Global Information Assurance Certification Paper

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at http://www.giac.org/registration/gcia

Intel IXP Network Processor Based Intrusion

Detection

Gold Certification

Author: Greg Pangrazio, gpangrazio@secureworks.com

Advisor: John C. A. Bambenek, bambenek@gmail.com

Accepted: 2007-05-22

Abstract

Intrusion Detection is a very important part of network security, and with the increasing bandwidth in large scale networks, Intrusion Detection and Prevention Systems must scale to handle the large internal LAN links. Common server and desktop hardware does not have the throughput to handle 10Gb/s network connections, so specialized hardware is required. The Intel IXP series Network Processor is one such system with the throughput to manage these network connections. This paper will introduce the IXP series processors as well as outline the steps to create a functioning Snort based IDS on the IXP 425.

Table of Contents

Greg Pangrazio

Introduction

Network Processors range in ability from simple interface cards in home computers to the large scale interfaces in core network routers and switches. Network Processors are nothing more than specialized processors with instruction sets designed to operate on packetized data such as that commonly seen on a network. These instruction sets allow these processors to handle billions of packets a second with relatively low clock speeds less than 1GHz. These processors can function as a small embedded system like the Intel IXP4xx series does, or can be a component of a much larger system. In larger systems these processors offload the handling of the TCP/IP stack from the main processor of the system allowing higher order functions to consume the main CPU's time. This can be done by inserting a card into a standard system's PCI bus as is done by the IXP2850 or as integrated components similarly to the implementation of Cisco Network processing engines.

The Cisco Network Processor line is highly proprietary, and is not released for use by independent groups. The Intel line of Network Processors however is used in both proprietary systems as well as custom applications. The availability of the IXP series and very controlled release of Cisco Network Processors limits the focus of this paper to the Intel systems.

Network Processors have many uses in today's security and performance based networking world. With the increase

Greg Pangrazio

4

in applications and use of the internet the average home internet connection has gone from dialup modems in the late nineties to high speed eight to twelve Mb/s links going to the home. If this bandwidth is summed for a small neighborhood it can easily be seen how larger cities would need high speed capabilities. The original routers were simple UNIX hosts that knew where to send the packet along the path to the next host. From there specialty hosts were required to keep up with the increasing demand. These specialty hosts became the first of the routers that later evolved into Network Processor based devices. Now Network Processors are even more specialized with some designed for cryptography, others with reduced instruction sets for speed.

Routers often use these Reduced Instruction Set Computer (RISC) Network Processors in conjunction with Application Specific Integrated Circuits (ASICs) to handle their load. The packet enters through the interface, is passed to the Network Processor for initial inspection, this inspection includes checksum calculations and access control list application. Next the packet is passed on to either the backplane of the router for forwarding to another interface or to the high speed ASICs for manipulation. ASICs are used today for network address translation, packet routing decisions, and route table calculation.

Firewalls started as routers with access control and have developed into specialized hardware. They are still very similar to the router, but have much more processing

power.   This additional processing power along with increased memory allows the firewalls to do multi-layer analysis.   This analysis takes place in the Network Processor. The packet can be picked apart, and sorted out by any component in the packet.   Network Processors are able to follow a packet stream just like a human can with logic structures that allow for streams to be classified and for packets to be dropped, passed or even modified. These Network Processors are specialized to handle packets at the Transmission Control Protocol (TCP) layer as well as the Internet Protocol (IP) layer, where their router based brethren are specialized for the lower network layers only.

Other specialized Network Processors are also used in high end network interfaces. Most of these are designed to serve out content on high bandwidth systems; however there is one Network Processor based interface system designed for the online gaming market. This interface card by Bigfoot Networks called Killer allows the customization of the network stack and separates out the network communication allowing the central processing unit to handle the game without hardware interrupts from the network interface card while the Killer NIC handles the interrupts and prioritizes packets. It has been shown to reduce ping latency by up to 35% in games like World of Warcraft. This also brings to light one unusual benefit of using a Network Processor. They do not have to play by normal network rules. Network Processors have the ability to manipulate any field in a packet, or to re-order packets exiting the system. Some individuals have been known to exploit this flexibility and change the way games measure

Greg Pangrazio

6

their connectivity.   Often algorithms give a slight edge to
gamers with a slower (higher latency) connection to even
out the playing field.   This is often measured by the echo
time to a host.   Increasing this latency, while keeping the
data rates the same for all other communication can allow
an in-game advantage.

