This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022

OSWireless: Enhancing Automation for Optimizing Intent-Driven Software-Defined Wireless Networks

Sabarish Krishna Moorthy[1], Zhangyu Guan[1], Nicholas Mastronarde[1],
Elizabeth Serena Bentley[2], and Michael Medley[2]
[1]Department of Electrical Engineering, University at Buffalo, Buffalo, NY 14260, USA
[2]Air Force Research Laboratory (AFRL), Rome, NY 13440, USA
Email: {sk382, guan, nmastron}@buffalo.edu, {elizabeth.bentley.3, michael.medley}@us.af.mil

***Abstract*—To control wireless networks with optimized configurations, engineers usually need to grapple simultaneously with network modeling, algorithm and protocol design as well as their implementation on distributed 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 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 flexibility of OSWireless in automated generation of control programs and the effectiveness of the generated programs considering a variety of network control problems. OSWireless can help enhance the automation of wireless network development, deployment and optimization.**

***Index Terms*—Intent-Driven Networks, Software-Defined Networking, Specification Abstraction.**

I. Introduction

Software-defined networking (SDN) has been envisioned as a key technique to enable *programmable* networks with high scalability and low management complexity, and to accelerate the adoption of new communication techniques and hence hasten the network evolution [1]–[4]. In the past decade, SDN has captured the attention of both academia and industry [5]–[9]. 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 [1]. Later, Google deployed their first SDN-enabled backbone wide area network called B4 based on OpenFlow for connecting their worldwide data centers [8]. Recently, Google deployed a new-generation SDN controller Orion in their B4 networks and data center Jupiter [9].

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 however rather challenging. The primary reason is that 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 [10]–[12]. As a result, to control wireless networks with optimized configuration the engineers need to grapple simultaneously with mathematical modeling, design of (possibly distributed) numerical algorithms, protocol development as well as the implementation of the resulting control logic in hardware. *This process is typically tedious and error prone.*

In this work we attempt to address the above challenges by designing OSWireless, a new SDN controller that can enable self-optimizing software-defined wireless networks. 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. We demonstrate and evaluate OSWireless by deploying it over NeXT, a newly developed software-defined network emulation and experimen-

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. (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.

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.

This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022

tation testbed. We showcase both effectiveness and flexibility of OSWireless in automated generation of control programs considering different network control problems. The source code of OSWireless will be released to the community via GitHub upon paper publication.

The remainder of the paper is organized as follows. We first discuss the related works in Sec. II. Then we describe the overall design objective of OSWireless in Sec. III and design the kernel of OSWireless (i.e., WiNAS Subplane) in Sec. IV and the OaaS Subplane in Sec. V. The testbed development and experimental evaluation are presented in Sec. VI. Finally, we draw the main conclusions and discuss future research directions in Sec. VII.

II. Related Work

There are a few SDN architectures for wireless networks in existing literature. For example, in SoftRAN [13], 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 [14] and MCC-SDWN [15] share the similar *split-design-control-plane* principles. Recently O-RAN has emerged to support interoperation between vendors' equipment by providing industry-wide standards for RAN interfaces and to enhance the RAN performance through virtualized network elements and integrated intelligence in RAN [16], [17]. Readers are referred to [18] 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.

Network automation has also attracted significant research attention [19]–[21]. For example, in [19] 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 [20] Harte et al. design a tool called "THAWS" to speed-up the development and deployment of heterogeneous WSNs. The authors of [21] design a toolflow that can help monitor the correctness of the implementation throughout the development of wireless sensor networks (WSN). *We are not aware of any existing frameworks that can provide open flexible APIs for hiding the specification complexity for optimizing software-defined wireless networks.*

The work most related to OSWireless is our prior work WNOS [22], [23] and its application in cellular [24] and ad hoc UAV networks [25]. WNOS is an optimization based wireless network operating system for principled software-defined wireless networking. Generally speaking, WNOS shares the same design goal with OSWireless, i.e., enhancing the automation of optimizing intent-driven software-defined wireless networks. In WNOS, the automated decomposition of centralized network control programs is enabled by a technique called Disciplined Instantiation (DI), based which it is complicated and not very flexible to instantiate abstract network control problems. In OSWireless we aim to provide a new control plane without relying on DI and hence it is easier to develop more sophisticated and practical wireless network applications.

