Magellan: Generating Multi-Table Datapath from Datapath Oblivious Algorithmic SDN Policies

Andreas Voellmy[+] Shenshen Chen[*] Yale University[+] Xing Wang[*] Tongji University[*] Y. Richard Yang[*+]

ABSTRACT

Despite the emergence of multi-table pipelining as a key feature of next-generation SDN data-path models, there is no existing work that addresses the substantial programming challenge of utilizing multi-tables automatically. In this paper, we present Magellan, the first system that addresses the aforementioned challenge. Introducing two novel, substantial algorithms, map-explore and table-design, Magellan achieves automatic derivation and population of multi-table pipelines from a datapath-oblivious, high-level SDN program written in a general-purpose language. Comparing the flow tables generated by Magellan with those produced from standard SDN controllers including OpenDaylight and Floodlight, we show that Magellan uses between 46-68x fewer rules.

CCS Concepts

•**Networks** → **Programming interfaces;**

Keywords

SDN, Programming model, Multi-table pipeline

1. INTRODUCTION

Multi-table pipelining has emerged as the foundation of the next generation SDN datapath models. Avoiding key issues such as unnecessary combinatorial explosions to substantially reduce datapath table sizes, multi-table pipelining is essential for making SDN practical. At the same time, the introduction of multi-tables also adds additional SDN programming tasks including designing effective layout of pipelines and populating the content of multiple tables.

In this work, we investigate how to automatically derive and populate multi-table pipelines from datapath-oblivious algorithmic policies (AP) [1]. We choose the algorithmic policies model because it is highly flexible and hence poses minimal constraints on SDN programming. The model is also general, and hence, can be used to express other models. As a result of the generality, if we can compute high-quality multi-table pipelines for algorithmic policies, we can convert other policies into malgorithmic policies, and use algorithmic policies as a powerful intermediate language for implementing other high-level SDN programming models.

On the other hand, effectively utilizing multi-table pipelines from algorithmic policies can be extremely challenging, because APs are expressed in a general-purpose programming language with arbitrary complex control structures (*e.g.*, conditional statements, loops), and the control structures of APs can be completely oblivious to the existence of multi-tables. Hence, it is not clear at all whether one can effectively program multi-table pipelines from such APs. We refer to this as the *oblivious multi-table programming challenge*.

The main contribution of this paper is the development of Magellan, the first system that addresses the oblivious multi-table programming challenge. The core of Magellan consists of two novel, substantial algorithms: the map-explore algorithm and the table design algorithm. Specifically, the map-explore algorithm conducts a novel, efficient form of hybrid *symbolic* (map) and *direct* (explore) execution of a multi-table oblivious program written in a general-purpose language, resulting in a data-structure called *explorer graph*. The table design algorithm partitions the dataflow graph of program and merges tables.

2. ARCHITECTURE COMPONENTS

The high-level objective of Magellan is simple to state: automate the tasks of table design and table population for general-purpose APs.

To achieve the goal, Magellan introduces a sophisticated compiler and runtime system shown in Figure 1.

- The static analysis and transformation proceeds in two steps: native compilation, and bytecode rewriting. A native compiler converts the user program to an objective code format to remove the extra complexity of high-level programming language which makes program analysis complex. We refer to the result as the bytecode. The purpose of bytecode rewriting is first to identify and organize the compact-mappable statements which can be represented

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil
 ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2959064

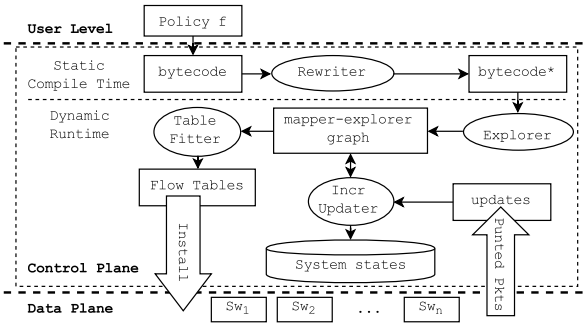

Figure 1: Magellan system components and work flow.

by a compact flow table with a small range output into `brk` statements and others into $xblocks$.

