International Journal of Research
Available at https://edupediapublications.org/journals

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December 2016

# High-Performance Monitor for a Network Processor

[1]Jarabhala Venkateswaramma
MTech (CSE)
venky.cse2015@gmail.com
GUDLAVALLERU ENGINEERING COLLEGE, Kakinada

[2]M.N.SATISH KUMAR MTech
ASSISTANT PROFESSOR
maganti.nagasatishkumar@gmail.com
GUDLAVALLERU ENGINEERING COLLEGE, Kakinada

## ABSTRACT

*As the Internet becomes integrated into nearly all aspects of everyday life, its reliability grows in importance. This vital communication resource, which has become an inviting target for attackers, must be protected with the same vigor as the end-systems it interconnects. Recent trends in network router architecture towards programmability and flexibility have increased the susceptibility of communication hardware to software attacks which modify intended data processing and forwarding functions. Contemporary routers typically feature network processors, whose protocol processing functions are determined via software. Prior work has shown that these general-purpose software-based processing systems can be attacked with data packets sent through the Internet. As a defense mechanism, the correct functionality of a network processor can be verified by a hardware monitor that observes processor operation and compares it*

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December 2016

*to expected behavior. In the event of an attack, the monitor can interrupt the network processor, suppress malicious behavior, and reset the processor to a usable state for processing of subsequent traffic. In this work, we present several significant advances in hardware monitoring for network processors. A low-overhead monitor architecture that evaluates correct network processor operation in real-time on an instruction-by-instruction basis is described and tested. The monitor is shown to effectively prevent stack smashing attacks on processors that use Harvard architecture, a widely used network processor configuration. Through experimentation, we show that our approach to hardware monitoring does not affect data plane packet throughput. In the event of an attack, malicious packets are dropped while packets of regular network traffic proceed through the network unaffected. A full evaluation of monitor architectural parameters is provided to create an optimized monitor design.*

## INTRODUCTION

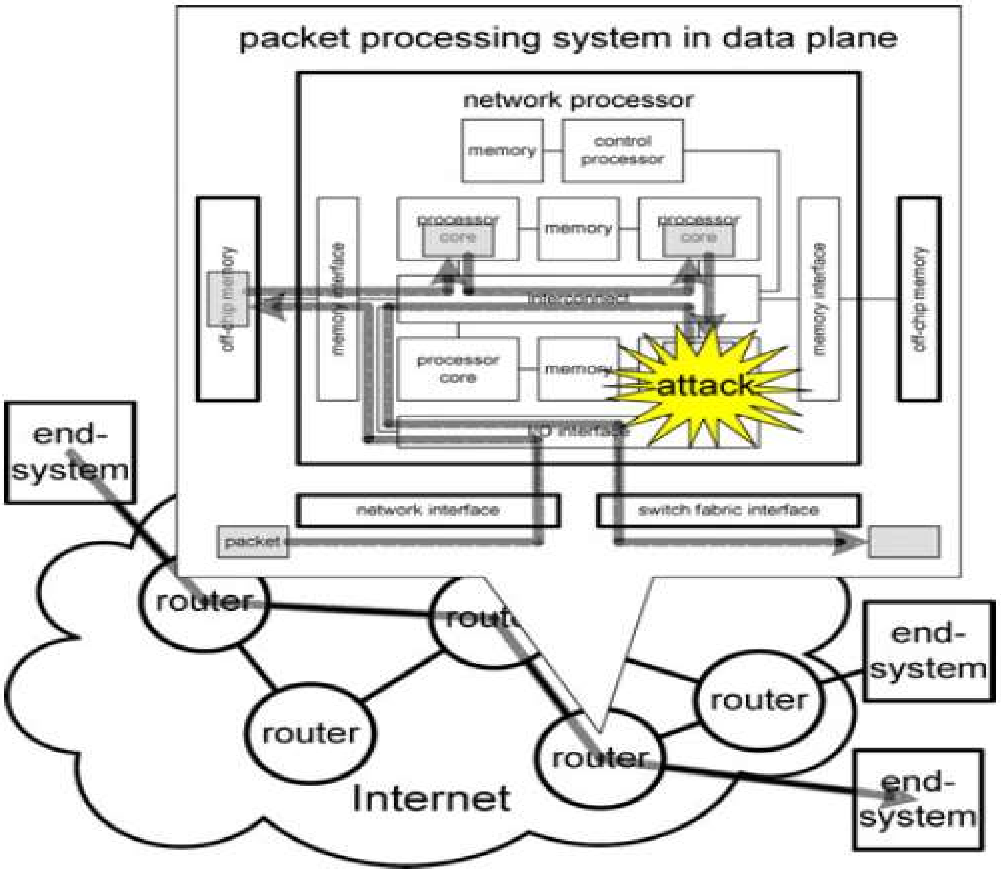
OVER the past 40 years, the Internet has grown from a modest research network to a critical communication resource used by billions of people across the world. Indeed, the reliable operation of this resource has become as critical to commerce, personal interaction, and government activities as traditional utilities, such as the power grid. With the continuing growth of the Internet, there are ongoing technical challenges to meet emerging needs for networking functionality, throughput performance, reliability, and security. To address these challenges, it is necessary to improve the security of networks, including the router devices that constitute the core of the network, with limited compromise in other networking goals. To address this need, we have developed techniques to secure the processing of packets in the data plane of network routers. In contemporary routers, network processors (NPs) are frequently used to perform packet processing. These embedded processors (Fig. 1) typically contain multiple simple RISC-based processing cores that can efficiently manipulate data packets but are potentially vulnerable to attacks initiated by data packets. The functionality of these programmable processors can easily be updated via software updates to provide a broad range of router functionality. However, this programmability leads to a significant drawback; the security of the router is only as strong as the software that programs it. Recently, it has been shown that network processors can be successfully attacked to generate denial-of-service attacks [1]. Using strategically crafted data packets, these attacks exploit weaknesses in the packet processing software of the network processor. Specifically, it is demonstrated that a malicious packet can exploit errors in packet size boundary checking software to overwrite a network processor's stack. This stack smashing attack can then be used to modify the return address of the NP program, forcing control flow jumps to user-supplied code or to library functions already present in the system that can be used in unintended and malicious ways. An important point to note is that this type of attack vector is entirely in the data plane of the network. That is, the attacker does not hack into the control interface of the router, but merely transmits a malformed data packet. Thus, these "in-network" attacks cannot easily be defended against with conventional security mechanisms. Instead, our new high-performance hardware monitoring approach is able to quickly identify this type of attack, drop the offending packet, and reset the router to continue processing normal traffic. Previous efforts have shown that specialized digital hardware can be used to monitor network processor operation and identify deviations from expected behavior [1], [2], [3]. This hardware typically examines the sequence of executed instructions by a processor core to determine if expected program sequencing is exhibited. Deviations from the expected instruction flow indicate that an attack is in progress and the monitor generates control signals to initiate processor core recovery. For most processor cores, hardware-based monitors, rather than software-based detection approaches, are needed since monitors operate at hardware speeds external to the core, thus avoiding packet processing slowdowns. As networking speeds increase, the need for rapid identification of attacks using a minimum amount of additional monitoring hardware is apparent. In particular, these mechanisms must be tuned to the processor configurations most commonly exhibited by contemporary NPs. The research described in this manuscript addresses several important monitoring issues for network processors that must be considered to keep NPs safe from packet-based attacks. For comprehensive protection, every instruction executed by the NP should be validated in real-time, necessitating a high-performance monitoring solution. In general, the tracking of instructions is easily modeled as a finite state machine with a finite number of known paths. Although a non-deterministic finite automaton (NFA) can be used to model instruction sequencing for hardware monitoring [1], [2], this approach can require numerous memory lookups to differentiate multiple parallel states, limiting performance. Alternatively, monitoring can be more quickly performed by tracking coarse basic blocks instead of instructions [3], although this approach can exhibit a lag between when an attack starts execution and when it is identified. Our new approach, based on a deterministic finite automaton (DFA), provides an advance over both of these previous techniques. It has been previously shown that network processors with combined data and instruction memory (von Neumann architecture) are susceptible to attacks that write executable code to the processor stack [1]. However, contemporary network processors generally use separated instruction and data memories (Harvard architecture) for increased code security and performance. These architectures make it impossible to execute code from a stack located in data memory, drawing into question whether data plane attacks are feasible in these types of architectures. In this work, we show that data plane attacks on Harvard architecture NPs are feasible and a new instruction-level hardware monitoring system can be used to defeat them. The specific contributions of our paper are:

