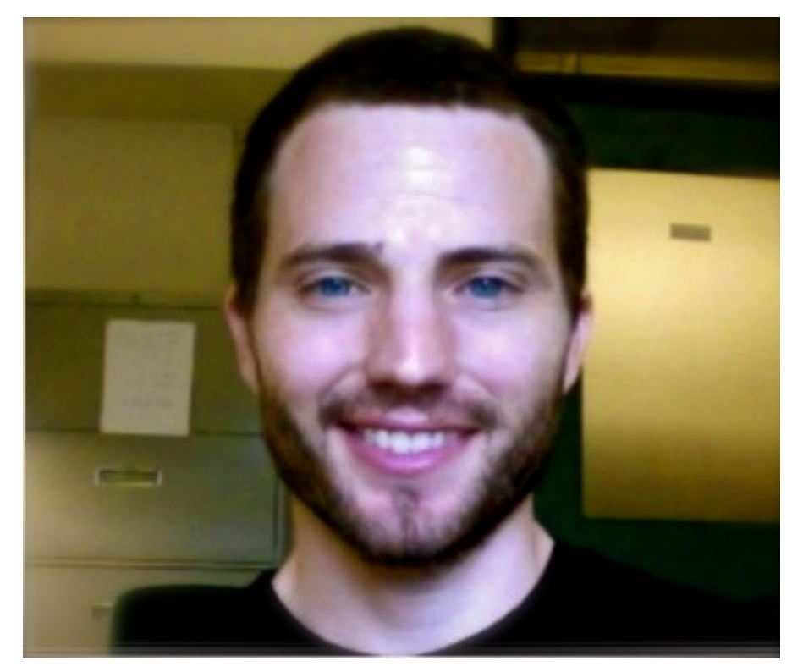
Magellan: Compiling Datapath-Oblivious Packet Processors to P4 Programs

Andreas Voellmy (`andreas.voellmy@gmail.com`) & Y. Richard Yang
Yale University, Dept of Computer Science



Problem

Datapath packet processors are difficult to program due to the need to program low-level constructs such as match-action tables and priorities. Furthermore, the more details that a program specifies explicitly, the less portable the program becomes. Previous work has proposed high-level programming models (e.g., Frenetic, FlowLog, and Maple) to hide packet processor details and achieve cross-platform portability. However, no existing work can handle multi-table pipelining, which is a key feature of the next-generation SDN packet processors.

Solution

We present Magellan, the first system that automatically synthesizes both multi-table pipelines and corresponding control programs from a single, high-level forwarding program written in a familiar, general-purpose, *datapath-oblivious* programming language. Magellan leverages static analysis to automatically extract tables, and applies a novel runtime algorithm called *Map-Explore* to proactively populate table entries based on current system state. Magellan supports both P4 and OpenFlow 1.3+ switches, achieving cross-platform portability for the state-of-art datapath packet processors.

Magellan Programming Model

In Magellan, the user writes a function that processes packets in a language with familiar, algorithmic features like variables, loops, and data structures. Conceptually, the user defines a function that is repeatedly executed on packets, taking packets in and producing some number of modified packets out:

p_0
$onPacket()$
p_1

To illustrate, the following is an example program that drops packets destined to some blacklisted IP prefixes, learns host location bindings, and forwards based on destination MAC address:

```
1  function onPacket() {
2      if (deny()) {egress = []; return; }
3      macTable[ethSrc] = ingressPort;
4      d = macTable[ethDst];
5      if (d == null) {
6          egress = members ports;
7      } else {
8          egress = [d];
9      }
10     return;
11 }
12 function deny() {
13   if (ethType != 0x0800) { return false; }
14   routes = members blacklist;
15   for (i = 0; i < len routes; i = i + 1) {
16     route = routes[i];
17     if (ipDst in route.addr<route.len>) { return true; }
18   }
19   return false;
20 }
```

This program exhibits several features of the programming model:

Input A set of distinguished *input variables* denoting values for packet fields and ingress info, such as $ethSrc$, $ethDst$, $ingressPort$.

Output A set of distinguished *output variables*, such as $egress$, denoting the ports on which to send a packet.

State User-defined state, (e.g. $macTable$, $blacklist$) which can be simple variables, sets, and maps, and can be accessed (e.g., line 4) and updated (e.g., line 3) by the packet handler.

Computation Full arithmetic (add, multiply, modulo, ...), data structures such as lists and records (e.g., routes), variables, assignments, conditional, loops (line 15), function calls (line 2).

