Building a Router on Linux

Andrew McRae (amcrae@netd.com)
NetDevices Inc.

Abstract

This paper presents a description of a mid-range services gateway that uses Linux as an internal infrastructure. A range of subjects are discussed, such as security issues, multi-CPU interaction, distribution, performance and scaling. The architecture of the NetDevices SG-8 is described, with emphasis on the technical aspects of employing Linux as the underlying infrastructure, and the benefits and costs involved.

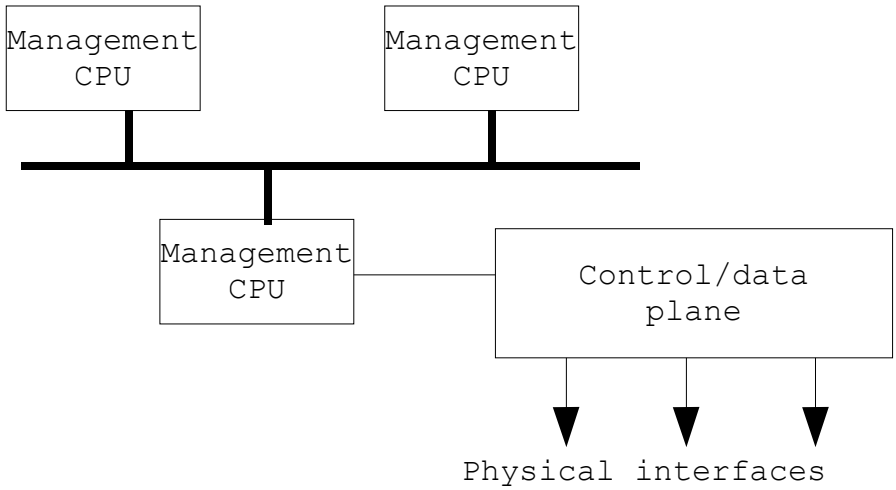
As part of this infrastructure, a networking subsystem is described that addresses the major issues with supporting a sophisticated and extensive packet processing environment on Linux without sacrificing performance or robustness.

1.Introduction

Traditionally, routers are developed using proprietary operating systems (such as Cisco's IOS), or by licensing an OS from a vendor (such as QNX, or VxWorks etc).
With the growing acceptance of Linux as an embedded or targeted OS, it is becoming more common to see Linux being used in this environment. In fact, with the networking support that is built in to Linux, a common arrangement is to use the networking facilities of Linux wrapped with some friendly and vendor specific user interface.
There are significant advantages to using Linux in an embedded environment, such as cost, portability, access to source etc. Sometimes, however, the developer can fall afoul of the GPL if careful consideration is not taken e.g the Linksys situation, where using Linux internally also meant that code tainted with the GPL must be freely offered when that code is used in a publicly available application.
NetDevices Inc. is a Silicon Valley startup founded in 2003 with the goal of developing a next generation services gateway, incorporating routing, firewall, VPN and other networking applications in a single integrated device. This paper describes the experience in implementing this device using Linux as an internal framework, but without using the Linux networking facilities directly.

2.Environment Overview.

The NetDevices SG-8 is a state-of-the-art services gateway that incorporates many different networking applications in a single integrated device, offering a high degree of manageability, performance and robustness.
Physically, the SG-8 consists of a number of separate hardware hot-swappable modules, interconnected with a high speed industry standard PCI-Express bus. A unique feature is the introduction of a cluster of management orientated CPUs, separating the management functions of the device from the control and data plane. These CPUs communicate via an internal high speed LAN:



Internally, there are a number of different CPUs, some with functions related to the management facilities of the device (providing user interface, CLI services, configuration, monitoring etc.), and some dedicated to packet processing and control plane functions (route protocol processing etc.).
Linux is running on each of these CPUs, and provides the infrastructure framework that hosts the various applications running internally. These applications provide the router and gateway functionality. Linux provides the following facilities:

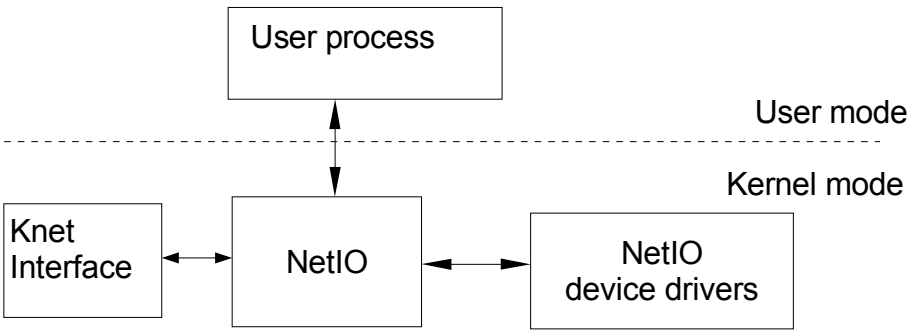
- An embedded kernel, providing the usual kernel/user process model.
- Internal network connectivity, for communication between the various CPUs in the device.
- A familiar and well understood environment for hosting applications.
- The ability to run the same applications on a host PC for development and testing.
- The readily available open source packages that can be run on Linux with no change, such as embedded web servers, SSH servers and clients, SNMP agents etc.

