A Dynamically Reconfigured Multi-FPGA Network Platform for High-Speed Malware Collection

Sascha Mühlbach
Secure Things Group
Center for Advanced Security Research Darmstadt (CASED)
sascha.muehlbach@cased.de

Andreas Koch
Embedded Systems and Applications Group
Dept. of Computer Science
Technische Universität Darmstadt
koch@esa.cs.tu-darmstadt.de

***Abstract*—Malicious software has become a major threat to computer users on the Internet today. Security researchers need to gather and analyze large sample sets to develop effective countermeasures. The setting of honeypots, which emulate vulnerable applications, is one method to collect attack code. We have proposed a dedicated hardware architecture for honeypots which allows both high-speed operation at 10 Gb/s and beyond, as well as offers a high resilience against attacks on the honeypot infrastructure itself. In this work, we refine the base NetStage architecture for better management and scalability: Using dynamic partial reconfiguration, we can now update the functionality of the honeypot during operation. To allow the operation of a larger number of vulnerability emulation handlers, the initial single-device architecture is extended to scalable multi-chip systems. We describe the technical aspects of these modifications and show results evaluating an implementation on a current quad-FPGA reconfigurable computing platform.**

I. Introduction

The significant increase of malicious software (malware) in recent years (see [1]) requires security researchers to analyze an ever increasing amount of samples for developing effective prevention mechanisms. One method for collecting a large number of samples is the use of low-interaction honeypots (e.g., [2]). Such dedicated computer systems emulate vulnerabilities in applications and are connected directly to the Internet, spanning large IP address spaces to attract many different attackers. A number of software applications are available helping in building up honeypot systems. But in addition to having performance limitations in high-speed environments (10+ Gb/s), such software systems also suffer from being compromisable themselves (they can be subverted to attack even more hosts).

In this context, we have proposed MalCoBox, a low-interaction malware-collection honeypot realized entirely in reconfigurable hardware without any software components in [3]. The core of the MalCoBox system is NetStage, a high-speed implementation of the basic Internet communication protocols, attached to several independent vulnerability emulation handlers (VEH), each emulating a specific security flaw of an application (see Fig. 1). We have demonstrated the feasibility of that approach by implementing a prototype on a FPGA platform, fully employing the power of dedicated hardware resources to support 10+ Gb/s network traffic.

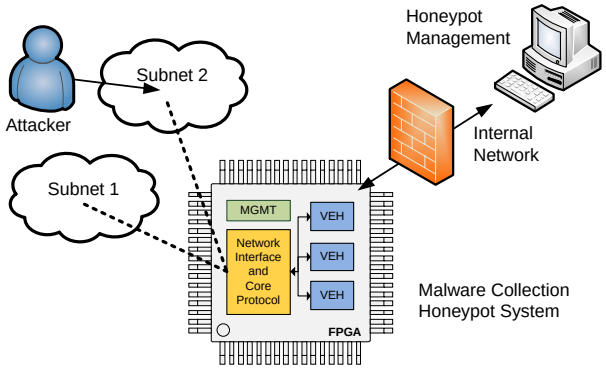

Figure 1. Hardware-Based Malware Collection

Beyond the performance aspects, in context of the network security domain, a purely hardware-based approach such as ours has the additional advantage, that no general-purpose software programmable processor is present that could be subverted if the honeypot itself is being attacked.

An important issue for potential MalCoBox users is how the platform can be updated during operation with new or improved Vulnerability Emulation Handlers (VEH) to react to the changing threat landscape. For an FPGA-based system, the hardware functionality can be altered during operation by using partial reconfiguration (PR). This approach has already been used for network routers in [4]. We now employ the technique in a larger scope to flexibly swap-in new VEHs while the rest of the system stays in operation. The initial discussion presented in [5] is expanded in this work.

Another aspect of great practical interest is the number of different vulnerabilities that can be emulated in *parallel*. The original MalCoBox relied on a single-device implementation of NetStage, and was limited to ca. 20 VEHs active in the system. This is a gap to software honeypots, where

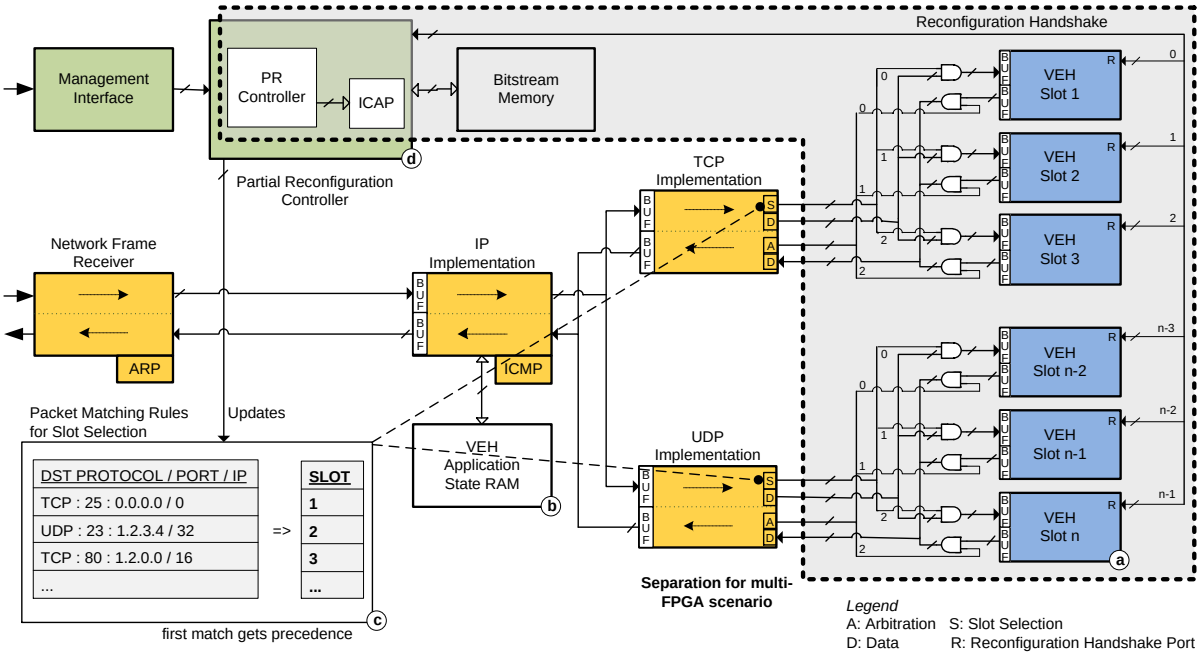

Figure 2. Core Architecture of the Partially Reconfigurable Malware Collection System

even the low-interaction variants often support 50 ... 100 different vulnerabilities implemented as scripts in languages such as Perl and Python. While it could be argued that the capacities of individual FPGA chips does increase from each generation to the next (which they do), the larger high-end devices are significantly more expensive per logic cell than the mid-range versions. Thus, it is worthwhile to examine how the MalCoBox capacity can be extended using a multi-device NetStage implementation. This approach has been introduced in [6] and is described in greater detail here.

