Design of a Secure Router System for Next-Generation Networks

Tilman Wolf and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
Email: {wolf,tessier}@ecs.umass.edu

***Abstract*—Computer networks are vulnerable to attacks, where the network infrastructure itself is targeted. Emerging router designs, which use software-programmable embedded processors, increase the vulnerability to such attacks. We present the design of a Secure Packet Processing Platform (SPPP) that can protect these router systems. We use an instruction-level monitoring system to detect deviations in processing behavior. If such deviations are detected, a recovery system is invoked to restore the system into an operational state. Our preliminary results show that most attacks can be detected within a single instruction. The system overhead for secure monitoring is limited to a fraction of the overall space, memory, and power budget.**

***Index Terms*—network security, router design, embedded processor, processor monitor**

I. Introduction

Computer networks, in particular the Internet, are an essential part of the global communication infrastructure. As our society relies increasingly on data networks for business and personal communication, the value of this infrastructure increases and makes it more attractive to potential attackers. Networks have been used for several decades to launch attacks against computer systems (e.g., remote end-system intrusion). More recently, the network itself has become a target for attacks, in which Distributed Denial-of-service (DDoS) attacks have been used to overwhelm Internet access links of target systems (e.g., for the purpose of extortion).

Separately, novel functions are being introduced into networks: ubiquitous network access to personal mobile devices, new network applications and business models, and new protocols for high-performance and secure communication. To support these features, the computer networking community is currently in the process of redesigning the fundamental architecture of the next-generation Internet [1] (with an intended deployment in the coming decade). The redesign of the Internet architecture encompasses many aspects ranging from all-optical transmission to sensor networks, and distributed protocols and applications. Routers, which represent the fundamental building blocks of any network, are the focus of our work.

Routers need to perform packet processing to implement protocols correctly. Traditionally, this processing has been implemented in application-specific integrated circuits (ASIC) for performance reasons. As networks have become more diverse and new protocols have been developed, the fixed functionality of ASICs has been replaced by software-programmable packet processing systems using general-purpose processors. This type of data path programmability is already available in some Internet routers and is expected to be present in a majority of network routers in the near future [2]. These router designs provide more functionality into the network core and thus increase the area of attack.

The vulnerability of network devices is inherent in their functionality. Routers connect links in a network and are thus remotely reachable. For attackers, this reachability translates into an easy path for targeting remote exploits onto an important infrastructure component. Current router systems do not provide security mechanisms at the hardware level and thus are at significant risk for attack. The combination of increased value and more potential vulnerabilities indicates that it is only a question of time until computer network infrastructure becomes a prime target of attack.

In this paper, we present a design that provides fundamental security capabilities in next-generation router systems to protect crucial networking infrastructure. Our main idea is to expand packet processing systems – which are the central components in router systems – to include monitoring subsystems that can verify correct operation. In particular, a monitor can determine when a packet processor deviates from the sequence of operations that is considered correct. Our specific contributions in this paper are:

- The design of a Secure Packet Processing Platform (SPPP), which uses hardware monitors to detect deviation from normal processing behavior at the instruction level.
- The design of a recovery system for the SPPP that can recover a router system in the case problems with packet processing are detected.
- The evaluation of the effectiveness of the system as well as an estimation of the resource overhead for SPPP.

The remainder of the paper is structured as follows. Section II discusses related work. The overall SPPP architecture is introduces in Section III. Performance estimates are presented in Section IV. Section V summarizes and concludes this paper.

II. Related Work

Extensions to the feature set of the original Internet architecture [3] have been proposed in many forms. Many proposed architectures include software processing in the data path of routers, spanning active networks [4], programmable

routers [2], and configurable protocol stacks [5]. In next-generation networks, where deviations from the current Internet architecture [1] can be considered, a variety of protocol features and data path services can be implemented in routers [6], [7]. These services could be implemented on a variety of platforms ranging from workstation routers [8] to programmable routers [9] and virtualized router platforms [10]. Most high-performance processing platform use an embedded multi-processor system-on-chip (MPSoC) at their core, for example a network processor (Intel IXP2400 [11], EZchip NP-3 [12], or LSI APP [13]). Several router designs that use such embedded packet processing platforms have been demonstrated [14], [15].

