OpenPipes: Prototyping high-speed networking systems

Glen Gibb, David Underhill, Adam Covington, Tatsuya Yabe, and Nick McKeown
Stanford University, NEC System Platforms Research Labs

ABSTRACT

High-speed networking systems are typically built from many modules strung together in a *pipeline*. Examples of such systems include routers, firewalls, load-balancers and IDSs. High-speed systems must run at line-rate, and so commonly require at least some modules to be implemented in custom hardware. A developer faces two problems when building these systems:

1. How do they partition their design over different subsystems; which modules belong in CPU, FPGA, ASIC, NPU? In current systems, they have to decide at design time; changing the partition later is very hard, and often requires significant redesign.
2. How do they test modules—realistically—before putting in all the effort/money to put them in specialized hardware (FPGA or ASIC)?

In our group, we build a lot of high speed networking systems, and often face these problems. We decided to create a reusable platform that we—and others—can use when designing and prototyping high speed networking systems. In particular, our system:

1. Allows a design to be repartitioned very easily—modules can be moved from one physical system to another, *even while the system is running*. This gives lots of flexibility to the designer, who is freed from worrying about how to "fit" the application on limited resources.
2. Allows modules to be tried in software in the running system, and later be replaced by hardware, without the design changing—in fact, we can do this *while the system is running*.

The operation of our platform can be summarized by three key ideas. The first two are shared with other platforms—the third is unique to our platform and provides us with the unique capabilities listed above. The three ideas are:

- We assume the system is made of modules.
- We connect the modules together using the network.
- We use OpenFlow [2] to connect the modules together.

As already mentioned, the use of an OpenFlow network is the key change—it allows modules to be moved around in the network, from one subsystem to another, while the system is running. This solves both of our problems: (1) It allows repartitioning dynamically; and (2) It allows modules to be implemented in hardware, software, or both. This allows the system to migrate, over time, from software to hardware; or for developers to verify their designs by comparing the output of both hardware and software modules in the running system.

We call our platform *OpenPipes* as it uses *OpenFlow* to construct a modular *pipeline*. To demonstrate OpenPipes, we have built an IDS out of multiple modules, and "plumbed" the modules together using OpenFlow. The modules are written in Verilog on NetFPGA [1] and as software process on Linux PCs; and between them, they form a pipeline that process packets for an IDS We show that—like Click—OpenFlow can be used to quickly build new systems from existing modules. But with OpenFlow, the modules can reside on different machines, and be moved around seamlessly during operation. For example, we can pick up a particular "signature matching module" while the IDS is running, and move it to another machine. Or we can insert a module that identifies a new set of threats while the system is running. Or we can run the pipeline in reverse. Or put some modules in software and some in hardware. All the modules are plumbed together using OpenFlow, regardless of where they reside in the network. Furthermore, there are no encapsulation or tunneling protocols used—the modules have unique addresses in the network.

Our platform's design was inspired by Click [3]. We wanted a system that has the simplicity of Click that also supports modules written in hardware and the ability to move modules to other CPUs/FPGAs in the network.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

OpenFlow, OpenPipes, modular design

1. DEMONSTRATION

We demonstrate a video processing application built using the OpenPipes platform. The video processing application takes a live video stream and performs a number of transformations on that stream. Video is fed into the system from a camera providing a live feed and the resultant output stream is displayed on a monitor to show the transformations perfomed by the application. The transformations that are performed can be altered by changing the active modules.

A graphical user interface (GUI) is used to demonstrate OpenPipes and our the video processing application. Figure 1 shows a conceptual overview of the GUI. The primary section of the display shows the active state of the system. Modules are located inside "module hosts" (NetFGPAs or PCs) and connections exist between modules. Using the GUI the user instantiates modules by dragging them from a palette of available modules, reconnects modules and configure individual modules.