The paper is organized as follows: Section II briefly describes the core architecture components. The next Section III covers details of the ring implementation and elaborates the differences between single-chip and multi-chip solution. Section IV continues with a description on the required modifications of the partial reconfiguration strategy. The implementation of the complete system on the BEEcube BEE3 quad-FPGA reconfigurable computing platform [7] is described in Section V, followed by experimental results given in Section VI. We close with a conclusion and an outlook towards further research in the last Section.

A. Related Work

To our knowledge, this is the first implementation of such a honeypot system using pure dedicated hardware blocks. In 2007 Pejovik et. al. [8] presented an initial concept for a hardware honeypot with RAM-based state machines for the emulations. Unfortunately, they did not give any detailed results on the achievable performance and the possible parallelism. It is likely, however, that the RAM-based state machines could become a bottleneck in high-speed environments due to limited bandwidth. Thus, our architecture contains only dedicated hardware blocks.

In terms of FPGA-based networking, a popular generic platform for research is the Stanford NetFPGA [9], containing an FPGA, four 1G interfaces, and various memories. The platform is the vehicle for a wide spectrum of research, e.g., on accelerated switches, routers and network monitoring devices. Internally, NetFPGA provides a flexible data-path structure into which custom processing modules can be easily inserted. With the widespread use of NetFPGA, a new version supporting 10G networks is currently being released.

Another related research project is DynaCORE [10]. It consists of a Network-on-Chip (NoC) oriented architecture for a generic reconfigurable network co-processor, combining general network processing in software with accelerated functions (e.g., encryption) in hardware units. By using current techniques such as partial reconfiguration, the platform can be adapted to different communication situations.

In contrast to these often packet-oriented approaches, our own research has always been aiming at higher-level Internet (e.g., TCP, UDP) and application protocols. We thus have created NetStage [3] as a novel base architecture for our honeypot appliance. NetStage is built around an all-hardware Internet protocol stack implementation (including TCP operation). We have chosen to follow some of the approaches proven useful with NetFPGA, e.g., the capability to insert “plug-in” hardware modules at various points in the processing flow. In contrast to DynaCORE, however,

we generate a light-weight application specific interconnect between these modules, instead of using a general-purpose, but larger NoC scheme.

II. Key Architecture Components

Figure 2 shows the base NetStage Architecture (discussed in greater detail in [3]), including extensions to support dynamic partial reconfiguration (DPR). The architecture provides module slots (Fig. 2-a) into which the partial VEH bitstreams can be loaded. These VEH slots are loosely interconnected with the core system by buffers, allowing all VEHs to have the same external interface (important for DPR). Thus, any VEH may be configured into any of the slots of the same size, with the buffers limiting the impact of brief VEH-level stalls on the system-level throughput.

VEHs share the underlying implementations of the core protocols (IP, TCP, UDP) in NetStage. These have been very carefully optimized to achieve a throughput of at least 10 Gb/s by using pipeline- and task-level parallelism to keep up with the line-rate of the 10 Gb/s external network interface.

In some cases, VEHs have to track session state to generate an appropriate response. NetStage provides a central facility for storing per-connection state (Fig. 2-b): When a packet is passing the IP implementation, the globally maintained state information is attached to the packet in a custom control header which accompanies every packet through the system. The VEH can read this information, act on it, and update the value if necessary. The modified header is written back to the state memory when a response packet (or an empty state write packet) passes the IP implementation on the transmit path. Such a centralized storage is more efficient than attempting to store state in each VEH (which would fragment the capacity of the on-chip memories).

The global VEH application state memory can also be used to save/restore VEH state during reconfiguration to allow the seamless swapping-in of newer (but state-compatible) versions of a VEH.

A. Vulnerability Emulation Handler

When a packet has passed through the NetStage core, it will be forwarded to the responsible slot where the VEH performs the actual malware detection and extraction. Packets are routed to the appropriate slots by means of a routing table (Fig. 2-c) that holds matching rules for the different vulnerability emulations currently active in the system. The table is writable to allow dynamic modification of the actual VEHs used. A basic set of matching rules includes the destination port, destination IP and netmask. The latter allows us to set-up separate IP address ranges which use VEHs for different vulnerabilities on the same port (e.g., many handlers will listen on the HTTP port 80).

With the processing speed achievable using reconfigurable hardware, these basic rules could also be extended to directly match payload contents. However, this would require dynamic reconfiguration of the actual matching units, instead of just writing new values into registers (as in the basic matcher). Since all our current VEHs are selected just based on protocol and port (independently of the payload), we can continue to use the simpler basic approach.

B. Management Section

The partial dynamic reconfiguration of VEHs is managed by the Partial Reconfiguration Controller (PRC, Fig. 2-d)), which is connected to the FPGAs internal configuration access port (ICAP). On the application side, the PRC is connected to the MalCoBox management interface (either by a PCI Express endpoint or a dedicated network link, depending on the selected deployment mode of the system). The PRC is also connected to the individual VEH slots by a number of handshake signals to inform the VEHs about their impending reconfiguration (for a clean shutdown etc.) and to check whether the VEH is idle. An attached bitstream memory can hold several partial bitstreams to allow the system to be reconfigured independently of the management station in future implementations.

C. Reconfigurable VEHs

To support independent partial reconfiguration of any of the VEH slots, a wrapper encapsulates the actual VEH implementation module (see Fig. 3). This wrapper includes glue logic controlled by the partial reconfiguration controller to disconnect/reconnect all inputs and outputs of the VEH module. This clean separation is essential to avoid introducing errors in the rest of the system when reconfiguring.

The wrapper also contains the send and receive buffers for each module as well as the corresponding buffer management logic. As all the handlers share the same buffer structure, it is more efficient to keep it static than configure it with each VEH. The inputs and outputs of the wrapper are directly connected to the MalCoBox core (see Fig. 2).

III. Multi-Device Architecture

To extend our system to multiple FPGAs, we will draw the boundaries between the static NetStage core (basic Ethernet and Internet protocol functions) and the dynamically exchangeable VEH slots (see the dotted line in Figure 2). One FPGA acting as Master node holds the network core and the network interfaces, the remaining other FPGAs, called VEH nodes contain the individual emulation blocks. The BEE3 platform supports a number of inter-device interconnection schemes. For future scalability independently of the BEE3 architecture, we decided to implement a ring structure. Such rings have already proven useful in multi-chip systems internally using NoCs [11].