1) Design of a high-performance hardware monitoring system for NPs. Our monitor design can perform instruction verification with a single memory read per instruction and thus can operate at speeds suffi- cient to maintain line rate networking data transfer.

2) Algorithm for construction of a deterministic monitoring graph. We present a method to convert the monitoring graph of NP instructions, which initially is nondeterministic due to control-flow changes (e.g. branches), into a deterministic automaton. The representation of the DFA is compacted to allow for a highly efficient implementation in the hardware monitor. 3) Demonstration of an attack on and defense of a Harvard architecture network processor. We demonstrate an in-

network attack through the data plane of the network that exploits an integer overflow vulnerability to smash the processor stack and launch a return-to-library attack. This attack propagates the attack packet and crashes the processor system. We also show that our hardware monitor is effective in defending against this attack and allowing for continued NP-based router operation after attack identi- fication and recovery.

4) A full evaluation of architectural parameters needed to build a hardware-based monitor for a broad collection of nine network processing benchmarks.



**IMPLEMENTATION MODULES:**

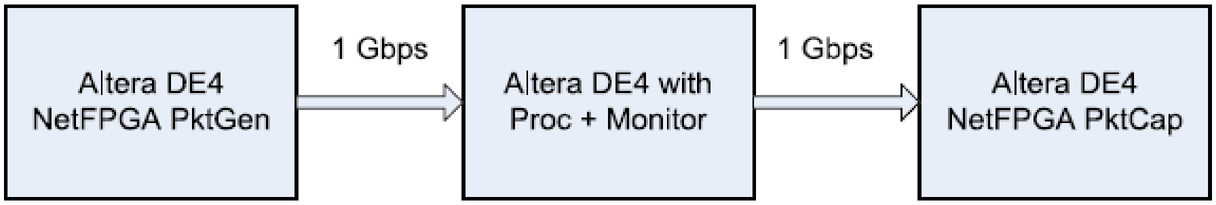
-  network setup
-  harvard architecture attacks

-  network processor and monitoring system
-  hardware monitor system

## MODULES DESCRIPTION:

-  **Network Setup:**

The simple test topology that was used to verify the performance of our monitoring system is shown in Fig.. For hardware experiments, packets were generated and transmitted to the DE4 with the network processor and the monitor at a 1 Gbps line rate by a separate DE4 card serving as a packet generator. This same card was used to receive the processed packets from the card with the NP. The packet generator tool allows for customizing the size and the throughput rate for the test packets.


Altera DE4 NetFPGA PktGen
1 Gbps
Altera DE4 with Proc + Monitor
1 Gbps
Altera DE4 NetFPGA PktCap

-  **hardware monitor system architecture:**

In a Harvard architecture, the code and data are placed in separate physical address spaces. Separate buses provide instruction and data access, with each potentially having different word widths, timing and memory address structures. The instructions are usually stored in read-only memory while data is stored in read-write memory. Since a program counter cannot point to addresses in the data memory, code injection attacks are difficult to perform in a Harvard memory architecture. Even if an attacker successfully writes a malicious code in the stack, it will not be executed. Even though general memory error techniques (integer overflow, heap overflow etc.) cannot be used to generate code injection attacks, Francillon and Castelluccia [17] demonstrated that code injection attacks are still feasible on a Harvard architecture processor using a return-oriented programming technique. Here, an attacker takes control of return instructions in the stack to chain attack code from an existing library function. Since the code is already present in executable memory, the attack will not be prevented from running. In this section, we describe how such an attack can be constructed for the networking environment and how our monitor can detect it. Fig. 9 shows portions of congestion management protocol (CM) and an IPV4 packet forwarding application used to build an attack on the network processor system. The congestion management protocol inserts a custom protocol header in the packet header space between the IP header and the UDP header. During this operation, the code needs to make sure the new packet size does not exceed the maximum datagram length (the boxed instruction in the CM code).

