SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware

Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, Nick Feamster
School of Computer Science, Georgia Tech

ABSTRACT

We present SwitchBlade, a platform for rapidly deploying custom protocols on programmable hardware. SwitchBlade uses a pipeline-based design that allows individual hardware modules to be enabled or disabled on the fly, integrates software exception handling, and provides support for forwarding based on custom header fields. SwitchBlade's ease of programmability and wire-speed performance enables rapid prototyping of custom data-plane functions that can be directly deployed in a production network. SwitchBlade integrates common packet-processing functions as hardware modules, enabling different protocols to use these functions without having to resynthesize hardware. SwitchBlade's customizable forwarding engine supports both longest-prefix matching in the packet header and exact matching on a hash value. SwitchBlade's software exceptions can be invoked based on either packet or flow-based rules and updated quickly at runtime, thus making it easy to integrate more flexible forwarding function into the pipeline. SwitchBlade also allows multiple custom data planes to operate in parallel on the same physical hardware, while providing complete isolation for protocols running in parallel. We implemented SwitchBlade using NetFPGA board, but SwitchBlade can be implemented with any FPGA. To demonstrate SwitchBlade's flexibility, we use SwitchBlade to implement and evaluate a variety of custom network protocols: we present instances of IPv4, IPv6, Path Splicing, and an OpenFlow switch, all running in parallel while forwarding packets at line rate.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks C.2.6 [Computer-Communication Networks]: Internetworking

General Terms: Algorithms, Design, Experimentation, Performance

Keywords: Network Virtualization, NetFPGA

SIGCOMM 2010, August 30-September 3, 2010, New Delhi, India.

1. INTRODUCTION

Countless next-generation networking protocols at various layers of the protocol stack require data-plane modifications. The past few years alone have seen proposals at multiple layers of the protocol stack for improving routing in data centers, improving availability, providing greater security, and so forth [3, 13, 17, 24]. These protocols must ultimately operate at acceptable speeds in production networks—perhaps even alongside one another—which raises the need for a platform that can support fast hardware implementations of these protocols running in parallel. This platform must provide mechanisms to deploy these new network protocols, header formats, and functions quickly, yet still forward traffic as quickly as possible. Unfortunately, the conventional hardware implementation and deployment path on custom ASICs incurs a long development cycle, and custom protocols may also consume precious space on the ASIC. Software-defined networking paradigms (e.g., Click [7, 16]) offer some hope for rapid prototyping and deployment, but a purely software-based approach cannot satisfy the strict performance requirements of most modern networks. The networking community needs a development and deployment platform that offers high performance, flexibility, and the possibility of rapid prototyping and deployment.

Although other platforms have recognized the need for fast, programmable routers, they stop somewhat short of providing a programmable platform for rapid prototyping on the hardware itself. Platforms that are based on network processors can achieve fast forwarding performance [22], but network processor-based implementations are difficult to port across different processor architectures, and customization can be difficult if the function that a protocol requires is not native to the network processor's instruction set. All other functions should be implemented in software. PLUG [8] is an excellent framework for implementing modular lookup modules, but the model focuses on manufacturing high-speed chips, which is costly and can have a long development cycle. RouteBricks [12] provides a high-performance router, but it is implemented entirely in software, which may introduce scalability issues; additionally, prototypes developed on RouteBricks cannot be easily ported to hardware.

This paper presents SwitchBlade, a programmable hardware platform that strikes a balance between the programmability of software and the performance of hardware, and enables rapid prototyping and deployment of new protocols. SwitchBlade enables rapid deployment of new protocols on hardware by providing modular building blocks to afford customizability and programmability that is sufficient for implementing a variety of data-plane functions. SwitchBlade's ease of programmability and wire-speed performance enables rapid prototyping of custom data-plane functions that can be directly deployed in a production network. SwitchBlade

relies on field-programmable gate arrays (FPGAs). Designing and implementing SwitchBlade poses several challenges:

