<https://p4.org/p4/prototyping-in-p4.html>

P4 stands for Programming Protocol-Independent Packet Processors. It has been developed as a next step in the evolution of Software-Defined Networking (SDN). As you may know, the classical telecommunications architecture can be divided into data plane, control plane and management plane. Data plane is the layer, where data packets are being processed and forwarded, while the control plane decides how these packets should be handled. **P4 has been designed to enable data plane programming.** Thus, using the P4 language you can specify which packet’s headers can be processed and which actions can be performed on packets. The data plane programming was the missing link in the software-based network systems, where control plane and management plane are programmable already. It is a high-level Domain-Specific Language (DSL) dedicated for programming of network devices. It allows to specify the format of packets (protocol’s headers) to be recognized by network devices and actions to be performed on incoming packets (forwarding, headers modification, adding protocol header, etc). Nevertheless, the P4 language is not consumed directly by the network device, but it must be compiled to the source code for particular platform (some target-specific binary).

The P4 program is composed of three main sections: Protocols definition (data declaration), Parser Logic (Parser & Deparser) and a number of control blocks containing Match-Action tables. The first section defines the protocols headers that the network device will be able to recognize.

The Parser Logic is a finite state machine defining the steps to read and parse incoming packets. Visually, the Parser can be represented as a cyclic graph, in which every node processes a protocol’s header.

P4 program programmer must define a number of control blocks, which contains Match-Action tables. The definition of simple IPv4 forwarding table can be implemented as follows:

table routing\_table {

key = {

hdr.ipv4.dstAddr: lpm;

}

actions = {

ipv4\_forward;

drop;

NoAction;

}

default\_action = NoAction();

}

The above routing\_table reads the IPv4 destination IP address and matches it based on the Longest Prefix Match algorithm. Then, on packets matching the rule there can be three actions performed: ipv4\_forward, drop or NoAction.

The last part is to define the Deparser, which defines the order of packet’s headers for outgoing packets.

<https://p4.org/p4/p4-language-evolution.html>

P4 aims to fundamentally change how we design network systems. With P4 you start from your system design requirements, write a P4 program to describe how your system needs packets to be processed, and then compile your program to tell the forwarding elements what to do.

Chips are being built this way, and we expect more in the future. These chips are protocol independent (their hardware specs don't mention any protocols, and they have programmable packet parsers), yet they have very high forwarding performance.

<https://p4.org/p4/an-open-source-p4-switch-with-sai-support.html>  
Whether you build your own chips or use off-the-shelf merchant silicon, it can be frustrating to write software for fixed-function switch chips. You are limited by the set of capabilities baked into the silicon. You can't add new features, and you have little or no control over how the switch resources are allocated. This should improve over the next few years as P4-programmable switch chips become available allowing us to: (1) Define and customize how a switch processes packets, and (2) Auto-generate an API to interface the chip to the switch software