Critically, in contrast to OpenFlow controllers or P4 configuration/-controllers, Magellan programs are:

Familiar : Constructs have ordinary, regular semantics. There are no oddities, such as out of order execution or only integer variables.

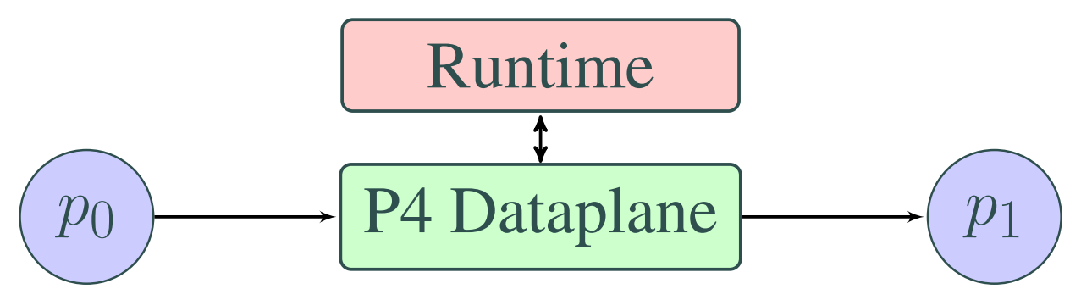
Expressive All programs are supported; that is, no missing arithmetic expressions or missing state update operations.

Safe : Errors in low-level dataplane specification/config are removed.

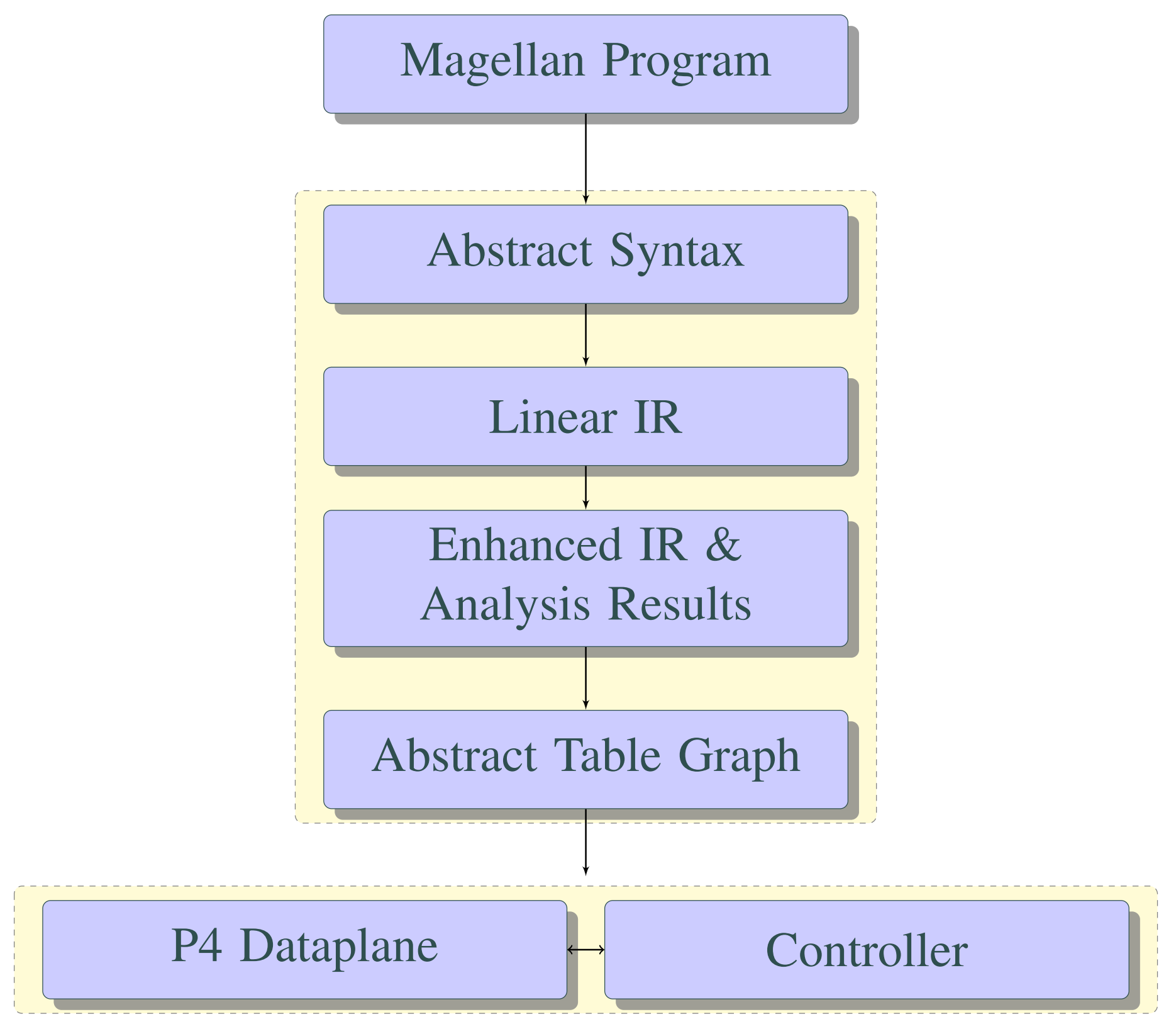
Portable : The program is independent of southbound forwarding models and protocols.

