Security Vulnerability in Processor-Interconnect Router Design

WonJun Song, John Kim
KAIST
Daejeon, Korea
{wjsong,jjk12}@cs.kaist.ac.kr

Jae W. Lee
SungKyunKwan University
Suwon, Korea
jaewlee@skku.edu

Dennis Abts
Google Inc.
Madison, WI
dabts@google.com

Abstract

Servers that consist of multiple nodes and sockets are interconnected together with a high-bandwidth, low latency processor interconnect network, such as Intel QPI or AMD Hypertransport technologies. The different nodes exchange packets through routers which communicate with other routers. A key component of a router is the routing table which determines which output port an arriving packet should be forwarded through. However, because of the flexibility (or programmability) of the routing tables, we show that it can result in security vulnerability. We describe the procedures for how the routing tables in a processor-interconnect router can be modified. Based on these modifications, we propose new system attacks in a server, which include both performance attacks by degrading the latency and/or the bandwidth of the processor interconnect as well as a livelock attack that hangs the system. We implement these system on an 8-node AMD server and show how performance can be significantly degraded. Based on this vulnerability, we propose alternative solutions that provide various trade-off in terms of flexibility and cost while minimizing the routing table security vulnerability.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General–security and protection; C.2.1 [**Network Architecture and Design**]: network communications, network topology

General Terms

Security

Keywords

processor-interconnect; routing table; router; vulnerability

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2660267.2660290.

1. INTRODUCTION

Interconnection networks can be found in many different domains, including on-chip network for multicore architecture as well as off-chip networks for large-scale system such as supercomputers [8]. The interconnection network is a critical component of modern systems as communicating or moving data between compute nodes are becoming more important in determining overall system performance and cost. With multisocket servers being commonly used in high-performance computing and datacenters, we focus on the processor-interconnect or the interconnection networks found in these modern servers with multiple sockets and nodes. Each socket can consist of multiple *nodes* with multi-chip modules and each node can consist of multiple cores. These nodes are interconnected with a processor-interconnect and some commonly used processor-interconnects include Intel QPI [29] and AMD Hypertransport [2]. In this work, we focus on the processor-interconnect found in these multi-socket (multi-node) SMP (symmetric multiprocessing) systems and the security vulnerability in the routing tables found within the processor-interconnect router microarchitecture.

The basic building block of any interconnection network is the router which receives packets and forwards them to the appropriate output ports. As a result, one of the first steps for an arriving packet is determining the output port and is done by some routing computation logic within the router. To provide flexibility, a routing table is commonly used for the routing computation. The routing table structure is a lookup table, with the lookup done often based on the packet's destination and the entries within the routing table specifying the output channel [1] that needs to be used to route the packet. Although the routing table provides flexibility, we show in this work that such flexibility can result in security vulnerability. To the best of our knowledge, this is one of the first work to investigate security vulnerability within a processor-interconnect; in particular, the routing table that is located within each router.

We first describe the methodology for how to modify the routing table, which includes first understanding the topology (or the connectivity) of the processor-interconnect. Based on the ability to modify the routing table, we present three different types of attacks. The livelock attack modifies the routing table such that packets circulate in the network and

[1]In this work, we use the terminology link and channel interchangeably. In addition, the term nodes and routers are also used interchangeably since a communication between "nodes" are done through the routers.

do not arrive at their destination and hangs the system. The other two types of attacks are different form of performance attack that degrades the latency or the bandwidth of the processor-interconnect. Modifying the routing table requires kernel access, which we assume can be gained by exploiting previously known techniques [5, 11, 22, 24]. With root access, there are many different ways of crashing or hanging the system as well as degrading system performance. The system attacks presented in this work are an alternative form of such attacks. However, the performance attacks from modifying the routing table not only results in performance degradation but also becomes very difficult to detect without understanding that the routing table could have been maliciously modified.

In particular, the contributions of this work include the following.

