A Rapid Prototyping for Wireless Virtual Network Embedding using MARVELO

Haitham Afifi, Sebastian Eikenberg, Alexander Makejkin, Arnold Müller,
Rafael Schellenberg, Lars Gansel, Kai Hannemann, and Holger Karl
Paderborn University (haitham.afifi@upb.de)

***Abstract*—One of the major challenges in implementing wireless virtualization is the resource discovery. This is particularly important for the embedding-algorithms that are used to distribute the tasks to nodes.**

MARVELO is a prototype framework for executing different distributed algorithms on the top of a wireless (802.11) ad-hoc network. The aim of MARVELO is to select the nodes for running the algorithms and to define the routing between the nodes. Hence, it also supports monitoring functionalities to collect information about the available resources and to assist in profiling the algorithms.

The objective of this demo is to show how MAVRLEO distributes tasks in an ad-hoc network, based on a feedback from our monitoring tool. Additionally, we explain the work-flow, composition and execution of the framework.

1. Introduction

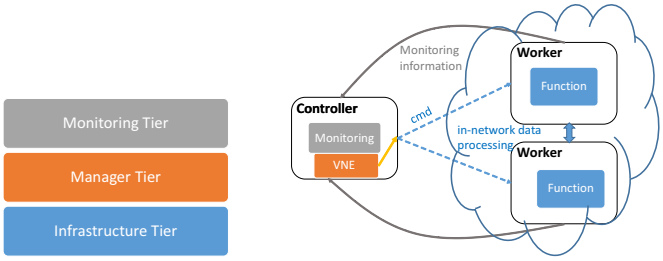
Nowadays, there exist many applications that rely on wireless infrastructure. Some application types read data (e.g., from sensors) and process them directly inside the network. When it comes to implementation, such applications are often structured as individual functions that are chained together.

Examples for such functions are load balancers for networking applications or filtering for signal processing. Instead of relying on various physical networks for each application, different applications can share the same network to run these functions, virtualizing the network.

Network virtualization replaces the need for dedicated hardware with virtual functions running on top of a generic hardware, thus, enabling different applications to coexist on the same hardware.

A typical use case would be for functions of the same application that need to be interconnected to exchange data. In this context, we need to answer two important questions: on which node will the virtual functions be running and which route will be taken between the nodes. This process is commonly called Virtual Network Embedding (VNE).

When it comes to wireless VNE, only few frameworks were implemented for IoT or Mobile networks [1]. Nevertheless, they assumed that wireless link conditions are fixed as in wired networks, and ignored the changes in wireless links due to interference. Moreover, some existing work limits the application of virtual functions to virtual network functions [2], rather than taking a more general approach, also embracing application-level functions.



(a) Tiers (b) Architecture

Figure 1: Representation of MARVELO's Tiers

We are the first to implement a framework for a wireless VNE in Wireless Sensor Networks (WSN). It supports executing generic virtual functions (let it be networking or other) that can be written in any programming language (e.g., Python, C, JAVA, etc.) in an executable form. Additionally, we jointly consider the wireless interference and computational resources during the allocation process. In our implementation, we use Raspberry Pis as our wireless nodes, and we show distributed processes running on multiple nodes.

2. MARVELO Framework

In this section we describe the implementation of the Multicast-Aware Routing for Virtual network Embedding with loopy Overlays (MARVELO) framework. The detailed mathematical formulation for resource allocation can be found in [3]; we focus here on the framework architecture and implementation. MARVELO has also been used to show the importance of distributing algorithms [4], but without explaining MARVELO's work-flow.

There are two main roles in the framework; *Controller node* and *Worker Node*. A controller is responsible for allocating functions to the wireless nodes and controlling the routing decisions. Additionally, it has an overview of the network performance as well as the virtual functions.

Worker nodes receive commands from the controller to start running the functions. The output from these functions is then forwarded according to the forwarding tables received from the controller. In other words, the controller

is an intelligent entity (brain) that is giving orders to the worker nodes (muscles) for processing and forwarding.

For internal communication between different roles, MARVELO's architecture is represented by three tiers: Infrastructure, Manager, and Monitoring tier (Figure 1).

2.1. Manager tier

The manager tier is responsible for taking the allocation and routing decisions for the infrastructure tier.