Magellan Compiler & Runtime

Based on a more general programming model, a Magellan program can include both computations that can be implemented by P4 and computations that are either impossible (e.g., table updates) or infeasible (e.g., `ethDst modulo 7`) in P4. Therefore, the Magellan compiler generates both (1) a statically designed P4 dataplane and (2) a runtime controller to perform computations missing in P4:



For computations that can be mapped to P4, Magellan must statically design tables, compound actions, and acyclic control flow. In addition it must choose how to encode variables and their values (including structured objects) into P4 metadata. These tasks are accomplished in the Magellan compiler pipeline, consisting of the following steps:



Intermediate Representation (IR)

Magellan uses a simple linear intermediate representation (IR) with two new, non-standard IR instructions: one for accessing input variables and state, and another for updating state. The following shows the instructions:

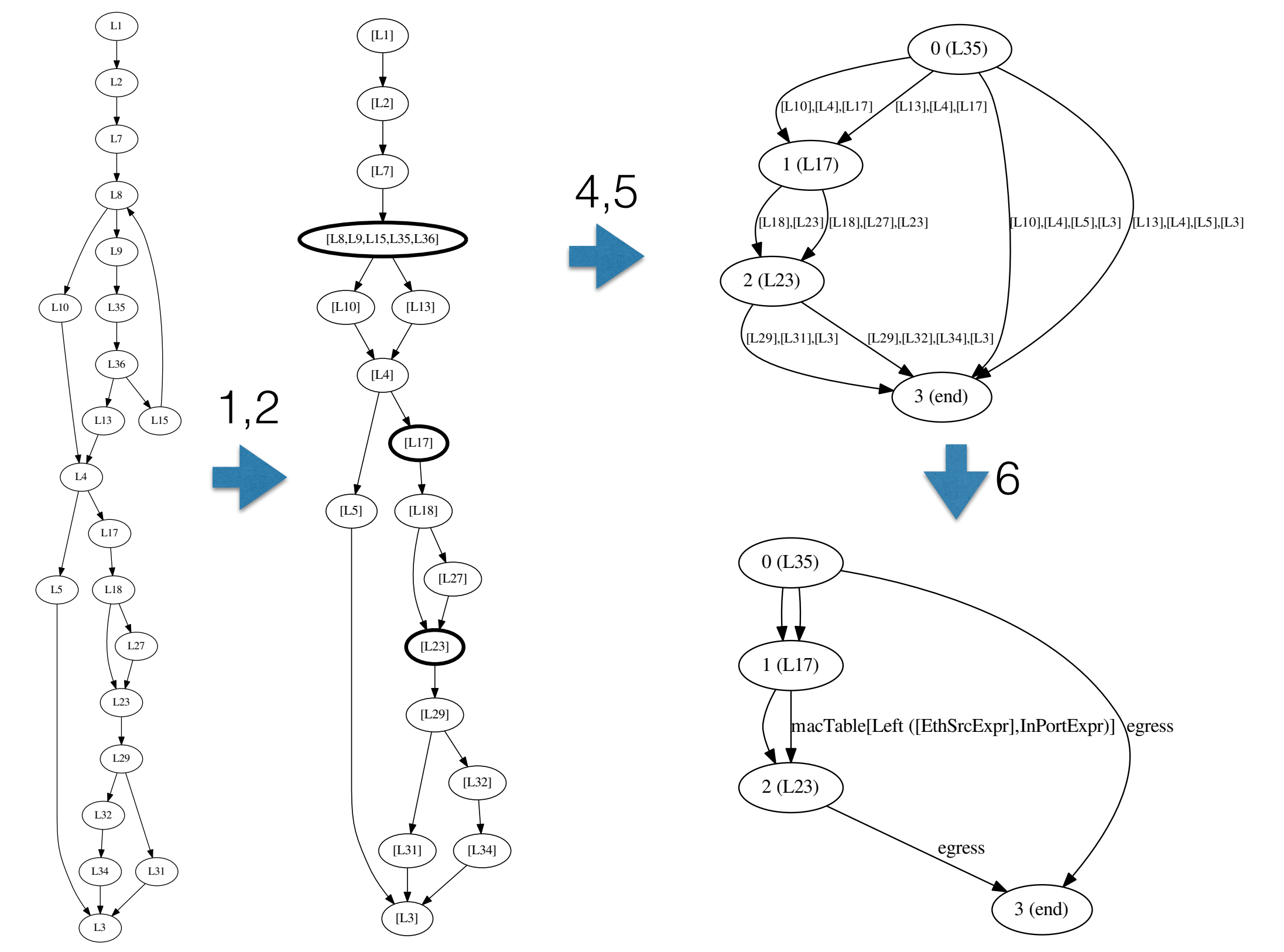
$$\begin{array}{lll}
\langle instruction \rangle & ::= & \langle ident \rangle \text{ `='} \langle accessexpr \rangle \\
 & | & \text{`}\texttt{insert}\text{'} \langle tableident \rangle \langle expr\text{-}list \rangle \langle expr \rangle \\
 & | & \langle etc \rangle \\
\langle accessexpr \rangle & ::= & \text{`}\texttt{readfield}\text{'} \langle fieldexpr \rangle \\
 & | & \text{`}\texttt{lookup}\text{'} \langle tableident \rangle \langle keyexpr\text{-}list \rangle \\
 & | & \text{`}\texttt{contains}\text{'} \langle tableident \rangle \langle keyexpr\text{-}list \rangle \\
 & | & \text{`}\texttt{members}\text{'} \langle tableident \rangle \\
 & | & \text{`}\texttt{boolean}\text{'} \langle fieldexpr\text{-}expr\text{-}pair\text{-}list \rangle
\end{array}$$