## Network Processor and Monitoring System

The model of an NP system, including a monitor, is shown in Fig. 4. System operation is coordinated by a control processor that forwards incoming packets to the NP. Offchip external memory provides bulk storage for the packets and programs used by the NP. The NP, control processor, monitor, and system interface ports are interconnected using an on-chip communications infrastructure. The same control processor used for the assignment of programs and packets to the NP is used for the loading of monitoring graphs into the monitor. Monitoring graphs for all applications executed by the NP are stored on-chip in a centralized monitor memory. For some network processor systems this storage could be implemented in non-volatile storage (e.g. EEPROM). Alternatively, the storage could be implemented in DRAM with monitoring graphs downloaded to the system each time power is applied. Encryption is used to cipher monitoring graph information when it is input into the system using the external interface port. A standard AES core is used to decipher the monitoring graphs and place them in centralized monitor memory.

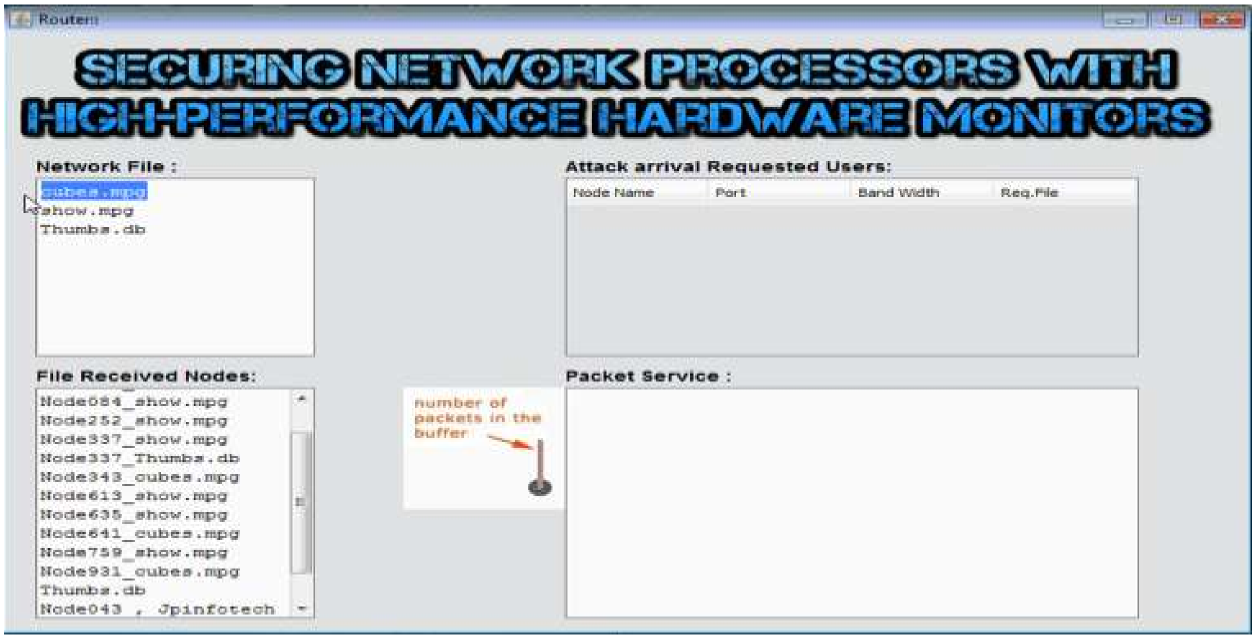
- **hardware monitor system:**

The system-level architecture of the network processor system with security monitor is shown in Fig. 2. The network processor shown on the left of the figure is based on a conventional Harvard architecture with separate data memory for network packets and processing state and instruction memory for packet processing code. For simplicity, only a single processor core is shown; the system can easily be extended for multiple processor cores. The processing monitor on the right side of the figure verifies the operation of the processor instruction-by-instruction. For every instruction that is executed on the processor core, a hash value of the executed operation is reported to the monitor. The monitor uses the comparison logic to compare the reported hash value to the information that is stored in the monitoring graph. The monitoring graph is derived by offline analysis of the packet processing code binary.

Any attack on the system necessarily needs to change the operation of the processor core (otherwise the attack is not effective). This deviation leads to the processor reporting hash values that do not match with the monitoring graph. The comparison logic can detect this deviation and reset the processor in response. In networking, such a reset and recovery operation is very simple: The current packet is dropped (i.e., the packet buffer is cleared), the processing state is reset (i.e., the stack is reset), and processing continues with the next packet. Since most packet processing operations are not statefull and there is no guarantee that packets are reliably delivered, no further recovery actions are necessary.