- We show that programmable routing tables in processor-interconnect has security vulnerability that can be exploited.
- We present a new type of attacks based on modifying the routing table. We describe a *livelock* attack where packets do not reach their destination and hangs the system. In addition, two different performance attacks are described – *round-about* attack, which increases interconnect latency, and *hotlink* attack, which reduces available bandwidth.
- We evaluate the impact of these attacks on real machine and show how the livelock attack can hang the system while the performance attack can result in significant performance degradation.
- We propose alternative solutions to avoid the security vulnerability that provide different trade-off in terms of flexibility and cost.

The rest of the paper is organized as follows. In Section 2, we provide a background on router designs and interconnection networks. In addition, we describe modern processor-interconnect and its routing table. The attack procedure on how to modify the routing table is described in Section 3 and the threat model and three different type of attacks are described in Section 4. We evaluate the impact of these attacks on a Dell server in Section 5. Alternative solutions to avoid the routing table vulnerability is discussed in Section 6. Section 7 presents related work and we conclude in Section 8.

2. BACKGROUND

2.1 Router Microarchitecture

A conventional router microarchitecture is shown in Figure 1. The different stages within a router pipeline include routing computation, resource allocation (including switch and virtual channels) and crossbar switch traversal [8]. In this work, we focus on the *routing computation* aspect within a router microarchitecture that determines the routing path for a given packet. Once a packet arrives at the input buffer, the packet first needs to go through the routing stage to determine the router output port that the packet should be routed through. The routing stage can implement different routing algorithms, including oblivious or adaptive routing.

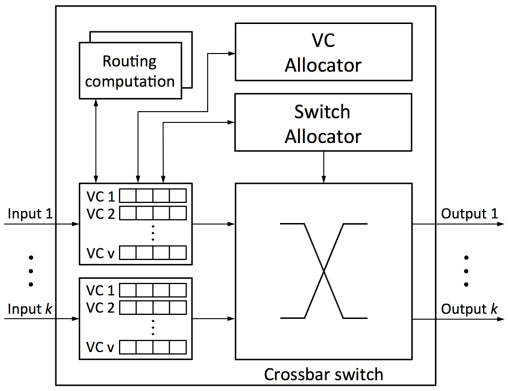

Figure 1: High-level block diagram of a router microarchitecture.

Given a particular routing algorithm, the routing logic for the routing stage can be implemented with algorithmic routing or table-based routing [8]. For algorithmic routing, dedicated combinational logic circuit is implemented which computes the next hop (or the output port) at runtime. With a dedicated hardware, the routing logic is often more efficient (lower latency and lower area); however, algorithmic routing is often limited to simple routing algorithms and lacks flexibility – e.g., if the topology changes, the routing logic needs to be re-designed. As a result, many routers use table-based routing where a routing table exists at each router to implement the routing algorithm. Based on the destination of the packet, a table lookup is done to determine the next hop. Both oblivious and adaptive routing can be implemented with table-based routing. In this work, we exploit this flexibility in the routing tables of the router microarchitecture and present different system attacks.

2.2 Interconnection Network Characteristics

Two important performance metrics in any interconnection network are latency and bandwidth. Latency is defined as the amount of time it takes for a packet from its source to its destination and has significant impact on overall performance. Bandwidth defines the throughput of the entire system and efficient utilization of the bandwidth maximizes performance for bandwidth sensitive workloads. In addition, the network must guarantee livelock and deadlock freedom. Deadlock is the process where dependencies in resources exist that prevent progress in the network while livelock is the condition where packets move in the network but do not reach their destination [8]. Both livelock and deadlock needs to be avoided (or properly recovered from when they occur) in the network to provide a stable system.

In this work, we show how both the latency and the bandwidth performance metrics can be degraded by modifying the routing table – i.e., increase the latency for remote memory access or consume unnecessary bandwidth through misrouting. In addition, we show how the routing table can be modified to create a livelock in the system by circulating a packet within the processor-interconnect and hang the system.

2.3 Processor-Interconnect Router

There are different applications of interconnection networks, including on-chip networks and larger-scale networks.

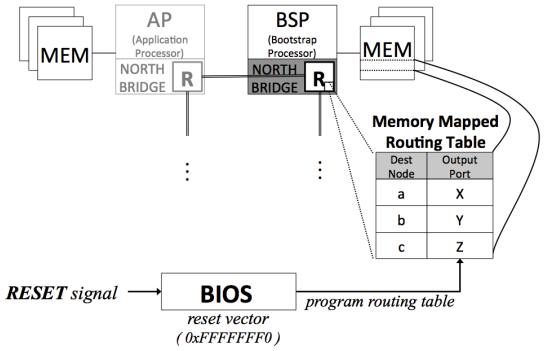

Figure 2: Routing table initialization process of memory-mapped routing table.

