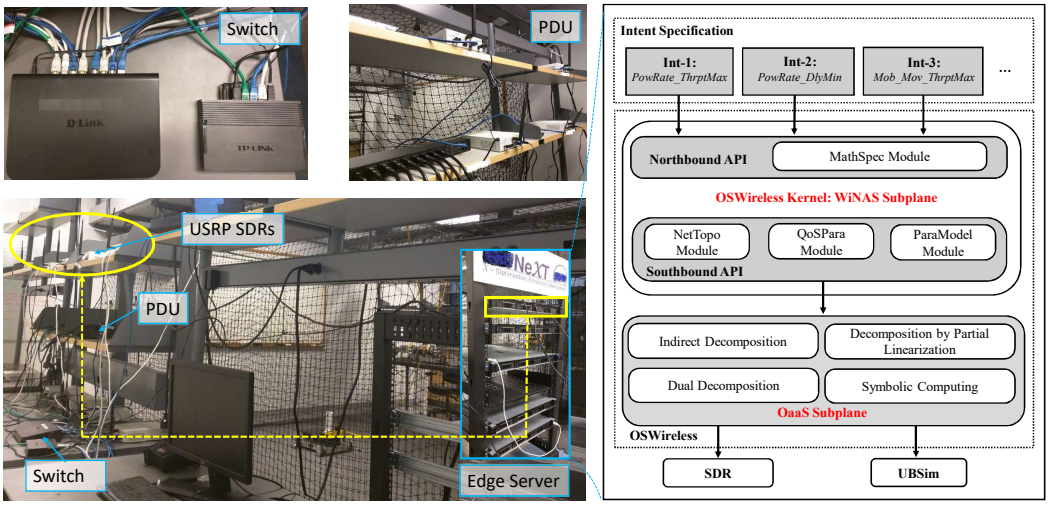


Figure 6: Snapshot of the software-defined testbed and software diagram for OSWireless prototyping.

A snapshot of the testbed is shown in Fig. 6. There are three major components in the NeXT testbed: edge server, front-end SDR, and programmable protocol stack. The edge server consists of five Dell EMC R340 poweredge workstations, each with Intel Xeon E-2246G 3.6 GHz CPU and Ubuntu v18.04. The front-end SDR consists of 20 N210 USRPs and is powered by three CyberPower PDU41001 Switched Power Distribution Units (PDU). The PDUs have been deployed to enable remote experimentation over the testbed during the COVID-19 pandemic and further share the testbed with the community in the future[6]. The edge servers and the front-end SDRs are connected using two switches with Gigabit Ethernet cables.

A programmable protocol stack has been developed operating over the software radio forwarding substrates. Each node can be programmed to transmit with Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK), Gaussian Minimum Shift keying (GMSK), or other modulation schemes, based on Reed Solomon Code for forward error correction with coding rate ranging from 0.14 to 0.30 at step of 0.02, and at transmission power that can be adapted on the fly by controlling the transmit gain of the Field Programmable Gate Arrays (FPGA) of the USRP SDRs. The packet length is set to 1000 bytes and the retransmission limit is set to 10 at each link; the lower and higher packet error rate thresholds are set respectively as 0.05 and 0.15 for triggering the coding rate reconfiguration. TCP Vegas has been implemented for transport-layer congestion control. The available spectrum band (2-2.6 GHz) is divided into three subchannels and shared by different nodes with configurable spectrum reuse factor.

In addition to the software-defined data plane as discussed above, the OaaS Subplane is also interfaced with *UBSim* (previously called SimBAG [55], [56]), as illustrated in Fig. 6. UBSim is a Python-based **U**niversal **B**roadband **Sim**ulator for event-driven broadband integrated aerial-ground wireless networks. UBSim comprises of four modules: *Network Configuration Module (NCM)*, *Network Element Module (NEM)*, *Discrete Event Driver (DED)*, and *Algorithm Repository Module (ARM)*. The NCM module provides the APIs for configuring various virtual network elements, such as the number of nodes, frequency band, bandwidth. The NEM module defines the classes for all the network elements in a hierarchical manner. The DED module provides the discrete network simulation environment based on the open-source library SimPy [57]. Finally, the ARM module is the place where the algorithms automatically generated by the OaaS Subplane are deployed and executed at network run time.

Finally, we develop OSWireless and deploy it on one of the workstations as indicated by the yellow rectangle in Fig. 6, including the WiNAS Subplane (Sec. 4) and the OaaS Subplane (Sec. 5). For the SDR data plane, the other workstations serve as the controlling hosts of the front-end SDRs for baseband signal processing and run the distributed control programs automatically generated by OSWireless to adapt the transmission parameters at different

[6]Currently we are extending the NeXT testbed to enable experiments for integrated aerial-ground wireless networks and make the testbed publicly accessible by interfacing it with a public cloud Amazon Web Service (AWS).

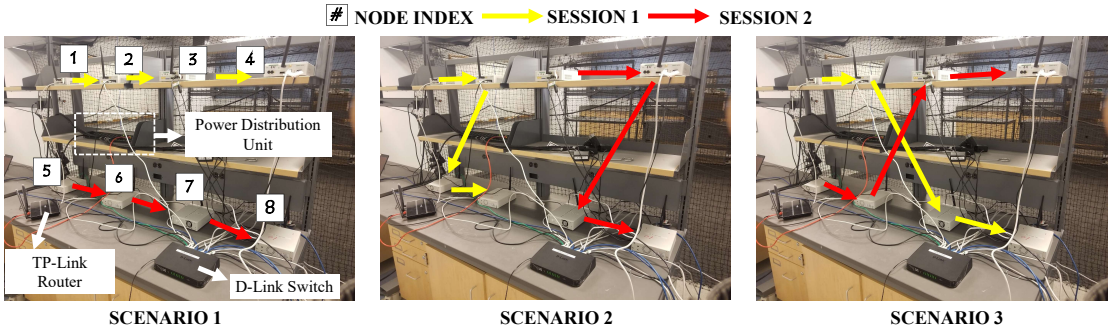


Figure 7: Network topology for OSWireless testing over NeXT testbed.

layers of the programmable protocol stack. For the UBSim data plane, UBSim shares the same server with OSWireless. Based on the kernel APIs provided the WiNAS Subplane, we prototype the OaaS Subplane by automating a set of widely adopted decomposition techniques for demonstration purpose, including dual decomposition, indirect decomposition and decomposition by partial linearization, while OSWireless can also be extended to incorporate other decomposition techniques. The symbolic mathematical operations required by the decomposition automation has been based on open-source symbolic computing library SymPy [58]. Based on the OSWireless prototype, next we conduct a series of experiments to test the effectiveness and flexibility of OSWireless in hiding the specification complexity for software-defined wireless networks.

