Computer Communications 34 (2011) 598–606

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

# Securing the data path of next-generation router systems

Tilman Wolf *, Russell Tessier, Gayatri Prabhu

*Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA*

ARTICLE INFO

*Article history:*
Received 9 December 2009
Received in revised form 21 February 2010
Accepted 18 March 2010
Available online 23 March 2010

*Keywords:*
Network security
Router design
Embedded processor
Processor monitor

ABSTRACT

As the technology used to implement computer network infrastructure advances, networking resources are becoming more vulnerable to attack. Recent router designs are based on general-purpose programmable processors, which increase their potential vulnerability. To address this issue, a Secure Packet Processing platform has been developed that can flexibly protect emerging router systems. Both instruction-level operation of embedded processors and I/O operations of router ports are monitored to detect anomalous behavior. If such behavior is detected, a recovery system is invoked to restore the system into an operational state. Experimental results show that processor-based attacks can generally be determined by a processing monitor within a single instruction. I/O anomalies, including unexpected packet broadcast or delay, can be detected by an I/O monitor with limited overhead. Overall, the system overhead for secure monitoring is limited to a fraction of the overall system space, memory, and power budget.



## 1. 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 (ASICs) 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 in 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. A recent study on network devices in today's Internet shows that 2.46% of enterprise-class devices and 41.6% of consumer-class devices exhibit vulnerabilities [3]. As the network's attack value increases and expanded amounts of network functionality expose the attack surface, it is only a question of time until computer network infrastructure becomes a prime target of attack.

In this paper, we present a router design that provides fundamental security capabilities for the data path of next-generation network 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

* Corresponding author. Address: ECE Department, Knowles 211C, University of Massachusetts, Amherst, MA 01003, USA. Tel.: +1 413 545 0757; fax: +1 413 545 1993.

*E-mail addresses:* wolf@ecs.umass.edu (T. Wolf), tessier@ecs.umass.edu (R. Tessier), gprabhu@ecs.umass.edu (G. Prabhu).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.03.019

the sequence of operations that is considered correct by monitoring the instruction flow. 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 if problems with packet processing are detected.
- The evaluation of the effectiveness of the system as well as an estimation of the resource overhead for the SPPP.

The remainder of the paper is structured as follows. Section 2 discusses related work. The overall SPPP architecture is introduced in Section 3. Performance estimates are presented in Section 4. Section 5 summarizes and concludes this paper.

## 2. Related work

Extensions to the feature set of the original Internet architecture [4] have been proposed in many forms. Many proposed architectures include software processing in the data path of routers, spanning active networks [5], programmable routers [2], and configurable protocol stacks [6]. 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 [7,8]. These services could be implemented on a variety of platforms ranging from workstation routers [9] to programmable routers [10] and virtualized router platforms [11]. Most high-performance processing platform use an embedded multi-processor system-on-chip (MPSoC) at their core, for example a network processor (Intel IXP2400 [12], EZchip NP-3 [13], or LSI APP [14]). Several router designs that use such embedded packet processing platforms have been demonstrated [15,16].

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" [17], various types of DoS attacks have been deployed [18], timing attacks can affect protocol behavior [19], and protocol vulnerabilities can be exploited [20]. 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 [21]. Ravi et al. describe mechanisms to achieve physical security by employing tamper resistant designs [22]. Wood et al. consider a networked scenario where systems are exposed to remote attacks [23]. Embedded systems are also susceptible to side-channel attacks (e.g., differential power analysis [24]), although we do not consider this issue in our work. Gogniat et al. [25] 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 [26]. 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. [27] and the IMPRES system [28], but we use a finer granularity of monitoring. The SAFE-OPS system by Zambreno et al. [29] 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 [30].

Abadi et al. [31] also use a control flow graph for monitoring program execution. Nakka et al. [32] introduce integrity checks into the micro-architecture and use special check instructions. Ragel et al. [33] introduce microinstructions to monitor for fault detection, return address checks, and memory boundary checks. This differs from our approach in that these approaches require changes in the machine code to implement the necessary checks.