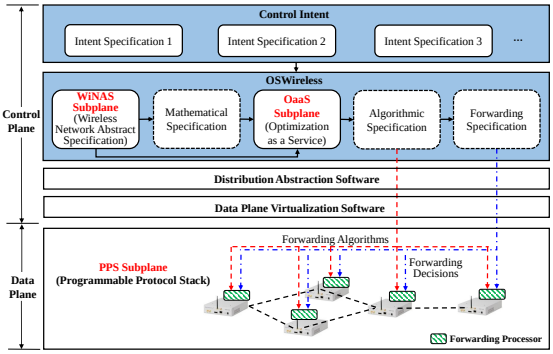


Fig. 1: Overall architecture of OSWireless.

III. 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

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

where f denotes the functionalities provided by OSWireless, $\mathcal{C}(\cdot)$ 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

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

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*.

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. Next, we first describe the basic design principles of OSWireless, focusing on *WiNAS Subplane* and *OaaS Subplane* in Secs. IV and V, respectively, and then showcase its application in wireless networks in Sec. VI.

IV. OSWireless Kernel Design: WiNAS Subplane

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 necessary notations.

A. Notation Definition

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

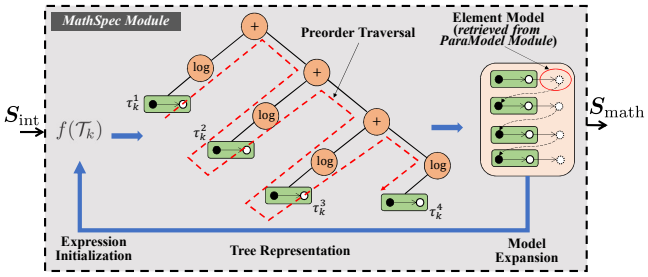


Fig. 2: Diagram of *MathSpec Module*. Solid and open dots are respectively, view and behavior elements. Open dots with dotted outlines denote models of behavior elements

(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. ii) 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 $\mathcal{B}(V_i)$. 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 as described later.

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.

B. 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.

This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022

Roughly speaking, each intent specification $\mathcal{S}_{\text{int}}$ consists of a set $\mathcal{K}$ of textual expressions. 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 (which will be described later in this section). 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 an example of $f(\mathcal{T}_k)$ with four VBEPs. If we consider VBEP $(\texttt{link}, \texttt{rate})$ 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 example as

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

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.

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.

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 in the expression are leaf behavior elements.

C. 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(·)`. These services

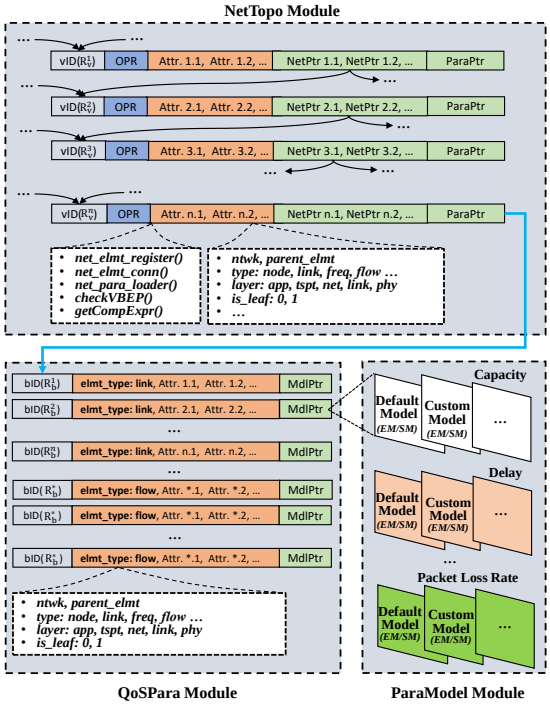


Fig. 3: Diagram of NetTopo, QoSPara and ParaModel modules. These three modules together provide various kernel services for the other modules of OSWireless.