7. Experimental Evaluation

In this section we aim to i) showcase how OSWireless can be used to hide the specification complexity by converting automatically a given control intent (*what to do*) to operational control programs (*how to do it*); and ii) understand how effective the automatically generated control programs are at network run time. To this end we conduct a series of experiments over the NeXT testbed considering network control problems at different protocol layers. OSWireless is designed to support cross-layer optimization of different wireless networks with ad-hoc or infrastructure-based architectures. In this paper, as a proof of concept, we have considered an ad-hoc wireless network topology. However, we expect that OSWireless can be deployed for any architecture.

7.1. Experiment 1: MSMH_PowRate_ThrptMax

In the first experiment, we consider network scenarios of device-to-device (D2D) communications, an important technique that can enable a wide set of new applications in NextG and IoT networks, such as sensor and actuator networks [59, 60] , vehicular networks [61, 62], and wireless backhaul networks [63]. In this work, we consider three scenarios as shown in Fig. 7. In each scenario, two source nodes communicate with their destination nodes via a set of relay nodes. The available subchannels are assigned to the nodes with spectrum reuse factor of 1/2. The control objective in this scenario is to maximize the sum end-to-end throughput of the two sessions by jointly controlling the transmission power of the nodes at the physical layer and source rate at the transport layer, subject to proportional fairness between the two sessions. We refer to this scenario as *MSMH_PowRate_ThrptMax*, where the first field (MSMH) represents the multi-session multi-hop network topology, the second field indicates the control variables, and the last field is the control objective. The same naming convention will be adopted for the other scenarios for easier source code sharing in the future. Next we first showcase how to define the intent specification based on the kernel APIs provided by OSWireless.

Figure 8(a) shows the main body of the intent specification for control problem *MSMH_PowRate_ThrptMax* written in Python 3.0, i.e., *what to do* described using the high-level APIs provided by the OSWireless kernel WiNAS Subplane. As shown in Fig. 8(a) line 5 represent the initialization of a blank network control problem. Furthermore. it can be seen that the network behavior description, including the specification of the network utility, constraints and control variables, can be accomplished by referring to three types of behavior elements and their associated view elements using the APIs. These behavior elements are `ssrate` (source rate of a session), `lkcap` (link capacity) and

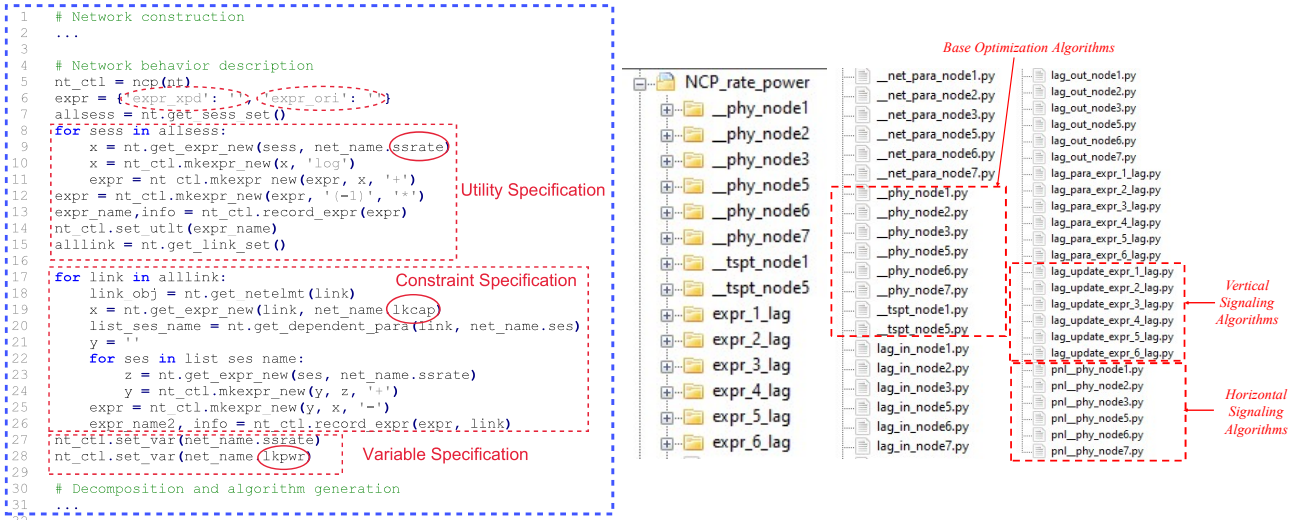


Figure 8: Example network control problem MSMH_PowRate_ThrptMax. (a) *What to do:* Intent specification written in Python 3.0. This is specified by network engineer in a centralized manner using OSWireless APIs and executed on the edge server running OSWireless (the yellow rectangle in Fig. 6); (b) *How to do it:* Automatically generated distributed control programs for different Software-Defined Radio (SDR) nodes at physical (starting with "__phy") and transport (starting with "__tspt") layers, Lagrangian coefficient updating algorithms (starting with "lag"), and penalization algorithms (starting with "pnl"). Node indices are indicated in the file names and the corresponding deployment is shown in Scenario 1 in Fig. 7. No control programs are generated for nodes 4 and 8 because they are destination nodes.