Although our work focuses on packet processing platforms, their vulnerability is certainly not the only security concern in networks: end-system vulnerabilities have led to large-scale "bot nets" [16], various types of DoS attacks have been deployed [17], timing attacks can affect protocol behavior [18], and protocol vulnerabilities can be exploited [19]. However, as programmability is increased, the protection of packet processing becomes an increasingly important concern. So far, this topic has received little attention. Some aspects of this problem tie into embedded system security due to the embedded nature of packet processing platforms.

A wide range of approaches can be used to attack embedded systems [20]. Ravi et al. describe mechanisms to achieve physical security by employing tamper resistant designs [21]. Wood et al. consider a networked scenario where systems are exposed to remote attacks [22]. Embedded systems are also susceptible to side-channel attacks (e.g., differential power analysis [23]), although we do not consider this issue in our work. Gogniat et al. [24] have developed a general, hardware-based architecture to protect embedded systems against a range of attacks, although the proposed monitors are not described.

To address general security concerns in packet processing systems, constrained programming environments have been proposed [25]. However, next-generation networks require a fully functional general-purpose programming environment. In our work, we achieve security by monitoring processors. Monitoring has been used in the system by Arora et al. [26] and the IMPRES system [27], but we use a finer granularity of monitoring. The SAFE-OPS system by Zambreno et al. [28] uses information that is collected across multiple executed instructions to determine valid operation. This system can detect errors and attacks at the end of such a sequence, whereas our monitor may immediately detect the first instruction that deviates [29].

The use of model comparison to perform anomaly and intrusion detection has been used in selected networking domains (e.g., mobile ad-hoc networks [30]). In our case, the problem is simpler since our model is derived from a protocol description or the actual binary of the protocol implementation. Thus, there is no guess-work on the accuracy of the model; it is exactly the same as the actual packet processing application.

The majority of previous efforts related to network router recovery have focused on recovery from hardware faults rather than network attacks. In Zhou et al. [31], whole packets are copied from the network router to an attached host processor memory. If a fault is detected, the router is restarted with the saved packets. Router state for this system is periodically checkpointed to facilitate router restart. In Huang et al. [32], a fault tolerant router structure is presented. Much of the hardware in the router is replicated to provide an alternate routing path if a single hardware failure occurs. In Luo and Fan [33], unused processors in a multicore network processor are used to provide redundancy. If a specific core fails, processing is moved to an idle core. Although effective, none of these approaches address monitoring and recovery from network attacks that we describe below.

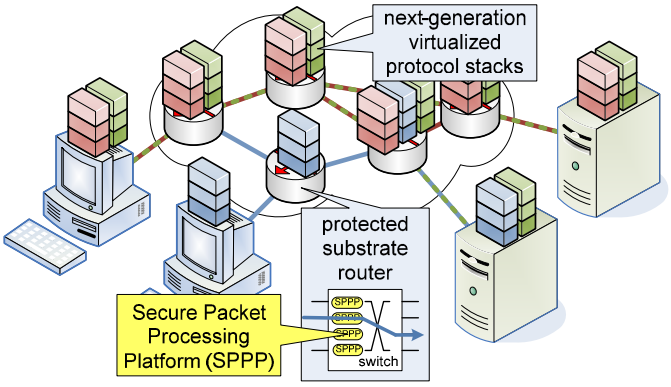

Fig. 1. Secure Packet Processing Platform (SPPP) in Next-Generation Virtualized Network Architecture.

III. System Architecture

The goal of our work is to design a secure packet processing platform (SPPP) for next-generation Internet routers. This SPPP can be embedded on router ports as illustrated in Figure 1. Before detailing the SPPP architecture, we briefly discuss the security model for our work.

A. Security Model

For our work, we define a security model that is representative of the current Internet and what we expect from the next-generation Internet. Attacks on network routers are motivated by a number of different goals. The following list illustrates this point but is not meant to be a complete enumeration of all possible scenarios:

- Denial of service attack (e.g., disabling links or an entire device, generating overwhelming traffic, configuring routing loops);
- Modification of stored or monitored data (e.g., tampering with log files);
- Extraction of secret information (e.g., reading of cryptographic key material); and
- Hijacking of a hardware platform (e.g., reprogramming of processors to send unsolicited emails).

To illustrate the types of attacks we consider in our work, we present two specific attack examples. Note that our system design can defend against a broader range of attacks than these

two, but the vulnerabilities shown are symptomatic of those encountered in router systems.