A completely different approach to ensuring secure execution of programs is the tagging of non-instruction memory pages with NX (No eXecute) or XD (eXecute Disable) bits. This approach prevents a control flow change to a piece of code that belongs to data memory. This mechanism is useful for the prevention of buffer overflow attacks. It does not consider a scenario where an attacker overwrites instruction memory. Another approach to defending against buffer overflow attacks is described by Shao et al. in [34], where bound checks are used and function pointers are protected by XORing them with a secret key.

The use of model comparison to perform anomaly and intrusion detection has been used in selected networking domains (e.g., mobile ad hoc networks [35]). 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. [36], 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. [37], 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 [38], 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.

This manuscript provides an extension of our previous work in secure router design [39]. An I/O monitor has been designed and implemented in an FPGA-based platform to verify functionality. As described in Section 3.3, the implementation is flexible enough to address a number of router weaknesses. Results presented in Section 4 illustrate the limited overheads needed to provide this security.

## 3. 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 Fig. 1. Before detailing the SPPP architecture, we briefly discuss the security model for our work.

### 3.1. 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).

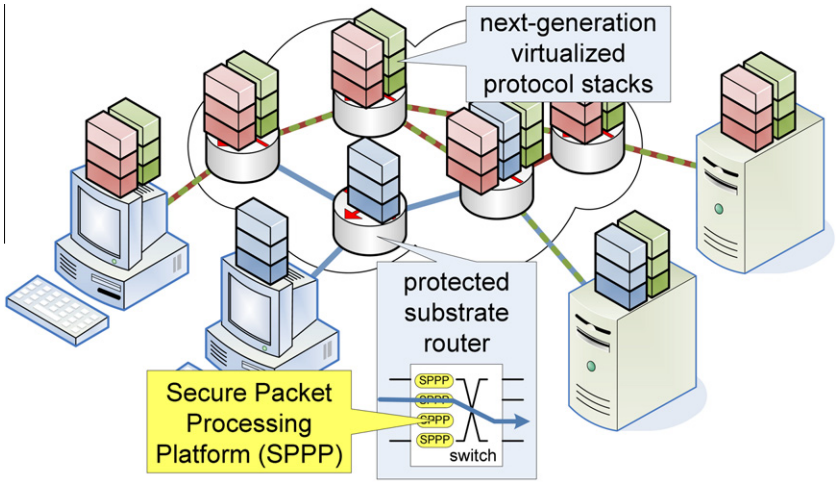


**Fig. 1.** Secure Packet Processing Platform (SPPP) in next-generation virtualized network architecture.

- 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.

*3.1.1. 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 [40,41]. 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.

*3.1.2. 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.

*3.1.3. 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 constraining assumptions on what the attacker can do, we believe the defined attacker capabilities are general and representative of typical network attacks.

### 3.2. 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 Fig. 2. 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 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.

### 3.3. 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 [30], and the I/O monitor, which is a new component that is designed specifically for network systems.

#### 3.3.1. Processing monitor

The main idea behind our processing monitor, which is illustrated in Fig. 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, the processor no longer executes instructions that are part of the correct program. Thus, 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