`lkpwr` (transmission power for a link), as indicated by the solid ellipses in Fig. 8(a) (lines 9, 19 and 28). Notice that there is no need to specify explicitly the low-level coupling among these view elements, e.g., the mathematical model of `lkcap` as a function of `lkpwr`, or their association to different protocol layers. Instead, this coupling will be integrated to the initial (original) expression defined by the intent specification (i.e., `expr_ori` in line 6) through the expression expansion operation in the MathSpec Module (see Fig. 2). The resulting expanded expression is recorded in `expr_xpd` (Fig. 8(a), line 6). In lines 13 and 19, we get the expression for the behavior element `ssrate` and `lkcap` using `get_expr_new` API. We then add the necessary operands and operators to construct the expressions as shown in lines 10, 11, 23 and 25. We then record the expression using `record_expr` API (lines 13 and 26). A unique name is assigned to each expression automatically by OSWireless and does not require any input from the user. In line 14, we set the utility function using `set_utlt` API (line 14). All other expressions recorded will be automatically considered as constraints. Finally, we set the optimization variables `ssrate` and `lkpwr` using `set_var` API (lines 27 and 28) which corresponds to network control problem (MSMH_PowRate_ThrptMax) of maximizing throughput by optimizing link power (`lkpwr`) and session rate `ssrate`. After specifying utility and constraints, the operational distributed optimization algorithms can be automatically generated by simply calling the following three lines of code:

1. `vdcp = xlydcp.ncp_xlayer_decomp(nt_ctl)`,
2. `hdcp = dstdcp.ncp_dist_decomp(vdcp)`,
3. `ag.alg_gen(hdcp)`,

where `nt_ctl` (line 1) stores the centralized intent specification; `vdcp` and `hdcp` are the specifications after vertical (line 1) and horizontal (line 2) decomposition, respectively; line 3 generates the operational optimization algorithms. The list of available key view elements and its corresponding behavior elements are reported in Table. 1. All the key high-level APIs exposed to users are presented in Table. 2.

It is worth pointing out that the centralized high-level specification in Fig. 8(a) is all that is needed for OSWireless to generate the operational distributed cross-layer control programs for *MSMH_PowRate_ThrptMax*, while all the specification complexity of the mathematical specification $\boldsymbol{S}_{\text{math}}$ and algorithmic specification $\boldsymbol{S}_{\text{alg}}$ will be orchestrated by OSWireless in an automated manner and hidden from the network engineers. The file directory of the automatically

generated algorithm specification $\boldsymbol{S}_{\text{alg}}$ is shown in Fig.8(b) for each node at different protocol layers (physical and transport in this experiment), which consists of base optimization algorithms, Lagrangian updating algorithms for vertical signaling among protocol layers, and horizontal penalization algorithms which forms the forwaring specification $\boldsymbol{S}_{\text{fwd}}$. These programs will be loaded to the control hosts (i.e., the edge server in Fig. 6) and executed to control the distributed USRP SDRs at network run time. As in traditional SDN domain, all the forwarding rules $\boldsymbol{S}_{\text{fwd}}$ and network behavior are defined as tuples of source and destination IP addresses as well as their port numbers.

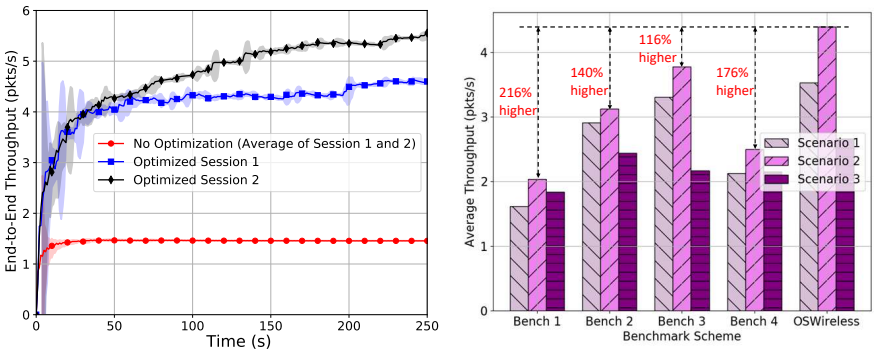
View Element	Behavior Element
Link	Link Power (`lkpwr`)
	Link SINR (`lksinr`)
	Link Distance (`lkdist`)
	Link Gain (`lkgain`)
	Link Interference (`lkitf`)
	Aggregare Link Rate (`lkinrate`)
	Link Capacity (`lkcap`)
	Link Delay (`lkdelay`)
	Link Tx Coordinates (`coord_x`, `coord_y`, `coord_z`)
	Link Rx Coordinates (`fx_coord_x`, `fx_coord_x`, `fx_coord_x`)
Session	Session Rate (`ssrate`)
	Session Delay (`ssdelay`)

Table 1: List of key View and Behavior Elements.

API/Function Name	Description	Arguments
`attach`	Attach network elements	`elmnt_name_str, elmnt_num_int, addi_info_dict`
`connect`	Connect network elements	`new_elmnt_name_str, [elmnt_1_str, elmnt_2_str])`
`install_model`	Install behavior models	`bhv_elmnt_name_str, mdl_name_str, depend_elmnt_list`
`get_expr`	Get expression	`ntwk_bhv_name_str, elmnt_name_str`
`mkexpr`	Make expression	`operand1_dict, operand2_dict operator_str`
`record_expr`	Record expression	`expr_dict, depend_elmt_str`
`set_para`	Set utility, constraint expressions and optimization variables	`para_str_or_list, para_type_str`
`ncp_xlayer_decomp`	Cross-Layer Decomposition	`ncp_obj`
`ncp_dist_decomp`	Distributed Decomposition	`vert_decomp_obj`
`alg_gen`	Algorithm Generation	`dist_decomp_obj`

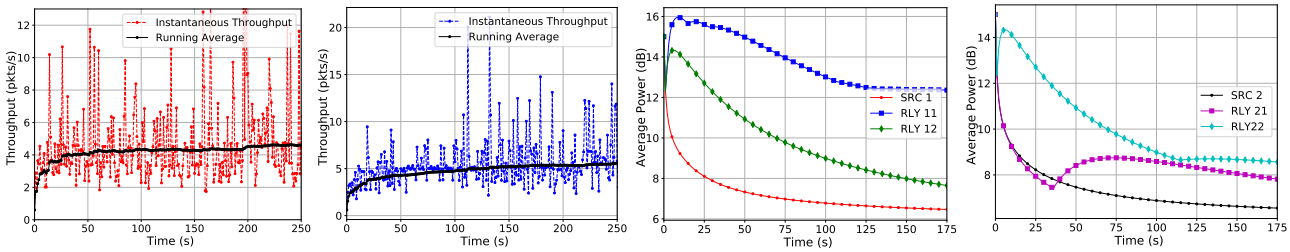
Table 2: List of key high-level API/Functions and its arguments.