Given the widespread availability, portability, and cost of Linux, it is of no surprise that it is rapidly becoming the preferred environment for embedded systems.

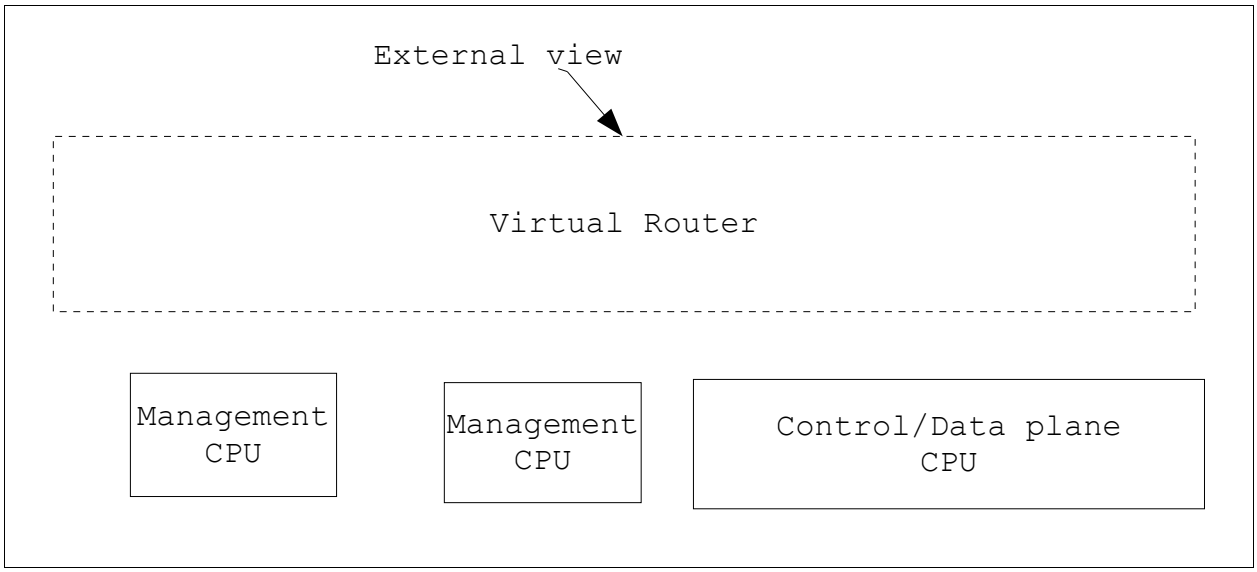
3.External/Internal View

Whilst Linux is being used internally on the SG-8, to all intents and purposes the user of the system will never be aware of this fact, nor will the user be exposed to any specific Linux user interface or Linux utility. Even though the device is being used to process network packets, none of these packets are processed using the Linux networking facilities.

Instead, a NetDevices proprietary packet switching framework is used to provide the packet processing applications. This framework is called NetIO, and forms the basis of the data and control plane of the SG-8.



As can be seen, NetIO is a Linux kernel module that interacts with a user process, so that the packet processing is actually performed in user space rather than kernel mode. NetIO device drivers are used to provide the physical device interface.
There are a number of advantages to this approach, such as robustness, performance, and modularity.
An internal Ethernet is used for the interconnection of the various CPUs in the system, so that management and control information can flow around the system (no packets being forwarded through the system ever appear on this internal network). For ease of addressing, a simple 10.x.x.x addressing scheme is used.
Externally, however, the device is treated as a `single system view', so that external users and network peers interact with the device just as if it were a single host with multiple interfaces (just like any other router). More importantly, there is a strong isolation and decoupling between the internal view of the system and the external view, for obvious security and robustness reasons.
The interesting implication that arises from this external/internal decoupling is that very extensive network applications and features can be implemented entirely according to the user applications running on the system, irregardless of whether Linux itself supports these features, or has anything to do with them. Conceptually, this model can be viewed as creating a *virtual router* operating internally over a set of CPUs running Linux:



This has a number of advantages:

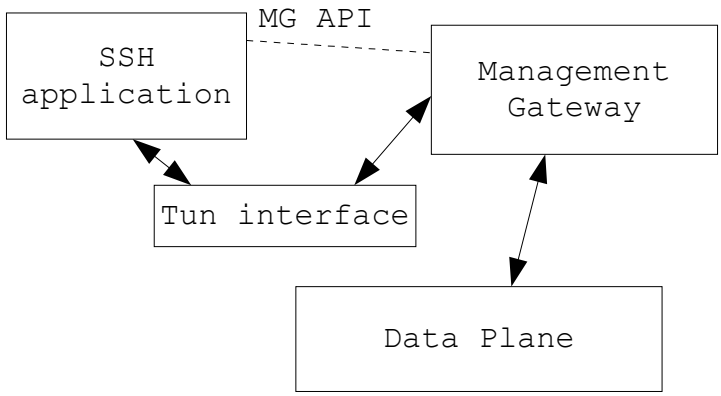
- The external addressing and interface management are entirely separate from the addressing of the internal LAN, so that there is no overlap or mixing of traffic.
- Performance is highly scalable by the use of multiple data plane CPUs, or offload CPUs, that are not externally visible.
- Sophisticated network features can be much more easily implemented, such as Virtual Routing and Forwarding (VRF), where multiple routing and forwarding domains can coexist within the device without any impact on the internal addressing.
- The system can be more robust, with the failure of individual elements not being visible externally e.g if one management CPU fails, then another can provide the same services.
- A more isolated and secure separation is achieved between the key internal facilities and the external packet traffic.