This work focuses on processor-interconnects found in multi-socket systems but the security vulnerability can be exploited in other interconnection networks. For the rest of this paper, we use an AMD-based system since the details of Hypertransport Interconnect are publicly available [2]. In the AMD architecture, the router for each node in the processor-interconnect network is placed on the Northbridge. At the high-level, the router microarchitecture is similar to the conventional router microarchitecture described earlier in Section 2.1. The multiple cores in a node are connected to the crossbar switch through the System Request Queue [6] and thus, implements a form of *external* concentration [25] as multiple cores share a single port into the router input. The routing computation is implemented with a routing table which is memory mapped. In the Hypertransport Interconnect that we consider, there are 4 physical channels for each router, with each physical channel supporting 4 virtual channels [7] to avoid protocol deadlock. With 4 physical channels, each router can connect to four other routers. However, each physical channel can also be physically partitioned into 2 sub-channels and thus, the number of routers connected to can be up to 8.

2.4 Routing Table Initialization

Since the routing computation is implemented with a routing table, the routing table provides flexibility and can be programmed with alternative entries to support different system size and provide fault tolerance for physical link failure [23]. Figure 2 shows the overall process for routing table initialization at boot-up time. Initially, alternative routing tables are stored in the flashed memory with the BIOS. The different routing tables represent various system configuration that is possible for the given CPU and the motherboard and the appropriate routing table is selected based on the system configuration. In a multiprocessor system, one of the cores is designated as the Bootstrap Processor (BSP), which is designated as the first processor that comes online and is responsible for proceeding with boot process, while the rest of the processors are designated as Application Processors (AP). When a reset signal is generated, the BSP jumps to the reset vector to fetch the BIOS code. As part of the BIOS, the BSP determines the channels available and the connectivity of the channels to the neighboring nodes. Once the BSP completes this process, the remaining APs go through the same process. Thus, the number of nodes and all available links are determined through the BIOS. Based on this information, the appropriate routing table is selected and loaded into the routing table register in each router within the Northbridge. Since the routing table registers for each router are mapped to the memory, the routing table entries can be modified through memory operations.

3. ROUTING TABLE ATTACK PROCEDURES

In this section, we provide the details on the process required for routing table attack. We assume kernel access throughout this section as the routing table entries can only be modified in kernel mode. The procedure described in this section is based on the AMD Hypertransport router but can be generalized to other systems as well.

3.1 Identifying Network Topology

The first step in the routing table attack is the need to understand the network topology or the connectivity between the nodes in the target system. In on-chip networks, the topology is determined at design time and fixed. Similarly, the topology for large-scale networks such as supercomputer networks is also known and determined by the design/vendor. However, processor-interconnect networks using either QPI or HT interconnect do not necessarily have a determined topology, and their topologies can be arbitrary. The topology is determined by the processor's routers that limit the number of ports per node and the motherboard vendors as they determine the layout of the channels and the node connectivity. The documentation provided by CPU vendors (e.g. AMD [10] and Intel [18]) provides some example topologies but the topology of the actual system is not necessarily the same, as we show later in Figure 4. As a result, since the topology can be irregular and the documentation to understand the processor-interconnect is not readily available, the network topology needs to be reverse-engineered.

To properly exploit the routing table vulnerability, two components of the topology needs to be known – the *logical* topology and the *physical* topology. The logical topology describes the logical connections between the nodes while the physical topology describes the physical information – e.g., which particular channels (or output ports) are used for the logical connection. To determine the topology and the connectivity, we leverage the information available from the SLIT (System Locality Information Table) table provided by ACPI (Advanced Configuration and Power Interface) [19], link status register, and the current existing routing table. The SLIT table provides the information necessary for the logical topology while the other two components are necessary to determine the physical topology.