Figure 9 shows the end-to-end throughput performance of the control programs automatically generated by OS-Wireless. The instantaneous throughput is reported in Fig. 9 (a) considering Scenario 1 (see Fig. 7) as an example. It can be seen that significant throughput gain (around 300%) can be achieved compared to the benchmark scheme (*Benchmark 1*), based on which the hardware transmit gain of the USRP SDRs is set to 15 dB (the middle of the range) and no rate or power adaptations are considered. Figure 10 reports the corresponding instantaneous throughput and transmission power. Figs. 10 (a) and (b) show the instantaneous throughput of session 1 (red curve) and 2 (blue curve), respectively. The black curve shows the average throughput of the respective sessions. Fig. 10 (c) show the instantaneous transmission power of source (SRC 1) and two relay nodes (RLY 11 and RLY 12) of Session 1. From both the figures, we can see that, the algorithm generated by OSWireless can achieve an optimized the throughput of 4.5 pkts/s for Session 1 and 5 pkts/s for Session 2. Similarly, Fig. 10 (d) show the instantaneous transmission power of source (SRC 2) and two relay nodes (RLY 21 and RLY 22) of Session 2. We can see that each node optimizes its own transmission power such that the overall interference caused to other session is minimized. For example, for both the sessions, the initial transmission power of all the transmitting nodes are set to 15 dB, but with OSWireless optimization algorithm, the transmission powers of SRC 1, RLY11, RLY12 are optimized to ~6.5 dB, ~12.5 dB and ~7.5 dB, respectively. Similar performance can also be observed for Session 2. The average performance is plotted in Fig. 9 (b) for the three scenarios by averaging over 10 experiments. Three more benchmark schemes are considered in



(a) (b)

Figure 9: (a) Single-run running average and (b) multi-run average end-to-end throughput achieved by OSWireless generated control programs for MSMH_PowRate_ThrptMax.



(a) (b) (c) (d)

Figure 10: Instantaneous throughput of (a) Session 1 and (b) Session 2; (c)(d): Instantaneous transmission power resulting from MSMH_PowRate_ThrptMax in Scenario 1.

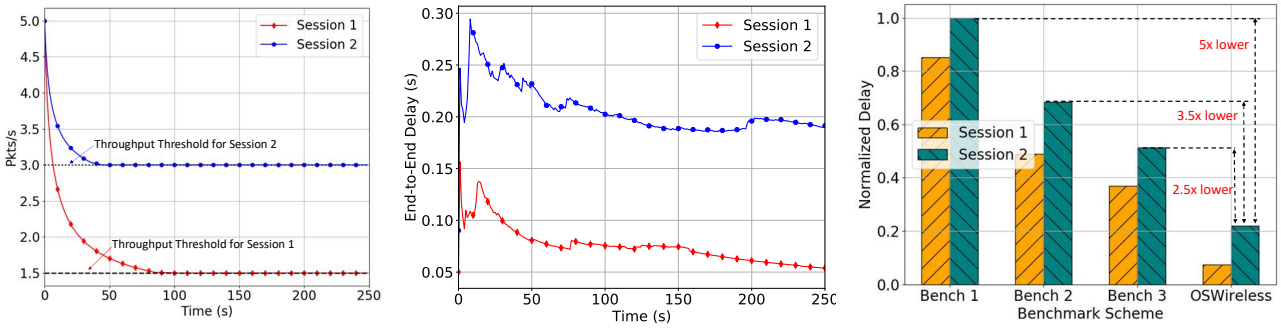
addition to that in Fig. 9 (a). These are *Benchmark 2*: maximum hardware transmit gain for the USRP SDRs (25 dB) and maximum source rate; *Benchmark 3*: fixed transmission power (15 dB) with adaptive source rate; and *Benchmark 4*: power adaptation subject to the target end-to-end throughput. It can be seen that OSWireless outperforms all the four benchmark schemes, which validates the effectiveness of automatically-generated control programs.

7.2. Experiment 2: MSMH_PwrRate_DlyMin

In the above experiment we have showcased through a toy example the operation of OSWireless, including intent specification, automated generation of the corresponding mathematical specification as well as the effectiveness of the resulting distributed control programs. In that example, the control variables `ssrate` and `lkpwr` are only loosely coupled through the network flow conservation constraint for each link (lines 17-26 in Fig. 8(a)) and hence can be decoupled vertically after constructing the corresponding dual problem of the mathematical specification. In the following experiments, we further test the flexibility of OSWireless in control intent definition considering more sophisticated control problems.

In this experiment, referred to as *MSMH_PwrRate_DlyMin*, the control objective is to minimize the average end-to-end delay of concurrent sessions in a multi-session multi-hop network, by jointly controlling the source rate at the transport layer and transmission power of the nodes at the physical layer under minimum end-to-end throughput constraints for each session. The M/M/1 queuing model [64] is considered for each session. In this case, the session rate `ssrate` and link capacity `lkcap` (hence the transmission power `lkpwr`) are closely coupled in the utility function through the queuing model - they both appear in the denominator of the model $\frac{1}{\texttt{lkcap(lkpwr)}-\texttt{ssrate}}$, and the resulting mathematical specification cannot be decomposed directly.

Fortunately, OSWireless can hide most of the specification complexity. Specifically, network engineer only needs to define the control intent by simply calling the model installation API provided by the OSWireless kernel WiNAS Subplane: "`nt.install_model(sess, ssdelay, mm1)`", where `sess`, `ssdelay` and `mm1` are respectively the view



(a) (b) (c)

Figure 11: Instantaneous (a) transmission rate and (b) end-to-end delay; (c) average end-to-end delay.

element, behavior element and the parameter model defined in the NetTopo, QoSPara and ParaModel modules of the WiNAS Subplane, as illustrated in Fig. 3. Then, as described in Sec. 5.1, the resulting mathematical specification will be analyzed for decomposability, reformulated by introducing auxiliary vertical signaling variables to uncouple the coupling between `lkpwr` and `ssrate` in the queuing model, and finally decompose the obtained network control problem based on indirect decomposition. *Notice that all these operations except the intent specification are conducted automatically by OSWireless rather than manually based on traditional network control. Hence, the specification complexity is hidden from the network engineers.*

The delay performance of the obtained algorithmic specification is reported in Fig. 11. In the experiment, we consider low-data-rate latency-critical applications, with the target end-to-end throughput set to 1.5 kbps and 3 kbps for the two sessions, respectively. From Figs. 11 (a) it can be seen that the end-to-end delay can be effectively reduced for both of the two sessions. The corresponding end-to-end delay is plotted in Fig. 11 (b). We can see that the target end-to-end throughput can be assured for them both as the delay is minimized. In Fig. 11 (b) we compare OSWireless to three benchmark schemes: benchmarks 1, 2 and 3 in Experiment 1 but tailored for this experiment. The results are obtained by averaging over 10 experiments. We can see that an average delay of 0.14 s can be achieved by OSWireless, which is 5x, 3.5x and 2.5x lower than the three benchmark schemes, respectively. This experiment further verifies the correctness of the control programs automatically generated by OSWireless and hence its capability and flexibility in hiding the specification complexity for more sophisticated network control intents.

