Notes on OF-PI Instruction Set

<<Editor’s Note: This is an initial draft and needs a lot of review/comment/revision. Ben M-C dreamed up the hardware abstraction and this needs review/discussion and may change. For example, the P4 specification seems to be based on a slightly different hardware abstraction; however, it is not clear if this is just different to support a higher-level language or if it requires a different hardware abstraction for the intermediate representation (IR) as well.>>

The OF-PI instruction set is intended to provide a necessary and sufficient base for configuration phase programming of a wide variety of packet switch datapaths. Each instruction should have a purpose for being included in the set. In general, actions that can be programmed using other (presumably more primitive) instructions should not be included in the primitive instruction set since it can be defined as a function instead; however, ultimately this is a judgment call as there may be advantages to including some instructions that could be programmed in other ways (e.g., to avoid forcing allocation of additional metadata).

A hardware abstraction is needed to provide a context for the instruction set. The hardware abstraction defines elements that control the execution of instructions that operate on well-defined data stores. The hardware abstraction is not intended to constrain implementations beyond providing the ability to program datapath behavior in a deterministic way. That is, each element of the hardware abstraction (model) is needed to enable unambiguous programming; however, these elements need not be implemented exactly as in the hardware abstraction as long as the net behavior is consistent the model.

Hardware Abstraction

An example hardware abstraction is shown in Figure 1. This model is used as the context for the OF-PI instruction set definitions below.

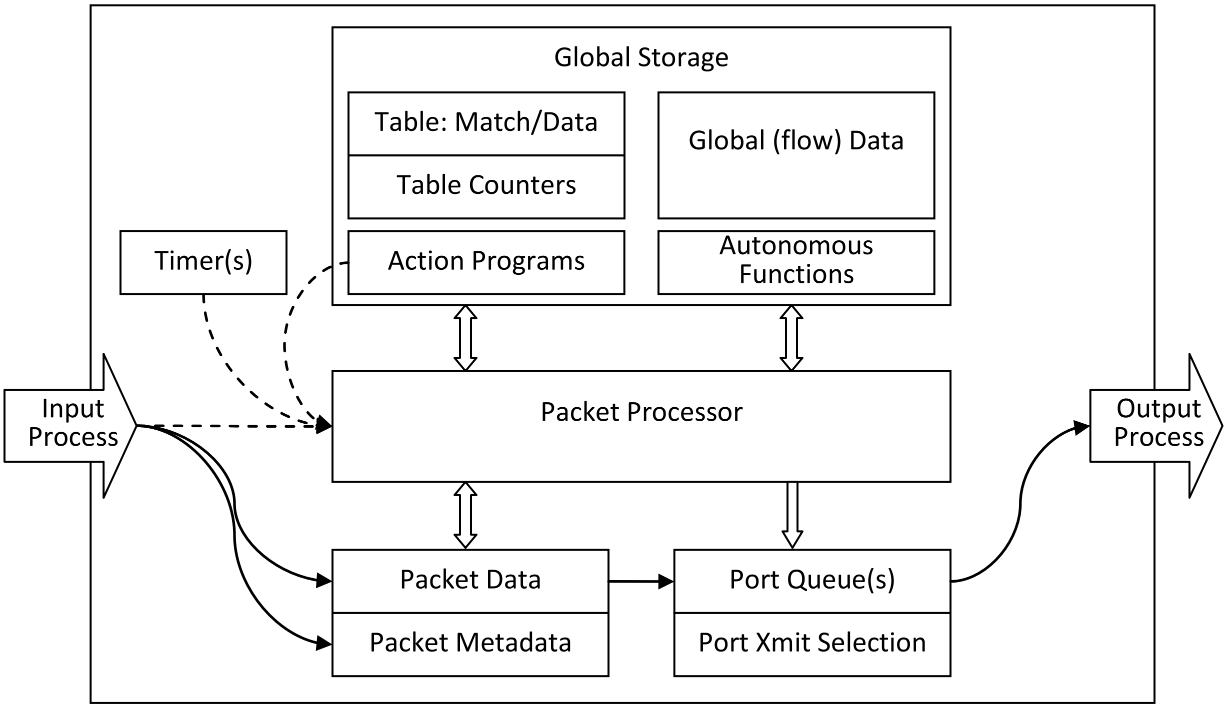

Figure 1 – Hardware Abstraction

The elements of the hardware abstraction are described below.