In a first phase, Magellan compiles the program's abstract syntax to IR in a straightforward way, translating every field access with a `readfield` access instruction. In a second phase, Magellan leverages dataflow analysis to rewrite expressions which use the field into specialized forms, such as field-based table lookups and boolean expressions.

Table, Action, & Control Flow Design

Magellan performs an initial table design using the following approach:

1. Form the strongly connected components (SCC) of the control flow graph (CFG) of the extended IR.
2. Allocate one table for each simple SCC that consists of a single access instruction and each compound SCC that includes one or more access instructions.
3. For every pair of tables s, t, we obtain the SCC-paths from the start label of s to the start label of t which traverse only SCCs that are not allocated tables.
4. The resulting tables and SCC-paths form the *abstract table graph (ATG)* for the input program.
5. For each edge of the ATG, a parameterized compound action is computed which simulates the effect of executing the code path along the SCC path.



The left-hand side graph is the cyclic control flow graph. The second graph is the strongly-connected control flow DAG, where bold nodes contain access instructions which will correspond to tables. Top-right is the abstract table graph with SCC-paths labeling edges, while bottom-right is the abstract table graph with each edge labeled by an action effect (writes and updates).

P4 Realization

In a final step, Magellan compiles the abstract table graph to a concrete P4 program, using metadata instances for program variables and generates code to interface with the P4 program's runtime API.