7.3. Experiment 3: MSMH_Mob_Mov_ThrptMax

In this experiment we further test OSWireless by considering mobile scenarios based on emulations on UBSim. Three scenarios (Scenarios 4, 5 and 6) are considered as shown in Fig. 12.[7] In each scenario, the locations of the nodes are randomly initialized. The control objective is to maximize the end-to-end throughput of concurrent sessions by jointly controlling the source rate and the movement of the relay nodes. Notice that the locations of the source and destination nodes are considered to be fixed during the experiments. We refer to this experiment as *MSMH_Mob_Mov_ThrptMax*. The rationale behind considering mobile relay nodes while having fixed source and destination nodes is to mimic a network scenario where a intermediate base station between two ground base stations gets damaged due to natural disaster or due to technical failures. In this case, we need an optimization algorithm to temporarily deploy UAVs between the fixed ground base stations to provide essential services to the users.

Based on OSWireless, the control intent in this experiment can be defined similarly as in Fig. 8(a). The only major modification is to specify node coordinates `coord_x` and `coord_y` as control variables. This can be done by simply using the APIs provided by the OSWireless kernel WiNAS Subplane: `nt_ctl.set_var(net_name.coord_x)` and `nt_ctl.set_var (net_name.coord_y)`. Figure 12 reports the optimized node locations, where the initial locations of the relay nodes of Sessions 1 and 2 are represented using blue solid circles and green 'x's, respectively. The movements of the relay nodes are indicated using dotted arrows. The corresponding end-to-end throughput is shown

[7]The network scenario index starts from 4 to avoid confusion with the three scenarios in Fig. 7.

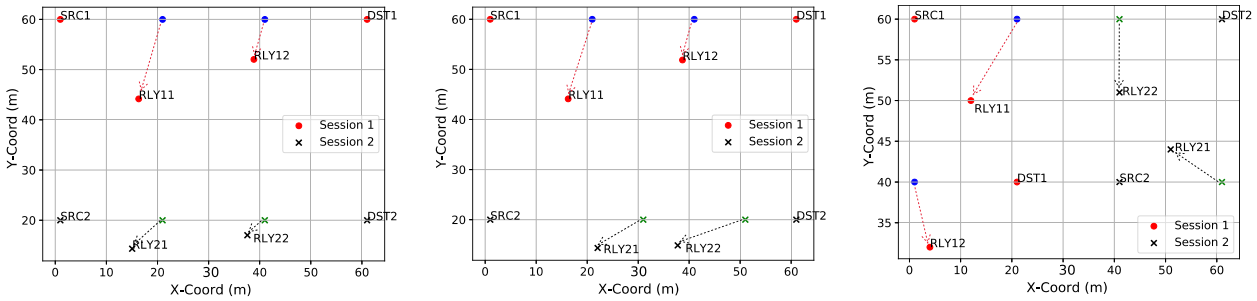


Figure 12: Optimized location for (a) Scenario 4, (b) Scenario 5 and (c) Scenario 6.

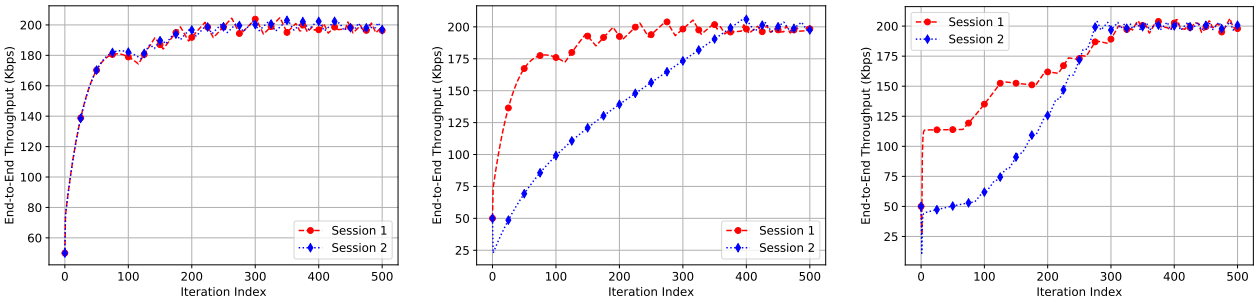


Figure 13: End-to-end throughput for (a) Scenario 4, (b) Scenario 5 and (c) Scenario 6.

in Fig. 13. It can be seen that, in all the tested scenarios, both of the two sessions are able to achieve the maximum end-to-end throughput of 200 kbps in around 200 time slots. This further verifies the flexibility and effectiveness of OSWireless in control intent specification.

8. Limitations and Future Work

We believe that our work on OSWireless provides a new approach to hiding the specification complexity for software-defined wireless networks. However, we acknowledge that there are several limitations, which will be addressed in future work.

Standardization of Interfaces. As discussed in Sec. 3, at the core of OSWireless is to separate those functionalities required by mapping intent specifications to the corresponding forwarding specifications and accomplish each functionality based on the services provided by the other functionalities. In the current implementation of OSWireless, while different modules have been developed for these functionalities, standardized interfaces among them are still missing that can allow third-party theoretical researchers, system engineers and field operation engineers to collaborate within the OSWireless framework.

Domain Knowledge-Integrated Data-driven Modeling. As discussed in the ParaModel Module of the WiNAS Subplane (Sec. 4), OSWireless currently supports two types of network element models: expression and script models. Both of the two types assume that the model can be defined analytically. To support better zero-touch control in heterogeneous wireless networks with more sophisticated control objectives and scenarios as well as more advanced standard-compliant radio interfaces (e.g., 5G NR), in future work the ParaModel Module can be extended by incorporating data-driven modeling, e.g., based on function approximation using neural networks, and integrating expert domain knowledge, including analytical models, cross-layer control and distributed control, with data-driven modeling.

Automated Control Intent Specification. In this work, while the mapping from intent specification to the algorithmic specification has been automated, the control intents are still specified manually by network engineers.

An interesting research direction is to automate this process as well for those scenarios requiring fast response, e.g., resilient networking in the presence of disruptive events or contested networking.