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 tuple, consisting of five categories of fields. These are $\texttt{vID}(\text{R}_{\text{v}}^n)$, which is the OSWireless-wide unique ID of view element R_{v}^n; $\texttt{vAttrList}(\text{R}_{\text{v}}^n)$, 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 $\texttt{vNetPtr}(\text{R}_{\text{v}}^n)$ consists of a set of pointers, each pointing to another associated view element $\mathcal{R}_{\text{v}}' \in \mathcal{V}$, e.g., the individual elements that form a set element; Finally, $\texttt{vBPtr}(\text{R}_{\text{v}}^n)$, i.e., field *ParaPtr*, maintains the set of behavior elements attached to R_{v}^n and $\texttt{vOprPtr}(\text{R}_{\text{v}}^n)$ defines the set of the operations supported by R_{v}^n. It is worth mentioning that these operations are designed to enable automated expression

This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022

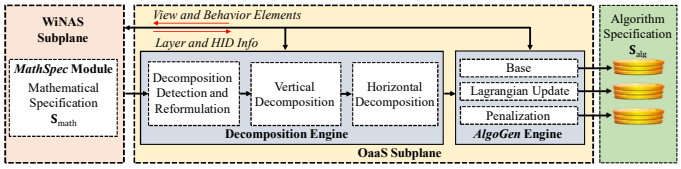


Fig. 4: Overall architecture of Optimization-as-a-Service (OaaS) Subplane.

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, each behavior record also consists of an OSWireless-wide unique ID $\texttt{bID}(R_{\rm b}^n)$ and an attribute list $\texttt{bAttrList}(R_{\rm b}^n)$. Examples of these behavior attributes include $\texttt{bLayer}(R_{\rm b}^n)$, which provides the protocol layer information of the behavior elements; and $\texttt{bHID}(R_{\rm b}^n)$, which defines the *node* view element that behavior element $R_{\rm b}^n$ is associated to. This information will be provided to the OaaS Subplane for automated distributed problem decomposition in Sec. V.

Finally, each behavior record $R_{\rm b}^n \in \mathcal{B}$ also has a `bMdlPtr` field, i.e., *MdlPtr*, 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. 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.

V. OaaS Subplane Design

With the mathematical specification $\boldsymbol{S}_{\rm math}$ obtained by the WiNAS Subplane in Sec. IV, 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 $\boldsymbol{S}_{\rm math}$. Since the mathematical specification $\boldsymbol{S}_{\rm math}$ is defined centrally and possibly across multiple protocol layers, we first need to decompose $\boldsymbol{S}_{\rm math}$ into distributed ones. 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. 4, which consists of two major components, i.e., *Decomposition Engine* and *AlgoGen Engine*.

A. 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 "+"[1]; 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. IV-B. Denote the resulting set of component expressions as $\mathcal{K}_{\rm comp}(f)$. In the second step, the component assignment is accomplished through two types of decomposition: vertical and horizontal decomposition. The overall decomposition architecture is illustrated in Fig. 4.