Intel Based Network Processors

Intel Network Processors are made up of two main
parts, a general execution core and one or more Network
Processing Engines (NPE).   The NPEs are connected directly
to the layer 1 interfaces and handle all of the packet
based instructions.   Figure 1 includes a diagram of a
Network Processing Engine from an IXP 1200.   The general
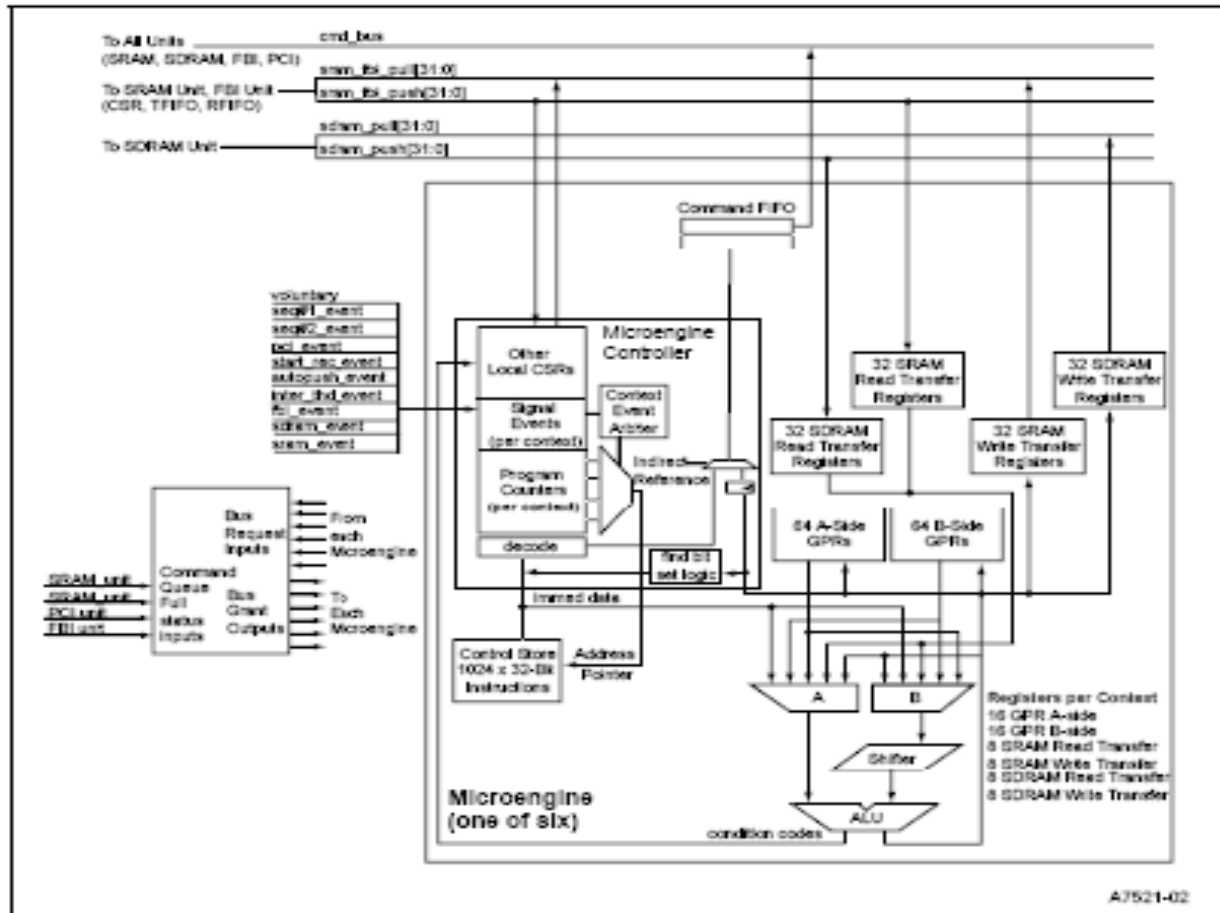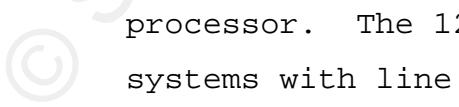execution core is typically an Intel XScale processor of
varying abilities.

Greg Pangrazio

7

Figure 1. Micro Engine Diagram

(IXP425 XScale Development Kit for Embedded Linux – Arcom, 2007).

Intel Network Processors are broken into three main categories, the 4xx series, 12xx series and 2xxx series. The 4xx series has three NPEs and one general instruction core provided by an XScale processor running at up to 533MHz. The NPEs provided on this system are of fixed configuration that is accessed through specific libraries and an API provided by Intel to the vendors of the completed systems. Each of the NPEs in this device is capable of handling 100Mb/s full duplex Ethernet data rates and 70Mb/s encrypted data rates. The 4xx series is

Greg Pangrazio

8

targeted for Small Office/Home Office applications and is
often used to make light weight security, gateway and
application servers.

Figure 2. IXP 425 Network Processor
(Intel IXP425 Network Processor, 2007).