For extending the NetStage-based MalCoBox to multiple devices, a unidirectional ring suffices. (Figure 4). The unidirectional ring needs fewer I/O pins on the FPGAs and avoids

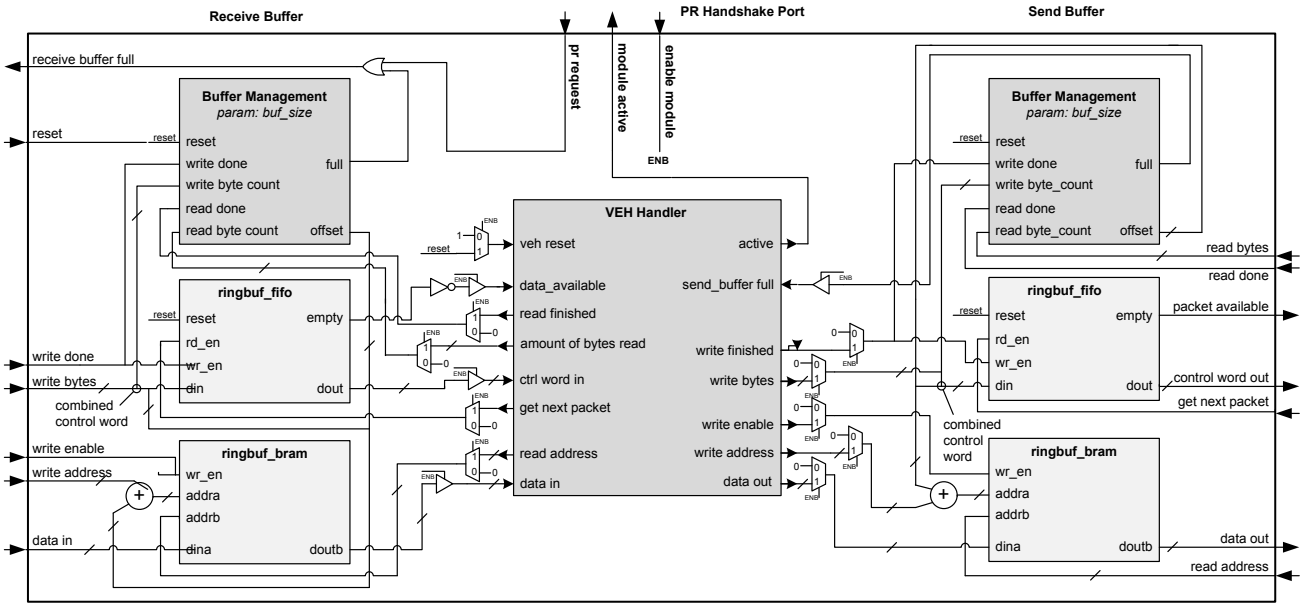

Figure 3. Wrapper encapsulating a Vulnerability Emulation Handler Slot

the increased latency of the bus-turnaround cycles that would be required when running a bidirectional bus over the same pins. Also note that, in contrast to the implementations in discussed in prior work, the one described here is using the external SDRAM instead of the internal BlockRAM to temporarily store bitstreams, conserving FPGA resources to allow more VEHs.

A. Ring Communication

For the communication on the ring we use 66 of the 72 available inter-FPGA data lines on the BEE3 which are run in DDR mode, resulting in 132 bits of data per clock cycle [6]. Four bits are reserved for status bits, the remaining 128 form the data transmission word. As data words are sent continuously on the ring for synchronization purposes (even if not actual data needs to be transmitted), a valid flag is used to indicate words holding actual message data. For the separation of individual messages, we use two flags to signal the first and the last word of a message. As the actual byte size of the message is already stored within the Internal Control Header (ICH, see Figure 8) used by NetStage (prefixed to the message body), no special-case processing is required for unused bytes in the last data word of a message. A final flag is used to denote special ring control messages used, e.g., to control the DPR process (see next Section). In a future extension, this could also be used, e.g., to enumerate the ring nodes automatically during initialization. Currently, the destination addresses of the available FPGAs inside the ring are set during compile time.

The ICH-prefixed message is prefixed yet again with a 128b Ring Control Header (RCH) when it is transmitted between devices. The RCH carries the type information of the message and the destination FPGA. The remaining bits are reserved to implement further control functions in the future.

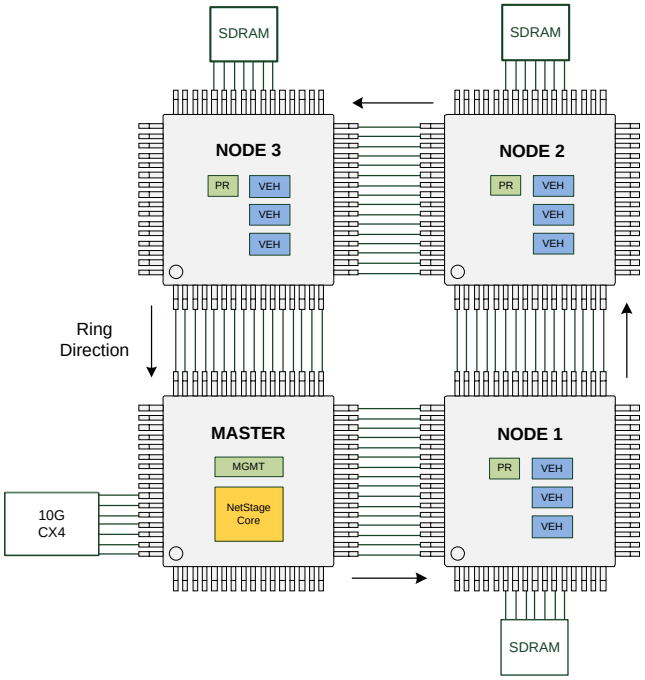

Figure 4. Multi-FPGA network processor in ring topology

Since we want to maintain a high bandwidth and low latency even when distributing the architecture across mul-