1) Attack Examples: Two illustrative attack scenarios are

- Processing Attack: In this scenario, an attacker uses the transmission of a data packet that causes the packet processing program to misbehave. The attack could be based on a buffer overflow, where certain data fields can smash the processor stack. In many cases, changes to the stack cause the program to crash. However, it is also possible for an attacker to change the control flow such that malicious code (e.g., contained in the packet) is executed. This type of attack has been used successfully to gain access to end-systems through widely deployed and vulnerable software. One of the most famous examples is the Code Red worm that exploited a vulnerability in a service of the Windows operating system and used it to spread itself around the globe [34], [35]. As routers provide more functionality, it becomes more difficult to formally validate the correct operation of all protocol features in all virtual slices. Thus, it becomes feasible for this type of attack to be widely used by attackers.
- Denial-of-Service Attack: In this scenario, we consider a situation where processing is performed correctly, but the overall router system still behaves incorrectly. One example is the use of a multicast function to (intentionally or accidentally) launch a denial of service attack. It can be envisioned that multicast can be implemented through a loop that iterates over the interfaces to which a packet needs to be forwarded. If this loop does not terminate (e.g., due to an incorrect parameter in the packet or multicast data structure) and continues to duplicate packets, the outgoing links can easily be saturated while the router cannot forward any other packets. Such a problem is particularly damaging in router systems as they are located inside the network and are typically connected via high-bandwidth links. Unlike end-systems, which have very limited uplink connections, a router could easily generate a denial of service attack with many Gigabits per second of traffic.

In our system, the monitoring subsystem is able to detect these attacks and initiate an immediate recovery process that foils the attempt. It is also possible to use I/O monitoring to detect an imbalance of incoming and outgoing packets caused by such an attack.

2) Security Requirements: The above attack scenarios rely on the ability of an attacker to gain remote access to the system and change its behavior (i.e., change in instruction memory) or its data (i.e., change in data memory). It is important to note that in most attack scenarios a modification of behavior is necessary even when modification of or access to data is the ultimate goal of the attack. This leads to two main security requirements, which ensure that the router continues to perform correct protocol processing:

- Benign packets should be processed according to protocol specifications without interference from possible attacks;
- Malicious traffic should be identified as quickly as possible (packets that belong to a connection that causes malicious processing on the SPPP may be discarded

In addition to these functional requirements, there are several performance requirements: fast detection, accuracy, low overhead, and quick recovery.

3) Attacker Capabilities: The capabilities of an attacker that define the potential attack space include the following:

- An attacker can send arbitrary data and control packets;
- An attacker can modify instruction and data memory through exploits. We do not specify which exploits can be used, but target a solution that can deal with the effects of any such attack;
- An attacker cannot modify the source code or binary of the protocol implementation before it is installed on the router. Thus, the basis of what we define as correct operation cannot be tampered with. However, once the binary is installed on the router, the attacker may change the binary as stated above;
- An attacker cannot physically access the router. Thus, attacks are limited to remote exploits.

While we make some constraining assumptions, we believe the defined attacker capabilities are general and representative of typical network attacks.

B. Secure Packet Processing Platform

Any security mechanism for packet processing needs to consider the following important criteria, which are met by our design:

- Independence: A monitoring subsystem should use independent system resources that overlap as little as possible with the target of a potential attack. In particular, the use of a single embedded processing system for both protocol processing and security-related monitoring is a bad choice. If an intruder can access the protocol processor, then the monitor may be vulnerable to attacks.
- Low Overhead: Embedded packet processing systems require a lightweight security solution that considers the limitations of MPSoCs in terms of additional logic and memory for monitoring.
- Fast Detection and Recovery: A monitoring subsystem should be able to react as quickly as possible to an attack. In particular, attacks that simply change memory state or extract private data may require only a few instructions to cause damage. Therefore, it is important to be able to detect an attack within a few instructions. To maintain the operation of a network router, it is also important to quickly recover from an attack.