The 12xx series contains six programmable NPEs and one
StrongARM processor. This configuration allows custom
applications to be run from the NPE allowing packets to be
handled without Operating System kernel contact. The
increase in Network Processors allows this device to have a
total throughput of 6.26Gb/s. Each NPE has a 2K
instruction store to reduce processor overhead. The main
processing is handled by a StrongARM 166, 200 or 232 MHz
processor. The 12xx series is targeted to the higher level
systems with line rate OC12 interfaces available for voice
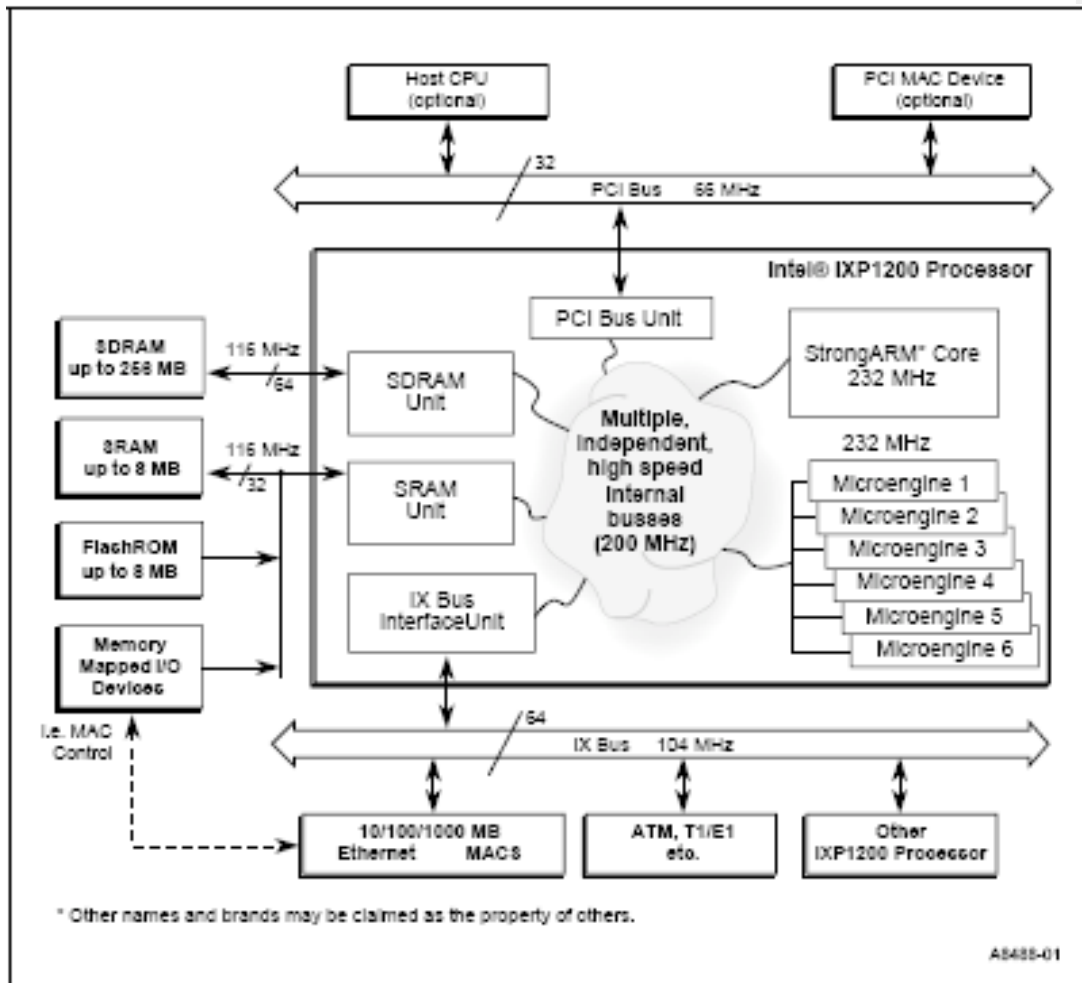and data communications.

Greg Pangrazio

9

Figure 3. IXP 1200 Network Processor
(Intel IXP1200 Network Processor Family, 2007).

The 2xxx series is much more powerful than either of
the other Network Processor lines released by Intel. This
processor was designed for high end routing, switching and
security applications. With ten 1Gb/s interfaces available
or one 10Gb/s on the 2855 model this series is capable of
handling today's highest performance markets. There are
sixteen programmable Micro Engines. The two cryptography
engines on board are used to accelerate cryptography
algorithms; the cipher path is able to perform 25 million
encryptions a second based on a 40 byte packet size (Intel