3.1.1 SLIT Table

The NUMA distance [19] is an integer value that represents the distance or hop count between two nodes. The distance information is provided by the SLIT(System Locality Information Table) table [26], which is a matrix that contains the relative distance information. Linux operating system exposes the relative distance information in user space such as *sysfs* and NUMA utilities *numactl* and the NUMA information can be easily obtained. An example of NUMA distance information for Dell PowerEdge R815 is shown in Figure 3(a) and the definition of the NUMA values are shown in Figure 3(b). The hop count between the

NODE	0	1	2	3	4	5	6	7
0	10	16	16	22	16	22	16	22
1	16	10	22	16	16	22	22	16
2	16	22	10	16	16	16	16	16
3	22	16	16	10	16	16	22	22
4	16	16	16	16	10	16	16	22
5	22	22	16	16	16	10	22	16
6	16	22	16	22	16	22	10	16
7	22	16	16	22	22	16	16	10

(a)

Relative Distance shown in SLIT	hop count
10	local node
16	1 hop
22	2 hop

(b)

Figure 3: (a) SLIT table with NUMA distance for Dell PE R815 server and (b) SLIT value definition.

nodes can be determined based on this information. The value of 10 represents the local access (local memory) and thus, the diagonal of the matrix in the SLIT table always consists of the value 10. For the nodes that are 1 hop away (e.g., NUMA value of 16), the two nodes are known to be directly connected.

By iterating through all of the nodes, the *logical* topology of the system can be determined. For example, starting with Node 0 (R0), a graph can be drawn to show all the nodes that are directly connected (or one hop away), which include Nodes 1,2,4 and 6 (Figure 4(a)) for the Dell PE R815 server. Each edge is bidirectional to represent communication in both directions. Once Node 0 connectivity is known, the same process can be repeated for the other nodes. For simplicity of the figure, only the connectivity analysis for Node 0 and Node 1 is shown in Figure 4(a). Once this process is repeated for all of the nodes, the complete logical topology can be obtained, as shown in Figure 4(b). However, this information is insufficient to determine which channels (or output ports) in the router is used for the connection and thus, does not provide overall *physical* connectivity information. To find this, we leverage information from the link status register and the current routing table itself.

3.1.2 Link Status Register

The link status register provides information on which links in the system interconnect are actually connected or not [10]. In addition, it provides information on whether the channels are used for *internal* connections (e.g., channel between two nodes within a multi-chip module) or for external connections to connect with other nodes or peripheral devices. For example, the dotted box in Figure 4(a) shows the nodes that are packaged within the same multi-chip module based on the link status register information.

3.1.3 Routing Table

The last information necessary is understanding which particular channels are used to connect to the nodes in other packaging modules. We leverage the existing routing table for this information. Within each router, a single routing table is shared by all of the ports. The routing table in the AMD Hypertransport router that we evaluate consists

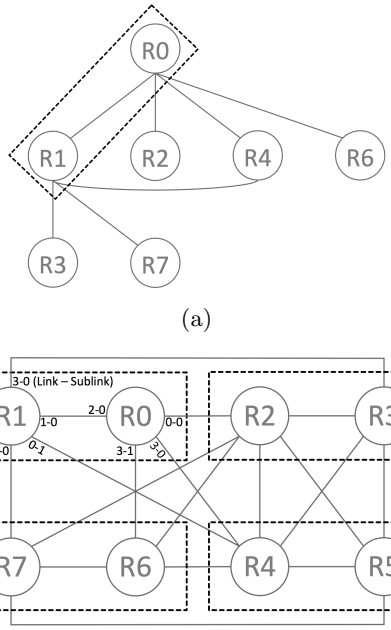

Figure 4: (a) Partial logical topology from the SLIT table NUMA distance information to show the connectivity between the nodes and (b) the full topology. The dotted box shows the nodes that are packaged together in the same module.

of 8 entries and is indexed by the destination node ID of a packet. Each entry in the routing table is 32-bit long, as shown in Figure 5, with entry containing routing information for three type of packets – broadcast packets, request packets, and response packets. The routing decision can be different for the different packet type. Each bit in the routing table entry indicates a link identifier (i.e., output port or the output channel and the sublink information) through which the packet needs to be routed for a given particular packet type.