- *Design and implementation of a customizable hardware pipeline.* To minimize the need for resynthesizing hardware, which can be prohibitive if multiple parties are sharing it, SwitchBlade's packet-processing pipeline includes hardware modules that implement common data-plane functions. New protocols can select a subset of these modules on the fly, without resynthesizing hardware.
- *Seamless support for software exceptions.* If custom processing elements cannot be implemented in hardware (*e.g.*, due to limited resources on the hardware, such as area on the chip), SwitchBlade must be able to invoke software routines for processing. SwitchBlade's hardware pipeline can directly invoke software exceptions on either packet or flow-based rules. The results of software processing (e.g., forwarding decisions), can be cached in hardware, making exception handling more efficient.
- *Resource isolation for simultaneous data-plane pipelines.* Multiple protocols may run in parallel on same hardware; we call each data plane a Virtual Data Plane (VDP). SwitchBlade provides each VDP with separate forwarding tables and dedicated resources. Software exceptions are the VDP that generated the exception, which makes it easier to build virtual control planes on top of SwitchBlade.
- *Hardware processing of custom, non-IP headers.* SwitchBlade provides modules to obtain appropriate fields from packet headers as input to forwarding decisions. SwitchBlade can forward packets using longest-prefix match on 32-bit header fields, an exact match on fixed length header field, or a bitmap added by custom packet preprocessing modules.

The design of SwitchBlade presents additional challenges, such as (1) dividing function between hardware and software given limited hardware resources; (2) abstracting physical ports and input/output queues; (3) achieving rate control on per-VDP basis instead of per-port basis; and (4) providing a clean interface to software.

We have implemented SwitchBlade using the NetFPGA board [2], but SwitchBlade can be implemented with any FPGA. To demonstrate SwitchBlade's flexibility, we use SwitchBlade to implement and evaluate several custom network protocols. We present instances of IPv4, IPv6, Path Splicing, and an OpenFlow switch, all of which can run in parallel and forward packets at line rate; each of these implementations required only modest additional development effort. SwitchBlade also provides seamless integration with software handlers implemented using Click [16], and with router slices running in OpenVZ containers [20]. Our evaluation shows that SwitchBlade can forward traffic for custom data planes—including non-IP protocols—at hardware forwarding rates. SwitchBlade can also forward traffic for multiple distinct custom data planes in parallel, providing resource isolation for each. An implementation of SwitchBlade on the NetFPGA platform for four parallel data planes fits easily on today's NetFPGA platform; hardware trends will improve this capacity in the future. SwitchBlade can support additional VDPs with less than a linear increase in resource use, so the design will scale as FPGA capacity continues to increase.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 explains our design goals and the key resulting design decisions. Section 4 explains the SwitchBlade design, and Section 5 describes the implementation of SwitchBlade, as well as our implementations of three custom data planes on SwitchBlade. Section 6 presents performance results. Section 7 briefly describes how we have implemented a virtual router on top of SwitchBlade using OpenVZ. We discuss various extensions in Section 8 and conclude in Section 9.

2. RELATED WORK

We survey related work on programmable data planes in both software and hardware.

The Click [16] modular router allows easy, rapid development of custom protocols and packet forwarding operations in software; kernel-based packet forwarding can operate at high speeds but cannot keep up with hardware for small packet sizes. An off-the-shelf NetFPGA-based router can forward traffic at 4 Gbps; this forwarding speed can scale by increasing the number of NetFPGA cards, and development trends suggest that much higher rates will be possible in the near future. RouteBricks [12] uses commodity processors to achieve software-based packet processing at high speed. The design requires significant PCIe interconnection bandwidth to allow packet processing at CPUs instead of on the network cards themselves. As more network interface cards are added, and as traffic rates increase, however, some packet processing may need to be performed on the network cards themselves to keep pace with increasing line speeds and to avoid creating bottlenecks on the interconnect.

Supercharged PlanetLab (SPP) [22] is a network processor (NP)-based technology. SPP uses Intel IXP network processors [14] for data-plane packet processing. NP-based implementations are specifically bound to the respective vendor-provided platform, which can inherently limit the flexibility of data-plane implementations.