9. Conclusions

In this work we have designed OSWireless, a new control plane for optimizing software-defined wireless networks with automated control program generation capabilities. We verified experimentally the effectiveness and flexibility of OSWireless in hiding specification complexity. We believe OSWireless can provide a new approach to hiding the specification complexity for optimizing software-defined wireless networks and hence accelerate the research towards zero-touch programmable networks. In future work we will i) incorporate AI/ML-based data-driven network control in OSWireless and further design specification APIs to enable automated control with integrated optimization and learning; ii) test the effectiveness and flexibility of OSWireless considering wireless networks with standards-compliant radio interfaces, such as 5G based swarm UAV networks [65]; iii) standardize the interfaces among different OSWireless modules to allow third-party theoretical researchers, system engineers and field operation engineers to collaborate within the framework of OSWireless; (iv) allow dynamic configuration of intent and addition/removal of nodes from the network at run time; (v) allow definition of multiple applications per node with different network control objective and (vi) extend OSWireless to support complex network toplogies introduced in our prior research [55], [56] and [66]. The source code of OSWireless is available in our lab GitHub page [27]. We will prepare an instruction manual and make it available to the community once our system is further matured, so that researchers in different areas of expertise can use OSWireless.

References

[1] S. K. Moorthy, Z. Guan, N. Mastronarde, E. S. Bentley, M. Medley, OSWireless: Enhancing Automation for Optimizing Intent-Driven Software-Defined Wireless Networks, in: Proc. of IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, 2022.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, SIGCOMM Computer Communications Review 38 (2) (2008) 69–74.

[3] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A Roadmap for Traffic Engineering in SDN-OpenFlow Networks, Computer Network (Elsevier) Journal 71 (2014) 1–30.

[4] I. T. Haque, N. Abu-Ghazaleh, Wireless Software Defined Networking: A Survey and Taxonomy, IEEE Communications Surveys & Tutorials 18 (4) (2016) 2713–2737.

[5] W. Wang, Y. Chen, Q. Zhang, T. Jiang, A Software-Defined Wireless Networking Enabled Spectrum Management Architecture, IEEE Communications Magazine 54 (1) (2017) 33–39.

[6] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-Aware Controller Placement for Software-Defined Networks, in: Proc. of IFIP/IEEE International Symposium on Integrated Network Management, Ghent, Belgium, 2013.

[7] M. Ambrosin, M. Conti, F. De G., R. Poovendran, LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking, IEEE/ACM Transactions on Networking 25 (3) (2017) 1206–1219.

[8] S. H. Yeganeh, A. Tootoonchian, Y. Ganjali, On Scalability of Software-Defined Networking, IEEE Communications Magazine 51 (2) (Feb. 2013) 136–141.

[9] A. Montazerolghaem, M. H. Y. Moghaddam, A. Leon-Garcia, OpenSIP: Toward Software-Defined SIP Networking, IEEE Transactions on Network and Service Management 15 (1) (2018) 184–199.

[10] L. E. Li, Z. M. Mao, J. Rexford, Toward Software-Defined Cellular Networks, in: Proc. of European Workshop on Software Defined Networking (EWSDN), Darmstadt, Germany, 2012.

[11] S. R. Chowdhury, M. F. Bari, R. Ahmed, R. Boutaba, PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks, in: Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS), Krakow, Poland, 2014.

[12] X. Ge, Z. Li, S. Li, 5G Software Defined Vehicular Networks, IEEE Communications Magazine 55 (7) (2017) 87–93.

[13] S. Bera, S. Misra, S. K. Roy, M. S. Obaidat, Soft-WSN: Software-Defined WSN Management System for IoT Applications, IEEE Systems Journal 2 (3) (2018) 2074–2081.

[14] A. Gudipati, D. Perry, L. E. Li, S. Katti, SoftRAN: Software Defined Radio Access Network, in: Proc. of HotSDN, Hong Kong, China, 2013.

[15] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, B4: Experience With a Globally-Deployed Software Defined WAN, in: Proc. of the ACM SIGCOMM, Hong Kong, China, 2013.

[16] M. Jimenez, H. Kwok, Building Express Backbone: Facebook's New Long-haul Network, 2017.
URL `https://code.facebook.com/posts/1782709872057497/`

[17] A. D. Ferguson, S. Gribble, C. Hong, C. Killian, W. Mohsin, H. Muehe, J. Ong, L. Poutievski, A. Singh, L. Vicisano, R. Alimi, S. S. Chen, M. Conley, S. Mandal, K. Nagaraj, K. N. Bollineni, A. Sabaa, S. Zhang, N. Zhu, A. Vahdat, Orion: Google's Software-Defined Networking Control Plane, in: Proc of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2021.

[18] Z. Shi, Y. Tian, X. Wang, J. Pan, X. Zhang, Po-Fi: Facilitating innovations on WiFi networks with an SDN approach, Elsevier Computer Networks 187 (2021) 107781.

[19] B. Cao, Y. Li, C. Wang, G. Feng, S. Qin, Y. Zhou, Resource Allocation in Software Defined Wireless Networks, IEEE Network 31 (1) (2017) 44–51.

[20] Open Networking Foundation, OpenFlow Switch Specification (Oct. 2013).

[21] D. E. Sarmiento, A. Lebre, L. Nussbaum, A. Chari, Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey, IEEE Communications Surveys & Tutorials 23 (1) (2021) 256–281.

[22] M. Priyadarsini, P. Bera, Software Defined Networking Architecture, Traffic Management, Security, and Placement: A Survey, Computer Networks (Elsevier) 192 (June 2021).

[23] Huawei, Autonomous Driving Network - Building Self-Fulfilling, Self-Healing, and Self-Optimizing Autonomous Networks, Striding Towards the Intelligent World White Paper (2019).

[24] L. Pang, C. Yang, D. Chen, Y. Song, M. Guizani, A Survey on Intent-Driven Networks, IEEE Access 8 (2020) 22862–22873.

[25] D. Palomar, M. Chiang, A tutorial on decomposition methods for network utility maximization, IEEE Journal on Selected Areas in Communications 24 (8) (2006) 1439–1451. doi:10.1109/JSAC.2006.879350.

[26] G. Scutari and Y. Sun, Parallel and Distributed Successive Convex Approximiation Methods for Big-Data Optimization, Lecture Notes in Mathematics, C.I.M.E, Springer Verlag Series (2018).

