ULLA-X: A Unified Programmatic Middleware for On-Demand Network Reconfiguration

Avishek Patra, Andreas Achtzehn, and Petri Mähönen
Institute for Networked Systems
RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
Email: {avp,aac,pma}@inets.rwth-aachen.de

***Abstract*—Software-defined networking provides new tools and offers more opportunities to actively manage large-scale network setups during runtime. However, controlling such installations in a unified and adaptive manner is inherently complex due to the remarkable diversity of management tools. In this paper we propose ULLA-X, a novel middleware that unifies network monitoring and on-demand reconfiguration through a universal network programming language. We introduce ULLA-X architecture and usage concept, which aims at bridging sophisticated network planning with practical network maintenance tools, resembling an "autonomic nervous system" for communication networks. Through its modular design, ULLA-X can be easily integrated into current network setups. While this paper is focused on introducing the ULLA-X concept and framework, we also present initial results from a prototype implementation for National Instruments wireless networked devices.**

I. Introduction

With the more widespread deployment of software-defined networks (SDNs) [5] the necessity for a unified way to manage these centrally coordinated and flexible networks is becoming more pressing. Currently, network operators need to employ up to 30 different vendor-specific administration interfaces to maintain their SDN devices [7], which requires highly skilled personnel and may cause outage due to misconfiguration and human errors. While OpenFlow has established itself for a long time as a de-facto standard for the "southbound" API between data and control plane, the "northbound" API for deployment, monitoring, and maintenance is lacking behind in terms of consolidation. In the absence of unified configuration APIs, the adaption of software networking principles into mainstream computer network design may soon be severely hampered [6]. This problem is even larger in the case of software defined wireless networking, where also the "southbound" APIs are mostly non-existent, especially towards PHY-layer configuration parameters.

Motivated by current standardization efforts, e.g. by the Open Networking Foundation NBI working group, we propose ULLA-X, a middleware that enables network monitoring and on-demand network reconfiguration for SDN and legacy network components. Our aim is to jointly define a distributed controller architecture and network programming language that enables network operators to specify the reactive behavior of a communication network. We envision ULLA-X to play a similar role in a network as the autonomic nervous system in humans and animals - automating regular tasks and quickly responding to threat conditions. As such, we see a potential for ULLA-X to operate in the post-deployment phase of a network, where it strikes a balance between purely statistics logging and a closed-loop control systems. To achieve this goal, ULLA-X implements vendor and technology-agnostic monitoring of network performance indicators and event-based triggering of reconfiguration tasks. It draws from our earlier experience in wireless media access unification [8], and significantly extends it for larger challenges in current networking paradigms.

We envision ULLA-X to be used in conjunction with other potentially highly sophisticated network control tools. By transforming high-level policies into simple network programming primitives ULLA-X implements an important transition layer, allowing network designers to outline what-if reactions based on more complex reasoning on state variables and conditional scenarios. ULLA-X thereby transfers some of the original ideas of cognitive networks [9] to the practical and implementable sphere of SDNs. While truly cognitive networks are still subject to intense research, the positioning of ULLA-X at the boundary between decision and action functions of the network cognition cycle makes it relevant and applicable already in today's practical network deployments.

There are other recent approaches towards creating a unified (re-)configuration and monitoring API for SDNs. Gember et al. propose OpenMB [4], a framework for the configuration of middleboxes such as firewalls or proxies. Contrary to ULLA-X, OpenMB's main application is the initial deployment phase; it hence lacks a terminology for defining triggers and reactions to network state changes in a holistic manner. Similar to OpenMB, OpenAFNV by Ge et al. [3] aims at matching virtualized network functions (VNFs) and available hardware resources, but misses a runtime control. Similar to ULLA-X, PayLess [1] defines a common southbound API monitoring interface for retrieving flow statistics. Another notable example of a southbound network programming language is Frenetic [2], where the focus lies on handling (potentially interdependent) traffic flows, making it a powerful tool for policy-based network steering. ULLA-X on the other hand resides at the northbound interface, in-between coarse VNF configuration and fine-grained flow-level policy execution. Using methods similar to those of PayLess, it defines responses in

978-1-4673-6770-7/15/$31.00 ©2015 IEEE

terms of parametric reconfigurations to changes in the overall network performance, uniting features of classical network monitoring tools with reconfiguration features for SDNs.

Functional primitive sets in the ULLA-X language are used by the ULLA-X infrastructure to derive necessary monitoring tasks and trigger the reactive behavior of the ULLA-X enabled SDN. Given its lightweight structure, and an adaptation layer for wrapping vendor-specific configuration APIs, we consider our middleware to be easily extensible for various network setups. While this paper is mainly describing the architecture and framework of ULLA-X, we demonstrate the potential of ULLA-X through some initial experiments with a flexible link setup using National Instruments equipment. Such platform typical exposes a large number of configuration parameters that need to be managed in a similar fashion as e.g. OpenFlow switches. We provide some initial test results, where we show the reconfiguration delay.