We use a simple video processing application in order to clearly show the main features of OpenPipes. The modules comprising the baseline application are depicted in Figure 2. In this configuration the video stream is fed initially to a classifier module which

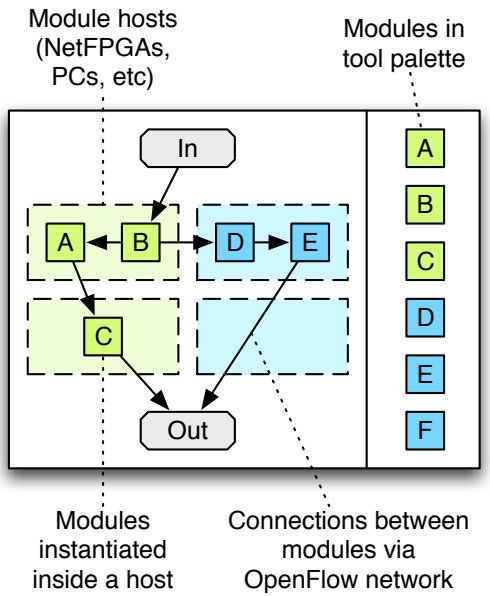

Figure 1: Basic overview of the OpenPipes GUI

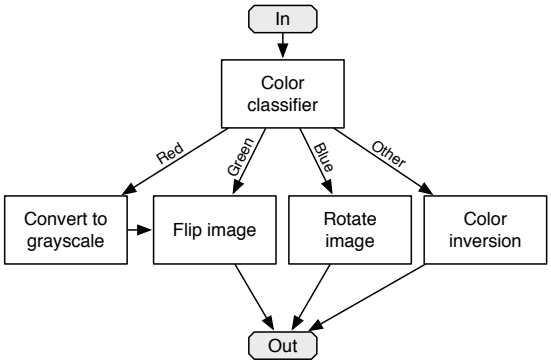

Figure 2: A very basic video processing system

determines the primary color-cast of the video stream. Simple celophane filters are placed in front of the video camera to provide the color-cast within the stream. The color-bast of the stream is used to determine which of the alternate processing pipelines to send the video over. Each pipeline performs a different sequence of transforms on the video—examples of tranformations include conversion to greyscale, flipping about either axis, rotation, and feature detection.

Using the baseline video application as a starting point we demonstrate the ability to insert modules into a running system and to verify the operation of a new module.

Dynamic insertion of a module into the system

A new software-based smoothing module is downloaded into the system. This is connected to the output of one of the color-cast pipelines as shown in Figure 3. The output of this pipeline now smoothes each frame.

Testing of new modules

Once confidence has been gained in the correct operation of the software-based smoothing module, the operator of the system can implement the module in hardware. To verify the operation of the

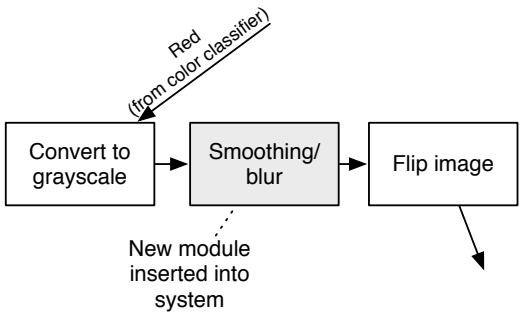

Figure 3: Insertion of a smoothing/blur transform into the red color-cast processing branch

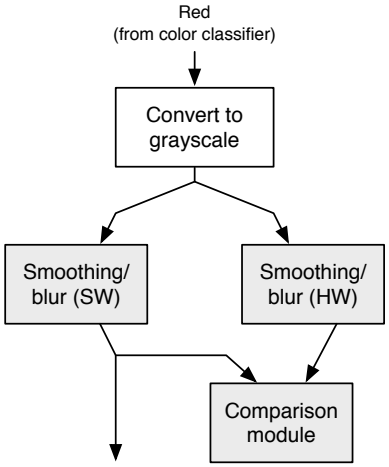

Figure 4: Comparison of a software and hardware implementation of the smoothing/blur transform

hardware module the operator operates the software and hardware modules in parallel and feeds the output into a comparison module that compares the output of the two modules. This is shown in Figure 4.

2. REFERENCES

[1] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA—an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 160–161, 2007.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.

[3] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 217–231, New York, NY, USA, 1999. ACM.