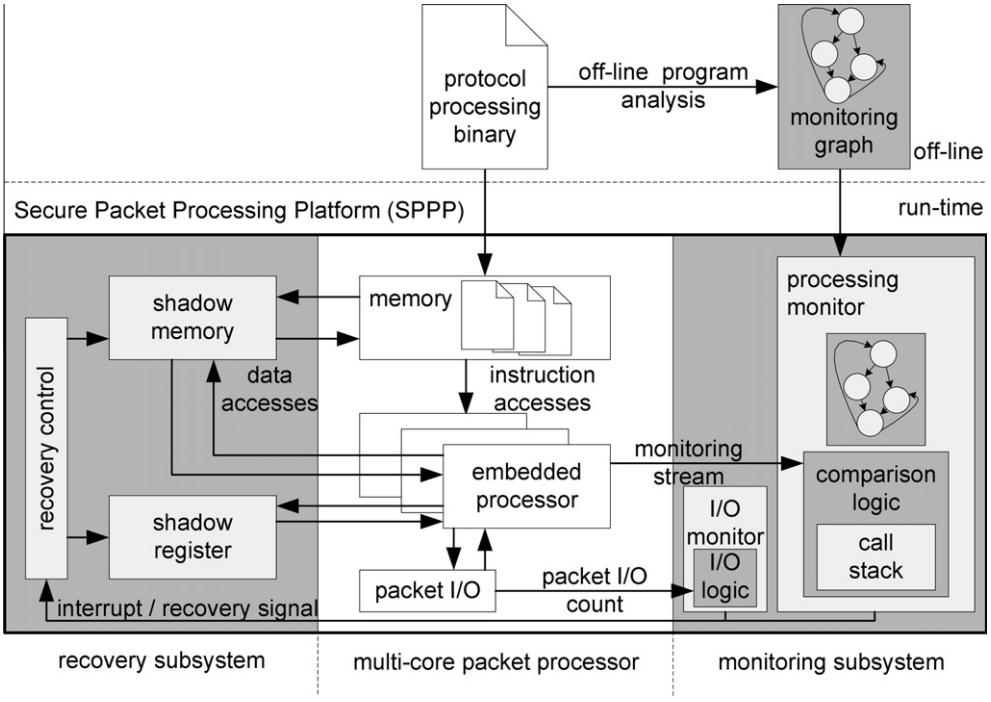


**Fig. 2.** Architecture of a Secure Packet Processing Platform.

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.

*3.3.2. 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-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.
- Integrity: By recording the CRC or checksum over portions of the packet that are not rewritten during normal packet forwarding (e.g., TCP header and payload when using IP forwarding), the monitor can verify the integrity of the packet at transmission. Unauthorized modifications to higher-layer headers and packet payload can be identified with this process. (This check can be disabled or limited to some range of the packet for services that require changes to the packet payload.)

A more detailed architecture of an I/O monitor that implements attribution and integrity checks is shown in Fig. 3. The I/O monitor performs the following actions for incoming packets:

(1) An incoming packet is assigned a specific packet identifier (packet ID) by an input packet control circuit. This identifier is carried with the packet through the packet processor as meta-information. The packet ID is also stored in tag storage to allow for matching against subsequent output packets. The identifier is unique to the packet (for the lifetime of the packet in the system). The tag storage may include additional information (e.g., packet arrival time, payload CRC).
(2) A cyclic redundancy check code (CRC) is computed for the incoming packet payload by a dedicated CRC circuit. The resulting CRC is stored in the tag storage under the control of the input packet control circuit.

After a packet has been processed, it is sent back to the I/O monitor for checking and, ultimately, for transmission. The identifier of the packet which was the source for the processed packet is still associated with the packet as meta-information. When the outgoing packet arrives at the I/O monitor, the following actions are performed:

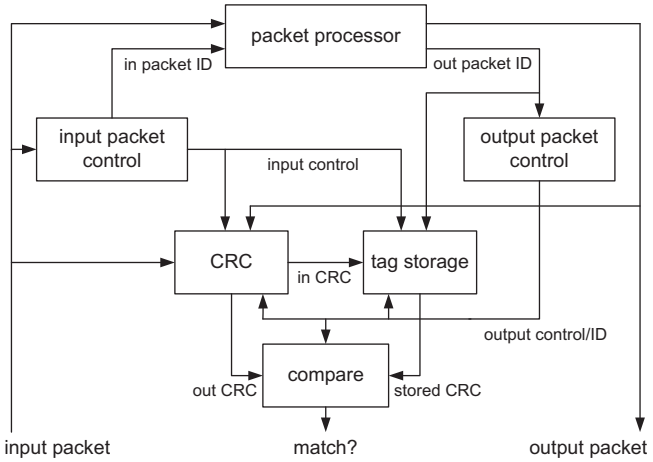


**Fig. 3.** Architecture of a sample I/O monitor.

(1) The outgoing packet's identifier is matched against stored identifiers in tag storage. A missing identifier in tag storage indicates that the outgoing packet cannot be attributed to an incoming packet (either because it was sent without a matching incoming packet or it is a retransmission and the initial tag has already been "used"). In either case, the transmission can be interpreted as unauthorized (e.g., caused by a denial-of-service attack) and be blocked.
(2) If the identifier is located in tag storage, its companion CRC is checked against the CRC of the payload of the processed packet. This validation protects against malicious modification of packet payloads. If there is CRC match, then the packet is transmitted.

The tag information of a packet is removed when a packet is transmitted and the buffer slot can be reused by future packets. If a packet is dropped during packet processing, its identifier and corresponding CRC can be removed from tag storage following a timed period measured by a counter.

The I/O monitor shown in Fig. 3 can be extended to address additional packet processing issues. Input and output packet counts can be easily correlated through the use of the packet identifiers. Using timestamps, processing delays can be determined. In Section 4, the overheads associated with a preliminary version of the I/O monitor are quantified.

*3.4. 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 [42,43]. For effective detection, it is necessary

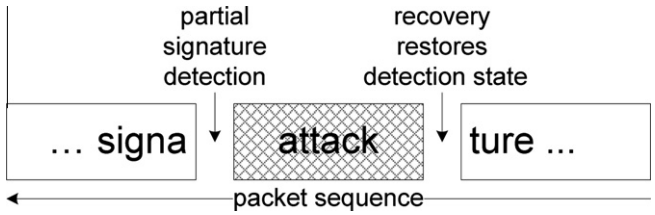


**Fig. 4.** Example of state recovery in intrusion detection.

to consider signatures that are split across multiple packets (see Fig. 4). 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 [44,45] 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.

## 4. Results

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

### 4.1. Monitoring stream information

As shown in Fig. 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 [30]. 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. 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 [30]. 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.

### 4.2. 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 [46] to generate workloads that are similar to what can be expected on an SPPP. We employ the SimpleScalar simulator [47] to extract relevant monitoring information and the `objdump` utility for binary analysis to generate monitoring graphs.

Table 1 shows the size requirements of different information streams graphs. As a comparison, the size of the application binary is also shown. Monitoring graphs require only in the order of 10% of the size of the application binary and thus do not incur significant overhead. The monitoring graph for the hash 4 pattern requires least overhead with an average of 7.1% of the size of the binary.

Table 2 shows the detection performance of the monitor. The hash 4 pattern can detect all but 6% of the attacks (due to hash collisions, which can be reduced with a larger hash size). There are no false positive detections. 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 hash 4 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 less than 10% additional instruction memory and the logic necessary for implementing the comparison monitor.

### 4.3. Multi-core monitor

When monitoring multiple processor cores in a single system, it is possible to amortize the overhead for storing the monitoring