Another solution to achieve wire-speed performance is developing custom high-speed networking chips. PLUG [8] provides a programming model for manufacturing chips to perform high-speed and flexible packet lookup, but it does not provide an off-the-shelf solution. Additionally, chip manufacturing is expensive: fabrication plants are not common, and cost-effective manufacturing at third-party facilities requires critical mass of demand. Thus, this development path may only make sense for large enterprises and for protocols that have already gained broad acceptance. Chip manufacturing also has a high turnaround time and post-manufacturing verification processes which can impede development of new protocols that need small development cycle and rapid deployment.

SwitchBlade is an FPGA-based platform and can be implemented on any FPGA. Its design and implementation draws inspiration from our earlier work on designing an FPGA-based data plane for virtual routers [4]. FPGA-based designs are not tied to any single vendor, and it scales as new, faster and bigger FPGAs become available. FPGAs also provide a faster development and deployment cycle compared to chip manufacturing.

Casado *et al.* argue for simple but high-speed hardware with clean interfaces with software that facilitate independent development of protocols and network hardware [9]. They argue that complex routing decisions can be made in software and cached in hardware for high-speed processing; in some sense, SwitchBlade's caching of forwarding decisions that are handled by software exception handlers embodies this philosophy. OpenFlow [19] enables the rapid development of a variety of protocols, but the division of functions between hardware and software in SwitchBlade is quite different. Both OpenFlow and SwitchBlade provide software exceptions and caching of software decisions in hardware, but SwitchBlade also provides selectable hardware preprocessing modules that effectively moves more flexible processing to hard-

ware. SwitchBlade also easily accommodates new hardware modules, while OpenFlow does not.

SwitchBlade provides wire-speed support for parallel customized data planes, isolation between them, and their interfacing with virtualization software, which would make SwitchBlade a suitable data plane for a virtual router. OpenFlow does not directly support multiple custom data planes operating in parallel. FlowVisor [1] provides some level of virtualization but sits between the OpenFlow switch and controller, essentially requiring virtualization to occur in software.

3. GOALS AND DESIGN DECISIONS

The primary goal of SwitchBlade is to enable *rapid development and deployment of new protocols working at wire-speed*. The three subgoals, in order of priority, are: (1) Enable rapid development and deployment of new protocols; (2) Provide customizability and programmability while maintaining wire-speed performance; and (3) Allow multiple data planes to operate in parallel, and facilitate sharing of hardware resources across those multiple data planes. In this section, we describe these design goals, their rationale, and highlight specific design choices that we made in SwitchBlade to achieve these goals.

Goal #1. Rapid development and deployment on fast hardware. Many next-generation networking protocols require data-plane modifications. Implementing these modifications entirely in software results in a slow data path that offers poor forwarding performance. As a result, these protocols cannot be evaluated at the data rates of production networks, nor can they be easily transferred to production network devices and systems.

Our goal is to provide a platform for designers to quickly deploy, test, and improve their designs with wire-speed performance. This goal influences our decision to implement SwitchBlade using FPGAs, which are programmable, provide acceptable speeds, and are not tied to specific vendors. An FPGA-based solution can allow network protocol designs to take advantage of hardware trends, as larger and faster FPGAs become available. SwitchBlade relies on programmable hardware, but incorporates software exception handling for special cases; a purely software-based solution cannot provide acceptable forwarding performance. From the hardware perspective, custom ASICs incur a long development cycle, so they do not satisfy the goal of rapid deployment. Network processors offer speed, but they do not permit hardware-level customization.

Goal #2. Customizability and programmability. New protocols often require specific customizations to the data plane. Thus, SwitchBlade must provide a platform that affords enough customization to facilitate the implementation and deployment of new protocols.

Providing customizability along with fast turnaround time for hardware-based implementations is challenging: a bare-bones FPGA is customizable, but programming from scratch has a high turnaround time. To reconcile this conflict, SwitchBlade recognizes that even custom protocols share common data-plane extensions. For example, many routing protocols might use longest prefix or exact match for forwarding, and checksum verification and update, although different protocols may use these extensions on different fields on in the packets. SwitchBlade provides a rich set of common extensions as modules and allows protocols to dynamically select any subset of modules that they need. SwitchBlade's modules are programmable and can operate on arbitrary offsets within packet headers.