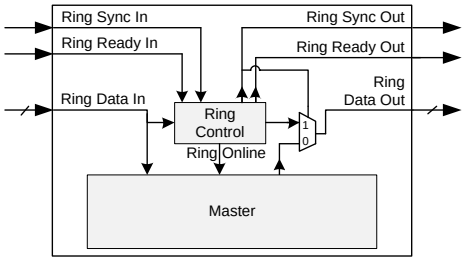

Figure 5. Schematic overview of the master

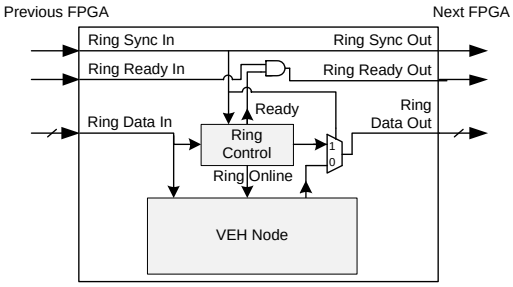

Figure 6. Schematic overview of the nodes

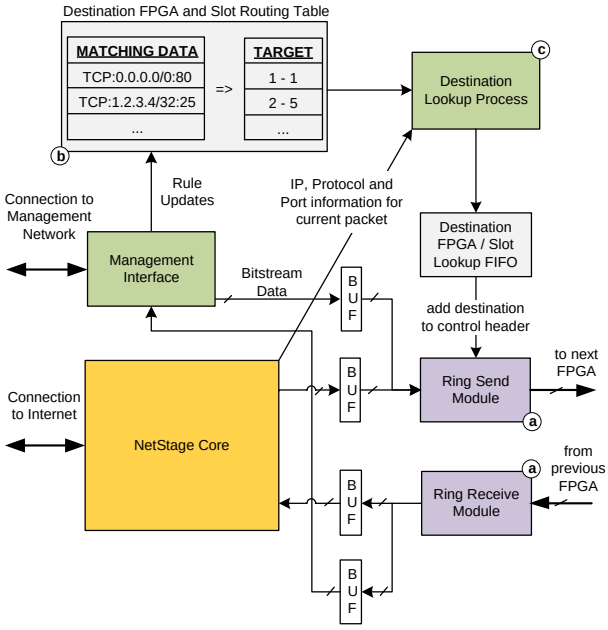

Figure 7. Master node architecture

tiple devices, we want to operate the inter-device links at maximum speed. To this end, we use the BEE3-provided global clock to all FPGAs as the ring clock. However, due to different trace lengths and other board-level signal integrity effects, reliable operation at our target frequency of 250 MHz requires a training of the individual FPGAs to the link characteristics. This is done using the technique proposed in [12]: A known training sequence is transmitted between adjacent FPGAs, and the receiver adjusts its delay until it receives a stable pattern from the transmitter.

Two additional signals are required to realize training procedure (see Figures 5 and 6): The Master starts the process by asserting a Sync signal, which is routed around the entire ring. It is used to both initiate training between neighboring FPGAs, as well as to test whether the nodes did configure correctly on start-up (an error is indicated if the Sync sent out by the Master does not match the incoming Sync passed around the ring). One the receiving FPGA of a synchronization pair has locked-on to the training pattern, it asserts its internal Ready signal, which is ANDed with the Ready incoming from its transmitting partner before being output to its receiving partner (the next device on the ring). Once the Master has received an asserted Ready signal passed around the entire ring, it stops training and releases the ring into normal operation.

The ring thus achieves a transfer rate of 32 Gb/s between nodes, more than sufficient for our current 10 Gb/s network environment. For simplicity, and since we did not experience any data integrity issues in our practical experiments once training completed, we do not perform error detection/correction on the ring communications. However, for long-term production use, CRC/ECC facilities could be added here. As there are still data lines available on the BEE3, this could be easily implemented without affecting the base architecture.

B. Master Node

Beyond the the network core and the management section that was already present in the single-chip NetStage implementation, the Master node (see Figure 7) now contains additional logic (Fig. 7-a) to handle the ring communication. In particular, this includes send and receive interface modules, as well as the FPGA addressing logic. Note that we do not place any VEHs in the Master node, the currently unused space is intended to be used for future extensions of the NetStage core (e.g., to IPv6). Thus, the Master itself will not be dynamically reconfigured and does not require an internal ICAP controller. However, the Master is responsible for initiating the reconfiguration of the VEH nodes. Thus, the management section in the Master and the configuration controllers in the VEH nodes interact, which is achieved by specialized ring control messages.

The payload data traffic around the ring is organized on two levels: The 32B ICH (see Figure 8) replaces the original protocol headers for a packet with a more compact representation. It also carries the carries the packet-to-handler routing information of a message on the ring in the form of a destination VEH node ID and the VEH slot on that node. Since the destination node ID is already specified in the RCH, this might be seen as redundant. However, the RCH is present only while a packet is transmitted between

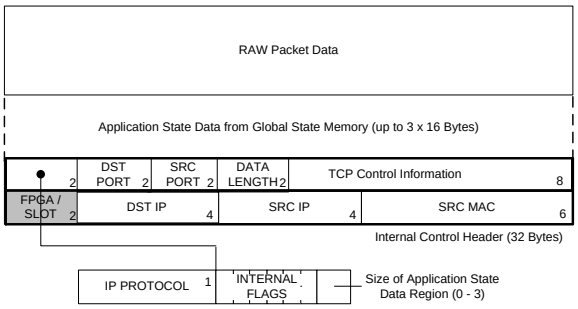

Figure 8. Structure of the internal control header

nodes, and stripped from it for node-internal processing. Since we want to give VEHs the ability to transparently forward packets to other VEHs which might also reside on other nodes, they can read/write that destination data in the ICH instead (which, due to alignment reasons, is not efficient for performing ring-level routing).

As we now have multiple destination FPGAs, the Master routing table, which associated packets just with the responsible VEH slot in the single-chip version, needs be extended to hold the destination node ID as well (Fig. 7-b). This is used to build the RCH when the packet is sent out over the ring. To reduce the latency, the process to lookup the destination address (Fig. 7-c) is pipelined between the core and the Ring Send module. This can be easily done as packets are not reordered between the two modules.

The Master will silently discard packets not matching any rule in its routing table to conserve bandwidth on the ring links. Furthermore, core IP protocols such as ARP and ICMP are usually handled with low-latency entirely inside the Master, and do not cause ring traffic, either.

C. VEH Node

The individual VEHs are housed in the VEH nodes (see Figure 9). For communication with the rest of the system, the VEH nodes need the same ring interface modules (Fig. 9-a) as the Master node. Furthermore, a node-local packet distributor and aggregator (Fig. 9-b) emulate the single-chip NetStage core interface so that VEHs can be attached directly connected to the network core of the single-chip implementation. VEHs are thus portable between the single- and multi-chip versions.

In contrast to the Master node, the VEH are actually dynamically reconfigured to exchange VEHs. Thus, they do need a PR controller and access to the ICAP. The details of this are described in greater detail in the next section.