With the additional information provided by the routing table (in addition to the SLIT table and the link status register), the complete topology (including the physical connectivity) of the system can be determined. An example of the different link (and sublink) assignments for the server used in our evaluation is shown in Figure 4(b) for two of the nodes (R0 and R1). Based on this understanding of the network topology, we describe how the routing table can be modified in the following section to achieve the different type of attacks that will be described later in Section 4.

3.2 Modifying Routing Table

Since the routing table in the Hypertranport-based interconnect is a memory-mapped I/O device, it can be accessed by memory operations. As shown in Figure 5, all of the routing tables in the system are mapped to specific memory address range sequentially where each node has its own memory address for a single routing table. The memory address range used for the routing table is in PCI-defined configuration space and thus, the routing table can be modified with PCI Kernel APIs. We implemented a malicious device driver which is inserted into kernel address space to

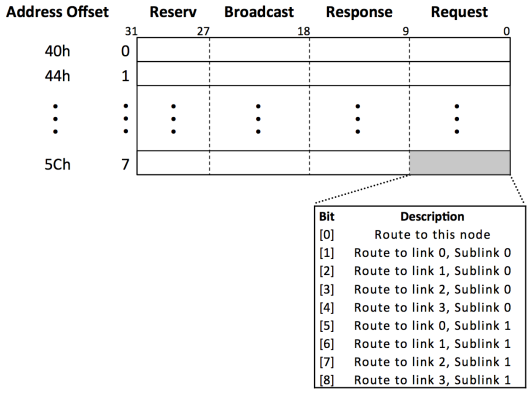

Figure 5: Routing table organization in Hypertransport Interconnect.

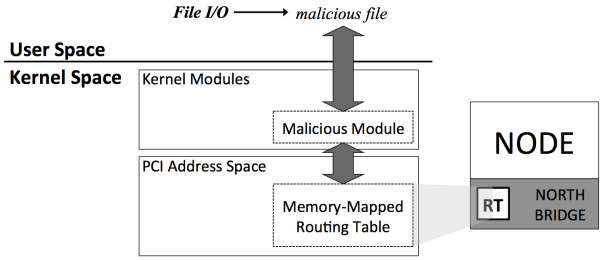

Figure 6: Routing table modification procedure.

access PCI-defined configuration space. This driver first creates a malicious file in *proc* virtual file system and it maps all of the file operations such as open, read and write into the routing table in the Northbridge. Since the malicious file in *proc* file system exists in user space with normal user privilege, the attacker can read and write the file via UNIX standard I/O which results in modifying the routing table in the Northbridge. The malicious file in the system can serve as a backdoor-like attack that allows any user to access the routing table and modify it. This malicious file can also be removed after the routing table modifications.

4. ROUTING TABLE ATTACK SCENARIO

In the previous section, we described the how the routing table in a system interconnect can be modified. In this section, we first begin by describing our threat model. We then introduce three different routing table attacks. The first attack (livelock attack) results in a system hang as packets continue to circulate within the network. The next two attacks (roundabout attack and hotlink attack) are performance attack as the latency and the bandwidth of the interconnect is degraded.

4.1 Threat Model

In this work, we consider an attacker that has kernel-level access in a commodity operating system. With the large number of lines of codes in an OS, different exploits that result in kernel-level access for commodity OSes have been shown [37, 30, 27]. The goal of the attacker is to crash the system or create system performance attack with security

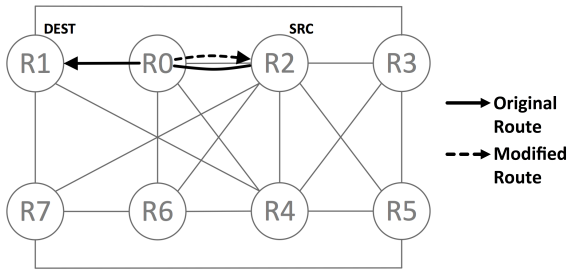

Figure 7: Block diagram of livelock attack between R2 and R0 nodes.

analyst (e.g., IT professional) unable to determine the source of the attack. We assume the attacker has no physical access to the hardware – either the router or the processor.