Feature	Design Goals	Pipeline Stages
Virtual Data Plane (§ 4.2)	Parallel custom data planes	VDP selection
Customizable hardware modules (§ 4.3)	Rapid programming, customizability	Preprocessing, Forwarding
Flexible matching in forwarding (§ 4.4)	Customizability	Forwarding
Programmable software exceptions (§ 4.5)	Rapid programming, customizability	Forwarding

Table 1: SwitchBlade design features.

For extensions that are not included in SwitchBlade, protocols can either add new modules in hardware or implement exception handlers in software. SwitchBlade provides hardware caching for forwarding decisions made by these exception handlers to reduce performance overhead.

Goal #3. Parallel custom data planes on a common hardware platform. The increasing need for data-plane customization for emerging network protocols makes it necessary to design a platform that can support the operation of several custom data planes that operate simultaneously and in parallel on the same hardware platform. SwitchBlade's design identifies functions that are common across data-plane protocols and provides those implementations shared access to the hardware logic that provides those common functions.

SwitchBlade allows customized data planes to run in parallel. Each data plane is called a Virtual Data Plane (VDP). SwitchBlade provides separate forwarding tables and virtualized interfaces to each VDP. SwitchBlade provides isolation among VDP using per-VDP rate control. VDPs may share modules, but to preserve hardware resources, shared modules are not replicated on the FPGA. SwitchBlade ensures that the data planes do not interface even though they share hardware modules.

Existing platforms satisfy some or all of these goals, but they do not address all the goals at once or with the prioritization we have outlined above. For example, SwitchBlade trades off higher customizability in hardware for easier and faster deployability by providing a well-defined but modular customizable pipeline. Similarly, while SwitchBlade provides parallel data planes, it still gives each data plane direct access to the hardware, and allows each VDP access to a common set of hardware modules. This level of sharing still allows protocol designers enough isolation to implement a variety of protocols and systems; for example, in Section 7, we will see that designers can run virtual control planes and virtual environments (e.g., OpenVZ [20], Trellis [6]) on top of SwitchBlade.

4. DESIGN

SwitchBlade has several unique design features that enable rapid development of customized routing protocols with wire-speed performance. SwitchBlade has a pipelined architecture (§4.1) with various processing stages. SwitchBlade implements Virtual Data Planes (VDP) (§4.2) so that multiple data plane implementations can be supported on the same platform with performance isolation between the different forwarding protocols. SwitchBlade provides customizable hardware modules (§4.3) that can be enabled or disabled to customize packet processing at runtime. SwitchBlade implements a flexible matching forwarding engine (§4.4) that provides a longest prefix match and an exact hash-based lookup on