Greg Pangrazio

IXP2855 Network Processor, 2007). A 15Gb/s interface is
included to connect multiple 28xx series processors
together for high demand environments. These features
combined with a 1.5 GHz clock speed and the RISC processor
cores make this by far the most powerful dedicated Network
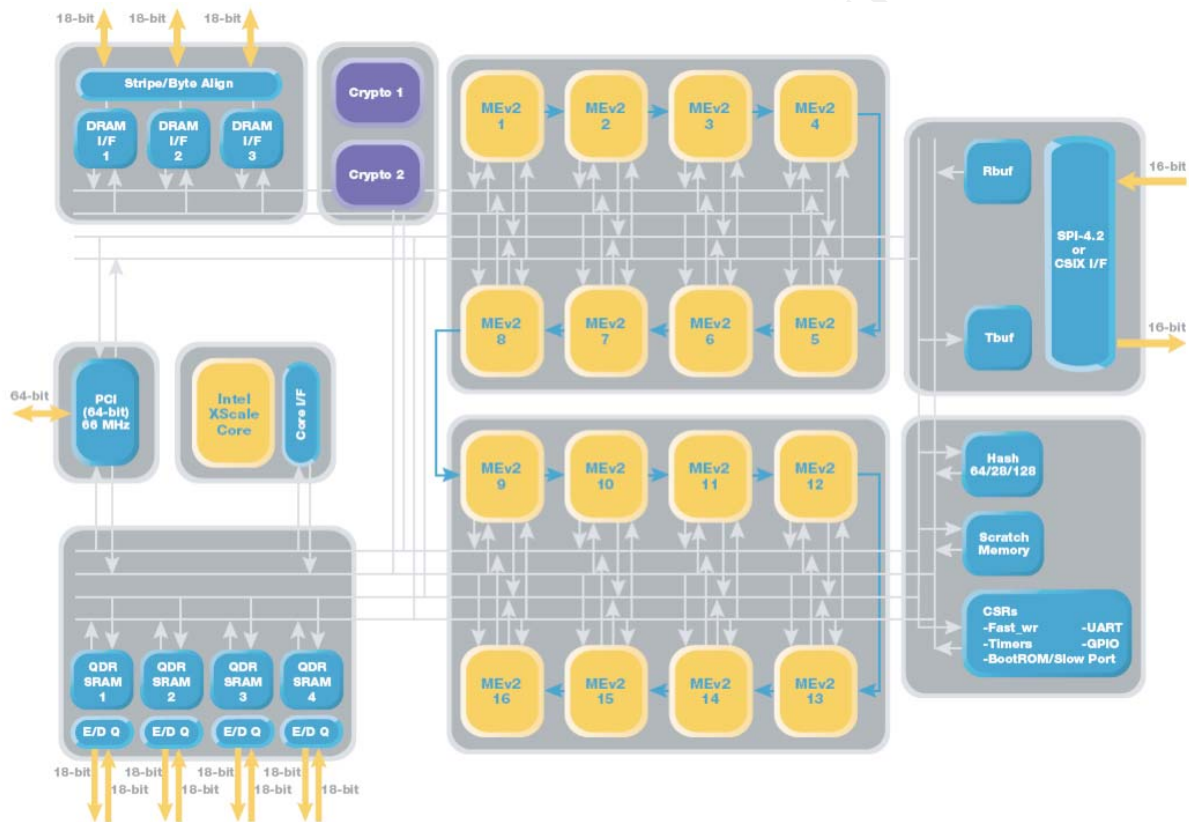Processor available.

Figure 4. Intel IXP 2855 Network Processor

(Intel IXP2855 Network Processor, 2007).

Vulcan IXP 425

The Intel IXP 425 can be purchased as a stand alone
embedded Linux system with almost any interface type that
can be conceived. Options include Ethernet, DSL, DSL
Access Manager, 802.11 or any other interface that can be

Greg Pangrazio   1

connected to a PCI or USB bus. The Arcom Vulcan
development kit provides two 10/100Mb/s Ethernet
interfaces, four USB 2.0 interfaces, four 921.6 KBaud
serial interfaces, a CompactFlash slot and a PC/104
interface. The CompactFlash slot can be used for extra
memory or interface with a Prism v2 CompactFlash Wireless
adapter. The logical diagram of the Vulcan development
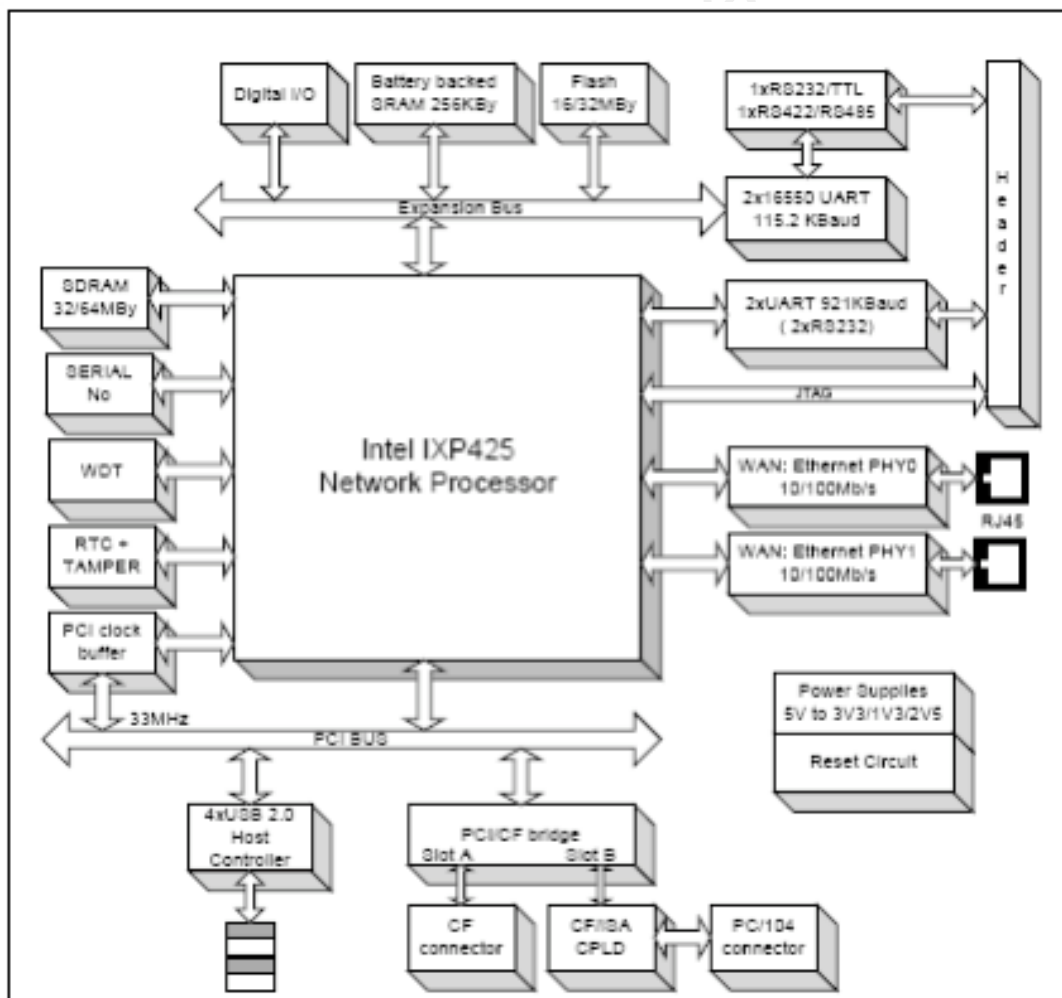board is included in Figure 5.