After the initial attack (or modifications to the routing table), the routing table state is modified and no malware is necessary to maintain the attack. Thus, the device driver can be removed and leave minimal trace on the system; however, this prevents any further modifications to the routing tables. If the attacker desires to scrub any traces (and maintain the possibility of further modifying the routing table), an appropriate Rootkit will be necessary.

4.2 Livelock Attack

As described earlier in Section 2.2, livelock in interconnection network is defined as a condition where a packet continues to move through the network but does not reach its destination [8]. Guarantee from livelock freedom is necessary in order for an interconnection network to function properly. In the routing table attack on processor-interconnect, a livelock can be created by forming a routing "loop" as shown in Figure 7 which results in a packet from circulating in the network without reaching its destination.

In the network shown in Figure 7, assume a packet is sent from source (R2) to destination (R1) by routing through an intermediate node (R0). In normal operation at R0, a packet that arrives at R0 destined for R1 will be routed through the appropriate channel to reach its destination. However, the routing table in R0 can be modified such that for packets destined for R1, instead of being routed through the "West" output port, the packet is re-routed towards the "East" output port (as shown with the dotted line in Figure 7). With this modification to the routing table in R0, the packet will arrive back at R2 and then, be sent to R0 again since the routing table in R2 has not been modified. As a result, the packet will continue to circulate between R0 and R2, without arriving at the destination R0. Since the processor-interconnect system is not intended to tolerate missing or dropped packets, the livelock condition will eventually results in a system hang.

Figure 8 shows how the routing table in node R0 is modified to implement this livelock attack. Figure 8(a) shows the original routing table, based on the channel (link) and sublink assignments described earlier in Figure 4(b). For packets destined for node 1 (R1), the original routing table in R0 shows that packets should be routed through link 2 as 4th bit of both the request and reply fields of the routing table is set to 1. However, if the routing table is modified such that these bits are set to 0 but the second bit (which corresponds to link 0 – Figure 8(b)) is set, packets will be

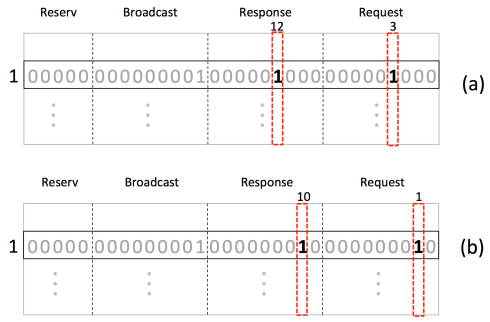

Figure 8: (a) Original routing table in R0 and (b) modified routing table to implement the livelock attack. For simplicity, only the routing table entry for destination node 1 (R1) is shown in the figure.

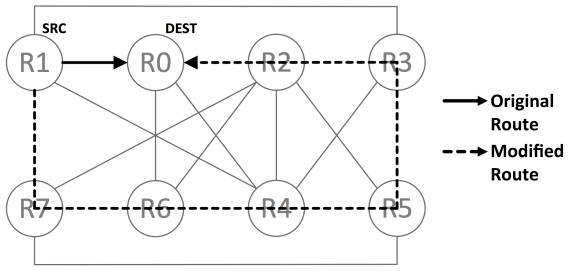

Figure 9: Block diagram of roundabout attack between R1 and R0 nodes.

sent through the channel that is connected to R2 (instead of R1) and result in a livelock. The example of livelock described in this section is one example of livelock but similar routing livelock condition can be created that involves more nodes or other nodes in the system.

4.3 Roundabout Attack

The latency of a remote memory access in a NUMA system can have significant impact on overall system performance. Compared with local memory access, the remote memory access has higher latency because of the latency to traverse one or more routers (and the channels) to reach the remote memory. The routing for the remote memory access is often done with minimal routing [2] where the number of hops (or intermediate routers) is minimized to reduce overall latency. However, in this section we describe a performance attack that exploits the routing table security vulnerability by increasing the remote memory access latency through the *roundabout* attack where the routing hop count is increased.