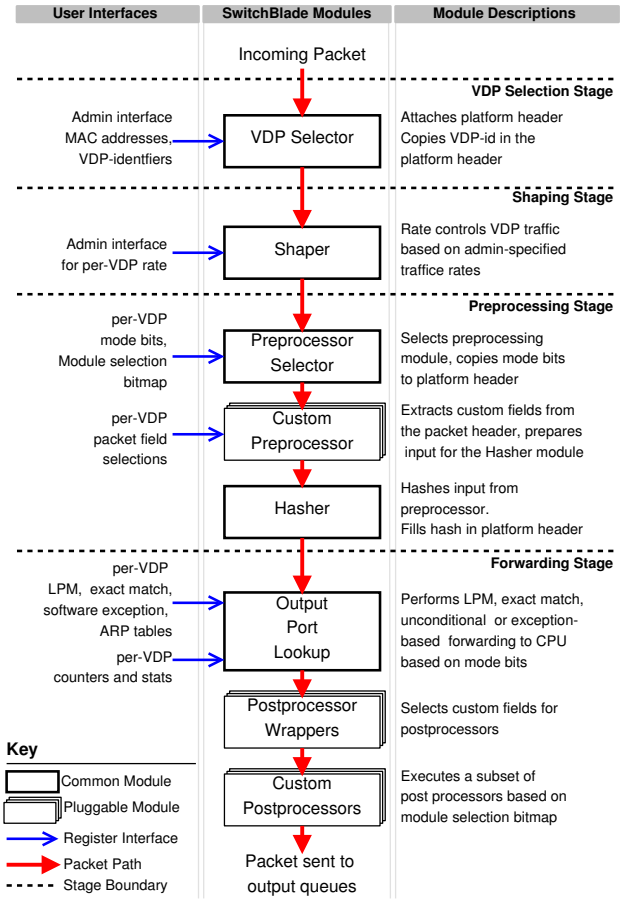

Figure 1: SwitchBlade Packet Processing Pipeline.

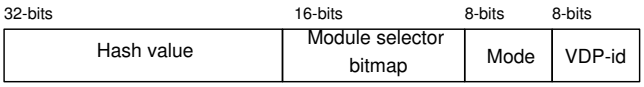

Figure 2: Platform header format. This 64 bit header is applied to every incoming packet and removed before the packet is forwarded.

various fields in the packet header. There are also programmable software exceptions (§4.5) that can be configured from software to direct individual packets or flows to the CPU for additional processing.

4.1 SwitchBlade Pipeline

Figure 1 shows the SwitchBlade pipeline. There are four main stages in the pipeline. Each stage consists of one or more hardware modules. We use a pipelined architecture because it is the most straightforward choice in hardware-based architectures. Additionally, SwitchBlade is based on reference router from the NetFPGA group at Stanford [2]; this reference router has a pipelined architecture as well.

VDP Selection Stage. An incoming packet to SwitchBlade is associated with one of the VDPs. The *VDP Selector* module classifies the packet based on its MAC address and uses a stored table that maps MAC addresses to *VDP identifiers*. A register interface populates the table with the VDP identifiers and is described later.

Field	Value	Description/Action
Mode	0	Default, Perform LPM on IPv4 destination address
	1	Perform exact matching on hash value
	2	Send packet to software for custom processing
	3	Lookup hash in software exceptions table
Module Selector Bitmap	1	Source MAC not updated
	2	Don't decrement TTL
	4	Don't Calculate Checksum
	8	Dest. MAC not updated
	16	Update IPv6 Hop Limit
	32	Use Custom Module 1
	64	Use Custom Module 2
	128	Use Custom Module 3

Table 2: Platform Header: The Mode field selects the forwarding mechanism employed by the Output Port Lookup module. The Module Selector Bitmap selects the appropriate postprocessing modules.

This stage also attaches a 64-bit *platform header* on the incoming packet, as shown in Figure 2. The registers corresponding to each VDP are used to fill the various fields in the platform header. SwitchBlade is a pipelined architecture, so we use a specific header format that to make the architecture extensible. The first byte of this header is used to select the VDP for every incoming packet. Table 2 describes the functionality of the different fields in the platform header.

Shaping Stage. After a packet is designated to a particular VDP, the packet is sent to the shaper module. The shaper module rate limits traffic on per VDP basis. There is a register interface for the module that specifies the traffic rate limits for each VDP.

Preprocessing Stage. This stage includes all the VDP-specific preprocessing hardware modules. Each VDP can customize which preprocessor module in this stage to use for preprocessing the packet via a register interface . In addition to selecting the preprocessor, a VDP can select the various bit fields from the preprocessor using a register interface. A register interface provides information about the mode bits and the preprocessing module configurations. In addition to the custom preprocessing of the packet, this stage also has the hasher module, which can compute a hash of an arbitrary set of bits in the packet header and insert the value of the hash in the packet's platform header.

Forwarding Stage. This final stage in the pipeline handles the operations related to the actual packet forwarding. The *Output Port Lookup* module determines the destination of the packet, which could be one of: (1) longest-prefix match on the packet's destination address field to determine the output port; (2) exact matching on the hash value in the packet's platform header to determine the output port; or (3) exception-based forwarding to the CPU for further processing. This stage uses the mode bits specified in the preprocessing stage. The *Postprocessor Wrappers* and the *Custom Postprocessors* perform operations such as decrementing the packet's time-to-live field. After this stage, SwitchBlade queues the packet in the appropriate output queue for forwarding. SwitchBlade selects the postprocessing module or modules based on the *module selection bits* in the packet's platform header.