## SCREEN SHOTS

**Router**


Router

SECURING NETWORK PROCESSORS WITH HIGH-PERFORMANCE HARDWARE MONITORS

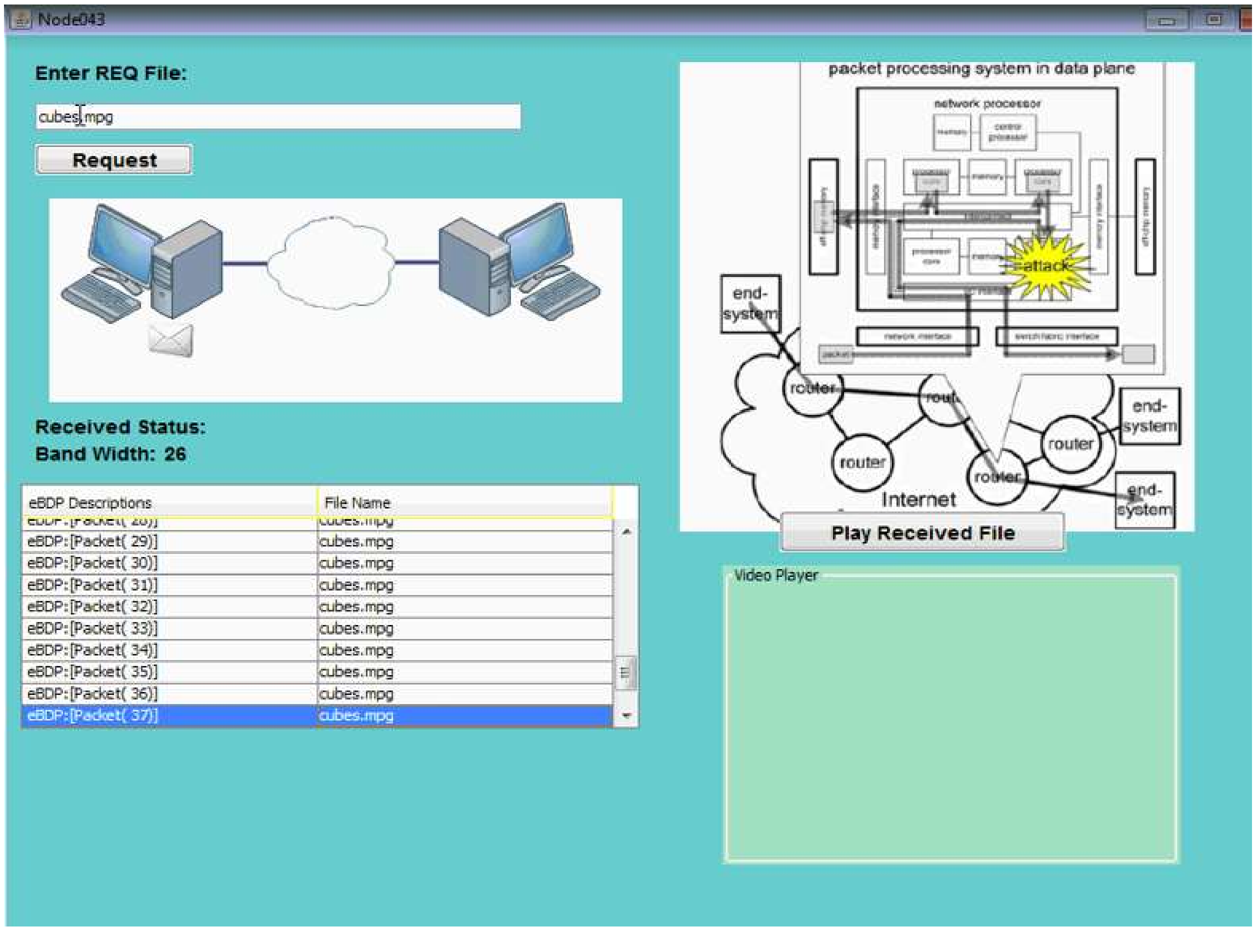
Network File :
cubes.mpg
show.mpg
Thumbs.db

Attack arrival Requested Users:

| Node Name | Port | Band Width | Req.File |
|---|---|---|---|

File Received Nodes:
Node084_show.mpg
Node252_show.mpg
Node337_show.mpg
Node337_Thumbs.db
Node343_cubes.mpg
Node613_show.mpg
Node635_show.mpg
Node641_cubes.mpg
Node759_show.mpg
Node931_cubes.mpg
Thumbs.db
Node043 , Jpinfotech

Packet Service :