An example of a roundabout attack is shown in Figure 9. Assume a packet needs to be sent from a source node R1 to a destination node R0 (e.g., application running on a processor located on the R1 node needs to access the memory that is located on the R0 node). Without any modification to the routing table, the minimal route would be taken as shown with the solid line in Figure 9. However, a performance attack can be done to increase the remote memory

[2]Non-minimal routing can be used to increase path diversity but most processor-interconnect do not leverage non-minimal routing because of its complexity [8].

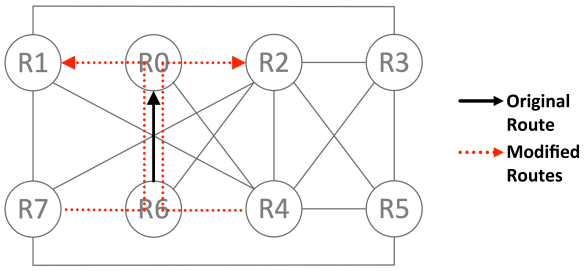

Figure 10: Block diagram of Hotlink Attack through a common channel between R6 and R0 nodes.

access latency by altering the routing path with routing table changes. In this particular example, we modify the routing table such that instead of the packet taking only 1 hop, the packet takes 7 hops, as shown with the dotted line in Figure 9. This requires modifying the routing table in not only R1 but also all the other intermediate nodes.

One side effect of this roundabout attack is that other packets destined to R0 from other nodes will also utilize this new path. For example, a different packet from node R4 to R0 would normally take the minimal path (R4,R0) with a single hop. However, the changes in the routing table will result in the packet taking the non-minimal route (R4, R5, R3, R2, R0). The roundabout attack can also have a different *degree* of indirectness or the number of intermediate routers that a packet will be routed through. The example in Figure 9 was the worst-case roundabout attack for this particular system since all of the other intermediate router nodes is traversed. Instead of increasing the hop count from 1 to 7, the hop count can also be increased to arbitrary numbers – e.g., three hop counts with (R1,R7, R6, R0) routing path – and have different impact on performance.

4.4 Hot-Link Attack

The roundabout attack described in the previous section attacked the latency aspect of interconnect performance. However, another important metric is bandwidth since the channel bandwidth is a resource that is shared by the different nodes and impact overall interconnection network throughput. In this section, we describe the hotlink attack which degrades the bandwidth in the system by re-routing packets through a common or a *hot* link in the system and degrade overall performance.

An example of the hotlink attack is shown in Figure 10. We assume there is a normal flow that uses the channel between R6 and R0 where a flow is defined as traffic originating from a source to a destination. However, by altering the routes for other flows, this particular channel can be overloaded and results in a bottleneck channel and performance degradation. In this particular example, two other flows (R7 to R1) and (R4 to R2) has been modified such that instead of routing minimally between the source and destination, the packets are routed through intermediate routers R6 and R0. Since all of the packets will end up sharing the same channel between nodes R6 and R0, the particular channel or link becomes the *hotlink* and degrades the bandwidth utilization and degrades overall performance. Unlike the livelock or the roundabout attack, multiple entries in the routing table needs to be modified – for example, in R6, the routing en-

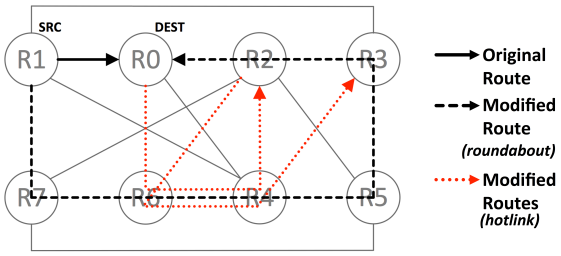

Figure 11: Block diagram of performance attack that combines both roundabout and hotlink attack.

Description	Value
System	AMD Opteron 6320
# of Sockets	4
# of Nodes	2 per socket
# of Cores	4 per node
L2 Cache	1 MB per node
L3 Cache	6 MB per node
Interconnect	6.4 GT/s HT 3.0
# of QPI Links	4 per node

Table 1: Dell PE R815 workstation using in our evaluation.

tries for destination R2 and R1 needs to be modified such that they are routed through the R6-R0 channel. In addition, the routing table in both R7 and R4 also needs to be modified accordingly as well.