**Table 1**
Size of monitoring graph for different MiBench benchmarks and information streams.

| Application | Binary size (kB) | Pattern | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Address | | Opcode | | Load/store | | Control flow | | Hash 4 | |
| | | Size (kB) | % of bin. | Size (kB) | % of bin. | Size (kB) | % of bin. | Size (kB) | % of bin. | Size (kB) | % of bin. |
| adpcm | 953 | 98 | 10.3 | 80 | 8.4 | 65 | 6.9 | 81 | 8.5 | 59 | 6.2 |
| basicmath | 1023 | 106 | 10.3 | 88 | 8.6 | 71 | 7.0 | 87 | 8.5 | 64 | 6.2 |
| bitcount | 1200 | 141 | 11.8 | 114 | 9.5 | 90 | 7.5 | 116 | 9.7 | 83 | 6.9 |
| blowfish | 969 | 100 | 10.3 | 82 | 8.5 | 67 | 6.9 | 83 | 8.5 | 60 | 6.2 |
| crc | 958 | 98 | 10.3 | 80 | 8.4 | 65 | 6.8 | 81 | 8.4 | 59 | 6.1 |
| dijkstra | 1107 | 129 | 11.6 | 104 | 9.4 | 83 | 7.5 | 105 | 9.5 | 76 | 6.9 |
| fft | 1001 | 104 | 10.3 | 84 | 8.4 | 69 | 6.8 | 85 | 8.5 | 62 | 6.2 |
| gsm | 1104 | 126 | 11.4 | 104 | 9.4 | 85 | 7.7 | 104 | 9.4 | 76 | 6.9 |
| ispell | 1186 | 130 | 11.0 | 105 | 8.9 | 85 | 7.1 | 108 | 9.1 | 77 | 6.5 |
| jpeg | 1204 | 156 | 13.0 | 128 | 10.7 | 104 | 8.6 | 129 | 10.7 | 93 | 7.8 |
| lame | 3454 | 946 | 27.4 | 743 | 21.5 | 631 | 18.3 | 779 | 22.6 | 569 | 16.5 |
| mad | 1430 | 151 | 10.5 | 126 | 8.8 | 100 | 7.0 | 123 | 8.6 | 90 | 6.3 |
| patricia | 1123 | 129 | 11.5 | 105 | 9.4 | 83 | 7.4 | 106 | 9.4 | 76 | 6.8 |
| quicksort | 1106 | 127 | 11.5 | 103 | 9.4 | 82 | 7.4 | 104 | 9.4 | 75 | 6.8 |
| rijndael | 998 | 98 | 9.9 | 83 | 8.3 | 68 | 6.8 | 81 | 8.1 | 60 | 6.0 |
| rsynth | 1370 | 148 | 10.8 | 122 | 8.9 | 97 | 7.1 | 121 | 8.9 | 88 | 6.4 |
| sha | 955 | 98 | 10.3 | 80 | 8.4 | 65 | 6.8 | 81 | 8.4 | 59 | 6.1 |
| sphinx | 2185 | 230 | 10.5 | 192 | 8.8 | 157 | 7.2 | 188 | 8.6 | 139 | 6.4 |
| stringsearch | 960 | 100 | 10.4 | 81 | 8.5 | 66 | 6.9 | 83 | 8.6 | 60 | 6.2 |
| susan | 1084 | 120 | 11.0 | 101 | 9.3 | 81 | 7.5 | 98 | 9.0 | 72 | 6.7 |
| tiff2bw | 1668 | 174 | 10.4 | 116 | 7.0 | 97 | 5.8 | 121 | 7.2 | 87 | 5.2 |
| tiff2rgba | 1739 | 185 | 10.6 | 126 | 7.2 | 106 | 6.1 | 129 | 7.4 | 94 | 5.4 |
| tiffdither | 1457 | 159 | 10.9 | 127 | 8.7 | 104 | 7.1 | 131 | 9.0 | 94 | 6.5 |
| tiffmedian | 1458 | 161 | 11.1 | 129 | 8.8 | 105 | 7.2 | 132 | 9.1 | 95 | 6.5 |
| typeset | 1899 | 369 | 19.5 | 336 | 17.7 | 315 | 16.6 | 302 | 15.9 | 256 | 13.5 |
| Average | | 11.9% | | 9.6% | | 7.9% | | 9.6% | | 7.1% | |

**Table 2**
Detection rate of monitor for bit flip attacks. The results are based on 100 simulations using the `gsm` application.

| Monitoring pattern | Detection rate of bit flips (%) | Avg. no. of instr. to detection |
|---|---|---|
| Address | 13 | 49.1 |
| Opcode | 40 | 1.2 |
| Load/store | 24 | 15.8 |
| Control flow | 26 | 23.6 |
| Hash 4 | 94 | 1 |

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 [48]. Similar to how shared instruction stores are used, shared monitoring graph storage can be employed. To illustrate the effectiveness of such a sharing architecture, Fig. 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.