The rest of this paper is organized as follows: In Section II we present the necessary architectural components of an ULLA-X enabled SDN. In Section III we continue by defining an initial command-set for our declarative ULLA-X programming language. Initial results presented in Section IV from an implementation of ULLA-X on top of software-defined radios demonstrate the technical feasibility. Our paper is concluded in Section V with a review and outlook to future research tasks.

II. ULLA-X Architecture

The ULLA-X architecture is logically decomposed into the **ULLA-X core**, which is the operational center of the system, and a number of **connector modules** that coordinate between the core and the network components, see Figure 1. **Management applications**, e.g. configuration tools or network optimization engines, may access the ULLA-X core through a standard message passing interface. While being a single logical entity, the ULLA-X core may be functionally spread over multiple physical network entities. The connector modules can reside in different parts of the SDN, and be equipped with different interfacing capabilities towards the network functions.

A. ULLA-X Core

The ULLA-X core consists of the following main components: the configurator, the collector, and the watchdog. For their operations they employ a parser, a storage engine, and the network interfacing block. User interaction with ULLA-X exclusively takes place through message passing with the configurator. Users pass sets of commands defined in the ULLA-X PL, see Section III, which are interpreted by the parser to extract relevant information such as the referenced network device, parameters, or necessary arithmetic or logical operations. Results are returned to the user by similar means. Monitoring tasks are handed on by the configurator to the collector, which will subsequently use network interface functions to retrieve updated values for the respective network state variables. Depending on the type of request sent by the

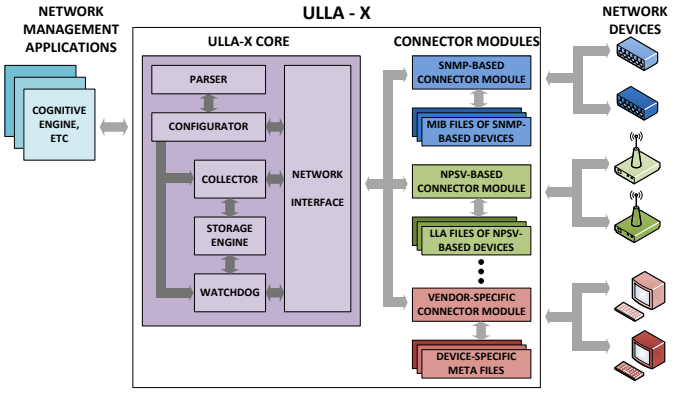

Fig. 1. ULLA-X Architecture Block Diagram.

user, such data collection may be carried out only once, or is repeated periodically in order to establish statistics of the overall network performance. Periodic data is stored in a background database, from which it is retrieved by the watchdog for further processing. The watchdog runs asynchronously, i.e. it becomes active only once new data for which a conditional execution has been defined, has changed. The watchdog is called at the periodicity of monitored parameter updates, allowing the network monitoring functions to operate with low computational and querying overhead. Once a conditional execution statement is triggered, the network is reconfigured according to a set of commands that have been provided to the watchdog by the configurator.

B. Connector Modules

Connector modules fulfill two purposes. First, they wrap vendor-specific APIs such as the Simple Network Management Protocol (SNMP) for traditional networks or specific configuration interfaces for SDNs [7], to enable a unified access. Using similar methods as for the user interaction in the ULLA-X core, we employ standard message passing interfaces for this purpose. A request from the ULLA-X core to one of the connector modules defines the requested parameter and device/function identifier, the response by the connector contains, apart from the actual parameter value, meta-data on data format, validity, and scope. Requests may also contain modification commands, to which the connector modules response with a set of error codes in order to indicate the success or failure of the alteration. Internally, the connector module translates the requests into vendor-specific commands, employing e.g. Management Information Base (MIB) files for SNMP devices or generic Link Layer Adapter (LLA) [8] definitions for non-standard network components.

As connector modules may be located on different network segments with individually limited reach, we have also incorporated as a second part of each connector a generic network discovery protocol. Some vendor-specific management interfaces, e.g. the Network-Published Shared Variables (NPSV) system of National Instruments LabVIEW, which we have been using for our prototype implementation, offer mechanisms to gather information on local network devices and their capabilities. The discovery mechanism creates individual device identifiers for all discovered devices, and passes these

to the ULLA-X core. Subsequent ULLA-X commands may refer to the devices by their device identifiers, which allows the core to pass the requests to the responsible connector module. Where device discovery is not possible, the network operator will need to configure the connector upon startup to become aware of its local network environment.