The ring receive module in each VEH node checks the type of an incoming ring message and its destination address field and either forwards the packet to the local distributor module, a reconfiguration message to the local PR controller, or immediately inserts the message into the forwarding queue if is intended for another node. VEH response network

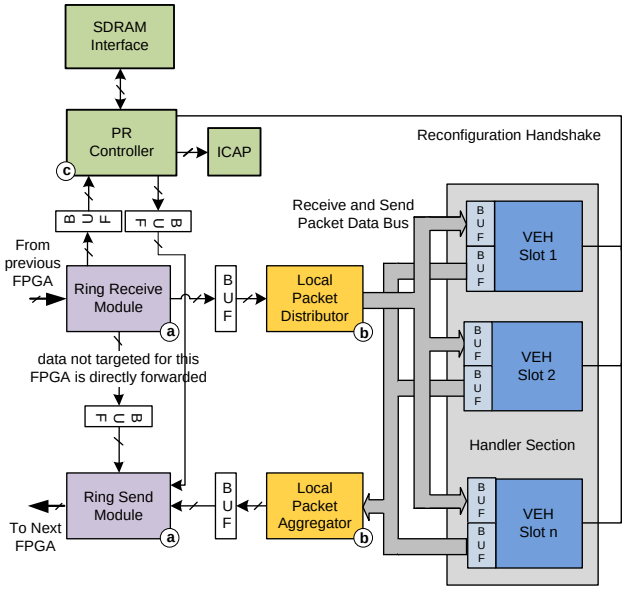

Figure 9. VEH node architecture

packets are picked up by the packet aggregator and inserted into an output queue, passing it on around the ring until it reaches the external network connection at the Master node.

IV. Partial Reconfiguration

Partial bitstream data is transferred from the management station (usually an external PC) to the MalCoBox via the management interface. As the MalCoBox should be able to run as an appliance under remote management, we implemented a stand-alone on-chip reconfiguration interface instead of using the JTAG port together with a software programmer on a host PC.

The underlying protocol used to transmit the bitstream to the MalCoBox consists of the raw bitstream prefixed by a reconfiguration header (see Figure 10). The header contains the bitstream size, the FPGA slot location information, and the rules for the Master node routing table to direct packets to the newly configure VEH.

In contrast to the single-chip implementation [5], in the multi-device scenario, the management interface housed in the Master does not have a direct connection to the Partial Reconfiguration Controller (PRC). Instead, the bitstream is transferred over the ring to the destination VEH node FPGA, but the routing rules table still remains inside the Master node. The management interface therefore extracts the header information from incoming reconfiguration data requests and updates the routing table, while the raw bitstream data is forwarded to the FPGA specified in the reconfiguration header (see also Figure 7). As the partial reconfiguration process is now distributed across multiple devices, the time between the Master-local routing rule update and the activation of new VEH in a remote node

is longer than on the single-chip system. Thus, to avoid misrouting of packets, the new routing rule is explicitly disabled until the VEH is actually ready to accept traffic.

In a VEH node, an incoming bitstream is stored in node-local external DDR-SDRAM memory. One an actual reconfiguration is requested, a fast DMA unit retrieves the bitstream data from memory and transfers it at maximum speed to the ICAP configuration interface. This two-step approach could also be used in a later extension to, e.g., integrity-check the bitstream for communication errors, or to accept only signed bitstreams [13], [14]. Since the ring communication has proven reliable in our tests, and the management console is trusted, the current prototype does not implement these facilities.

A. Partial Reconfiguration Process

The distributed reconfiguration process is performed in the following order:

1) The rule header of incoming bitstream data is extracted and the rule table is updated with the new rule (eventually replacing an existing one), having the active flag set to zero.
2) Incoming bitstream data is forwarded to the corresponding VEH node.
3) After complete reception of bitstream data, the PRC in the VEH node starts the reconfiguration process.
4) After completion of reconfiguration, the PRC sends a DONE status to the Master as a ring control message.
5) The Master management interface receives the message and activates the routing rule so that packets will actually be forwarded.

Network packets and reconfiguration messages (including the bitstream data) share the ring. However, since reconfiguration management is crucial for the reliable operation of the system, these ring control messages receive priority over regular packet transmissions.

Internally, the reconfiguration process in the VEH nodes follows the approach implemented for the single-chip solution [5]: When the node-local PRC receives a reconfiguration request, it initially informs the wrapper of the target slot that the slot is about to be reconfigured. This will stop the receive buffer of the VEH from accepting new packets. The VEH is allowed to process all of the packets held in the buffer at this time, asserting a signal to the PRC on completion. The PRC then deactivates the VEH, and the now inactive VEH is disconnected from the slot wrapper. The actual bitstream data is then read from the DDR-SDRAM and fed into the ICAP. Once reconfiguration is completed, the PRC re-enables the VEH-wrapper connections and allows the new VEH to wake up in its reset state.

V. Implementation

The MalCoBox running on the multi-device NetStage architecture has been implemented on the BEEcube BEE3

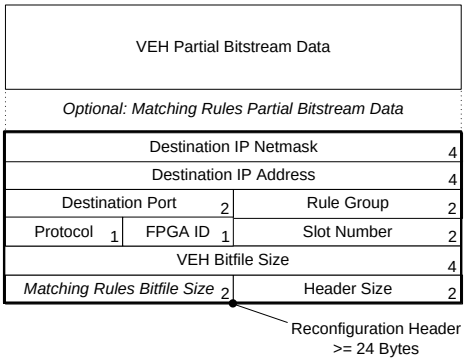

Figure 10. Custom PR header and bitstream data

FPGA-based reconfigurable computing platform, which is equipped with eight 10 Gb/s network interfaces and four Xilinx Virtex 5 FPGAs (2x LX155T, 2x LX95T). The Master node is realized as one of the smaller SX95Ts to have both of the larger LX155T devices available for VEHs.

Network connectivity is provided by the Xilinx XAUI and 10G MAC IPs. The network core in the Master runs at the speed of the 156.25 MHz clock of the 10G network interface. Together with the internal bus width of 128 bit, this leads to a maximum core throughput of 20 Gb/s. This overprovisioning allows us to react to brief stalls in the data flow: Affected handlers in the Master are able to "catch-up" with the normal 10 Gb/s traffic by burst-processing the data accumulated in the buffers at 20 Gb/s. In order to allow more complex VEHs (having longer combinational paths), the VEH nodes currently run at 125 MHz (however, if desired, other clock speeds would be possible). Since they also use 128 bit buses to the VEH slots, the VEHs achieve a peak processing rate of 16 Gb/s, thus still having burst-processing headroom over the 10 Gb/s network line rate.

The ICAP is operated at 32b data width and driven by a separate clock to support variable reconfiguration speeds (and thus support experiments with overclocking the ICAP). Management access is implemented as dedicated network interface with a unique MAC address, directly connected to a standard PC. The management interface receives bitstream data and control operations over the network using a custom protocol. Perl scripts are used to assemble the appropriate network packets. The DDR2-SDRAM interface in the VEH nodes is realized by a Xilinx MIG core and fully uses the DDR2-SDRAM bandwidth.