The SPPP architecture is shown in Figure 1. The two key components are the monitoring subsystem shown on the right and the recovery subsystem shown on the left. We discuss each subsystem in more detail below. It is important to note that computer networks can be logically divided into two parts: the data plane, where data traffic is handled, and the control plane, where routers exchange control information (e.g., routing

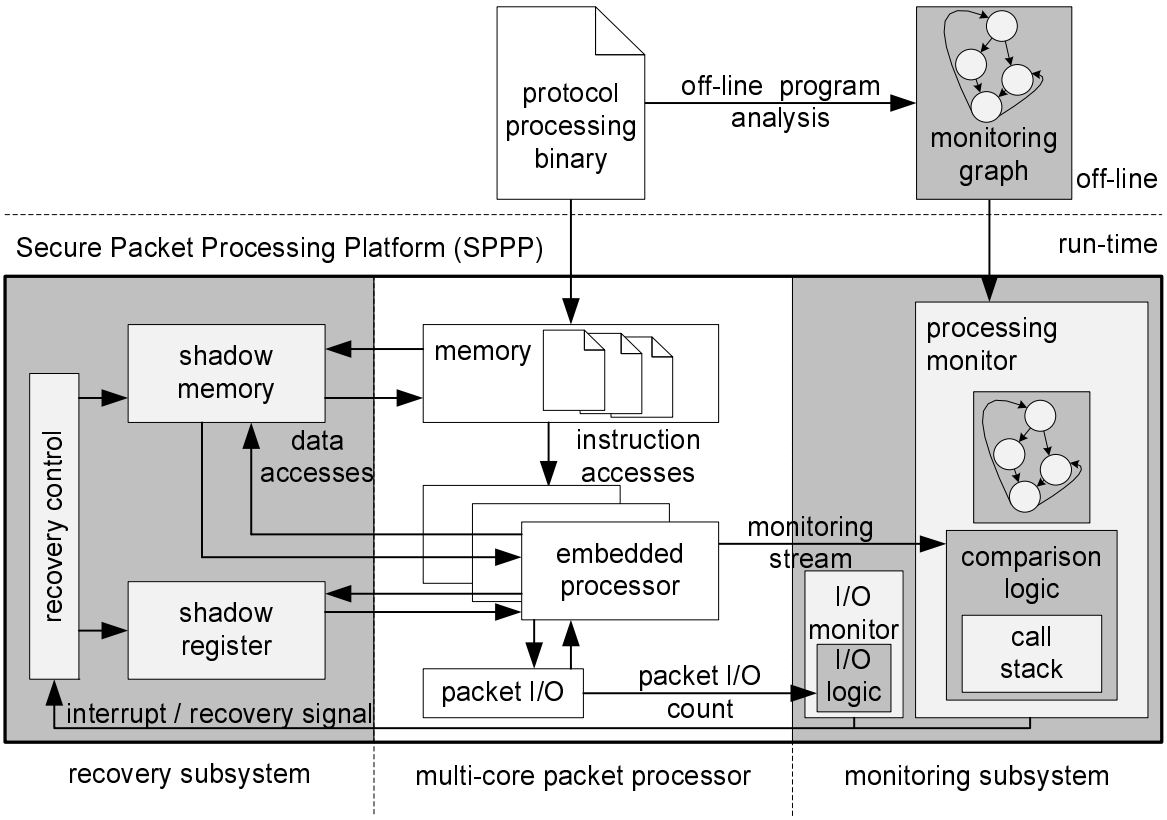

Fig. 2. Architecture of a Secure Packet Processing Platform.

updates). The SPPP focuses on data plane processing since it is the most crucial functionality of a router, but the monitoring concept is equally applicable to control processors. Control processors are typically simple unicore processors with small numbers of processing tasks and thus they are less challenging than data path network processors. Therefore, we believe that a good solution for data plane processing can also be used to protect the control plane.

C. Monitoring Subsystem

The monitoring system consists of two components: the processing monitor, which is based on our prior work on security in embedded single core systems [29], and the I/O monitor, which is a new component that is designed specifically for network systems.

1) Processing Monitor: The main idea behind our processing monitor, which is illustrated in Figure 2, is to analyze the binary code of a protocol processing implementation and derive an augmented control flow graph. The embedded processor reports on the progress of application processing at run time by sending a stream of information to the monitoring system. The monitoring system compares the stream to the expected behavior of the program as derived from the executable code. If the processor deviates from the set of possible execution paths, it is assumed that an attacker has altered the instruction store or program counter to alter the behavior of the system. In that case, the recovery subsystem restores the processing state stored before a particular packet was processed. Our evaluation of an embedded system benchmark shows that this monitoring technique can detect deviations from expected program behavior within a single instruction while only requiring a small amount of additional logic and memory on the order of one tenth of the size of the protocol processing implementation binary.

It is important to note that this design does not use intrusion detection heuristics, which may be slow and computationally expensive. Instead we use a novel multi-core monitoring platform that can detect deviations from normal protocol processing steps within a few instruction cycles. Such rapid detection is essential for high-speed networks since the processing time for a packet totals just a few microseconds.