- The dynamic runtime part has two key components: Explorer and Table Fitter. The goal of Explorer is to generate the mapper-explorer graph whose nodes indicating instructions and links indicating control flow between instructions. By exploration of the program, a node in the graph includes all possible values for inputs and outputs of the instruction generating the node. After generating the mapper-explorer graph, a simple algorithm can generate flow tables from the graph. Finally, a table design algorithm in the Table Fitter merges tables to reduce the number since the number of flow tables is limited in real switches. In the next section, we will show a main step by using an example in the table design. The Incremental Updater will directly update mapper-explorer graph and then Magellan recomputes the content of flow tables.

3. EXAMPLE: AP TO DATAFLOW GRAPH

Here we give an example to show the translation from an AP program to a dataflow graph which is used in the table design. Bellow is the AP example:

```
// Program: Static-Example
onPacket(p) {
 x = macSrc;
 if (x > 4) {y = hostTable[macDst];} else {y = 1;}
 egress = [2 * y]; }
```

To remove language-specific constructs and simplify program analysis, we convert programs to a generic, simple, streamlined labeled instruction set (IR)) that uses conditional and unconditional jumps for all control flows, and also we do a simple compile optimizaiton to replace all x with macSrc and remove x = macSrc:

```
L2: cjump (macSrc > 4) L3 L5
L3: y = hostTable[macDst]
L4: jump L6
L5: y = 1
L6: egress = [2 * y]
```

In order to generate Magellan dataflow graph, we also need to convert jump statements (belong to control flow) to data dependency. So we introduce guard variable (g in the following example):

```
L1: g = (macSrc > 4)
L2: if g: y = hostTable[macDst]
L3: if !g: y = 1
L4: egress = [2 * y]
```

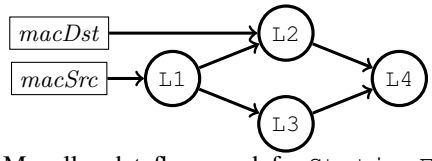

Figure 2: Magellan dataflow graph for `Static-Example`.

System	Hosts	Rules	Time (s)	Med RTT(ms)
POX	70	18787	96	9.7
Floodlight	70	4699	37	2.1
OpenDaylight	70	4769	32	0.6
Pyretic	70	-	> 1500	
Magellan	70	142	25	0.3
POX	140	13107	389	11.9
Floodlight	140	16451	200	6.1
OpenDaylight	140	19349	150	1.2
Pyretic	140	-	-	
Magellan	140	282	123	0.6

Figure 3: End-to-end performance comparison.

Then we generate the dataflow graph for this program as shown in Figure 2. The table design algorithm will partition the dataflow graph into regions, and merge all nodes in one region. The dataflow graph guarantees the merging is correct comparing with control flow graph.

4. PRELIMINARY EVALUATIONS

We compare Magellan with a range of state-of-the-art commercial and academic SDN systems, including OpenDaylight, Floodlight, POX, and Pyretic. We evaluate all systems using Open vSwitch (OVS) version 2.0.2, and conduct evaluations in a range of settings. In this poster, we report the results of the L2-learning-and-routing policy, because it is available in each system from the system's authors (with minor variations). Specifically, for each system, after allowing appropriate initialization of hosts and controller, we perform an all-to-all ping among the hosts, record the RTT of each ping, measure the time for all hosts to complete this task. After completing the task, we retrieve and count all Openflow rules installed in the switch. Figure 3 lists the number of rules, task completion time, and median ping RTT [1]for each system with $H = 70$ and $H = 140$ hosts and. We observe that for 70 hosts, Magellan uses 33x fewer rules than OpenDaylight and Floodlight, while for 140 hosts, Magellan uses between 46-68x fewer rules than other systems. This rule compression is due to leveraging multi-table pipelines: all other systems generate rules into a single table, and therefore generate approximately H^2 rules, while Magellan generates approximately only $2 * H$ rules.

5. REFERENCES

[1] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak. Maple: Simplifying sdn programming using algorithmic policies. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 87–98. ACM, 2013.

[1]Tests of Pyretic at both 70 and 140 hosts failed and these measurements are therefore omitted.