Figure 5. Vulcan Logical Block Diagram
(IXP425 XScale Development Kit for Embedded Linux -
Arcom, 2007).

Arcom provides a Fedora Core 3 development environment
that is used to create new kernels, boot files; essentially
any program components that are needed for the creation of
a Vulcan based Network Processor Application. Included in
the development software is the Intel Application library.
It provides C libraries that can access and configure the
NPEs in the IXP425 processor. These applications are
loaded into kernel modules for the Redboot v2.6 kernel that
is on the Vulcan at shipping. By default the Vulcan system
has Secure Shell, a small HTTP server, Secure Copy, console
access and kernel modules to drive the IXP425 processor and
its interfaces loaded.

For the development machine an Intel 3.0Ghz Pentium 4
machine with 2GB of RAM, 500GB of hard disk space, two
Ethernet interfaces and two serial interfaces was used.
Once new applications were compiled on the host machine,
they were loaded to a 512 MB CompactFlash memory card that
was inserted into the Vulcan enclosure. The onboard 16MB
of flash was saved for the original kernel for disaster
recovery. The files could also be transferred via SCP to
the Vulcan system, however they would not survive a reboot
unless the files were added to the binary boot file,
compressed and the binary was updated on the device.

Implementation

The code that was created contained a new kernel
driver, and a modified libpcap. Libpcap is the code that
allows Snort, TCPdump and several other tools to directly

Greg Pangrazio

1

read packets from the interface in raw format. Typically
each packet is processed by the four layers of the IP stack
(Physical, Data Link, Network and Transport) removing the
extraneous information at each pass. This requires extra
processing power, and the main CPU is required to handle
each packet. This reduces the amount of packets that can
be analyzed by the CPU and hinders the overall
effectiveness of the IDS. Using a modified version of
libpcap allowed the packets to be taken from the Network
Processor Micro-Engine directly to the main memory without
involving the CPU for packet fetching. The processor is
then free to only handle the IDS application and is not
concerned with servicing the interfaces. The packets are
then stored in a queue in the memory until the IDS
application has a chance to process these packets. Libpcap
reads from the main memory to fetch packets for the IDS
while the custom kernel module accepts incoming packets and
passes them on to the main memory through the bus. This
allowed a default IDS application to be used and only a new
interface type was created in libpcap, making this a
minimally invasive change to increase throughput.
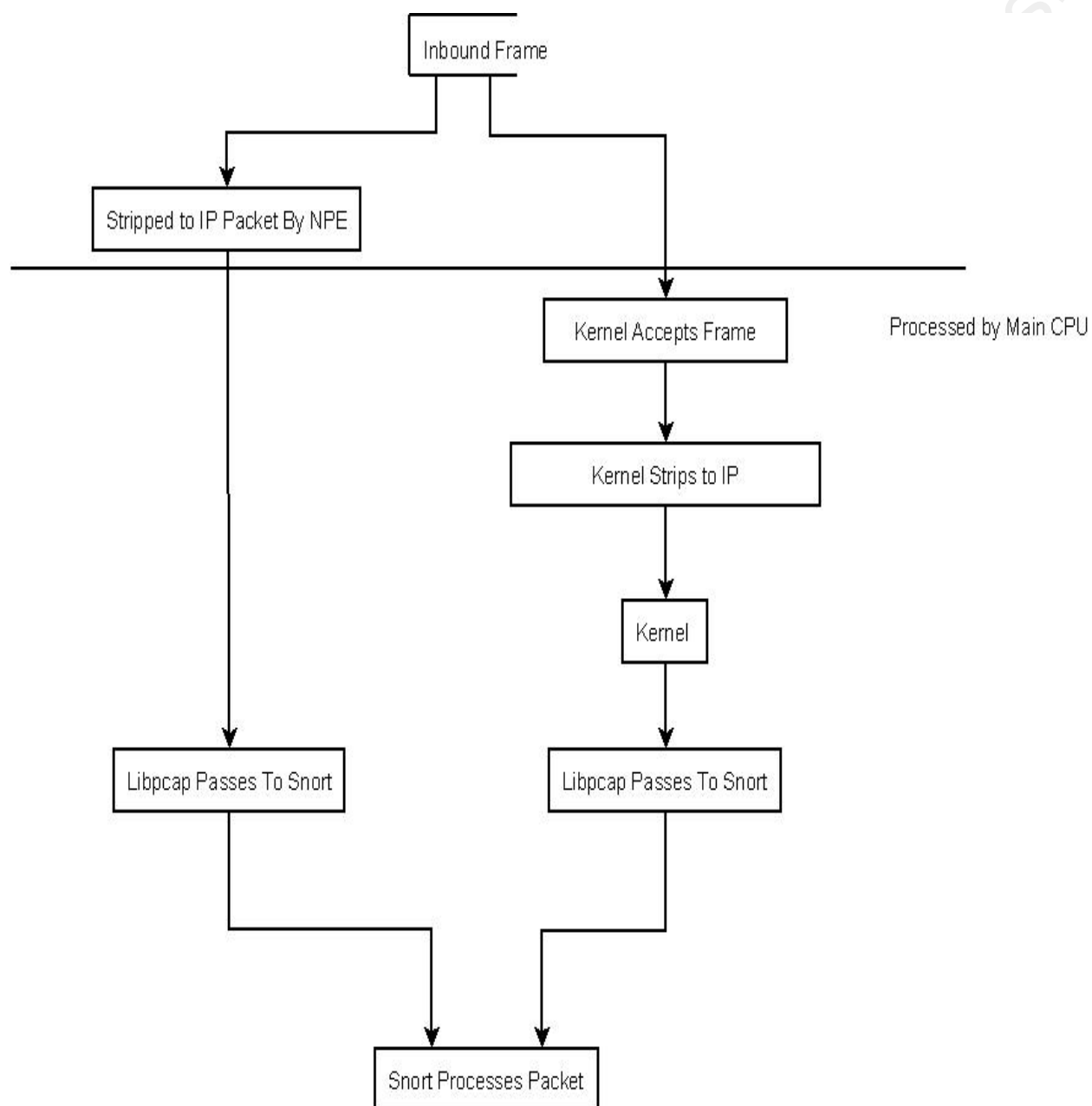
Greg Pangrazio