2) I/O Monitor: The I/O monitor is designed to track the I/O behavior of the router system. Monitoring takes place at a granularity that is coarser than the per-instruction monitoring of the processing monitor. The I/O monitor correlates the flow of outgoing packets to the flow of incoming packets. By tracking such information, the monitor can determine when conditions occur that are considered unusual from a networking perspective. These conditions may not be detected by the processing monitor since they may be caused by the correct execution of instructions. Examples of such conditions include the dropping of incoming packets that is not due to congestion and the transmission of large numbers of packets that is not triggered by incoming packets (e.g., denial of

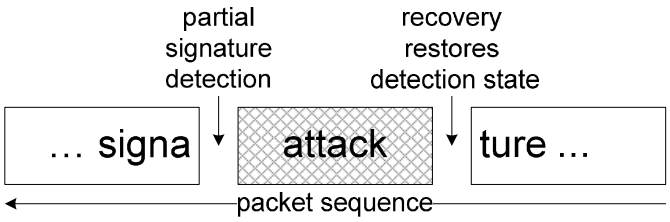

Fig. 3. Example of State Recovery in Intrusion Detection.

service attack, etc.).

The I/O monitor uses information gathered at the I/O interfaces (i.e., not at the processor cores) to infer the correct operation of the router. Possible types of information in the design of the I/O logic are:

- Count: The count of outgoing packets is related to the count of incoming packets (over a window) to indicate the general flow of packets. This approach can be extended to account for dropped packets, multicast, and similar special cases.
- Attribution: By uniquely marking incoming packets and passing this marking through the system, it is possible to attribute outgoing packets to their "origin." Such attribution allows the identification of misbehaving processing features (or network slices). Based on this technique, per-flow or per-slice transmission limits can be enforced.
- Timing: By recording timing information between incoming and outgoing packets, the system delay can be measured (assuming that packets can be distinguished using attribution). Delay information can be used to identify functional and performance problems.

Based on the information obtained from the I/O monitor, higher-level security problems can be identified and system recovery can be initiated.

D. Recovery Subsystem

The recovery of the router system after an attack (i.e., deviation from protocol processing) has occurred is an important aspect of a security system. Ideally, an attack should have as little impact as possible to avoid a denial-of-service abuse of the monitoring system. It would seem that recovery in network systems is a simple process because the Internet Protocol inherently does not provide delivery guarantees and thus packet loss is acceptable (and can be dealt with through TCP and similar protocols). However, as more stateful processing features are introduced into the network, we require more effective recovery mechanisms.

To illustrate the importance of recovery, consider an intrusion detection system that scans for a signature of malicious traffic in a stream of packets [36], [37]. For effective detection, it is necessary to consider signatures that are split across multiple packets (see Figure 3). An attacker could hide a split signature by introducing an "attack packet" between the packets. The attack could possibly alter or destroy the detection state that was stored at the end of the first packet. In such a case, the second part of the signature could not be matched successfully and malicious traffic would reach its target. To avoid this and many similar problems, we introduce a recovery mechanism into our Secure Packet Processing Platform.

The recovery mechanism design is based on per-flow checkpointing. In such a system, processing state is preserved at the granularity of packets. The recovery mechanism is based on the following process:

1) When processing a packet, record all memory operations to a shadow memory. This allows for state recovery if processing of subsequent packets causes a processor failure.
2) When the processing of a packet has successfully completed, backup the processor register values to shadow registers and commit shadow memory operations.
3) When the processing of a packet causes a processor failure (as identified by the processing monitor), restore register values from shadow registers and clear (i.e., do not commit) shadow memory operations. This step restores the processing state to values stored before processing was started.

The processor pipeline is flushed at the end of packet processing (successful or not). In the case of processor failure, the packet causing the problem is discarded.

Our design differs from previous checkpointing approaches used by microprocessors [38], [39] by providing checkpointing at a finer granularity (a packet). The lack of caches in network processors simplifies the per-packet backup of critical register and data values. Note that maintaining the state information for checkpointing is not significantly more expensive than maintaining per-flow processing state, which is already necessary to support the level of custom processing that can be expected to be encountered in the next-generation Internet.

IV. Results

We show results on the monitoring effectiveness for single-core and multi-core systems.