The size of all inter-module and slot buffers is set to 4 kB (to hold 2 packets with a maximum size of 1500 B), which is sufficient to assure stall-free operation as the modules regularly consume the data at a minimum rate of 10 Gb/s. The only exception are the ring receive buffers, which are set to 16 kB to provide sufficient headroom to receive bursts of packets on the ring. The size of the global application state memory in the Master node is set to 1 Mb of BRAM, which

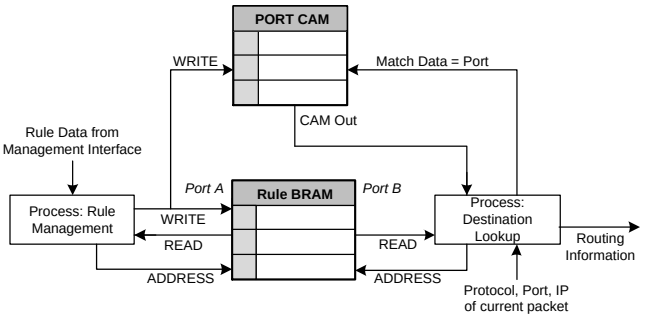

Figure 12. Implementation of the destination lookup process

is sufficient to manage the short running sessions we expect in the honeypot use-case. In further refinements, some of this state could be also placed in the Master node-local DDR2-SDRAM available on the BEE3 platform (incurring longer access latencies, though).

A. Destination Lookup

The rules that control the message routing to VEHs in different nodes are stored in the destination routing table (see Fig. 11) inside the Master node. This table is implemented as BRAM (currently with a size of 1024 rules), to achieve high lookup speeds and flexible scaling. In addition to the routing information, each rule entry contains a rule ID that is used for management purposes, and an active flag used during the reconfiguration process. The table supports multiple rules for the same destination VEH (e.g., to let it respond to different IP addresses).

As the destination routing decision is on the critical path with regard to latency, we use a hierarchical approach for lookups: Rules with the same destination port are represented as a linked list, and a CAM is used to retrieve the BRAM address of the list head for a given port (see Fig. 12). Then, the individual rules for this port are searched in list order by following the rules' next pointers. Beyond quick lookups, this also ensure that rules with the longest IP address prefix will be matched first. The management process ensures that rules are inserted in the correct order.

For efficiency, we have restricted the CAM size to 256 entries, reasoning that 256 different active ports should be sufficient for most cases. Since the CAM has an 8b wide output, the heads of the per-port rule lists always start in the bottom 256 addresses of the routing table BRAM.

B. Example VEHs

To test the system, we have created a number of VEHs emulating different vulnerabilities and applications. In addition to controlling FSMs, the VEHs contain additional logic to perform tasks such as fast parallel pattern matching.

1) SIP: The SIP VEH looks for packets exploiting a vulnerability of the software SIP SDK sipXtapi [15]. The exploit uses a buffer overflow occurring if a SIP INVITE packet contains a CSeq field value exceeding 24 bytes in length. This VEH is based on the UDP protocol.

2) MSSQL: Another UDP-based VEH has a similar structure and is emulating a vulnerable MSSQL 2000 server looking for exploits targeting the resolution service [16]. This exploit was used in the past by, e.g., the Slammer worm.

3) Web Server: As a VEH for a further popular application, we implemented a simple web server emulation, that contains a ROM with predefined HTML pages to be served to clients. The HTTP headers needed for response generation are also stored inside the ROM. A FSM checks the URL of incoming requests and fetches the corresponding output data to be sent from the ROM. This VEH can be flexibly used, e.g., to emulate a login page for a company intranet and monitor attack attempts (e.g., brute force logins), or attacks to the web server itself.

4) Mail Server: As spam is amongst the widespread distribution techniques for malware, we implemented a mail server VEH that accepts incoming mails and pretends to be an open relay server. It contains a FSM that implements the basic SMTP dialog for the reception of mails.

VI. Results

The design was synthesized and mapped using Xilinx ISE 12.4, targeting a SX95T for the Master node and both SX95T and the LX155T devices as VEH nodes. Partial reconfiguration was implemented using the latest partial reconfiguration flow available in PlanAhead 12.4 [17]. Each VEH node was configured to include 24 slots. The VEH module slots were placed manually on the FPGA and sized based on the resource usage trends shown by the sample VEH synthesis results. The resulting layout can be seen in Figure 13. To support VEHs with different resource needs (BRAM vs. LUTs), four kinds of slots, differing in the number and types of contained ressources (see Table V), are provided.

As techniques to dynamically relocate bitstreams on the FPGA matrix are not yet production ready, and even research versions have significant limitations (e.g., only support for outdated device families), we have to create separate bitstreams for the all of the different slots a VEH can be placed. In addition to requiring more storage, this also necessitates to run the place-and-route tools multiple times with different area constraints. Each run produces the partial bitfile for a specific VEH-Slot combination. However, due to using partial reconfiguration, bitfile sets for different VEHs can be created and used independently, e.g., exploiting a multi-core server by executing many tool runs in parallel.

System tests were performed by simulation as well as on an actual BEE3 machine connected to a quad-XEON Linux

TARGET FPGA AND SLOT SELECTION TABLE (BRAM)									
Addr.	Protocol	Target FPGA	Target Slot	Port	Netmask	IP Addr.	Rule ID	Rule Act.	Next Rule
(10b)	(8b)	(8b)	(8b)	(16b)	(32b)	(32b)	(16b)	(1b)	(10b)
0	0x06	1	0	80	0x00000000	0x00000000	13	0	0
1	0x06	1	1	25	0xFFFFFFFF	0x53251021	25	1	256
…	…	…	…	…	…	…	…	…	…
256	0x11	2	3	25	0xFFFFFF00	0x10102500	47	1	257
257	0x06	1	1	25	0xFFFFFF00	0x32122500	69	1	0
…	…	…	…	…	…	…	…	…	…

Linked list to speed up lookups and to maintain IP/Netmask prefix order for matching

Figure 11. Layout of the destination lookup table

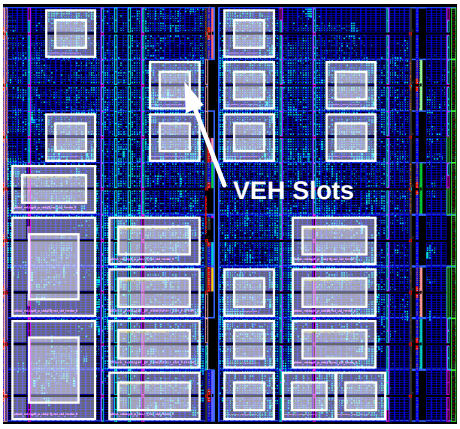

Figure 13. FPGA Layout for 24 VEH Slots

Table I
SYNTHESIS RESULTS FOR MASTER NODE COMPONENTS

Module	LUT	Reg. Bits	BRAM
Network Core incl. Management	12,297	8,884	93
Ring Interface	788	1,489	16
Mapped incl. MAC, XAUI and Clocks	**16,532**	**13,526**	**117**
in % of SX95T	28	22	47

server, sending data to the VEHs at 10 Gb/s. Partial reconfiguration was performed under operator control, loading in new bitstreams via network from the management station.