Node043

Enter REQ File:
cubes.mpg
Request

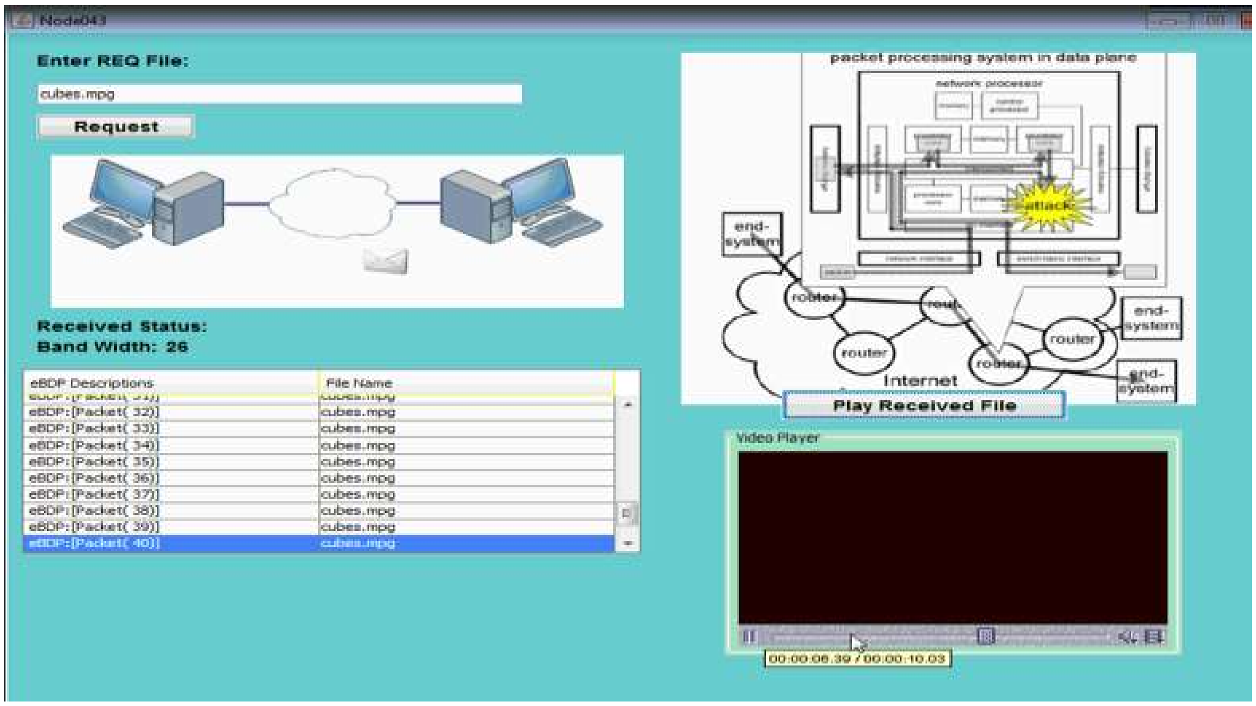
Received Status:
Band Width: 26

| eBDP Descriptions | File Name |
|---|---|
| eBDP:[Packet( 28)] | cubes.mpg |
| eBDP:[Packet( 29)] | cubes.mpg |
| eBDP:[Packet( 30)] | cubes.mpg |
| eBDP:[Packet( 31)] | cubes.mpg |
| eBDP:[Packet( 32)] | cubes.mpg |
| eBDP:[Packet( 33)] | cubes.mpg |
| eBDP:[Packet( 34)] | cubes.mpg |
| eBDP:[Packet( 35)] | cubes.mpg |
| eBDP:[Packet( 36)] | cubes.mpg |
| eBDP:[Packet( 37)] | cubes.mpg |

Play Received File

Video Player

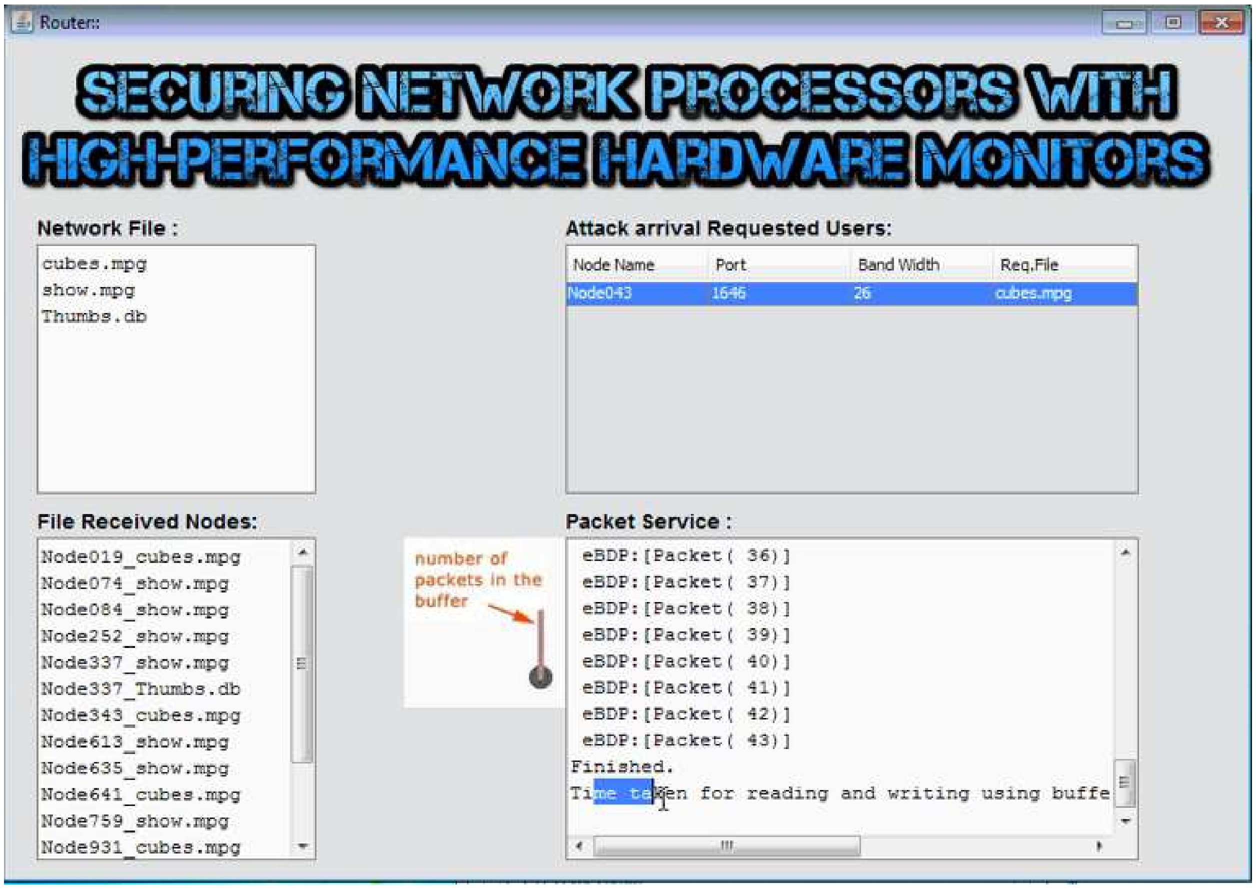
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December 2016



cubes.mpg

Received Status:
Band Width: 26

| eBDP Descriptions | File Name |
|---|---|
| eBDP:[Packet( 32)] | cubes.mpg |
| eBDP:[Packet( 33)] | cubes.mpg |
| eBDP:[Packet( 34)] | cubes.mpg |
| eBDP:[Packet( 35)] | cubes.mpg |
| eBDP:[Packet( 36)] | cubes.mpg |
| eBDP:[Packet( 37)] | cubes.mpg |
| eBDP:[Packet( 38)] | cubes.mpg |
| eBDP:[Packet( 39)] | cubes.mpg |
| eBDP:[Packet( 40)] | cubes.mpg |