A. Monitoring Stream Information

As shown in Figure 2, the monitoring subsystem tracks processing progress via the monitoring stream provided by the processor. In our prior work, we have evaluated different monitoring stream information and their effectiveness for detecting attacks [29]. We briefly provide an overview on the results from this work, which was focused on a single-core implementation, before discussing extensions to multi-core systems.

We consider the following "patterns" as design alternatives for the monitoring information stream:

- Address Pattern: The address of an instruction is a unique indicator, but it does not contain any information about the operation that corresponds to the instruction. An attacker can simply replace instructions without being detected.
- Opcode Pattern: The opcode pattern tracks the instruction opcode and thus represents a program's functionality.

TABLE I
PERFORMANCE OF MONITOR TO DETECT BIT FLIP ATTACKS (FROM [29]). THE RESULTS ARE BASED ON 100 SIMULATIONS USING THE GSM APPLICATION.

Monitoring pattern	undetected bit flips out of 100 runs	avg. no. of instr. to detection
address	87	49.1
opcode	60	1.2
load/store	76	15.8
control flow	74	23.6
hash4	6	1

An attacker would need to replace a program with an identical sequence of opcodes.

- Load/Store Pattern: This pattern tracks memory access operations and their target registers (since target addresses cannot be determined statically). This pattern shows the same vulnerabilities as the opcode pattern.
- Control Flow Pattern: The control flow pattern tracks control flow operations (i.e., branches, calls, returns) and allows the monitor to track any change in the program counter. This pattern exhibits a vulnerability that is similar to an address pattern since there it contains no information about the actual operation of the processor.
- Hashed Pattern: A pattern that we developed in prior work and that addresses the shortcomings of the above patterns is the hashed pattern [29]. In this case, several pieces of information (in our case an instruction address and an instruction word) can be compacted to a smaller hash value. This is particularly useful since opcodes, operands, etc. can consume a lot of memory space. This pattern can be used with different lengths of hash functions. We use the function name hash$n$ to indicate that an n-bit hash function is used. To circumvent this monitor, an attacker would need to craft an instruction sequence with identical hash values, which is very difficult, especially for larger values of n.

The quantitative tradeoffs between these patterns are considered below.

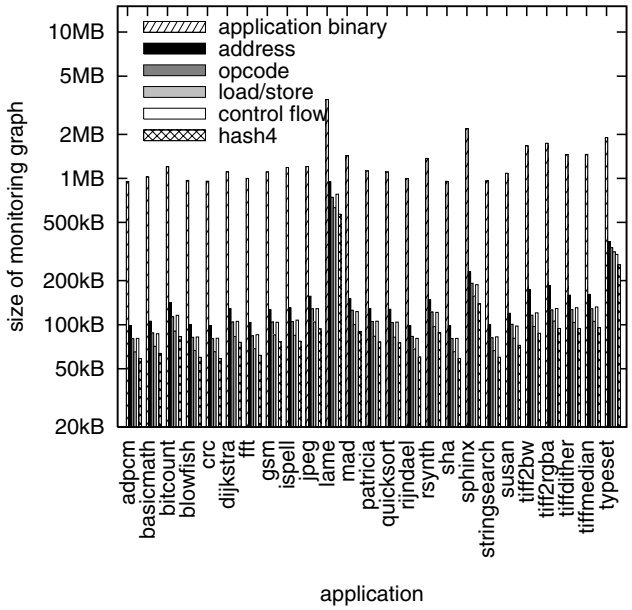

Fig. 4. Size of Monitoring Graph for Different Benchmarks and Information Streams (from [29]).

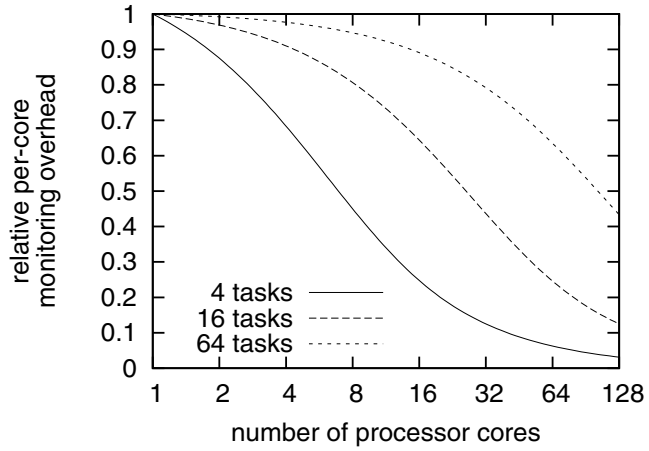

Fig. 5. Multi-Core Monitoring Overhead.