This model of separation between the internal and external view of the system has been successful in delivering these advantages.

4.Management Gateway.

Given that there is now a separation between the external view of the system (and how packets get in and out of the device), and the internal network, how does this impact applications, especially applications that use the standard Linux networking facilities (such as sockets etc.)? If I want to run an application like a SSH server using the standard OpenSSH application, how does it interface internally so as to be visible in the external view of the system?

This is achieved through the use of a facility called the Management Gateway (MG):



The MG is the bridge between the external view of the system, and the internal applications that provide the management facilities of the system. When an application is to be accessed from external packets, it will use a MG API to register itself. The MG will interact with the data plane to register the UDP/TCP port number associated with that application. Meanwhile, as part of the application initialisation, the application will open and listen on a socket on the internal LAN. The application can be running on the same CPU, or a different CPU, as the MG.
When packets arrive for that application, the data plane will forward the packets to the MG. The MG will perform a Network Address Translation on that packet to map the packet address to the address of an internal tunnel interface, and forward it to the application's listening socket via the tunnel driver (used to deliver a user level packet into the kernel networking stack). Packets sent from the application will be sent back to the tunnel interface, where the MG will receive the raw packet, perform the appropriate reverse NAT, and forward the packet to the data plane for transmitting.
The application may need to discover the peer address of the connection. If it used the standard socket function (getpeeraddress), it would see the internal address of the tunnel interface, rather than the real external address. So the MG API provides facilities for the application to discover the peer address, port number and other context (for AAA or other purposes).
The MG performs a critical role in mapping the external network interfaces into the internal applications so that standard applications can be used with very little modification. The MG will perform a range of functions:

- Knowledge of the various external network interfaces and IP addresses attached to these interfaces.
- Awareness and mapping of multiple routing domains (VRFs) so that applications can use the internal network for interacting with potentially overlapping private routing and addressing domains (part of the MG API is to provide VRF information for the application if it requires it, something that is unavailable through the stock Linux kernel.
- Security filtering of application packets, so that Denial of Service attacks will not impact the manageability of the device.
- Load balancing when multiple instances of the application may be running on multiple CPUs.

5.GPL Issues

As a lawyer quoted in a presentation when advising NetDevices concerning the GPL, lawyers generally hate the GPL because it is so clear and watertight in its treatment of Open Source – they would much prefer a BSD style license, which is quite different in its commercial applicability. It is interesting to discuss the various considerations that needed to be taken into account when actively developing in a mixed proprietary and GPL environment. Care must be taken to avoid tainting GPL source with proprietary code (either because the code is separately licensed, or being internally developed), and to maintain strict boundaries and awareness of the areas where the GPL applies.

One approach that is used in the NetDevices system is to run specific GPL programs as separate processes, using standard IPC mechanisms to present the data for processing. For example, the use of Snort as an Intrusion Detection System is very common, so that many customers and users have developed a high degree of trust in this tool. There is significant advantage to be able to employ such a tool as an internal process incorporated seamlessly into the main packet processing of the system by running Snort as a process on one of the data plane CPUs, and internally passing packets to it for processing. In this way, there is no violation of the GPL, and customers can use the best tool for the job, externally seamless.

In some areas where it may be somewhat grey, it is easier to simply provide various interface libraries or code and make it Open Source to avoid any kind of GPL taint. The

Linksys experience shows that commercial organisations ignore the GPL at their peril, and will quite correctly be taken to task for any kind of violation or license issue. Netdevices position is that the GPL should be honoured in every situation, and pains are taken to ensure that no licensing boundary is crossed.

Part of the approach here is that it can be shown that in a larger system or environment, GPL and proprietary code can co-exist quite easily as long as care is taken to ensure the appropriate boundaries are maintained; of course, there is the wider issue of whether *all* code, proprietary or otherwise, should be Open Source, but the commercial reality is that this is not a goal that will be achieved.

One interesting implication is the acceptance in the startup community of Linux and Open Source in general, to the extent that most Venture Capital providers are well aware of Open Source, and the significant advantages it holds. The Open Source environment has become well wired into the DNA of the general software engineering community.

6.Conclusion.

The NetDevices SG-8 is an interesting product because it successfully employs many of the advantages of using Linux and other Open Source product, whilst providing a sophisticated and fully featured networking device.

The use of a decoupled external/internal view of the networking processing has successfully allowed a greater degree of scalability, performance, security and robustness. The use of Linux as an internal framework for an embedded system has delivered many significant benefits, such as speed of development, testing, and application portability.