SECURING NETWORK PROCESSORS WITH HIGH-PERFORMANCE HARDWARE MONITORS

Network File :

```
cubes.mpg
show.mpg
Thumbs.db
```

Attack arrival Requested Users:

| Node Name | Port | Band Width | Req.File |
|---|---|---|---|
| Node043 | 1646 | 26 | cubes.mpg |

File Received Nodes:

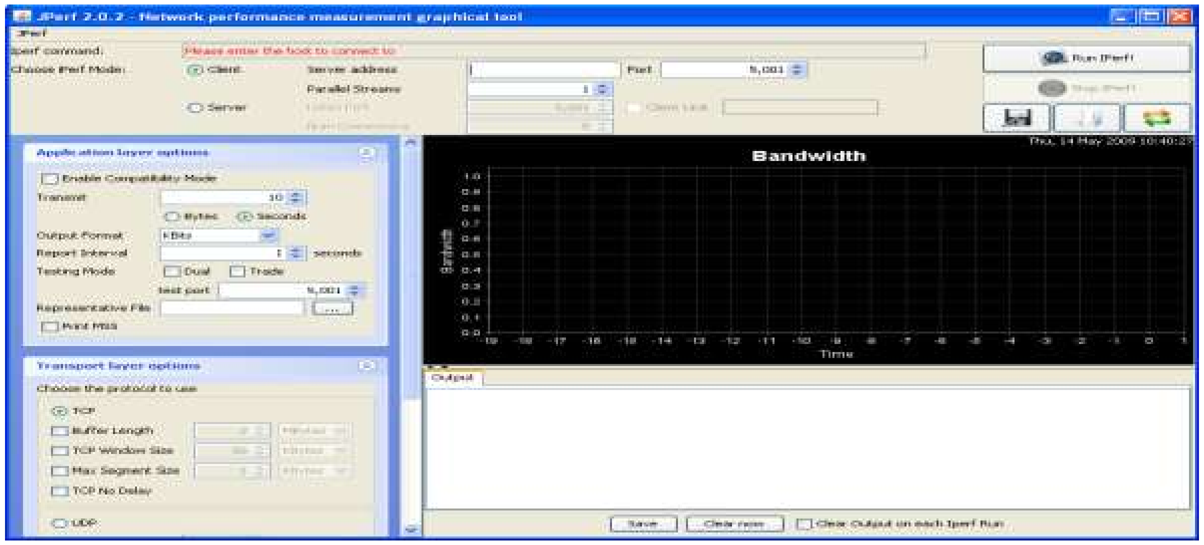
```
Node019_cubes.mpg
Node074_show.mpg
Node084_show.mpg
Node252_show.mpg
Node337_show.mpg
Node337_Thumbs.db
Node343_cubes.mpg
Node613_show.mpg
Node635_show.mpg
Node641_cubes.mpg
Node759_show.mpg
Node931_cubes.mpg
```

Packet Service :

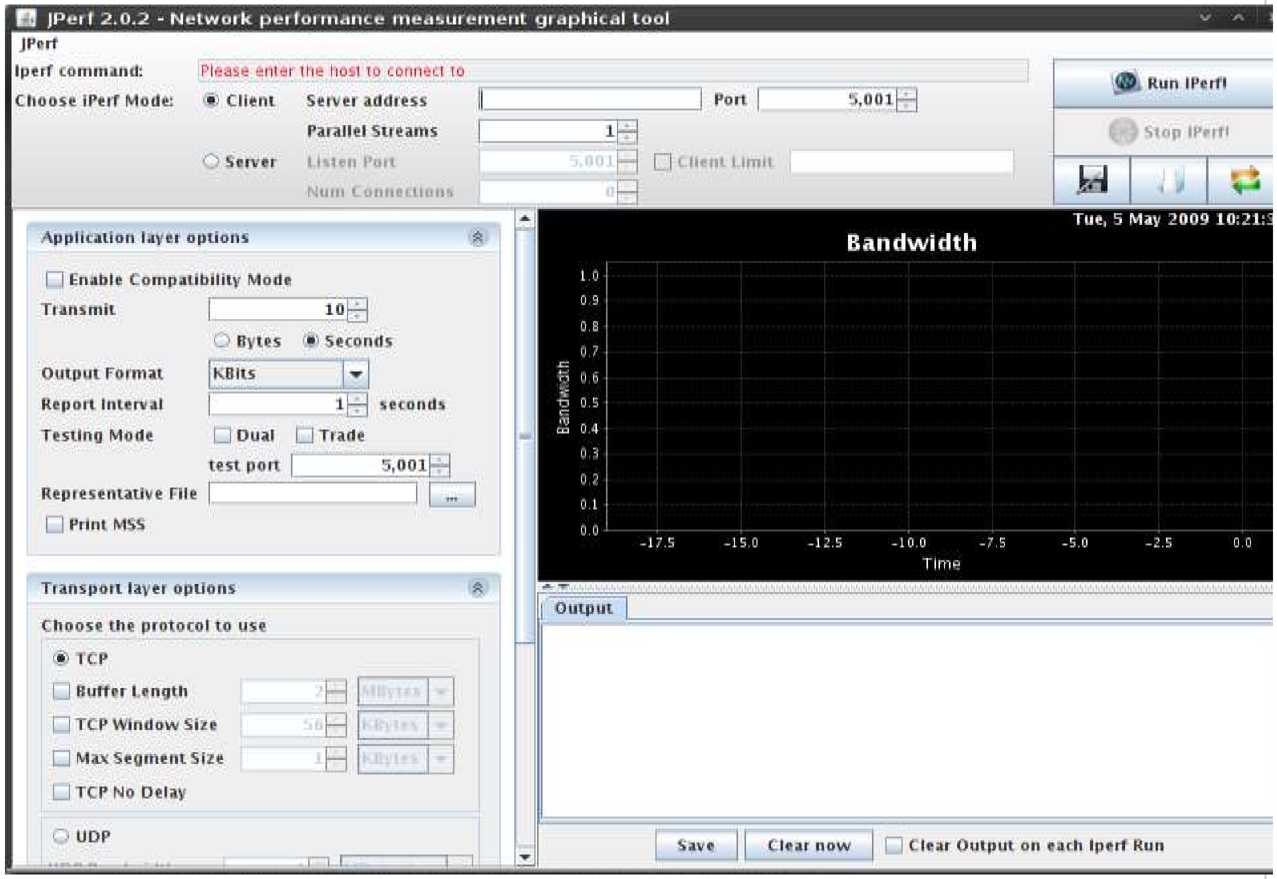
```
 eBDP:[Packet( 36)]
 eBDP:[Packet( 37)]
 eBDP:[Packet( 38)]
 eBDP:[Packet( 39)]
 eBDP:[Packet( 40)]
 eBDP:[Packet( 41)]
 eBDP:[Packet( 42)]
 eBDP:[Packet( 43)]
Finished.
Time taken for reading and writing using buffe
```