B. Single-Core Monitor

The single-core monitor is the basic building block for monitoring. The main performance considerations are the ability to detect attacks quickly and to do so with low overhead. We use the MiBench benchmark suite [40] to generate workloads that are similar to what can be expected on an SPPP. We employ the SimpleScalar simulator [41] to extract relevant monitoring information and the `objdump` utility for binary analysis to generate monitoring graphs.

Figure 4 shows the size requirements of different information streams graphs. As a comparison, the size of the application binary is also shown. Most graphs require only in the order of 10% of the size of the application binary and thus do not incur significant overhead.

Table I shows the detection performance of the monitor. The hash4 pattern can detect all but 6% of the attacks (due to hash collisions, which can be reduced with a larger hash size). The important metric is the number of instructions that are executed until the attack is detected. This delay provides a potential window for the attacker. For the hash4 pattern, the attack can be detected in a single instruction.

These results show that effective attack detection on real applications can be achieved with an overhead of around 10% additional instruction memory and the logic necessary for implementing the comparison monitor.

C. Multi-Core Monitor

When monitoring multiple processor cores in a single system, it is possible to amortize the overhead for storing the monitoring graph among cores that execute the same code.

While packet processing systems typically do not use SIMD processing, they often execute the same code independently on multiple cores [42]. Similar to how share instruction stores are used, shared monitoring graph storage can be employed. To illustrate the effectiveness of such a sharing architecture, Figure 5 shows the overhead for monitoring for varying numbers of cores and distinct tasks. Processors that execute the same task can share the storage used by monitors (but not the comparison logic). It is assumed that task allocations are done independently of each other. The overhead is the per-core overhead relative to a single monitor. As the number of cores increases, more sharing is possible and the overhead decreases. With an increase in task diversity, sharing becomes more difficult and the overhead increases. Nevertheless, the relative overhead in a multi-core architecture with sharing is less than that of a single-core system. Thus, our proposed monitoring system will require relatively less system resources when using highly parallel multi-core packet processors.

V. Summary and Conclusions

We have presented a novel approach to addressing security vulnerabilities within the networking infrastructure itself. Our SPPP architecture uses monitoring to detect an attack and a recovery subsystem to limit its impact. Our results show that the proposed architecture can detect attacks and can be implemented efficiently. The system may be deployed in next-generation network testbeds to assess the practical impact of defending network infrastructure.

References

[1] A. Feldmann, "Internet clean-slate design: what and why?" *SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 59–64, Jul. 2007.

[2] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Proc. of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.

[3] D. D. Clark, "The design philosophy of the DARPA Internet protocols," in *Proc. of ACM SIGCOMM 88*, Stanford, CA, Aug. 1988, pp. 106–114.

[4] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 5–18, Apr. 1996.

[5] N. T. Bhatti and R. D. Schlichting, "A system for constructing configurable high-level protocols," in *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, Cambridge, MA, Aug. 1995, pp. 138–150.

[6] T. Wolf, "Service-centric end-to-end abstractions in next-generation networks," in *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Arlington, VA, Oct. 2006, pp. 79–86.

[7] S. Ganapathy and T. Wolf, "Design of a network service architecture," in *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, HI, Aug. 2007, pp. 754–759.

[8] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64–76, Jan. 1991.

[9] L. Ruf, K. Farkas, H. Hug, and B. Plattner, "Network services on service extensible routers," in *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)*, Sophia Antipolis, France, Nov. 2005.

[10] 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.

[11] *Intel Second Generation Network Processor*, Intel Corporation, 2005, http://www.intel.com/design/network/products/npfamily/.

[12] *NP-3 – 30-Gigabit Network Processor with Integrated Traffic Management*, EZchip Technologies Ltd., Yokneam, Israel, May 2007, http://www.ezchip.com/.

[13] *APP3300 Family of Advanced Communication Processors*, LSI Corporation, Aug. 2007, http://www.lsi.com/.

[14] J. S. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman, and D. Zar, "Supercharging PlanetLab: a high performance, multi-application, overlay network platform," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Kyoto, Japan, Aug. 2007, pp. 85–96.

[15] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, Aug. 2006, pp. 3–14.

[16] D. Geer, "Malicious bots threaten network security," *Computer*, vol. 38, no. 1, pp. 18–20, 2005.