Figure 6. IXP Snort VS *NIX based Snort

The kernel module was implemented mostly through the
use of the IXP development API provided by Intel. For the
IXP 425 this is the only method to access the Micro Engines
as they cannot be accessed directly. The API provides
functionality to configure the interface, buffers, and for
packet handling. In order for the network interfaces to

Greg Pangrazio   1

send or receive packets the interface needs to be
initialized. There are three main steps to initializing
the interface; configuration, memory allocation, and packet
handling subroutine creation. This code is provided in the
Intel development binaries in the Arcom Development Kit.
The configuration process requires that a MAC address is
assigned to the interface, speed and the interface is
initialized. The MAC addresses are assigned using the
IxEthAccMacAddr macro. The addresses are assigned in
hexadecimal. The line looks like this:

```
IxEthAccMacAddr npeMacAddr1={{0x2,0x0,0x0,0x0,0x0,0x0,}};
```

This assigns the MAC of 0200:0000:0001 to the interface of
NPE1. The interfaces need to be initialized, and this is
done with the following code:

```
if (ixEthAccPortInit(IX_ETH_PORT1)!= IX_ETH_ACC_SUCCESS)
{
    */Error Initializing Ethernet port 1*/
    return(IX_FAIL);
}
ixEthAccCodeletPhyInit();
if (IxEthAccPortUnicastMacAddressSet(IX_ETH_PORT_1,
&npeMacAddr1) != IX_ETH_ACC_SUCCESS)
{
    return(IX_FAIL);
}
```

Greg Pangrazio

1

The port is now initialized but not enabled for traffic.
The configuration to provide transmit and receive
capabilities is the next part of the configuration.
Transmit and receive are each initialized separately, so
for a Snort monitoring interface, only the receive is
needed. The following line of code establishes the receive
registers:


```
If(ixEthAccPortRxCallbackRegister(IX_ETH_PORT_1,
ethPortReceiveCallBack, IX_ETH_PORT_1) !=
IX_ETH_ACC_SUCCESS)

    {

    return(IX_FAIL);

    }
```


The port is now ready to receive frames, however the memory
needs to be allocated so that there is somewhere to send
the information.   A simple statically allocated buffer was
used to reduce complication.   Since the packets are only
used for alerts no response is taken, the packets do not
have to be processed in any specific time.

        After this point one of two kernel modules are loaded.
The first will allow frames to be transmitted and received
by the interface.   This is the standard module used for IP
traffic on one of the Network Processing Engines.   It is
supplied by default with the development library.   The
second module simply puts packets into the buffer that was
described above allowing libpcap to read these packets to
Snort.

Greg Pangrazio

1

Cost Analysis

The Vulcan development kit from Arcom had a purchase price of $997.00 at the time of writing (IXP425 XScale Development Kit for Embedded Linux – Arcom, 2007). This provided a dual interface system capable of processing 100Mb/s of data, on a Linux system. Add to this several hours of development time to get the system processing packets, and building a special binary of the operating system that would keep changes through reboot to create a Snort Network Processor. This system does not have the disk space to store Snort logs and they will need to be sent to another host for storage and analysis. This requires a separate system to store and maintain the logs, as well as to create the operating system for the Network Processor.