4.2 Custom Virtual Data Plane (VDP)

SwitchBlade enables multiple customized data planes top operate simultaneously in parallel on the same hardware. We refer to each data plane as Virtual Data Plane (VDP). SwitchBlade provides a separate packet processing pipeline, as well as separate lookup ta-

bles and register interfaces for each VDP. Each VDP may provide custom modules or share modules with other VDPs. With SwitchBlade, shared modules are not replicated on the hardware, saving valuable resources. Software exceptions include VDP identifiers, making it easy to use separate software handlers for each VDP.

Traffic Shaping. The performance of a VDP should not be affected by the presence of other VDPs. The *shaper* module enables SwitchBlade to limit bandwidth utilization of different VDPs. When several VDPs are sharing the platform, they can send traffic through any of the four ports of the VDP to be sent out from any of the four router ports. Since a VDP can start sending more traffic than what is its bandwidth limit thus affecting the performance of other VDPs. In our implementation, the shaper module comes after the *Preprocessing* stage not before it as shown in Figure 1. This implementation choice, although convenient, does not affect our results because the FPGA data plane can process packets faster than any of the inputs. Hence, the traffic shaping does not really matter. We expect, however, that in the future FPGAs there might be much more than the current four network interfaces for a single NetFPGA which would make traffic shaping of individual VDPs necessary. In the existing implementation, packets arriving at a rate greater than the allocated limit for a VDP are dropped immediately. We made this decision to save memory resources on the FPGA and to prevent any VDP from abusing resources.

Register interface. SwitchBlade provides a register interface for a VDP to control the selection of preprocessing modules, to customize packet processing modules (*e.g.*, which fields to use for calculating hash), and to set rate limits in the shaper module. Some of the values in the registers are accessible by each VDP, while others are only available for the SwitchBlade administrator. SwitchBlade divides the register interfaces into these two security modes: the *admin* mode and the *VDP* mode. The admin mode allows setting of global policies such as traffic shaping, while the VDP mode is for per-VDP module customization.

SwitchBlade modules also provide statistics, which are recorded in the registers and are accessible via the admin interface. The statistics are specific to each module; for example, the VDP selector module can provide statistics on packets accepted or dropped. The admin mode provides access to all registers on the SwitchBlade platform, whereas the VDPmode is only to registers related to a single VDP.

4.3 Customizable Hardware Modules

Rapidly deploying new routing protocols may require custom packet processing. Implementing each routing protocol from scratch can significantly increase development time. There is a significant implementation cycle for implementing hardware modules; this cycle includes design, coding, regression tests, and finally synthesis of the module on hardware. Fortunately, many basic operations are common among different forwarding mechanisms, such as extracting the destination address for lookup, checksum calculation, and TTL decrement. This commonality presents an opportunity for a design that can reuse and even allow sharing the implementations of basic operations which can significantly shorten development cycles and also save precious resources on the FPGA.

SwitchBlade achieves this reuse by providing modules that support a few basic packet processing operations that are common across many common forwarding mechanism. Because SwitchBlade provides these modules as part of its base implementation, data plane protocols that can be composed from only the base modules can be implemented without resynthesizing the hardware and can be programmed purely using a register interface. As an example, to implement a new routing protocol such as Path Splicing [17], which requires manipulation of splicing bits (a custom field in the packet header), a VDP can provide a new module that is included at synthesis time. This module can append preprocessing headers that are later used by SwitchBlade's forwarding engine. A protocol such as OpenFlow [19] may depend only on modules that are already synthesized on the SwitchBlade platform, so it can choose the subset of modules that it needs.

SwitchBlade's reusable modules enable new protocol developers to focus more on the protocol implementation. The developer needs to focus only on bit extraction for custom forwarding. Each pluggable module must still follow the overall timing constraints, but for development and verification purposes, the protocol developer's job is reduced to the module's implementation. Adding new modules or algorithms that offer new functionality of course requires conventional hardware development and must still strictly follow the platform's overall timing constraints.