The objective of vertical decomposition is to uncouple the coupling among the protocol layers involved in mathematical specification $\boldsymbol{S}_{\rm math}$. To this end, for each component expression $k_{\rm comp} \in \mathcal{K}_{\rm comp}(f)$ its associated layer is obtained using WiNAS Subplane API $\texttt{bLayer}(R_{\rm b}(k_{\rm comp}))$, where $R_{\rm b}(k_{\rm comp})$ represents the VBEPs contained in component expression $k_{\rm comp}$. 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. 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 design a new attribute called horizontal index (HID) for each network element. 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 $\boldsymbol{S}_{\rm math}$. By default, the HID will be the parent view element (i.e., `parent_elmt` in QoSPara Module) 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 $k_{\rm comp} \in \mathcal{K}_{\rm comp}(f)$ to a distributed problem, we only need to get its HID using WiNAS Subplane API $\texttt{bHID}(k_{\rm comp})$ and further get the HID of $\texttt{bHID}(k_{\rm comp})$. This process is repeated until the HID of the view element is itself and then component expression $k_{\rm comp}$ 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. 4, 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

[1]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.

This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022

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. Roughly speaking, a specification expression can 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 $\texttt{link\_capacity} > \texttt{src\_rate}$, otherwise the average delay hence the response time will become infinity. This expression is not directly decomposable since $\texttt{link\_capacity}$ and $\texttt{src\_rate}$ 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 $\texttt{v}_{\text{aux}} == \texttt{link\_capacity} - \texttt{src\_rate}$. 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. We omit the details of these rules due to space limitations.

B. 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. 4, 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 [26]. To construct the base algorithm, we first detect the optimization variables present in each of the distributed problems obtained in Sec. V-A 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 $\texttt{bVar}(\cdot)$ 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 [27].

VI. OSWireless-based Wireless Network Control

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.

Testbed Development. A snapshot of the NeXT testbed is shown in Fig. 5. There are three major components in the

This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022

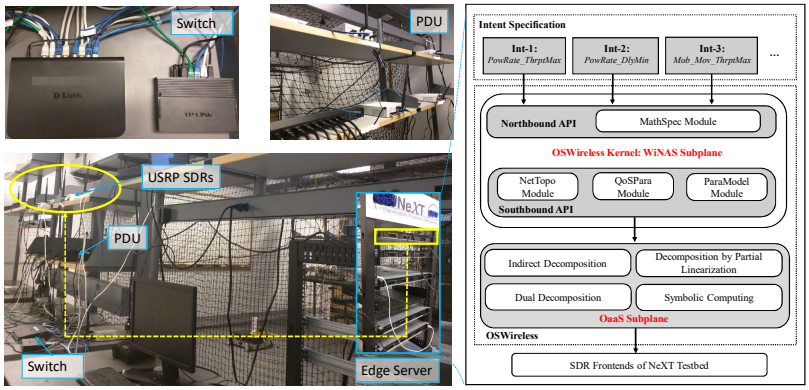


Fig. 5: Snapshot of the software-defined testbed and software diagram for OSWireless prototyping.

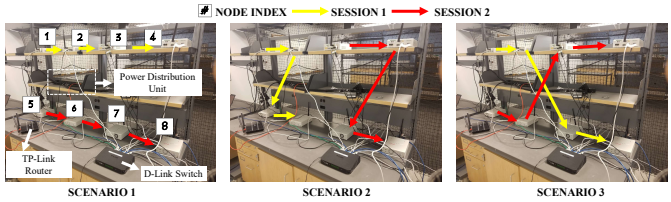


Fig. 6: Network topology for OSWireless testing over NeXT testbed.

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.[2] 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 bpsk, qpsk, 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 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 with data rate ranging from 1-200 kbps. The available spectrum band (2-2.6 GHz) is divided into three subchannels and shared by different nodes with configurable spectrum reuse factor.

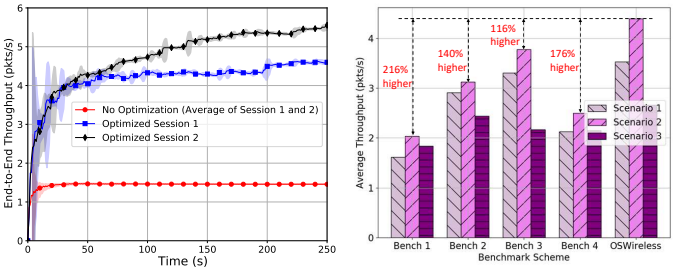
Finally, we prototype OSWireless and deploy it on one of the workstations as indicated by the yellow rectangle in Fig. 5, including the WiNAS Subplane (Sec. IV) and the OaaS Subplane (Sec. V). For the SDR data plane, the other workstations serve as the controlling hosts of the front-end SDRs for baseband signal processing and run the distributed control programs automatically generated by OSWireless to adapt the transmission parameters at different layers of the programmable protocol stack. 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 [28]. 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.