Inputs to this tier are divided into two categories. **First** the workers' status, which is given as the available resources for each worker and the wireless channel status. **Second**, the required processing resources by the functions (e.g., CPU and memory utilization).

Next, data is processed to find the optimal allocation for the nodes using the algorithms proposed in [3]. The output from the algorithms is written in an XML file to be used by the

2.2. Infrastructure tier

This tier is responsible for distributing the functions to the worker nodes. The decisions are taken by the controller node (in the manager tier) and written in an XML format, which expresses routing decisions as well. The XML file can also be edited manually to specify a custom allocation and routing. Next, the controller node sends to each worker which functions shall be executed and specify the routes (i.e., inputs to read data and outputs to send data).

To exchange data between the functions, we use the `netcat` tool, allowing us to support streaming data if needed. The `netcat` instances are auto-generated by the controller and shared automatically with the workers.

We use a parser for the command-line arguments to specify the input and output `netcat` instances for each function. Additional parameters (i.e, logging files and configuration) can optionally be added as arguments.

2.3. Monitoring tier

There exists many tools for monitoring the system utilities and processes running on the Raspberry Pi's. But existing network-monitoring tools cannot provide sufficient information about the wireless channel behaviour, due to the limitation by the Broadcom chip installed on the Raspberry Pi, which does not enable promiscuous mode, which is used to scan the devices in the network. Although there exists other workarounds [5], they do not support ad-hoc and promiscuous simultaneously.

This tier provides an API that is used to monitor both the processing (e.g., CPU and memory utilization) and wireless (e.g., interference and channel scanning) parameters on the Raspberry Pis.

It relies on three main modules. **First**, *CommonMonitor* module is responsible for monitoring the overall process and system utilities. Accordingly, it delivers information about the available resources per each node in the network. **Second**, *ProcessMonitor* similarly monitors the processes on the node, but only for specified process. Consequently, this tool is used for profiling the selected applications to estimate the required resources for each one. **Third**, *NetworkMonitor* is used to collect the received wireless signal strength from the source and other interfering nodes. It relies on NEXMON [6], a firmware patch to enable the promiscuous mode on the Broadcom chips of Raspberry Pis.

On the one hand, the common and process monitoring modules can collect information only from the clients running the framework. On the other hand, the networking information (e.g., mac addresses and received signals) can be collected from all nodes, even if they are not running the framework. This allows us to have information about the wireless network as reliable as possible.

The monitoring tier is supported also with a notification system that can: 1) send alerts if a node or a running process are shut down. 2) report new incoming nodes as well as their available resources. Other types of alerts (e.g., temperature or over-utilization) can also be added.

3. Demo Description

In this demo we show how to use the MARVELO framework to profile processes and collect information about the available resources in the network. Then, we give this information to MARVELO's controller in order to select the nodes for processing and define the routes between the nodes. Next, the controller will distribute the functions among the workers and start exchanging information. The running application will be in the context of acoustic sensor networks. Additionally, we demonstrate other features of MARVELO such as collecting the logs from the clients after finishing the process and sending notifications.

Acknowledgment

This work was supported by *Deutsche Forschungsgemeinschaft* (DFG) under contract no. KA2325/4-1 within the framework of the Research Unit FOR2457 "Acoustic Sensor Networks".

References

[1] N. Bizanis and F. A. Kuipers. SDN and virtualization solutions for the internet of things: A survey. *IEEE Access*, 4:5591–5606, 2016. ISSN 2169-3536. doi: 10.1109/ACCESS.2016.2607786.

[2] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo. SDN-WISE: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 513–521, April 2015.

[3] Haitham Afifi and Holger Karl. An approximation algorithm for power allocation in wireless virtual network embedding. In *2019 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE WCNC 2019)*, Marrakech, Morocco, 2019. (submitted).

[4] Afifi H, Schmalenstroeer J, Ullmann J, Haeb-Umbach R, and Karl H. Marvelo–a framework for signal processing in wireless acoustic sensor networks. *13th ITG conference on Speech Communication*, October 2018.

[5] Vivek Ramachandran and Cameron Buchanan. *Kali Linux: Wireless Penetration Testing Beginner's Guide*. Packt Publishing, 2015. ISBN 1783280417, 9781783280414.

[6] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. Nexmon: The c-based firmware patching framework, 2017. URL `https://nexmon.org`.