2025

Lucid

FUTURE INTERNET FINAL ASSIGNMENT

JUDITH HERSHKO & YONAH THIENPONT

INTRODUCTION

Lucid is a data plane programming language and framework designed to provide efficient, flexible solutions for networking applications. It simplifies the implementation of data-plane algorithms and structures by offering simple and modular abstractions. Lucid is built to run on programmable networking hardware, such as the Tofino ASIC, allowing for high performance while maintaining ease of use. With Lucid, programs are often significantly shorter—up to 10 times fewer lines of code—compared to traditional alternatives like P4. Additionally, its syntax resembles programming languages such as Go, Python, and C, making it more accessible and readable. This paper will explain what we implemented for this project and the steps used to generate our results. From writing code in lucid, compiling it to generating the equivalent p4 code.

INSTALLATION

The installation process is described here in detail. Following the steps is straightforward and will ensure that you can use Lucid and explore all its functionality. We will go through some common functionalities to have a better understanding of the lucid language. We will focus on the event-based functionality as it forms the core of making rules and switches in lucid.

FUNCTIONALITIES

EVENT DECLARATIONS

An event is declared using the following syntax:

```
event foo(type1 id1, type2 id2, ..., typeN idN)
```

This statement defines an event named ***foo*** that accepts a list of arguments ***id1, id2, ..., idN*** of corresponding types ***type1, type2, ..., typeN***. Events serve as fundamental building blocks for structured communication in network systems, encapsulating specific data types.

EVENT VALUES

An event value is instantiated using the syntax:

```
event x = foo(arg1, arg2, ..., argN);
```

Here, an instance of the event ***foo*** is created with the arguments ***arg1, arg2, ..., argN*** and assigned to the variable **x**. This allows the system to dynamically create event instances with varying content while maintaining type safety.

EVENT GENERATION

Events can be serialized and transmitted to designated destinations using **generate** statements. Three primary forms of event generation are supported:

- Port-Specific Generation:

```
generat_port(n, x);
```

In this form, the event ***x*** is serialized into a packet and transmitted to the designated port ***n.***

- Group-Specific Generation:

```
generate_ports(g, x);
```

This form allows the event ***x*** to be sent to all ports within the specified group ***g***.

- Recirculation:

```
generate(x);
```

The event ***x*** is queued for recirculation, enabling it to be reprocessed within the current network switch.

PACKET EVENTS

Packet events encapsulate arguments directly into packet structures. The declaration syntax is as follows:

```
packet event foo(type1 id1, type2 id2, ..., typeN
idN);
```

This form is particularly useful for constructing custom network packets where each argument is tightly integrated into the packet structure.

EXAMPLE: IP PACKET CREATION

An example of a packet event declaration is:

```
packet event eth_ip(eth_hdr_t eth, ip_hdr_t ip,
Payload.t pl);
```

In this case, the event ***eth_ip*** encapsulates Ethernet and IP header fields (***eth_hdr_t*** and ***ip_hdr_t,*** respectively) alongside a payload (***Payload.t***). This structure supports the generation of standard IP packets. However, custom parsers are required for deserializing these packets when received.

An overview of these functionalities can be found here in our repository.

EXAMPLES

In the example directory, we've put some examples to use lucid in creating switches. Some more examples can be found throughout the lucid project in tutorials and examples.

FILTERED TABLE

The files in this directory outlines a **network filtering and handling system** that processes Ethernet and IP packets. Its primary goal is to decide the fate of incoming packets. That is, deciding whether to forward them, drop them, or generate a report. This is based on predefined filtering rules declared here.

CORE FEATURES:

- Defines packet types (Ethernet and IP).
- Implements filtering logic using a table (***filter_table***) to match packets against rules based on source and destination IPs.
- Handles specific events like forwarding packets, dropping them, and creating reports.

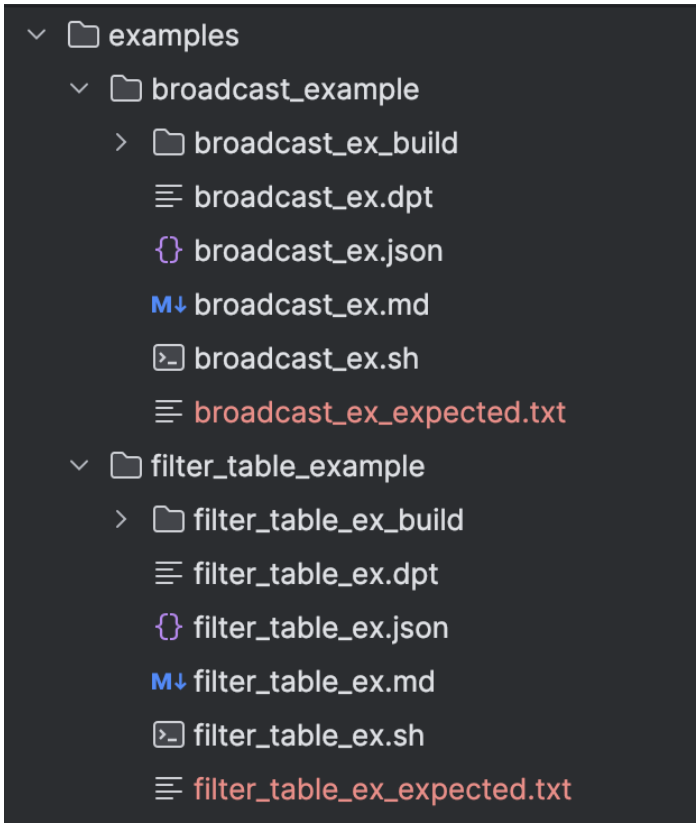
BROADCAST

This file demonstrates a **broadcast mechanism in a defined network topology**. It manages how events (e.g., broadcasts) are disseminated across a group of interconnected switches.

CORE FEATURES:

- Defines a simple network of switches and their connections.
- Implements a broadcasting process where events are sent to all switches in the network group.
- Includes functionality to determine connected ports and propagate events efficiently.

Both directories are accompanied by a readme file (*broadcast_ex.md* and *filter_table_ex.md*) for an in-depth explanation of what the examples do, a json-input file for example data to test out the functions and a script to translate the Lucid program into p4, optimize it and produces a build directory with a P4 program, Python control plane, and a directory that maps control-plane modifiable Lucid object names to their associated P4 object names. The directory will look as follows after running filter_table_ex.sh and broadcast_ex.sh:



Broadcast_ex_build and filter_table_ex_build contain all the necessary files for the p4 program. The text-files contains the default output generated by lucid when running the functions from the script files.

When exploring the files in the build-directories we see that the generated p4 code is much larger and more complex than the lucid-based program we wrote. This highlights one of the main advantages of using lucid.

SUMMARY

This paper explored the use of the Lucid programming language in creating network switches, focusing on its event-based functionalities for packet filtering and broadcasting. Lucid offers a highly modular and efficient way to implement network systems, making it an ideal choice for applications requiring flexibility and performance. Through the examples provided, such as the filtered table for packet processing and the broadcast mechanism for event dissemination, we demonstrated how Lucid simplifies complex network tasks with less code and greater clarity compared to traditional methods. The resulting P4 code, generated from Lucid programs, shows significant promise for optimizing and implementing networking applications, making Lucid a powerful tool for the future of programmable networking hardware.