A. Synthesis Results

The synthesis results for all components are given in Table I and II. For the VEH nodes, we show results only for the LX155T, as the results for the SX95T are very similar (in terms of resource requirements).

The NetStage core on the SX95T Master node requires around 20% of the LUTs and 38% of the BRAMs. The high

Table II
SYNTHESIS RESULTS FOR VEH NODE COMPONENTS

Module	LUT	Reg. Bits	BRAM
Ring Interface	976	2,048	20
PR Controller	722	544	4
VEH Section with 24 Slots (w/o VEHs)	15,494	6,426	120
Total incl. MIG, without VEHs	**19,540**	**12,428**	**150**
in % of LX155T	20	12	70

number of BRAMs is due to several buffers and the global application state memory. The mapped design including IP blocks occupies around 28% of the LUT and 47% of the BRAM resources, distributed amongst 50% of the slices. This still leaves sufficient area unoccupied on the SX95T to allow for further extension of the Master node functionality.

The number of available BRAMs is crucial for our platform (due to the multiple buffers). As the SX95T and the LX155T have nearly the same number of BRAMs, these mapping results confirm our decision to put the Master node into the SX95T and leave the large number of LUTs inside the LX155T available for VEHs.

In the VEH node, the ring interface, the partial reconfiguration controller, and the VEH slot interfaces occupy around 20 % of the FPGA. This leaves nearly 80% of the the LUT ressources available for the actual VEH implementations. In practice, the total number of slots per FPGA is limited by the number of BRAMs available to implement the slot buffers (five BRAMs are needed per slot). From these results, we conclude that we could theoretically support up to 36 VEH slots per FPGA on both the LX155T and SX95T.

In comparison to our single-chip implementation, which could hold 20 VEHs together with the NetStage core on a single LX155T device, the multi-FPGA approach is a significant improvement of the total processing power of our platform. When using all three VEH node FPGAs to

Table III
SYNTHESIS RESULTS FOR THE VEHS

Module	LUT	Reg. Bits
SIP VEH	1082	358
MSSQL VEH	875	562
Web Server VEH	1026	586
Mail Server VEH	741	362

Table IV
RTT FOR A 1000 B PACKET: OVERALL AND PER SYSTEM COMPONENT

Buffer Fill Level	RTT	Core	Ring	VEH
empty	5.5μs	3.6μs	1.4μs	0.5μs
half	28.7μs	9.8μs	17.4μs	1.5μs
full	51.9μs	16μs	33.4μs	2.5μs

their full extent, the system supports the parallel operation of 100 VEHs (depending on module size), which should suffice even for very complex honeypot use-cases.

B. VEHs

Table III summarizes the area requirements for the various VEH modules. They are only showing little variation, which is advantageous for putting them into different slots on the FPGA. Amongst them, the SIP VEH requires the most LUTs, as it contains the most complex pattern matching algorithm. Overall, the VEHs are relatively small compared to the device capacity, thus we are confident that our slot numbers are realistic.

C. Performance

The actual response time depends on the latency of the platform and the speed of the VEHs. As these numbers are, in turn, highly dependent on the implemented functionality, and the distribution of incoming network traffic, we show numbers for the upper and lower limits. For these experiments, we consider different fill levels of the buffers inside the NetStage core and the ring: All buffers empty (the best case), nearly half full (average case), and nearly full (worst case). For simplicity, we assume that all buffers in the system have the same fill level, and that the VEHs are able to actually sustain a speed of 10 Gb/s (possible using the sample VEHs described above).

Table IV lists the total round-trip-times (RTT) for a 1000 byte request packet that generates a 1000 byte response packet. As the packets have the same size, the time is independent of the ring location of the device holding the measured VEH.

Obviously, the fill level of the buffers inside the ring nodes has a severe impact on the latency, inducing a 10x increase in latency between empty and nearly full buffers. However, as we are currently feeding the system with only one 10G interface, and the VEHs are all designed for high-speed operation, the buffers should not fill up in practice. We thus expect the average latency of the current system to be between 10-20μs.

Table V
SLOT SIZE DISTRIBUTION AND RECONFIGURATION TIME

Qty.	LUT / BRAM	Bitfile Size	Reconfiguration Time			
			Raw		Total	
			w/o SD	w/ SD	10 Gb/s	1 Gb/s
14	1440 / 0	59KB	151μs	154μs	218μs	749μs
4	2304 / 0	119KB	305μs	308μs	432μs	1503μs
4	2304 / 2	128KB	328μs	332μs	465μs	1617μs
2	4864 / 0	237KB	607μs	610μs	852μs	2985μs

D. Partial Reconfiguration Results

Table V lists the raw reconfiguration time and the total time needed to update a VEH. The raw reconfiguration time is measured from the beginning of the DMA transfer between SDRAM and ICAP and the end of the reconfiguration process. The total update time is measured from the first reception of an bitstream packet request at the management interface until the DONE message sent by the node PRC has been received at the Master. This time includes all data transfers of bitstream data from the management PC to the system using the dedicated management network interface, sending bitstream messages on the ring from Master to the VEH node, and the actual raw configuration time. The measurements were made both for using a a 1G and a 10G interface at 80% utilization for management). As the bitstream data has priority on the ring, we assume empty buffers here. Furthermore, we assume that the target FPGA is the middle one of the three nodes in the ring.

We also distinguish two cases for when looking at the raw reconfiguration times: On a clean shutdown (labeled "w/ SD"), an outgoing VEH is allowed to fully process the packets already present in its input queue. Without a clean shutdown ("w/o SD"), the enqueued packets are discarded when the slot is reconfigured. For the clean shutdown measurement, we assume that the receive buffer of the VEH to be replaced is half full and that the VEH is able to process data at 10 Gb/s (being conservative, since all of our current VEHs can actually handle more).

The time required for cleanly shutting down the outgoing VEH is negligible: Most of the reconfiguration time is actually taken by feeding the bitstream into the ICAP, which limits the overall reconfiguration speed. Thus, a small size of the VEHs is important for fast reconfiguration (see also Section VI-E) and justifies our approach of heterogeneously sized VEH slots (we can configure the 14 smaller VEH slots much faster than the 4+4+2 larger ones).

When looking at the total reconfiguration time including transfer of the bitstreams from the management PC, using

a 10 Gb/s management link adds only roughly 40% of overhead while still achieving numbers below 1 ms. Thus, even when accessing bitstreams not already stored in the node-local DDR2-SDRAM, the MalCoBox can quickly be adapted to changing attack behavior. For conventional use (update of VEHs once in a while), even a 1 Gb/s management link would suffice, as even the largest VEHs require less than 3 ms to transfer and configure.