A challenge with reusing modules is that different VDPs may need the same postprocessing module (*e.g.*, decrementing TTL), but the postprocessing module may need to operate on different locations in the packet header for different protocols. In a naïve implementation, SwitchBlade would have to implement two separate modules, each looking up the corresponding bits in the packet header. This approach doubles the implementation effort and also wastes resources on the FPGA. To address this challenge, SwitchBlade allows a developer to include *wrapper modules* that can customize the behavior of existing modules, within same data word and for same length of data to be operated upon.

As shown in Figure 1 custom modules can be used in the preprocessing and forwarding stages. In the preprocessing stage, the customized modules can be selected by a VDP by specifying the appropriate selection using the register interface. Figure 3 shows an example: the incoming packet from the previous shaping stage which goes to a demultiplexer which selects the appropriate module or modules for the packet based on the input from the register interface specific to the particular VDP that the packet belongs to. After being processed by one of the protocol modules (*e.g.*, IPv6, OpenFlow), the packet arrives at the hasher module. The hasher module takes 256 bits as input and generates a 32-bit hash of the input. The hasher module need not be restricted to 256 bits of input data, but a larger input data bus would mean using more resources. Therefore, we decided to implement a 256-bit wide hash data bus to accommodate our design on the NetFPGA.

Each VDP can also use custom modules in the forwarding stage, by selecting the appropriate postprocessor wrappers and custom postprocessor modules as shown in Figure 1. SwitchBlade selects these modules based on the *module selection bitmap* in the platform header of the packet. Figure 4(b) shows an example of the custom wrapper and postprocessor module selection operation.

4.4 Flexible Matching for Forwarding

New routing protocols often require customized routing tables, or forwarding decisions on customized fields in the packet. For example, Path Splicing requires multiple IP-based forwarding tables, and the router chooses one of them based on splicing bits in the packet header. SEATTLE [15] and Portland [18] use MAC address-based forwarding. Some of the forwarding mechanisms are still simple enough to be implemented in hardware and can benefit from fast-path forwarding; others might be more complicated and it might be easier to just have the forwarding decision be made in software. Ideally, all forwarding should take place in hardware, but there is a tradeoff in terms of forwarding performance and hardware implementation complexity.

Preprocessor Selection		
Code	Processor	Description
1	Custom Extractor	Allows selection of variable 64-bit fields in packet on 64-bit boundaries in first 32 bytes
2	OpenFlow	OpenFlow packet processor that allows variable field selection.
3	Path Splicing	Extracts Destination IP Address and uses bits in packet to select the Path/Forwarding Table.
4	IPv6	Extracts IPv6 destination address.

Table 3: Processor Selection Codes.

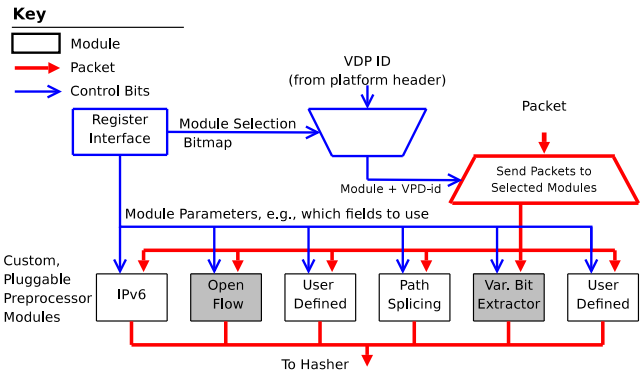

Figure 3: Virtualized, Pluggable Module for Programmable Processors.