### 4.4. I/O monitor

A preliminary version of the I/O monitor shown in Fig. 3 has been developed and prototyped using an Altera Stratix III FPGA. A series of implementation choices were made regarding the architecture described in Section 3.3. The tag storage is implemented as an $n$-slot FIFO. (For simplicity, we assume in-order processing for this prototype – a requirement that can be loosened in a real system.) Each incoming data packet is tagged with a $\log n$-bit packet identifier generated with a standard counter. The identifier and a 32-bit CRC for the payload portion of the packet is stored in the tag storage FIFO. As packets arrive at the output packet control portion of the I/O monitor, the identifier is compared against the identifier at the front of the FIFO. If its value is greater, the intermediate packets must have been dropped and the next FIFO location is examined. If the incoming identifier is less than the first identifier in the FIFO, an unauthorized retransmission is determined. Finally, the stored CRC and the newly calculated CRC for the outgoing packet payload are compared to determine if the payload has been maliciously modified. If a match is determined, the packet is permitted to be transmitted.

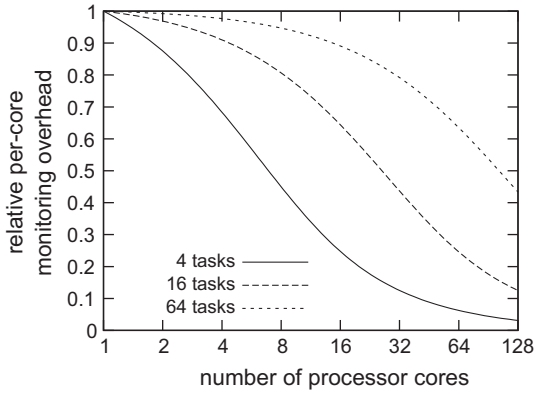


**Fig. 5.** Multi-core monitoring overhead.

The results in Table 3 show the resource requirements for the prototype implementation (four-input lookup tables (LUT), flip-flops (FF), and block memory bits) and its performance (maximum

**Table 3**
Performance and size of I/O monitor.

| Tag storage size | LUTs | FFs | Memory bits | Clock rate (MHz) | Throughput (Gbps) |
|---|---|---|---|---|---|
| 64 packets | 356 | 118 | 2688 | 316.86 | 10.14 |
| 128 packets | 361 | 121 | 5372 | 308.45 | 9.87 |
| 256 packets | 364 | 124 | 10,752 | 306.84 | 9.82 |
| 512 packets | 367 | 127 | 21,504 | 317.26 | 10.15 |
| 1024 packets | 371 | 130 | 43,008 | 305.62 | 9.77 |
| 2048 packets | 375 | 133 | 86,016 | 287.03 | 9.18 |
| 4096 packets | 378 | 136 | 172,032 | 265.39 | 8.49 |
| 8192 packets | 368 | 139 | 344,064 | 258.60 | 8.28 |
| 16384 packets | 372 | 142 | 688,128 | 196.73 | 6.30 |

clock rate, throughput). The resource requirements for the implementation are quite small and are dominated by the need for memory for tag storage. The throughput performance of around 10 Gigabits per second is sufficient to support state-of-the-art core routers and thus show that the presented architecture is a feasible approach to securing the data path in routers.