III. Programming Language

The design of the ULLA-X network programming language (ULLA-X PL) is aimed at resembling other widely used declarative programming languages, e.g. SQL. We assume that this will allow users to quickly learn how to define network monitoring and control statements, and lower the hurdle for integrating ULLA-X into higher-level network management tools. At the current stage, the ULLA-X PL defines statements for management, data definition, data manipulation, and triggered execution, and, as ULLA-X is still under development, we are extending the PL towards broader application scenarios and requirements. We will now give a brief overview of commonly used commands in the different command classes.

A. Management and discovery commands

In the initial deployment phase, the ULLA-X core needs to be made aware of the connector modules through which network functions may be controlled. Those are added by means of the

`ADD CONNECTOR <host>`

command, where the host definition may be an IP address or fully-qualified hostname of the network instance running the ULLA-X connector functionality. All other ULLA-X PL statements are oblivious to the underlying serving connector module infrastructure, i.e. the ULLA-X infrastructure and device access hierarchy is hidden from the users.

New managed network entities, e.g. switches or VNFs, are discovered through a broadcast request to all connector modules, i.e.

`SHOW ALL DEVICES [WHERE <cond>],`

where typical condition statements would be to select a particular device technology (e.g. all switches), location in the network (e.g. a particular data center), or status (e.g. whether the function is online). The output of this command may be nested, in order to define device condition or device property-specific data definition or manipulation statements. It returns a unique device identifier for subsequent commands.

B. Data definition statements

Data definition statements define the network parameters that should be monitored on a regular basis, and for which data should be stored through the ULLA-X core-embedded storage engine. Issuing a command,

`SET READ <var_name>[,<var_name>]* FROM <device> AS <storage_name> PERIOD <interval>,`

the user specifies the network parameter name(s) and their origin, the periodicity of the retrieval, and the storage name for later reference. Subsequent retrieval of parameter values is carried out through the collector component in the ULLA-X core. Over time, the collector will store increasing numbers of data elements. We have also defined purge-and-collapse statements to effectively manage the hold-back time for the monitored parameters, and reset them in case of configuration changes that would void the previous observations. We are further investigating the requirements for conditional sampling statements to enable on-demand collection, e.g. to create snapshots of network states in case of errors.

C. Data manipulation statements

Network parameter values are either provided by direct inquiry through the responsible connector module, or through retrieval of stored data from the storage engine. By default, a

`SELECT <var_name>[,<var_name>]* FROM <device>`

will yield the current state of a parameter. More complex statements that include functional operations on parameter values are possible through querying the storage engine as

`SELECT <function_op>,`

where the function `function_op` takes `storage_name` or constants as input. Those may be hard-coded, or result of nested function operations. For example,

`SELECT AVG(ADD(in_err,out_err),5),`

would yield the average value over the last five sum values of the stored parameters `in_err` and `out_err`.

Updating network parameters is possible by calling

`UPDATE <device> <var_name> = <function_op>|<constant>,`

whereby this command is always carried out directly on the network device.

D. Triggered execution

Network reconfiguration statements will execute as a result of changes in the stored parameters that match a certain condition. One example would be a rapid increase in packet errors on one network interface, which may indicate a failure of the device. All triggered executions are carried out by the ULLA-X core watchdog, which maintains a direct connection to the storage engine. As it only operates on stored values, the conditional execution statements only need to be evaluated once the storage engine modifies the stored values that are referenced. This way, the periodicity of the watchdog is implicitly defined through the periodicity of updates through the collector. The elementary conditional execution statement,

`IF <cond> EXECUTE <command>[,<command>]*,`

takes singular and chained condition statements as inputs. Its grammatic definition

`cond := <cond> {||,&&,XOR} <cond>`
`cond := {<function_op>,<constant>, <storage_name>} {==,!=,<,>,<=,>=} {<function_op>,<constant>,<storage_name>}`

covers a variety of options to trigger conditional execution. For the aforementioned example, a conditional command may be

`IF MIN(in_err,5) > 1000 EXECUTE UPDATE switch0 is_up = 0, UPDATE switch1 is_up = 1,`

to move operations from switch0 to switch1 if over the last five rounds of data collection at least 1000 errors were counted.