Intent Specification Example. 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, such as sensor and actuator networks [29], [30], vehicular networks [31], [32], and wireless backhaul networks [33]. In this work, we consider three scenarios as shown in Fig. 6. 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.

In this experiment, the network behavior description, in-

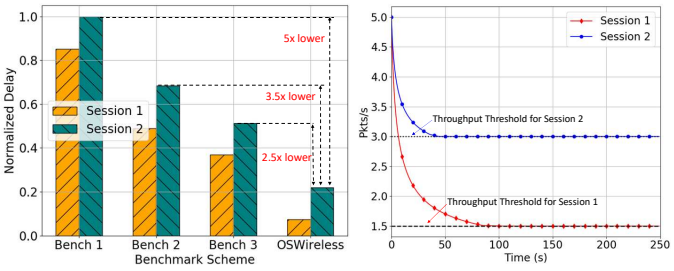
[2]Currently we are extending the NeXT testbed to enable experiments for integrated aerial-ground wireless networks and make the testbed publicly accessible by interfacing it with a public cloud Amazon Web Service (AWS).

This paper has been accepted for publication on IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), Denver, Colorado, October 2022



(a) (b)

Fig. 8: (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)

Fig. 9: (a) Average end-to-end delay and (b) instantaneous transmission rate.

cluding 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 `lkpwr` (transmission power for a link). 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 expression defined by the intent specification through the expression expansion operation in the *MathSpec Module* (see Fig. 2). 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. We omit the details of the generated algorithms due to lack of space.

Figure 8 shows the end-to-end throughput performance of the control programs automatically generated by OSWireless. The instantaneous throughput is reported in Fig. 8 (a) considering Scenario 1 (see Fig. 6) 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. The average performance is plotted in Fig. 8 (b) for the three scenarios by averaging over 10 experiments. Three more benchmark schemes are considered in addition to that in Fig. 8 (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.

It is worth pointing out that the centralized high-level specification 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 and algorithmic specification will be orchestrated by OSWireless in an automated manner and hidden from the developers.

Flexibility of OSWireless. In the above experiment we have showcased through an 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 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 [34] 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 element, behavior element and the parameter model defined in the NetTopo, QoSPara and ParaModel modules of the WiNAS Subplane. Then, as described in Sec. V-A, 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.

The delay performance of the obtained algorithmic specification is reported in Fig. 9. In this experiment, we consider low-data-rate applications with the target end-to-end throughput set to 1.5 pkts/s and 3 pkts/s for the two sessions, respectively, as shown in Fig. 9 (b). In Fig. 9 (a) we plot the normalized delay of the two sessions for OSWireless and compare it with three benchamrk schemes: benchmarks 1, 2 and 3 in Experiment 1 but tailored for this experiment. We can see that OSWireless can achieve up to $5\times$ lower delay.

VII. Conclusions and Future Work

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 [35]; and 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.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Computer Communications Review*, vol. 38, no. 2, pp. 69–74, March 2008.

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

[3] I. T. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2713–2737, Fourthquarter 2016.

[4] W. Wang, Y. Chen, Q. Zhang, and T. Jiang, "A Software-Defined Wireless Networking Enabled Spectrum Management Architecture," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 33–39, Jan. 2017.

[5] M. Ambrosin et al., "LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1206–1219, April 2017.

[6] A. Montazerolghaem et al., "OpenSIP: Toward Software-Defined SIP Networking," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 184–199, March 2018.