E. Impact of data path width

To evaluate the impact of the 128b data path on the VEH size, we created 64b versions of the SIP and the Web Server VEHs, and compared them to the original 128b implementation (Table VI). Data path conversion between the NetStage core and the VEHs can be easily performed by the wrappers at the cost of a reduced throughput for the attached VEH.

The area overhead of the 128b version is roughly 75% for the SIP VEH and 65% for the Web Server VEH. This was to be expected, since these VEHs mostly read data from the input buffer and write data to the output buffer. The area required is thus strongly related to the bus width. Together with the data path area, the BlockRAM usage is also reduced: With 64b operation, we can now narrow the buffers and only require three BlockRAMs per wrapper instead of five for 128b VEHs.

Given these results, the number of parallel VEHs in the system could be increased even further by using the smaller datapath width, but only at a loss of per-VEH throughput (8 Gb/s with 64b width and 125 MHz VEH node clocks). Assuming a heterogeneous traffic distribution across all VEHs, this would not actually lead to a slow-down, since the NetStage core would keep its 20 Gb/s-capable 128b data path width and *distribute* the traffic across multiple of the smaller-but-slower VEHs. The bottleneck would only become apparent if all traffic was to be directed at a *single* VEH, which then would not be able to keep up with the 10 Gb/s line rate.

VII. Conclusion and Next Steps

With this refinement of our MalCoBox system, we have presented a scalable architecture to build a high-speed hardware-accelerated malware collection solution that offers great flexibility through partial reconfiguration and the distribution of VEHs over multiple FPGAs. In the multi-device scenario, the total amount of VEH processing power is significantly improved in contrast to the single-chip implementation, allowing us to implement even large-scale honeynets with a single appliance. A dedicated management interface allows quick updates or replacements of single vulnerability emulation handlers by loading new partial bitstreams, without interrupting the operation of the rest of the system.

Enabled by the high performance of the dedicated hardware, the VEHs actually performing the malware detection and extraction can contain a wide range of functionality: They can embed complex regular expression logic as well as simple request-response patterns, while still reaching the required throughput of 10 Gb/s. Furthermore, our hardware approach is resilient against compromising attacks and significantly reduces the risk of operating honeypots in a production environment.

The presented implementation of the multi-FPGA system on the BEEcube BEE3 quad-FPGA reconfigurable computing platform demonstrated the feasibility of the approach. Operators have a great flexibility to adapt the system to their needs: A trade-off can easily be made between individual VEH complexity and total vulnerability coverage using many different VEHs just by altering the distribution of VEH slots sizes; throughput and area can be traded-off by selecting between VEH implementations with 64b and 128b processing widths, and the overall system size can be scaled by selecting either the single-chip or the multi-FPGA approach.

We will continue our work in this area. MalCoBox is planned to be stress-tested in a real production environment connected to the Internet (e.g., university or ISP). From this, we expect to gain valuable insights on how to improve the architecture and its parameters in the future. Furthermore, we will combine the multi-FPGA system with our recent work on self-adapting by dynamic partial reconfiguration based on the observed traffic characteristics. We expect to achieve a platform that exploits many of today's cutting edge technologies in reconfigurable computing to enable a system presenting maximal flexibility, performance and security to the user.

Table VI
Synthesis results for 128b and 64b VEHs

VEH	LUT	Reg. Bits
SIP 128 Bit	1082	358
SIP 64 Bit	619	278
Web Server 128 Bit	1026	586
Web Server 64 Bit	663	244

Acknowledgment

This work was supported by CASED and Xilinx, Inc.

References

[1] "Internet security threat report, volume xv," Symantec, 2010. [Online]. Available: http://www.symantec.com

[2] "Honeyd." [Online]. Available: http://www.honeyd.org

[3] S. Mühlbach, M. Brunner, C. Roblee, and A. Koch, "Malcobox: Designing a 10 gb/s malware collection honeypot using reconfigurable technology," in *FPL '10: Proceedings of the 20th International Conference on Field Programmable Logic and Applications*. IEEE Computer Society, 2010, pp. 592–595.

[4] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable network packet processing on the field programmable port extender (fpx)," in *FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*. ACM, 2001, pp. 87–93.

[5] S. Mühlbach and A. Koch, "A dynamically reconfigured network platform for high-speed malware collection," in *ReConFig '10: Proc. of the Intl. Conf. on ReConFigurable Computing and FPGAs*, 2010.

[6] ——, "A scalable multi-fpga platform for complex networking applications," in *FCCM '11: Proc. of the 19th Annual IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2011.

[7] "Bee3 hardware user manual," BEEcube Inc., 2008.

[8] V. Pejovic, I. Kovacevic, S. Bojanic, C. Leita, J. Popovic, and O. Nieto-Taladriz, "Migrating a honeypot to hardware," in *SECUREWARE '07: Proc. Intl. Conf. on Emerging Security Information, Systems, and Technologies*. IEEE Computer Society, 2007, pp. 151–156.

[9] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing," in *Proc. of the 2007 IEEE International Conference on Microelectronic Systems Education*, ser. MSE '07. IEEE Computer Society, 2007, pp. 160–161.

[10] C. Albrecht, R. Koch, and E. Maehle, "DynaCORE: A Dynamically Reconfigurable Coprocessor Architecture for Network Processors," in *Proc. of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE Computer Society, 2006, pp. 101–108.

[11] S. Bourduas and Z. Zilic, "A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing," in *Proceedings of the First International Symposium on Networks-on-Chip*, ser. NOCS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 195–204. [Online]. Available: http://dx.doi.org/10.1109/NOCS.2007.3

[12] C. Thacker, "DDR2 SDRAM Controller for BEE3," Microsoft Research, 2008.

[13] K. v. d. Bok, R. Chaves, G. Kuzmanov, L. Sousa, and A. v. Genderen, "Fpga reconfigurations with run-time region delimitation," in *Proceedings of the 18th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, 2007, pp. 201–207.

[14] Y. Hori, A. Satoh, H. Sakane, and K. Toda, "Bitstream encryption and authentication using aes-gcm in dynamically reconfigurable systems," in *IWSEC '08: Proceedings of the 3rd International Workshop on Security*. Springer-Verlag, 2008, pp. 261–278.

[15] M. Thumann, "Buffer overflow in sip foundry's sipxtapi," 2006. [Online]. Available: http://www.securityfocus.com/archive/1/439617

[16] D. Litchfield, "Microsoft sql server 2000 unauthenticated system compromise." [Online]. Available: http://marc.info/?l=bugtraq\&m=102760196931518\&w=2

[17] "Partial reconfiguration user guide," Xilinx, 2010.