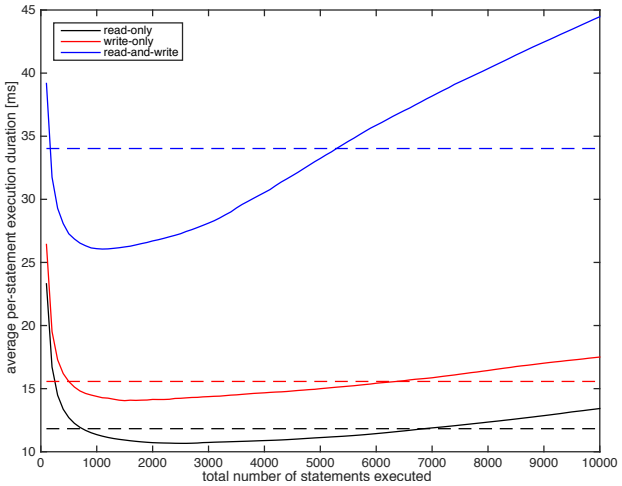

Fig. 2. Statement execution duration of prototype ULLA-X implementation.

Note that real command calls would also reset the monitoring statements and install monitors for the newly activated switching device.

IV. Empirical Results

We have developed a prototype to demonstrate the features of our proposed middleware. While the core of this ULLA-X implementation is generic, a vendor-specific connector module had to be created. We have opted to use National Instruments (NI) devices as the first target networking platform, because they are widely used in academic experimentation and industrial automation setups. More recently, NI hardware and software have become available to e.g. emulate sophisticated radio communication systems such as 5G prototypes, which, due to the freedom to implement highly customized setups, in practice often face a similar configuration complexity and diversity problem as productive network setups.

For common wireless communication tasks, low latency reconfiguration is crucial. Hence, we have initially simulated a highly loaded system running a Wi-Fi prototype. First, we sent consecutive `SELECT` statements to collect parameter values. Each command was sent immediately after the result of the previous read had been provided by the blocking retrieval operation. Then, we repeated the experiment with parameter-modifying `UPDATE` statements. In the final experiment, we alternated between read and write operations, and used the integer output of the read operation as an input to the write operation. In order to enforce potential cache misses, the write operation incremented the retrieved integer value in each cycle.

Figure 2 shows the average per-statement execution time over the total number of consecutively executed statements, where the dashed lines denote the respective average for all numbers of total executions. We find that the average for a read statement is approximately 11.8 ms, while the average writing operation is approximately 15.6 ms. Both values indicate that at the currently this vendor's API is too slow to implement heavy control loop duties, but that the reconfiguration time is reasonable for infrequent reconfiguration requests. Furthermore, the execution time is superlinear; after the initial decline that origins from the reducing impact of connection ramp-up and teardown time on the total runtime, some additional delays from non-linear processing units can be observed. Our final experiment with alternating read-write operations reveals that there is presumably a cache implemented in the vendor API, as we note that that the average duration of a paired read-write, 34.0 ms, is longer than then sum of the individual averages. Nevertheless, our data also shows that cache consistency is preserved, i.e. that no intermediate values are lost.

V. Conclusions

In this paper we have proposed ULLA-X, a middleware for post-deployment operations and maintenance of software-defined and legacy communication networks. ULLA-X aims at easing network management in potentially large installations. Our vision for ULLA-X is to resemble an autonomic nervous system for networks, capable of quickly reacting to state changes and optimizing network behavior.

We have presented the ULLA-X architecture, which decomposes generic and vendor-specific functions to allow rapid adaptation in current installations. We presume that the real strength of our middleware will be realized if used in combination with complex reasoning and decision making tools, as it decouples the (potentially heavy) planning from the responsive actions required to achieve the overall optimization targets. With the ULLA-X PL, we have provided the foundation for a novel network programming language that unites aspects of declarative programming with active device management.

Our initial results from a prototype implementation are promising, showing low increases in delay after introduction of the middleware. In our future work, we plan to broaden the range of supported platforms and investigate further extensions to our programming language. We are particularly interested in deploying ULLA-X into large production environments.

References

[1] S. Chowdhury *et al.*, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. IEEE Network Operations and Management Symp. (NOMS)*, May 2014.

[2] N. Foster *et al.*, "Frenetic: A network programming language," in *Proc. 16th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP)*, 2011, pp. 279–291.

[3] X. Ge *et al.*, "OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 353–354, August 2014.

[4] A. Gember *et al.*, "Design and implementation of a framework for software-defined middlebox networking," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 467–468, August 2013.

[5] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, January 2015.

[6] S. Sezer *et al.*, "Are we ready for SDN? implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, July 2013.

[7] A. Shalimov *et al.*, "Advanced study of SDN/OpenFlow controllers," in *Proc. 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR)*, 2013, pp. 1:1–1:6.

[8] M. Sooriyabandara *et al.*, "Unified link layer API: A generic and open API to manage wireless media access," *Comput. Commun.*, vol. 31, no. 5, pp. 962–979, March 2008.

[9] R. Thomas *et al.*, "Cognitive networks," in *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*, H. Arslan, Ed. Springer, 2007, pp. 17–41.