To create a PC based stand alone Snort sensor is much cheaper. If you ignore the cost of the development time, the difference is essentially one Network Processor system.

Expanding to Large Networks

The point where a Network Processor based IDS becomes attractive is when traffic throughput is much larger than a single host can handle. A normal desktop PC or light duty server can act as the host for multiple IXP 28xx series devices. This allows for a log host to store alerts as well as the development environment. The real differences are when the data links on the monitored segments reach multiple Gb/s and even 10Gb/s. There are few systems on the market that can handle this kind of throughput and the

Greg Pangrazio

1

prices quickly become astronomically prohibitive. To
monitor two 10Gb/s links the approximate costs would be
about $12,000 providing two 10Gb/s monitor ports and a
1Gb/s management port. This price change becomes much
smaller when compared to the scale of the networks that
would be monitored by this IDS system.

To create the 28xx based system a completely different
type of program would need to be created. The macros that
were defined in the 425 do exist, but there is no Linux
operation system on the 28xx. The IDS system would need to
be created; the rules partitioned into blocks for the Micro
Engines and pushed out to the device. One implementation
uses one Micro Engine to control the interface, and the
rule set is broken up into fifteen blocks. This can be
seen in Figure 7. The packets would then be passed from
Micro Engine to Micro Engine sequentially. Since packets
only trigger an alert once, the first rule set should
contain the most common or simplest rules. This would push
the more difficult rules to the later Micro Engines, where
they would see less traffic and allow the string matching
to have more time. This would reduce overall load on the
IXP, and increase throughput. This is much different than
the system proposed for the IXP425 which was a single
processor that just queues packets until they were
analyzed. Breaking the rules into the subsets allows
parallel processing similar to that of the superscalar
processors that have been proposed for high throughput
applications. Additionally, the 15Gb/s backplane connection
for the IXP28xx would allow the creation of a very
similarly based IPS but instead of the last Micro Engine

Greg Pangrazio   1

discarding the copy of the packet received; it would
transmit it out of the other interface. This very simple
bridging code is the example provided in many of the texts,
and is provided by Intel in the sample code for all of
their IXP based systems. The Snort type rules could then be
duplicated for incoming and outgoing traffic and tuned
based on direction. This would create a high throughput
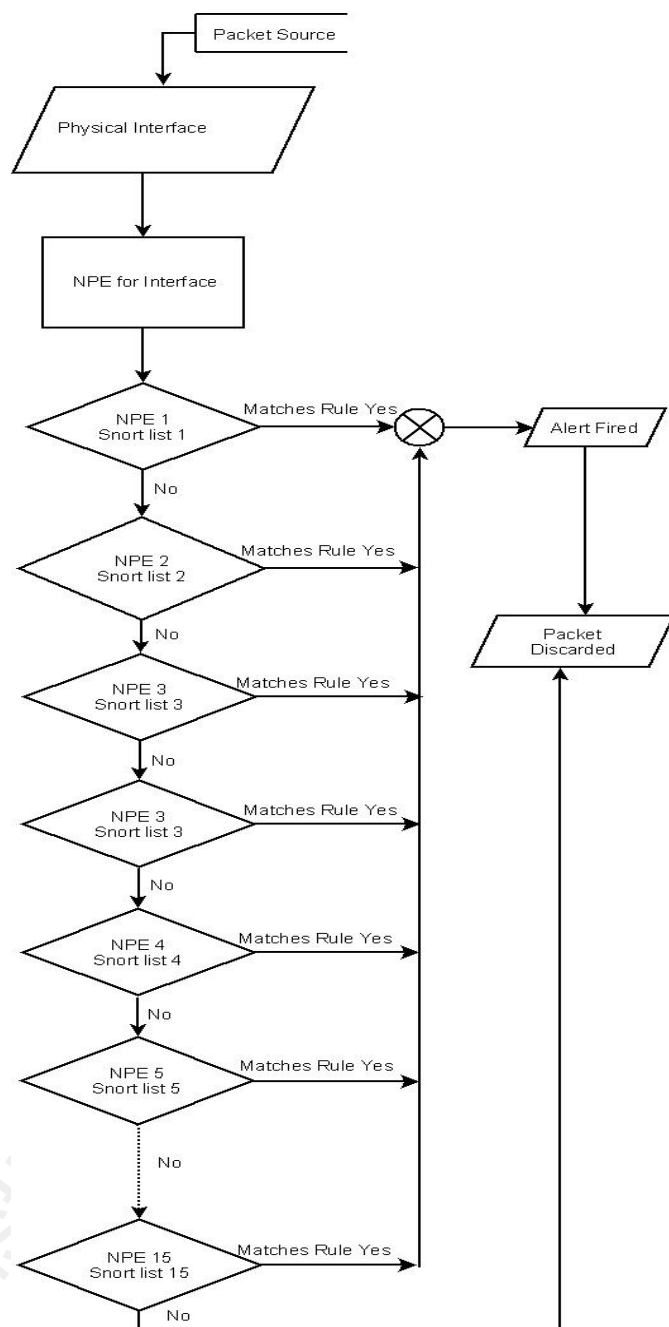very reliable 10Gb/s throughput IPS system.