[27] https://github.com/ubwingslab/OSWireless_G1 (2023).

[28] Open RAN Alliance, O-RAN: Towards an Open and Smart RAN, white paper (October 2018).

[29] S. Niknam, A. Roy, H. S. Dhillon, S. Singh, R. Banerji, J. H. Reed, N. Saxena, S. Yoon, Intelligent O-RAN for Beyond 5G and 6G Wireless Networks, arXiv.org (May 2020).
URL `https://arxiv.org/pdf/2005.08374.pdf`

[30] H. Kang, V. Yegneswaran, S. Ghosh, P. Porras, S. Shin, Automated Permission Model Generation for Securing SDN Control-Plane, IEEE Transactions on Information Forensics and Security 15 (2020) 1668–1682.

[31] M. Gharbaoui, B. Martini, P. Castoldi, Implementation of an Intent Layer for SDN-enabled and QoS-Aware Network Slicing, in: Proc. of IEEE International Conference on Network Softwarization (NetSoft), Tokyo, Japan, 2021.

[32] E. O. Zaballa, D. Franco, E. Jacob, M. Higuero, M. S. Berger, Automation of Modular and Programmable Control and Data Plane SDN Networks, in: Proc. of International Conference on Network and Service Management (CNSM), Izmir, Turkey, 2021.

[33] L. Bonati, M. Polese, S. D'Oro, S. Basagni, T. Melodia, Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead, Computer Networks 182 (2020) 1–28.

[34] Platforms for Advanced Wireless Research (PAWR) Program.
URL `https://advancedwireless.org/`

[35] J. Breen, A. Buffmire, J. Duerig, K. Dutt, E. Eide, M. Hibler, D. Johnson, S. K. Kasera, E. Lewis, D. Maas, A. Orange, N. Patwari, D. Reading, R. Ricci, D. Schurig, L. B. Stoller, J. Van der Merwe, K. Webb, G. Wong, POWDER: Platform for Open Wireless Data-Driven Experimental Research, in: Proc. of ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization (WiNTECH), London, United Kingdom, 2020.

[36] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, C. Gutterman, Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless, in: Proc. of International Conference on Mobile Computing and Networking, London, United Kingdom, 2020.

[37] M. L. Sichitiu, I. Guvenc, R. Dutta, V. Marojevic, B. Floyd, AERPAW Emulation Overview, in: Proc. of ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization (WiNTECH), London, United Kingdom, 2020.

[38] G. Hampel, D. Abush-Magder, A. Diaz, L. M. Drabeck, M. J. Flanagan, J. M. Graybeal, J. D. Hobby, M. MacDonald, P. A. Polakos, J. Srinivasan, H. Trickey, L. Zhang, G. Rittenhouse, The New Paradigm for Wireless Network Optimization: A Synergy of Automated Processes and Human Intervention, IEEE Communications Magazine 43 (3) (2005) S14–S21.

[39] L. S. Bai, R. P. Dick, P. H. Chou, P. A. Dinda, Automated Construction of Fast and Accurate System-Level Models For Wireless Sensor Networks, in: Proc. of Design, Automation Test in Europe, Grenoble, France, 2011.

[40] C. Di Martino, M. Cinque, D. Cotroneo, Automated Generation of Performance and Dependability Models for the Assessment of Wireless Sensor Networks, IEEE Transactions on Computers 61 (6) (2012) 870–884.

[41] M. Ge, J. B. Hong, W. Guttmann, D. S.Kim, A Framework for Automating Security Analysis of the Internet of Things, Journal of Network and Computer Applications 83 (2017) 12–27.

[42] K. Gökarslan, Menes: Towards a Generic, Fully-Automated Test and Validation Platform for Wireless Networks, arXiv:2006.02270 (2020).
URL `https://arxiv.org/abs/2006.02270`

[43] S. Harte, E. Popovici, B. O'Flynn, C. O'Mathúna, THAWS: Automated Design and Deployment of Heterogeneous Wireless Sensor Networks, WSEAS Transactions on Circuits and Systems archive 7 (2008) 829–838.

[44] J. Beutel, M. Dyer, R. Lim, C. Plessl, M. Wohrle, M. Yucel, L. Thiele, Automated Wireless Sensor Network Testing, in: Proc. of International Conference on Networked Sensing Systems, Braunschweig, Germany, 2007.

[45] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. D. Freitas, Learning to Learn by Gradient Descent by Gradient Descent, in: Proc. of International Conference on Neural Information Processing Systems (NIPS), Barcelona, Spain, 2016.

[46] S. Salman, C. Streiffer, H. Chen, T. Benson, A. Kadav, DeepConf: Automating Data Center Network Topologies Management with Machine Learning, in: Proc. of Workshop on Network Meets AI ML, Budapest, Hungary, 2018.

[47] R. Shimizu, K. Tei, Y. Fukazawa, S. Honiden, Model Driven Development for Rapid Prototyping and Optimization of Wireless Sensor Network Applications, in: Proc. of Workshop on Software Engineering for Sensor Network Applications, Waikiki, Honolulu, HI, 2011.

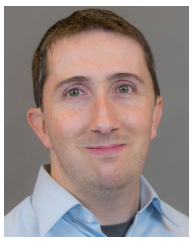
[48] H. Deng, Q. Li, Y. Li, S. Lu, C. Peng, T. Raza, Z. Tan, Z. Yuan, Z. Zhang, Towards Automated Intelligence in 5G Systems, in: Proc. of International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, 2017.

[49] L. Bertizzolo, S. D'Oro, L. Ferranti, L. Bonati, E. Demirors, Z. Guan, T. Melodia, and S. Pudlewski, SwarmControl: An Automated Distributed Control Framework for Self-Optimizing Drone Networks, in: Proc. of IEEE International Conference on Computer Communications (INFOCOM), Toronto, Canada, 2020.