- Packet processor
    - Packet processing is driven by Match-Action Tables (MAT). Each table has a defined key structure, possibly including priority, used to match and return a unique table entry. Each table entry comprises a reference to an action function and zero or more parameters for that function. MATs can also include counters that track the number of times each entry is matched.
    - The packet processor can be considered to have some number of registers that contain state data related to the execution of packet processing (e.g., table pointer, instruction pointer, packet pointer, data address base register(s), etc.); however, these registers do not have to be explicitly represented unless the programmer is allowed to control their contents.
    - The packet processor implements the OF-PI instruction set and is capable of executing action functions written in this instruction set. The packet processor may also have built-in functions that have a defined behavior but are not explicitly programmed in the OF-PI instruction set.
- Memory blocks
    - Packet Data – primarily header fields, but the entire packet may be made available. The header fields can be referenced symbolically, relative to a set of parsing rules for the PACKET_TYPE, or referenced indirectly relative to a packet base register and <offset, length> values.

- Packet Metadata – per packet storage that includes some pre-populated data (e.g., IN_PORT, PACKET_TYPE, etc.) and some data space that can be defined by the programmer.
    - Action Set – a portion of per packet metadata containing actions to be executed when the pipeline ends or on command. This can be considered a specialized part of Packet Metadata. If this storage is not shared with the rest of packet metadata it must have a separate capacity parameter.
- Global Data – data that can be referenced from any action program (flow metadata, meter counters, etc.); flow metadata can be referenced by direct or indirect address; can use symbol association (global flow datum, or global flow_ID plus local datum)
- Tables – a table is a set of <match, data> elements, match (a match condition, defining the size and structure of the key) plus data (action data, size and structure) define a logical table; the mapping of logical tables to physical tables is implementation dependent. Table entries can be referenced using the match information. Logical tables can employ a variety of matching methods, including:
    - Exact – match condition is an exact value
    - Masked – match condition is a value and mask
    - LPM – longest prefix match (value, prefix mask and associated priority)
    - Range – match condition is a value range
    - RegEx – match condition is a regular expression
      (Is this a table matching method or is it a different type of processing?)
- Table Counters – counters that can be referenced by table entry key. These are accessible by control functions (not shown in Figure 1) but not by the Packet Processor.
- Action Programs – storage for instruction blocks (re-entrant function definitions); functions can be named or unnamed (e.g., defined in flow table entry and automatically created and referenced) – a function definition is not stored in table entry data (the table entry only holds a function reference and parameter values)
- Autonomous Functions – these objects can be created in the context of a flow to control autonomous flow processing (e.g., OAM, protection switching, etc.). They can register for events and are invoked when a registered event occurs (i.e., an action function is invoked when the event occurs). An autonomous function can have internal state that can be queried and sometimes modified by the controller.

- Port Processes
    - Input Process – input processing on a port can be configured in some cases. For example, initial packet parsing can be configured by a set of parsing rules to determine a set of header fields that can be referenced by packet processing actions. Or IP packet reassembly can be indicated by setting a flag (while the actual reassembly action is specified by reference and not directly programmed). The input process can also include implicit functions, such as setting initial packet metadata values (e.g., IN_PORT, IN_PHY_PORT, PACKET_TYPE, PACKET_LENGTH, etc.)

- Output Process – similar to input processing, output processing on a port can be programmed or configured or implicit. Output processing can include actions like MTU enforcement (discard or fragmentation), tunnel encapsulation, signal modulation, etc.
- Metadata transfer (virtual ports) – Currently OpenFlow allows creation of recirculation ports (called “virtual ports” here since physical and logical ports are already defined in OpenFlow) that act like logical ports but also convey some packet metadata. (Are these virtual ports needed?)

- Events
  - Packet Receipt – The OpenFlow pipeline is implicitly triggered by the receipt of a packet on a port, so this is an important event type.
  - Timer Expiry – to allow for autonomous actions a timer events are needed. These can also trigger processing; however, these events are associated with autonomous functions rather than the OF pipeline (MATs). Autonomous functions may generate packet receipt events that then trigger pipeline processing.
  - Exceptions – action programs can generate exception events and these may be associated with autonomous functions or action programs that handle these exceptions. Exceptions handlers run in the context of packet processing have access to packet data and packet metadata. Exception handlers may generate packet receipt events that then trigger pipeline processing.