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December 2016

User interface in JPerf 2.0.2 on Windows


JPerf 2.0.2 - Network performance measurement graphical tool
Run IPerf!
Stop IPerf!
Bandwidth
Thu, 14 May 2009 10:40:27
Time
Output
Save
Clear now

User interface in JPerf 2.0.2



Iperf command: Please enter the host to connect to
Choose iPerf Mode: [✓ Client] Server
Server address:
Port: 5,001
Parallel Streams: 1
Listen Port: 5,001
Client Limit:
Num Connections: 0

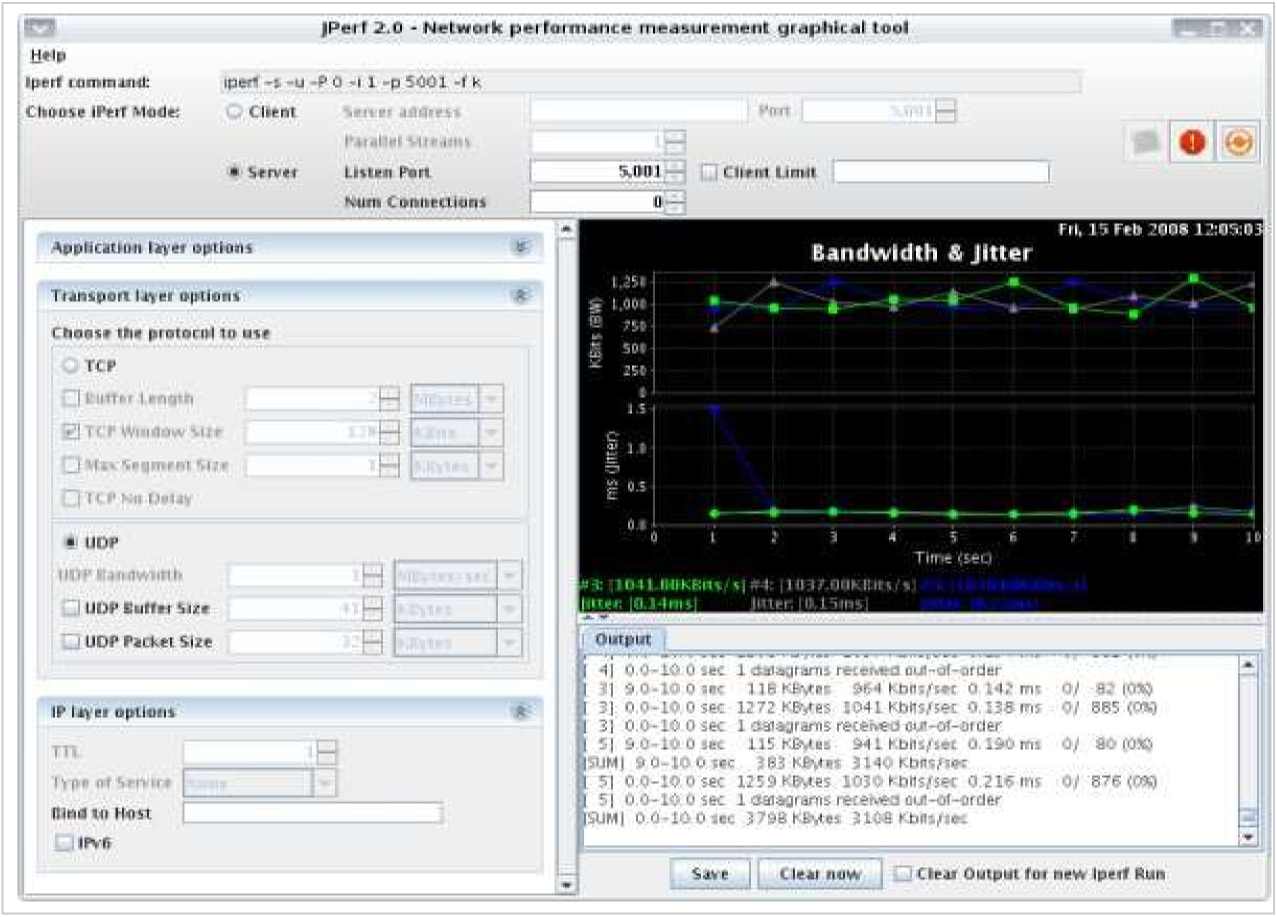
Application layer options
Enable Compatibility Mode
Transmit: 10
Bytes [✓ Seconds]
Output Format: KBits
Report Interval: 1 seconds
Testing Mode: Dual Trade
test port: 5,001
Representative File:
Print MSS