[50] Z. Guan, L. Bertizzolo, E. Demirors, T. Melodia, WNOS: An Optimization-based Wireless Network Operating System, in: Proc. of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Los Angeles, USA, 2018.
[51] Z. Guan, L. Bertizzolo, E. Demirors, T. Melodia, WNOS: Enabling Principled Software-Defined Wireless Networking, IEEE/ACM Transactions on Networking 29 (3) (2021) 1391–1407.
[52] L. Bonati, S. D'Oro, L. Bertizzolo, E. Demirors, Z. Guan, S. Basagni, T. Melodia, CellOS: Zero-touch Softwarized Open Cellular Networks, Computer Networks (COMNET) 180 (Oct. 2020).
[53] https://pypi.python.org/pypi/cogapp (2019).
[54] J. Hu, M. McManus, S. K. Moorthy, Y. Cui, Z. Guan, N. Mastronarde, E. S. Bentley, M. Medley, NeXT: A Software-Defined Testbed with Integrated Optimization, Simulation and Experimentation, in: Proc. of IEEE Future Networks World Forum (FNWF): WS5: Federated Testbed as a Service for Future Networks: Challenges the State of the Art, Montreal, Canada, 2022.
[55] S. K. Moorthy, Z. Guan, Beam Learning in MmWave/THz-band Drone Networks Under In-Flight Mobility Uncertainties, IEEE Transactions on Mobile Computing 21 (6) (2022) 1945–1957.
[56] S. K. Moorthy, M. McManus, Z. Guan, ESN Reinforcement Learning for Spectrum and Flight Control in THz-Enabled Drone Networks, IEEE/ACM Transactions on Networking 30 (2) (2022) 782–795.
[57] https://pypi.org/project/simpy/ (2020).
[58] https://www.sympy.org/en/index.html (2021).
[59] A. Badi, I. Mahgoub, ReapIoT: Reliable, Energy-Aware Network Protocol for Large-Scale Internet-of-Things (IoT) Applications, IEEE Internet of Things Journal 8 (17) (2021) 13582–13592.
[60] M. La Manna, P. Perazzo, G. Dini, SEA-BREW: A scalable Attribute-Based Encryption revocable scheme for low-bitrate IoT wireless networks, Journal of Information Security and Applications 58 (2021) 102692.
[61] L. Zhao, T. Zheng, M. Lin, A. Hawbani, J. Shang, C. Fan, SPIDER: A Social Computing Inspired Predictive Routing Scheme for Softwarized Vehicular Networks, IEEE Transactions on Intelligent Transportation Systems (2021) 1–12.
[62] H. Zeng, H. Pirayesh, P. K. Sangdeh, A. Quadri, VehCom: Delay-Guaranteed Message Broadcast for Large-Scale Vehicular Networks, IEEE Transactions on Wireless Communications 20 (6) (2021) 3883–3896.
[63] R. K. Sheshadri, E. Chai, K. Sundaresan, S. Rangarajan, SkyHaul: An Autonomous Gigabit Network Fabric in the Sky, arXiv:2006.11307 (2020).
URL `https://arxiv.org/abs/2006.11307`
[64] D. Bertsekas, R. Gallager, Data Networks, Prentice Hall, USA, 2000.
[65] Z. Guan, N. Cen, T. Melodia, S. Pudlewski, Joint Power, Association and Flight Control for Massive-MIMO Self-Organizing Flying Drones, IEEE/ACM Transactions on Networking 28 (4) (2020) 1491–1505.
[66] S. K. Moorthy, Z. Guan, S. Pudlewski, E. S. Bentley, FlyBeam: Echo State Learning for Joint Flight and Beamforming Control in Wireless UAV Networks, in: Proc. of IEEE International Conference on Communications (ICC), Virtual/Montreal, Canada, 2021.

Sabarish Krishna Moorthy is a Ph.D. student in the Department of Electrical Engineering at State University of New York at Buffalo (SUNY B̃uffalo), NY, USA. He is currently working in the UB WINGS lab under the guidance of Professor Zhangyu Guan. He received his B.E. degree in Electronics and Communication Engineering and M.S. in Electrical Engineering from Velammal Engineering College, Chennai, India and SUNY Buffalo, NY, USA in 2016 and 2018, respectively. His current research interests are in new spectrum technologies and network design automation.

Zhangyu Guan (SM'11) received the Ph.D. degree in Communication and Information Systems from Shandong University in China in 2010. He is c̆urrently an Assistant Professor with the Department of Electrical Engineering at the State University of New York at Buffalo, where he directs the W̃ireless Intelligent Networking and Security (WINGS) Lab, with research interests in network design automation, new spectrum technologies, and wireless network security. He has served as an Area Editor for Elsevier Journal of Computer Networks since July 2019. He has served as TPC Chair for IEEE INFOCOM Workshop on Wireless Communications and Networking in Extreme Environments (WCNEE) 2020, Student Travel Grants Chair for IEEE Sensor, Mesh and Ad Hoc Communications and Networks (SECON) 2019-2020, Information System (EDAS) Chair for IEEE Consumer Communications Networking Conference (CCNC) 2021. He has also served as TPC member for IEEE INFOCOM 2016-2021, IEEE GLOBECOM 2015-2020, IEEE MASS 2017-2019, IEEE IPCCC 2015-2020, among others.



Nicholas Mastronarde received the B.S. and M.S. degrees in electrical engineering from the University of California, Davis, CA, USA, in 2005 and 2006, respectively, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles, CA, USA, in 2011. He is currently an Associate Professor with the Department of Electrical Engineering, University at Buffalo, Buffalo, NY, USA. His research interests include reinforcement learning, Markov decision processes, resource allocation and scheduling in wireless networks and systems, UAV networks, and 4G/5G networks.

Elizabeth Serena Bentley received the B.S. degree in electrical engineering from Cornell University, the M.S. degree in electrical engineering from Lehigh University, and the Ph.D. degree in electrical engineering from University at Buffalo. She was a National Research Council Postdoctoral Research Associate with Air Force Research Laboratory (AFRL), Rome, NY, USA. She is currently employed by the AFRL Information Directorate, performing in-house research and development in the Communication Systems and Technology Branch and managing the Aerial Layer Communication Technology (ALCT) program. Her research interests are in cross-layer optimization, swarm networking, directional networking, wireless multiple-access communications, and modeling and simulation.

Michael Medley (M'91-SM'02) received the BS, MS, and PhD degrees in electrical engineering from the Rensselaer Polytechnic Institute, Troy, NY, in 1990, 1991, and 1995, respectively. Since 1991, he has been a research engineer for the United States Air Force at the Air Force Research Laboratory, Rome, NY, where he has been involved in communications and signal processing research related to adaptive interference suppression, spread spectrum waveform design, spectrum management, covert messaging, and airborne networking and communications links. In 2002, he joined the State University of New York Polytechnic Institute in Utica, NY, where he currently serves as an associate professor in the Electrical and Computer Engineering Department. He is a senior member of the IEEE.