Instruction Set

The OF-PI instruction set is described in the following table. This instruction set is intended to be suitable as an intermediate representation (IR) and so should include only those instructions that are necessary and sufficient for representing programmed datapath functions and should not be target-specific or protocol-specific.

<<Editor's Note: This is an initial draft and needs a lot of review/comment/revision.>>

Name	Parameters	Description	Use	Notes	From
These instructions are necessary					
output	*port_id*	Queue packet for transmission on the identified port.	Forwarding	Could add queue selection to this primitive. Currently queue is set separately.	OF POF
invoke	*function*, *parameter_list*	Call a function with the given parameters. After the function executes, execution continues at the instruction after **invoke**.	Action program structuring and code reuse.	Do functions need local data or can packet metadata play this role? Do we only need call by value for parameters? Is there a limit to the depth of the calling tree?	POF **group**?
write	*action*, *parameter_list*	Add an action to the Action Set.	Defer execution of actions (e.g., packet modifications) until later in the pipeline.	Do we need the Action Set concept? Can this be replaced by a model with two packet copies – original and modified?	OF
set_field	*base*, *offset*, *length*, *mask*, *value*	Set a field in packet data, packet metadata, or global data.	Many.	Should we include symbolic references? Is this sufficient for timers or do we need separate timer instructions?	POF P4 **modify_field**
inc	*value*, *dst*, *wrap*	Increment *dst* field by *value*, wrapping or not according to the *wrap* flag. *dst* is a <*base*, *offset*, *length*> tuple.	Process sequence numbers, byte counters, etc.	Should this be extended to include the Field Assignment and Saturation Attributes from P4?	POF **inc_field** P4 **add_to_field**
dec	*value*, *dst*, *wrap*	Decrement *dst* field by *value*, wrapping or not according to the *wrap* flag. *dst* is a <*base*, *offset*, *length*> tuple.	Process TTL, meter buckets, etc.		POF **dec_field**
copy	*src*, *dst*, *mask*	Copy one field to another, where *src* and *dst* are <*base*, *offset*, *length*> tuples.	Copy fields between tags, e.g., TTL or priority.	Is *mask* needed here?	OF P4 **modify_field**

Name	Parameters	Description	Use	Notes	From
add_header	*offset*, *length*	Add packet data space of size *length* bits at *offset* from the beginning of the current packet data.	Add a header or tag to packet data.		POF **add_field**
del_header	*offset*, *length*	Remove packet data space of size *length* bits at *offset* from the beginning of the current packet data.	Remove a header or tag from packet data.		POF **del_field**
jump	*location*	Continue execution at *location* in an action program.	Allow simple logic decisions to be made in action programs (to avoid creating additional tables in the pipeline).	Do we need both relative and absolute jumps or is just relative sufficient?	POF **relative_jump** **absolute_jump**
jump_c	*cond*, *location*	If *cond* is TRUE, continue execution at *location* in an action program, otherwise continue execution at the next instruction.	Allow simple logic decisions to be made in action programs (to avoid creating additional tables in the pipeline).	Need to define TRUE.	POF **conditional_relative_jump** **conditional_absolute_jump**
goto	*table*	Continue pipeline processing with the identified *table*.	Main pipeline control flow.		OF POF
			These instructions may be necessary		
lsl	*value*, *dst*	Logical shift left *dst* field *value* positions. *dst* is a <*base*, *offset*, *length*> tuple.			POF **sll_field**
lsr	*value*, *dst*	Logical shift right *dst* field *value* positions. *dst* is a <*base*, *offset*, *length*> tuple.	Process source route position indicator.		POF **srl_field**
or	*value*, *dst*	Replace the value of *dst* field with the Logical OR of *dst* field and *value*. *dst* is a <*base*, *offset*, *length*> tuple.			POF **or_field**