## 5. 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 37 (3) (2007) 59–64.
[2] T. Wolf, Challenges and applications for network-processor-based programmable routers, in: Proceedings of the IEEE Sarnoff Symposium, Princeton, NJ, 2006.
[3] A. Cui, Y. Song, P.V. Prabhu, S.J. Stolfo, Brave new world: pervasive insecurity of embedded network devices, in: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID), vol. 5758, Lecture Notes in Computer Science, Saint-Malo, France, 2009, pp. 378–380.
[4] D.D. Clark, The design philosophy of the DARPA Internet protocols, in: Proceedings of the ACM SIGCOMM 88, Stanford, CA, 1988, pp. 106–114.
[5] D.L. Tennenhouse, D.J. Wetherall, Towards an active network architecture, ACM SIGCOMM Computer Communication Review 26 (2) (1996) 5–18.
[6] N.T. Bhatti, 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, 1995, pp. 138–150.
[7] T. Wolf, Service-centric end-to-end abstractions in next-generation networks, in: Proceedings of the Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN), Arlington, VA, 2006, pp. 79–86.
[8] S. Ganapathy, T. Wolf, Design of a network service architecture, in: Proceedings of the Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, 2007, pp. 754–759.
[9] N.C. Hutchinson, L.L. Peterson, The x-kernel: an architecture for implementing network protocols, IEEE Transactions on Software Engineering 17 (1) (1991) 64–76.
[10] L. Ruf, K. Farkas, H. Hug, B. Plattner, Network services on service extensible routers, in: Proceedings of the Seventh Annual International Working Conference on Active Networking (IWAN 2005), Sophia Antipolis, France, 2005.
[11] T. Anderson, L. Peterson, S. Shenker, J. Turner, Overcoming the Internet impasse through virtualization, Computer 38 (4) (2005) 34–41.
[12] Intel Corporation, Intel Second Generation Network Processor, 2005. Available from: <http://www.intel.com/design/network/products/npfamily/>.
[13] EZchip Technologies Ltd., Yokneam, Israel, NP-3 – 30-Gigabit Network Processor with Integrated Traffic Management, May 2007. Available from: <http://www.ezchip.com/>.
[14] LSI Corporation, APP3300 Family of Advanced Communication Processors, August 2007. Available from: <http://www.lsi.com/> .
[15] J.S. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman, 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, 2007, pp. 85–96.
[16] A. Bavier, N. Feamster, M. Huang, L. Peterson, 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, 2006, pp. 3–14.
[17] D. Geer, Malicious bots threaten network security, Computer 38 (1) (2005) 18–20.
[18] A. Hussain, J. Heidemann, 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, 2003, pp. 99–110.
[19] A. Kuzmanovic, 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, 2003, pp. 75–86.
[20] J. Xia, L. Gao, T. Fei, Flooding attacks by exploiting persistent forwarding loops, in: IMC '05: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, Berkeley, CA, 2005.
[21] S. Parameswaran, T. Wolf, Embedded systems security – an overview, Design Automation for Embedded Systems 12 (3) (2008) 173–183.
[22] S. Ravi, A. Raghunathan, S. Chakradhar, Tamper resistance mechanisms for secure, embedded systems, in: Proceedings of the 17th International Conference on VLSI Design (VLSI Design 2004), Mumbai, India, 2004, pp. 605–611.
[23] A. Wood, J.A. Stankovic, Denial of service in sensor networks, IEEE Computer 35 (10) (2002) 54–62.
[24] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '99), Lecture Notes in Computer Science, Springer-Verlag, London, United Kingdom, 1999, pp. 388–397.
[25] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, R. Vaslin, Reconfigurable hardware for high-security/high-performance embedded systems: the SAFES perspective, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16 (2) (2008) 144–155.
[26] M. Hicks, P. Kakkar, J.T. Moore, C.A. Gunter, S. Nettles, PLAN: a packet language for active networks, in: Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages, ACM, 1998, pp. 86–93.
[27] D. Arora, S. Ravi, A. Raghunathan, N.K. Jha, Secure embedded processing through hardware-assisted run-time monitoring, in: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05), Munich, Germany, 2005, pp. 178–183.
[28] R.G. Ragel, S. Parameswaran, IMPRES: integrated monitoring for processor reliability and security, in: Proceedings of the 43rd Annual Conference on Design Automation (DAC), San Francisco, CA, USA, 2006, pp. 502–505.
[29] J. Zambreno, A. Choudhary, R. Simha, B. Narahari, N. Memon, SAFE-OPS: an approach to embedded software security, Transactions on Embedded Computing Systems 4 (1) (2005) 189–210.
[30] S. Mao, T. Wolf, Hardware support for secure processing in embedded systems, in: Proceedings of the 44th Design Automation Conference (DAC), San Diego, CA, 2007, pp. 483–488.
[31] M. Abadi, M. Budiu, Ú. Erlingsson, J. Ligatti, Control-flow integrity principles, implementations, and applications, in: ACM Conference on Computer and Communication Security (CCS), Alexandria, VA, 2005, pp. 340–353.
[32] N. Nakka, Z. Kalbarczyk, R.K. Iyer, J. Xu, An architectural framework for providing reliability and security support, in: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN), Florence, Italy, 2004, pp. 585–594.
[33] R.G. Ragel, S. Parameswaran, S.M. Kia, Micro embedded monitoring for security in application specific instruction-set processors, in: Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), San Francisco, CA, 2005, pp. 304–314.
[34] Z. Shao, Q. Zhuge, Y. He, E.H.-M. Sha, Defending embedded systems against buffer overflow via hardware/software, in: Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC), Las Vegas, NV, 2003, pp. 352–363.
[35] G.F. Cretu, J.J. Parekh, K. Wang, S.J. Stolfo, Intrusion and anomaly detection model exchange for mobile ad-hoc networks, in: Proceedings of the 3rd IEEE on Consumer Communications and Networking Conference (CCNC 2006), Las Vegas, NV, 2006, pp. 635–639.
[36] Y. Zhou, V. Lakamraju, I. Koren, C.M. Krishna, Software-based failure detection and recovery in programmable network interfaces, IEEE Transactions on Parallel and Distributed Systems 18 (11) (2007) 1539–1550.
[37] N.-F. Huang, Y.-T. Chen, Y.-C. Chen, C.-N. Kao, J. Chiou, A network processor-based fault-tolerance architecture for critical network equipments, in: Proceedings of the Information Networking, Networking Technologies for Broadband and Mobile Networks, International Conference (ICOIN), Lecture Notes in Computer Science, 3090, Springer-Verlag, Busan, Korea, 2004, pp. 763–772.
[38] Y. Luo, J. Fan, Fault tolerant practices on network processors for dependable network processing, in: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS), Miami, FL, 2008.
[39] T. Wolf, R. Tessier, Design of a secure router system for next-generation networks, in: Proceedings of the Third International Conference on Network and System Security (NSS), Gold Coast, Australia, 2009.
[40] CERT Coordination Center, Carnegie Mellon University, Pittsburgh, PA, CERT Advisory CA-2001-19 "Code Red Worm Exploiting Buffer Overflow In IIS Indexing Service DLL, July 2001.