[7] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat, "Soft-WSN: Software-Defined WSN Management System for IoT Applications," *IEEE Systems Journal*, vol. 2, no. 3, pp. 2074–2081, Sept. 2018.

[8] S. Jain et al., "B4: Experience With a Globally-Deployed Software Defined WAN," in *Proc. of the ACM SIGCOMM*, Hong Kong, China, Aug. 2013.

[9] A. D. Ferguson et al., "Orion: Google's Software-Defined Networking Control Plane," in *Proc of the USENIX NSDI*, April 2021.

[10] Open Networking Foundation, "OpenFlow Switch Specification," Oct. 2013.

[11] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 256–281, First Quarter 2021.

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

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

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

[15] B. Cao, Y. Li, C. Wang, G. Feng, S. Qin, and Y. Zhou, "Resource Allocation in Software Defined Wireless Networks," *IEEE Network*, vol. 31, no. 1, pp. 44–51, Jan./Feb. 2017.

[16] Open RAN Alliance, "O-RAN: Towards an Open and Smart RAN," *white paper, October*, 2018.

[17] S. Niknam et al., "Intelligent O-RAN for Beyond 5G and 6G Wireless Networks," *arXiv.org*, May 2020. [Online]. Available: https://arxiv.org/pdf/2005.08374.pdf

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

[19] G. Hampel et al., "The New Paradigm for Wireless Network Optimization: A Synergy of Automated Processes and Human Intervention," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S14–S21, March 2005.

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

[21] J. Beutel et al., "Automated Wireless Sensor Network Testing," in *Proc. of International Conference on Networked Sensing Systems*, Braunschweig, Germany, June 2007.

[22] Z. Guan, L. Bertizzolo, E. Demirors, and 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, June 2018.

[23] ——, "WNOS: Enabling Principled Software-Defined Wireless Networking," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1391–1407, June 2021.

[24] L. Bonati et al., "CellOS: Zero-touch Softwarized Open Cellular Networks," *Computer Networks (COMNET)*, vol. 180, Oct. 2020.

[25] L. Bertizzolo et al., "SwarmControl: An Automated Distributed Control Framework for Self-Optimizing Drone Networks," in *Proc. of IEEE INFOCOM*, Toronto, Canada, July 2020.

[26] https://pypi.python.org/pypi/cogapp, 2019.

[27] B. Buchberger, "Symbolic Computation (An Editorial)," *Journal of Symbolic Computation*, pp. 1–6, 1985.

[28] https://www.sympy.org/en/index.html, 2021.

[29] A. Badi and I. Mahgoub, "ReapIoT: Reliable, Energy-Aware Network Protocol for Large-Scale Internet-of-Things (IoT) Applications," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13 582–13 592, Sept. 2021.

[30] M. La Manna, P. Perazzo, and G. Dini, "SEA-BREW: A scalable Attribute-Based Encryption revocable scheme for low-bitrate IoT wireless networks," *Journal of Information Security and Applications*, vol. 58, p. 102692, May 2021.

[31] L. Zhao et al., "SPIDER: A Social Computing Inspired Predictive Routing Scheme for Softwarized Vehicular Networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, Oct. 2021.

[32] H. Zeng et al., "VehCom: Delay-Guaranteed Message Broadcast for Large-Scale Vehicular Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 6, pp. 3883–3896, June 2021.

[33] R. K. Sheshadri, E. Chai, K. Sundaresan, and S. Rangarajan, "SkyHaul: An Autonomous Gigabit Network Fabric in the Sky," *arXiv:2006.11307*, 2020. [Online]. Available: https://arxiv.org/abs/2006.11307

[34] D. Bertsekas and R. Gallager, *Data Networks*. USA: Prentice Hall, 2000.

[35] Z. Guan, N. Cen, T. Melodia, and S. Pudlewski, "Joint Power, Association and Flight Control for Massive-MIMO Self-Organizing Flying Drones," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1491–1505, August 2020.