Name	Parameters	Description	Use	Notes	From
and	*value*, *dst*	Replace the value of *dst* field with the Logical AND of *dst* field and *value*. *dst* is a <*base*, *offset*, *length*> tuple.			POF **and_field**
xor	*value*, *dst*	Replace the value of *dst* field with the Logical XOR of *dst* field and *value*. *dst* is a <*base*, *offset*, *length*> tuple.			POF **nor_field**
nor	*value*, *dst*	Replace the value of *dst* field with the Logical NOR of *dst* field and *value*. *dst* is a <*base*, *offset*, *length*> tuple.			POF
These instructions may be useful					
add_header **copy_header** **remove_header**	*header_instance*			These seem to be based on a slightly different hardware model that has more innate knowledge of header structure. It may be possible to program these as action functions using other primitives. This needs study.	P4
truncate	*length*	Indicate that the packet should be truncated to *length* on egress.		What is this used for? How does it behave (the description seems to indicate delayed action)?	P4
resubmit	*field_list*	The packet is marked for resubmission.		What is the operation of the field_list? It seems to be based on a slightly different hardware abstraction (containing a Parsed Representation).	P4
ingress_clone	*profile, field_list*	A copy of the original packet is passed to the ingress parser as an independent packet instance.		What is this used for?	P4
egress_clone	*profile, field_list*	The packet is marked for cloning at egress.		How does this work (marking instead of actually cloning directly)?	P4

Name	Parameters	Description	Use	Notes	From
parse	*packet_type*	Parse the current packet header(s) and make header fields available (i.e., make references valid)	Re-parsing after a header that does not contain a "next protocol" field, e.g., MPLS.	Not needed as a primitive – can be implicit in setting PACKET_TYPE. Exception can be raised if parse rules do not support a header field reference in a particular context.	
search	*table*,*key*	Perform a lookup in *table* using *key* and return table data for the matching entry or NULL.	Allows explicit table lookup in a function definition. (This is in addition to the implicit lookup used in pipeline execution.)	Is this needed?	POF
checksum	*range*	Calculate a checksum over the indicated range of packet data.	Calculate header checksum (e.g., for IP or TCP) or packet checksum (e.g. Ethernet FCS).	Should this be a primitive or a built-in function? I.e., is there a need for the controller to control this behavior?	POF **Alg**?
set_reg	*value*	Set a base register to value.	Adjust packet data base pointer (reference for <offset, length> addressing).	Do we need this or can all addressing support be handled implicitly?	POF **set_packet_offset** **move_packet_offset**
add_entry	*table*, *cond*, *data*	Add an entry to *table*. flow_mod is a <*cond*, *data*> tuple consistent with the identified *table*. If an entry with *cond* already exists, its *data* is replaced.	Allow actions that modify flows (e.g., MAC learning, protection switching, etc.).	Do we want this as a datapath instruction or should embedded control functions operate in a different space, e.g., operate in control space triggered by a notification from the datapath function?	POF **add_table_entry** **set_table_entry**
del_entry	*table*, *cond*	Delete the entry matching *cond* from *table*.	Allow actions that modify flows (e.g., MAC aging, protection switching, etc.).		POF **delete_table_entry**
fork	*thread id*	Create a parallel thread of execution.	Allow explicit parallelism in the pipeline.	Do we need explicit control for parallel execution or can compilers handle this without the need for explicit programming?	
wait	*thread id*	Wait on completion of a parallel thread of execution.	Allow parallel execution in the pipeline to be completed before continuing.		

These instructions do not seem to belong in the primitive set

Name	Parameters	Description	Use	Notes	From
set_field_update_cksum	*base*, *offset*, *length*, *mask*, *value*	Set a field in packet data and update the IP or TCP checksum.	IP or TCP header changes.	This can be programmed as an action function, but it is protocol-specific so does not belong in the OF-PI primitive instruction set.	POF
set_flow_metadata get_flow_metadata		Set and get flow metadata in global data space.		These can be programmed using the **set** primitive.	POF
count	counter_ref, value	The given counter is incremented by value.		This can be programmed using the **inc** primitive.	P4
meter	*meter_ref*, *field*	The given meter is executed.		Can this be programmed as an action function using other primitives?	P4
order_enforce				How is this instruction defined?	POF

Addressing modes

Instructions that reference fields in packet data, packet metadata, or global data must support one or more conventions for referencing a data field. Some possible conventions are:

- Offset/length (from base register)
- Header field id (is this needed in the HAL/IR? Or is this just symbolic reference resolved during compilation?)
- Direct (global memory)

<<Editor's Note: Need to discuss which addressing modes need to be supported by the primitive instruction set (IR) and which can be used by a higher level language (e.g., P4) but can be resolved to an IR form by a compiler.>>