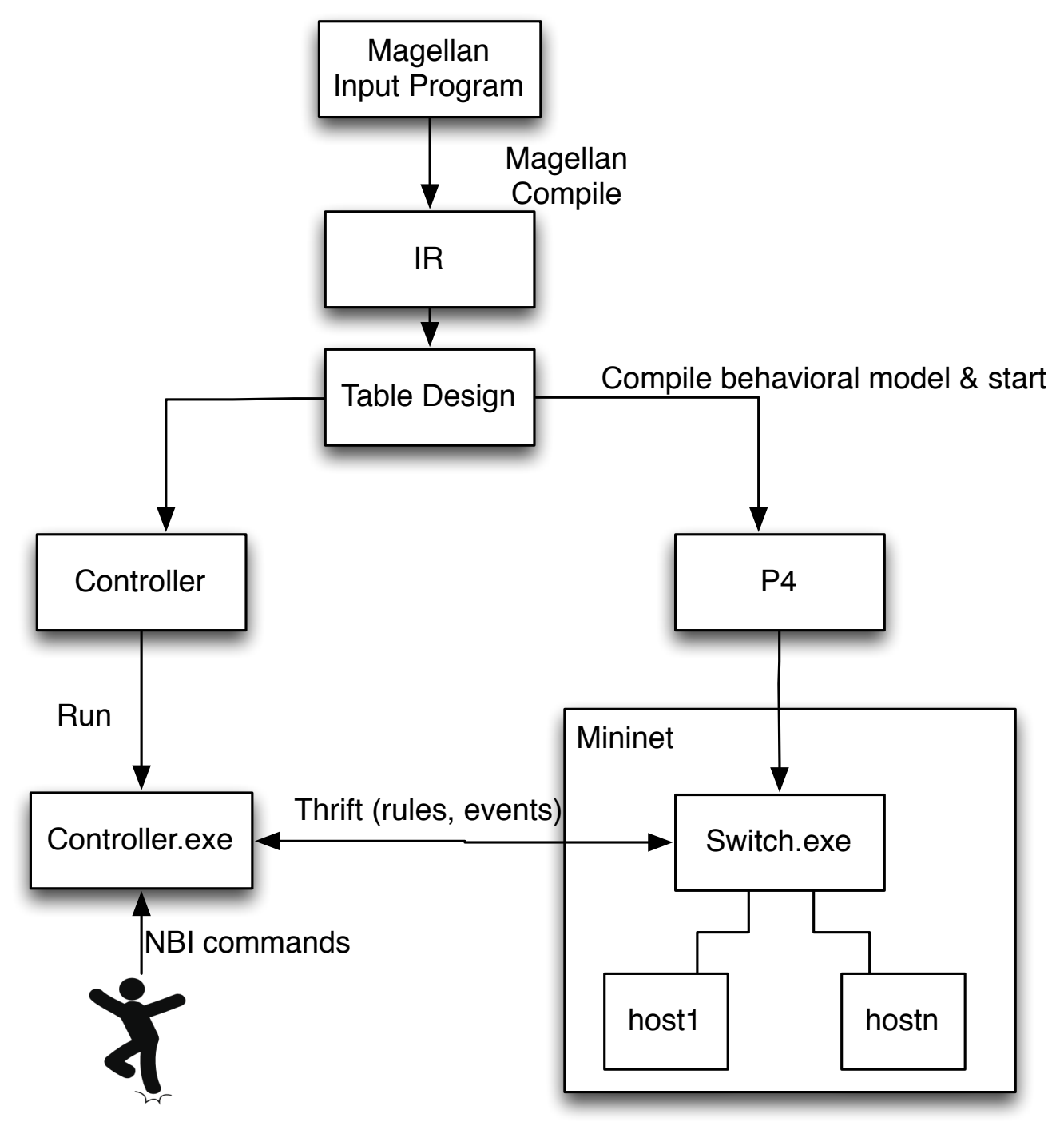
Runtime: Rule Population

Proactive Population In order to minimize or, in some cases, entirely eliminate packets failing to forward in the data plane, Magellan proactively populates dataplane match tables when possible. Given the values of state components (e.g. `macTable`), the Magellan runtime applies graph exploration to compute (1) the set of variable bindings that can reach each table, and (2) for each reaching variable binding and outcome possible at a table, the parameterized compound action equivalent to the program execution starting at that table with the given outcome and variable binding. With these values, the runtime can populate tables proactively.

Reactive Population Exploring all outcomes at $ReadField(f)$ instructions would be infeasible (e.g., $srcMac$ field has 2^{48} outcomes); instead the outcomes are determined by *sampling* the field values observed for f on real packets which reach the instruction. The Magellan compiler identifies sampled tables and the runtime generates default entries that generate "packet-ins" on misses in sampled tables.

Demo

We demonstrate the Magellan system targeting P4 using the *behavioral model* P4 software switch using mininet to configure interfaces and hosts.


Magellan Input Program
Magellan Compile
IR
Table Design
Compile behavioral model & start
Controller
P4
Run
Mininet
Controller.exe
Thrift (rules, events)
Switch.exe
NBI commands
host1
hostn