[17] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, Aug. 2003, pp. 99–110.

[18] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, Aug. 2003, pp. 75–86.

[19] J. Xia, L. Gao, and T. Fei, "Flooding attacks by exploiting persistent forwarding loops," in *IMC '05: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, Berkeley, CA, Oct. 2005.

[20] S. Parameswaran and T. Wolf, "Embedded systems security – an overview," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 173–183, Sep. 2008.

[21] S. Ravi, A. Raghunathan, and S. Chakradhar, "Tamper resistance mechanisms for secure, embedded systems," in *Proc. of 17th International Conference on VLSI Design (VLSI Design 2004)*, Mumbai, India, Jan. 2004, pp. 605–611.

[22] A. Wood and J. A. Stankovic, "Denial of service in sensor networks," *IEEE Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.

[23] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '99)*, ser. Lecture Notes in Computer Science, vol. 1666. London, United Kingdom: Springer-Verlag, 1999, pp. 388–397.

[24] 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 Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 144–155, Feb. 2008.

[25] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles, "PLAN: A packet language for active networks," in *Proc. of the Third ACM SIGPLAN International Conference on Functional Programming Languages*. ACM, 1998, pp. 86–93.

[26] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Secure embedded processing through hardware-assisted run-time monitoring," in *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)*, Munich, Germany, Mar. 2005, pp. 178–183.

[27] R. G. Ragel and S. Parameswaran, "IMPRES: integrated monitoring for processor reliability and security," in *Proc. of the 43rd Annual Conference on Design Automation (DAC)*, San Francisco, CA, USA, Jul. 2006, pp. 502–505.

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

[29] S. Mao and T. Wolf, "Hardware support for secure processing in embedded systems," in *Proc. of 44th Design Automation Conference (DAC)*, San Diego, CA, Jun. 2007, pp. 483–488.

[30] G. F. Cretu, J. J. Parekh, K. Wang, and S. J. Stolfo, "Intrusion and anomaly detection model exchange for mobile ad-hoc networks," in *Proc. of 3rd IEEE on Consumer Communications and Networking Conference (CCNC 2006)*, Las Vegas, NV, Jan. 2006, pp. 635–639.

[31] Y. Zhou, V. Lakamraju, I. Koren, and C. M. Krishna, "Software-based failure detection and recovery in programmable network interfaces," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1539–1550, Nov. 2007.

[32] N.-F. Huang, Y.-T. Chen, Y.-C. Chen, C.-N. Kao, and J. Chiou, "A network processor-based fault-tolerance architecture for critical network equipments," in *Proc. of Information Networking, Networking Technologies for Broadband and Mobile Networks, International Conference (ICOIN)*, ser. Lecture Notes in Computer Science, vol. 3090. Busan, Korea: Springer Verlag, Feb. 2004, pp. 763–772.

[33] Y. Luo and J. Fan, "Fault tolerant practices on network processors for dependable network processing," in *Proc. of IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, Miami, FL, Apr. 2008.

[34] *CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL*, CERT Coordination Center, Carnegie Mellon University, Pittsburgh, PA, Jul. 2001.

[35] D. Moore, C. Shannon, and J. Brown, "Code-Red: a case study on the spread and victims of an internet worm," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, Marseille, France, Nov. 2002, pp. 273–284.

[36] H.-J. Jung, Z. Baker, and V. Prasanna, "Performance of FPGA implementation of bit-split architecture for intrusion detection systems," in *Proc. of 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece, Apr. 2006.

[37] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An fpga-based network intrusion detection architecture," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 118–132, Mar. 2008.

[38] B. T. Gold, J. C. Smolens, B. Falsafi, and J. C. Hoe, "The granularity of soft-error containment in shared memory multiprocessors," in *Proc. of Workshop on System Effects of Logic Soft Errors*, Urbana-Champaign, Apr. 2006.

[39] R. Teodorescu, J. Nakano, and J. Torrellas, "SWICH: A prototype for efficient cache-level checkpointing and rollback," *IEEE Micro*, vol. 26, no. 5, pp. 28–40, Sep. 2006.

[40] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. of IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, Dec. 2001.

[41] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," Department of Computer Science, University of Wisconsin in Madison, Tech. Rep. 1342, Jun. 1997.

[42] Q. Wu and T. Wolf, "On runtime management in multi-core packet processing systems," in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, San Jose, CA, Nov. 2008.