Greg Pangrazio

Figure 7. Proposed IXP2855 IDS

Another concern with any system today is security. New vulnerabilities are found each day. Currently in IDS the packet is managed by the IP Stack, libpcap and the Snort process. This leaves three main areas of vulnerability. If any one of these were exploited there

are two possibilities, the sensor could be rendered blind,
or the box could be owned by an unauthorized person.
Blinding the sensor would allow other attacks to continue
unnoticed possibly allowing the attacker to exploit other
systems on the network. If the sensor was compromised, it
could be blinded as well or even used as an attacking
point. The best way to reduce this threat is to take the
software out of the chain. If the TCP Stack and the Kernel
that are handling the packets are replaced with hardware as
would be done in the 28xx based sensor. Which would be
completely hardware and not as likely to be vulnerable to
these attacks. If a vulnerability were discovered the
worst that could happen is a temporary blinding of the
sensor. The sensor would not have the ability to connect
outbound to propagate the attack. Since the code is
downloaded to the Micro Engines on boot, and the compiled
code is not stored on the local device, a reboot would
return the sensor to normal operation and original
condition. Restoring a normal sensor of this type may
require a complete format and rebuild from scratch which
may take days.

Results

      Although this shows much real world promise in larger
scale networks, 100Mb/s is not enough to tax most Snort
sensors, so this is not a very good system to test the
feasibility of an IXP based Snort type sensor. It is much
more economical to create a normal host based sensor.
Another huge drawback of the IXP based systems is that they
are not able to log locally. The IXP425 uses flash memory
for storage. By default the Vulcan used in this project has

Greg Pangrazio

a very limited with either 16 or 32 MB of memory which can be expanded with CompactFlash memory cards. Flash memory has a limited life span with approximately 10,000 writes guaranteed. To provide a sample calculation, a life time 100 times the estimated, or 10 million writes over a 5 year period it breaks down to 228 writes an hour or about 3 writes a minute possible. There is however no theoretical limit on the read life time of a Flash memory, so if the alerts were offloaded, the only writes to the Flash memory would be rule set updates. Since Flash memory is written in blocks, as long as multiple rule changes are made at one time, the memory live span would not be an issue. Flash memory has a relatively slow access speed running below 10MB/s of read speed and much lower write speeds. This means that logging entire packets or even just alerts is not possible for a large scale network or a large rule set. The alerts would need to be stored on another device, but the flash memory would be an excellent storage device for the rule set and configuration.

Since the IXP based Snort is a custom application this is by far its biggest drawback. There is no support infrastructure in place for this device, it would be similar to using a home grown IDS. If it breaks the coder would be the support team fixing the problem. The learning curve is steep; this is not programmed in normal C or C++. Every rule change would need to be a custom write, a new kernel build, a new binary image compilation and a TFTP transfer on reboot. This severely slows the implementation of new rules, changing rule updates from editing a text,

Greg Pangrazio

2

file and restarting a process to a major build that takes
hours if not days to implement and debug.

While the IXP based systems show significant promise
in large scale performance, the mediocre performance per
dollar compared to the price of a simple PC pushes the
Network Processor based Snort IDS out of the range of the
average user for Small Office/Home Office applications.
The large scale systems show significant promise in
throughput and scalability, but without a large vendor to
provide technical support and development they ar1e not a
feasible system.

Greg Pangrazio

2

References

Barry, Peter, and Gerard Hartnett. Designing Embedded Networking Applications. Intel Corporation, 2005.

Comer, Douglas E. Network Systems Design Using Network Processors. Boston: Prentice Hall, 2005.

Forouzan, Behrouz A. Data Communications and Networking. 3rd ed. Boston, 2004.

Hallinan, Christopher. Embedded Linux Primer. Upper Saddle River, NJ: Prentice Hall, 2006.

Harris, Guy, Bill Fenner, and Michael Richardson, eds. The Libpcap Project. Vers. 0.9.5. 07 May 2002. 9 May 2007 <http://sourceforge.net/projects/libpcap/>.

"Intel IXP1200 Network Processor Family." Advertisement. 9 May 2007 <http://download.intel.com/design/network/ProdBrf/27904001.pdf>.

"Intel IXP2855 Network Processor." Advertisement. 9 May 2007 <http://download.intel.com/design/network/ProdBrf/30943001.pdf>.

"Intel IXP425 Network Processor." Advertisement. 9 May 2007 <http://download.intel.com/design/network/ProdBrf/27905105.pdf>.

"IXP425 XScale Development Kit for Embedded Linux - Arcom." Arcom.Com. 9 May 2007 <http://www.arcom.com/devkit-linux-vulcan.htm>.

"Network Interface Card – Killer NIC – Network Card." Killer NIC. Bigfoot Networks. 9 May 2007 <http://www.killernic.com/KillerNic/>.

"Snort - the De Facto Standard for Intrusion Detection/Prevention." Snort.Org. 9 May 2007 <http://www.snort.org/>.

Greg Pangrazio

2