[41] D. Moore, C. Shannon, 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, 2002, pp. 273–284.
[42] H.-J. Jung, Z. Baker, V. Prasanna, Performance of FPGA implementation of bit-split architecture for intrusion detection systems, in: Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, 2006.
[43] A. Das, D. Nguyen, J. Zambreno, G. Memik, A. Choudhary, An FPGA-based network intrusion detection architecture, IEEE Transactions on Information Forensics and Security 3 (1) (2008) 118–132.
[44] B.T. Gold, J.C. Smolens, B. Falsafi, J.C. Hoe, The granularity of soft-error containment in shared memory multiprocessors, in: Proceedings of the Workshop on System Effects of Logic Soft Errors, Urbana-Champaign, 2006.
[45] R. Teodorescu, J. Nakano, J. Torrellas, SWICH: a prototype for efficient cache-level checkpointing and rollback, IEEE Micro 26 (5) (2006) 28–40.
[46] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, MiBench: a free, commercially representative embedded benchmark suite, in: Proceedings of the IEEE 4th Annual Workshop on Workload Characterization, Austin, TX, 2001.
[47] D. Burger, T.M. Austin, The Simple Scalar tool set, version 2.0, Tech. Rep. 1342, Department of Computer Science, University of Wisconsin in Madison, June 1997.
[48] Q. Wu, T. Wolf, On runtime management in multi-core packet processing systems, in: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS), San Jose, CA, 2008, pp. 69–78.