Transport layer options
Choose the protocol to use
[✓ TCP]
Buffer Length: 2 MBytes
TCP Window Size: 56 KBytes
Max Segment Size: 1 KBytes
TCP No Delay
UDP

Clear Output on each Iperf Run

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December 2016

The server mode


JPerf 2.0 - Network performance measurement graphical tool
Help
Application layer options
Transport layer options
IP layer options
Bandwidth & Jitter
Fri, 15 Feb 2008 12:05:03
KBits (BW)
ms (Jitter)
Time (sec)
#3: [1041.00KBits/s] #4: [1037.00KBits/s]
Jitter: [0.14ms] Jitter: [0.15ms]
Output
Save
Clear now
Clear Output for new Iperf Run

Iperf command: `iperf -s -u -P 0 -i 1 -p 5001 -f k`

Choose iPerf Mode: Client [✓ Server]

Listen Port: 5,001 · Client Limit: ☐ · Num Connections: 0

Choose the protocol to use: TCP [✓ UDP]

UDP Buffer Size ☐ · UDP Packet Size ☐

TTL: 1 · Bind to Host: · IPv6 ☐

```
[  4]  0.0-10.0 sec  1 datagrams received out-of-order
[  3]  9.0-10.0 sec   118 KBytes   964 Kbits/sec  0.142 ms   0/   82 (0%)
[  3]  0.0-10.0 sec  1272 KBytes  1041 Kbits/sec  0.138 ms   0/  885 (0%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
[  5]  9.0-10.0 sec   115 KBytes   941 Kbits/sec  0.190 ms   0/   80 (0%)
[SUM]  9.0-10.0 sec   383 KBytes  3140 Kbits/sec
[  5]  0.0-10.0 sec  1259 KBytes  1030 Kbits/sec  0.216 ms   0/  876 (0%)
[  5]  0.0-10.0 sec  1 datagrams received out-of-order
[SUM]  0.0-10.0 sec  3798 KBytes  3108 Kbits/sec
```

## CONCLUSION

In this paper, we have described a high-performance monitor for a network processor that requires only a single memory lookup per network processor instruction. This single memory lookup is maintained regardless of the complexity of the NP program using an NFA-to-DFA translation of the monitoring graph. Our monitor, which tracks individual NP instructions, has been verified in hardware using an NP with a Harvard architecture. Our results show that the use of DFA only increases memory size by 5.7 percent, compared to previous NFA approaches. Our prototype implementation of the monitoring systems shows our design is so efficient that even extremely large amounts of attack traffic do not lead to a degradation of throughput performance of the system. We believe that this work presents an important step towards deploying effective and efficient hardware protection mechanisms for network processors in the Internet. Future work includes optimizing the monitoring memory architecture to consider the caching of frequently used monitoring graphs. Also, the possibility of crafting attacks which include instructions with the same sequence of hash values as legitimate code could be evaluated. Finally, the use of pre-deployment simulation to determine dynamic branch targets for monitoring could be considered.

## REFERENCES

[1] D. Chasaki and T. Wolf, “Attacks and defenses in the data plane of networks,” IEEE Trans. Dependable Secure Comput., vol. 9, no. 6, pp. 798–810, Nov. 2012. [2] S. Mao and T. Wolf, “Hardware support for secure processing in embedded systems,” IEEE Trans. Comput., vol. 59, no. 6, pp. 847– 854, Jun. 2010.

[3] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, “Secure

embedded processing through hardware-assisted run-time monitoring," in Proc. Des., Autom. Test Eur. Conf. Exhibition, Mar. 2005, pp. 178–183.

[4] The Cisco QuantumFlow Processor: Cisco's Next Generation Network Processor, Cisco Systems, Inc., San Jose, CA, USA, Feb. 2008.

[5] OCTEON Plus CN58XX 4 to 16-Core MIPS64-Based SoCs, Cavium Networks, Mountain View, CA, USA, 2008.

[6] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," Computer, vol. 38, no. 4, pp. 34–41, Apr. 2005. [7] D. Geer, "Malicious bots threaten network security," Computer, vol. 3F8, no. 1, pp. 18–20, 2005.

[8] D. Moore, C. Shannon, and J. Brown, "Code-Red: A case study on the spread and victims of an Internet worm," in Proc. 2nd ACM SIGCOMM Workshop Internet Meas., Nov. 2002, pp. 273–284.

[9] A. Cui, Y. Song, P. V. Prabhu, and S. J. Stolfo, "Brave new world: Pervasive insecurity of embedded network devices," in Proc. 12th Int. Symp. Recent Adv. Intrusion Detection, Sep. 2009, pp. 378–380.

[10] R. G. Ragel and S. Parameswaran, "IMPRES: Integrated monitoring for processor reliability and security," in Proc. 43rd Annu. Conf. Des. Autom., Jul. 2006, pp. 502–505.

[11] J. Zambreno, A. Choudhary, R. Simha, B. Narahari, and N. Memon, "SAFE-OPS: An approach to embedded software security," Trans. Embedded Comput. Sys., vol. 4, no. 1, pp. 189–210, Feb. 2005.

[12] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," in Proc. ACM Conf. Comput. Commun. Secur., Nov. 2005, pp. 340–353.

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December 2016

[13] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, and R. Vaslin, "Reconfigurable hardware for high-security/high-performance embedded systems: The SAFES perspective," IEEE Trans. Very Large Scale Integration Syst., vol. 16, no. 2, pp. 144–155, Feb. 2008.

[14] D. Wagner and D. Dean, "Intrusion detection via static analysis," in Proc. IEEE Symp. Secur. Priv., May 2001, pp. 156–168.

[15] H. Chandrikakutty, D. Unnikrishnan, R. Tessier, and T. Wolf, "High-performance hardware monitors to protect network processors from data plane attacks," in Proc. IEEE/ACM Des. Autom. Conf., Jun. 2013, pp. 1–6.