SwitchBlade uses a hybrid hardware-software approach to strike a balance between forwarding performance and implementation complexity. Specifically, SwitchBlade's forwarding mechanism implementation, provided by the *Output Port Lookup* module as shown in Figure 1, provides the following four different methods for making forwarding decision on the packet: (1) conventional longest prefix matching (LPM) on any 32-bit address field in the packet header within the first 40-bytes; (2) exact matching on hash value stored in the packet's platform header; (3) unconditionally sending the packet to the CPU for making the forwarding computation; and (4) sending only packets which match certain user defined exceptions, called software exceptions 4.5, to the CPU. The details of how the output port lookup module performs these tasks is illustrated in Figure 4(a). Modes (1) and (2) enable fast-path packet forwarding because the packet never leaves the FPGA. We observe that many common routing protocols can be implemented with these two forwarding mechanisms alone. Figure 4 is not the actual implementation but shows the functional aspect of SwitchBlade's implementation.

By default, SwitchBlade performs a longest-prefix match, assuming an IPv4 destination address is present in the packet header. To enable use of customized lookup, a VDP can set the appropriate mode bit in the *platform header* of the incoming packet. One of the four different forwarding mechanisms can be invoked for the packet by the mode bits as described in Table 2. The output port lookup module performs LPM and exact matching on the hash value from the forwarding table stored in the TCAM. The same TCAM is used for LPM and for exact matching for hashing therefore the mask from the user decides the nature of match being done.

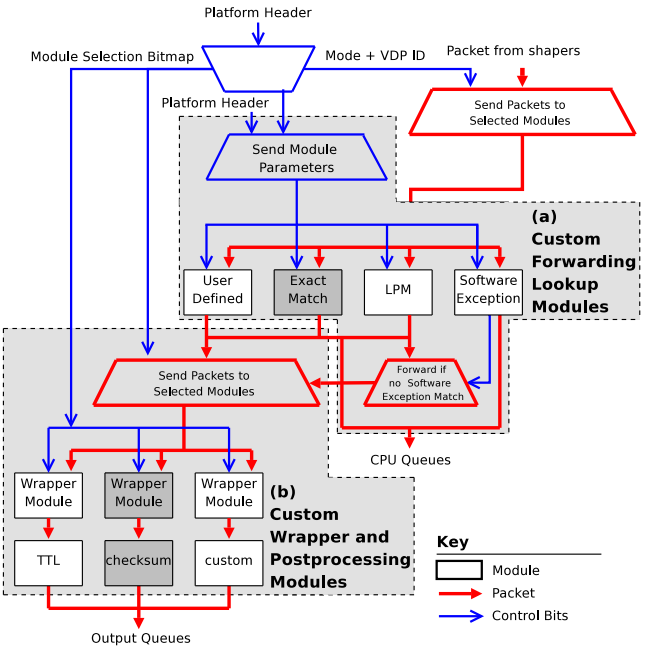

Figure 4: Output Port Lookup and Postprocessing Modules.

Once the output port lookup module determines the output port for the packet it adds the output port number to the packet's platform header. The packet is then sent to the postprocessing modules for further processing. In Section 4.5, we describe the details of software work and how the packet is handled when it is sent to the CPU.

4.5 Flexible Software Exceptions

Although performing all processing of the packets in hardware is the only way to achieve line rate performance, it may be expensive to introduce complex forwarding implementations in the hardware. Also, if certain processing will only be performed on a few packets and the processing requirements of those packets are different from the majority of other packets, development can be faster and less expensive if those few packets are processed by the CPU instead (*e.g.*, ICMP packets in routers are typically processed in the CPU).

SwitchBlade introduces *software exceptions* to programmatically direct certain packets to the CPU for additional processing. This concept is similar to the OpenFlow concept of rules that can identify packets that match a particular traffic flow that should be passed to the controller. However, combining software exceptions with the LPM table provides greater flexibility, since a VDP can add exceptions to existing forwarding rules. Similarly, if a user starts receiving more traffic than expected from a particular software exception, that user can simply remove the software exception entry and add the forwarding rule in forwarding tables.

There is a separate exceptions table, which can be filled via a register interface on a per-VDP basis and is accessible to the output port lookup module, as shown in Figure 4(a). When the mode bits field in the platform header is set to 3 (Table 2), the output port lookup module performs an exact match of the hash value in the packet's platform header with the entries in the exceptions table for the VDP. If there is a match, then the packet is redirected to the CPU where it can be processed using software-based handlers, and if there is none then the packet is sent back to the output port