For simplicity, the example in Figure 11 is shown where two flows are added to create the hotlink. However, this attack can also be generalized such that the number of flows contributing to the hotlink can be increased – e.g., R5 to R3 flow can be modified to route through (R5, R4, R6, R0, R2, R3) etc. In addition to roundabout attack and hotlink attack as individual attacks, the two type of attacks can be combined to maximize the performance degradation as shown in Figure 11. The hotlink shown in this example (between nodes R6 and R4) degrades the bandwidth usage of this channel and thus, the remote access latency between the R1 to R0 is further degraded, compared with the standalone roundabout attack.

5. EVALUATION

In this section, we describe the methodology used to evaluate the alternative performance attack. We then present the results of the different performance attacks and their performance degradation. We also evaluated the impact of the livelock attack, using the routing table changes described earlier in Section 4.2 to verify that the system hangs. Even though the system hangs, we were not able to identify any machine check error messages that are logged.

5.1 Methodology

In the evaluation, we used a Dell PowerEdge R815 system [9] with AMD processors [10]. The system uses a multi-chip module (MCM) and thus, a single socket consists of two nodes. The Dell PE R815 system consists of 4 sockets and thus, 8 nodes in our evaluation. The Hypertransport is used as the interconnect and the topology used to interconnect

Workloads	MPKI	Workloads	MPKI	Workloads	MPKI
gromacs	0.03	sphinx3	0.82	omnetpp	17.91
hmmer	0.04	astar	2.05	lbm	18.90
tonto	0.07	cactusADM	3.29	milc	21.28
bzip2	0.11	zeusmp	4.69	soplex	23.93
namd	0.34	gcc	6.40	libquantum	32.03
wrf	0.41	bwaves	10.12	mcf	40.30
sjeng	0.43	leslie3d	14.15		
gobmk	0.77	GemsFDTD	16.76		

L : 0 ~ 1
M : 1 ~ 10
H : 10 ~

Figure 12: MPKI measured from the Dell PE R815 for the different SPEC workloads.

the 8 nodes was described earlier in Figure 4(b). The details of the system are summarized in Table 1. SPEC CPU 2006 [13] are used in the evaluation and the performance degradation of the workload is measured by using execution time as the performance metric. We evaluate SPEC workloads that have different memory intensity (i.e., different amount of MPKI (miss per kilo-instruction)) to evaluate the performance degradation for different type of workloads. MPKIs from the last level cache (L3) for SPEC2006 benchmark suite [13] are measured using the target machine (see Figure 12). Higher MPKI often leads to more severe performance degradation from higher main memory latency and/or lower bandwidth. The workloads can be divided into three groups based on the memory intensity – L (memory non-intensive), M (medium), and H (memory-intensive). In our evaluation, we selected three representative workloads from each group : L(hmmer, tonto, namd), M(astar, zuesmp, gcc), H(soplex, libquantum, mcf). To minimize the impact of system variation and interference from background process, each evaluation is measured 10 times and the minimum execution time is used [20].

5.2 Performance Attack Evaluation

The roundabout attack is evaluated using the system description shown earlier in Figure 9. A single workload is executed on node R1 and we assume that the memory that it needs to access is located in R0. We vary the degree or the number of hop count in the roundabout attack and the results are shown in Figure 13(a). The results plot the execution time of the workload and the results are normalized to the baseline without any modification to the routing table. As expected, the L workloads are not impacted significantly by the roundabout attacks while M workloads has some impact and H workloads have the highest impact. In addition, for the M and H workloads, the performance degradation also depends on the degree of the roundabout attack – as the degree (or the number of hop count) is increased, the performance degradation also increases. For H workloads, the roundabout attack can result in up to 90% degradation with 7 hop roundabout attack.

Since the roundabout attack increases the processor-interconnect latency, we use the performance counters to measure the read average latency for memory read commands [10]. The latency values are shown in Figure 13(b) and the latency values are normalized to the baseline (i.e., 1 hop) where the routing table is not modified. As expected, the roundabout attack increases the latency and the increase is higher with higher hop count in the roundabout attack – on average, by introducing 7-hop roundabout attack, the latency increases by approximately 3× – and thus, resulting in the overall performance degradation. It is interesting to note

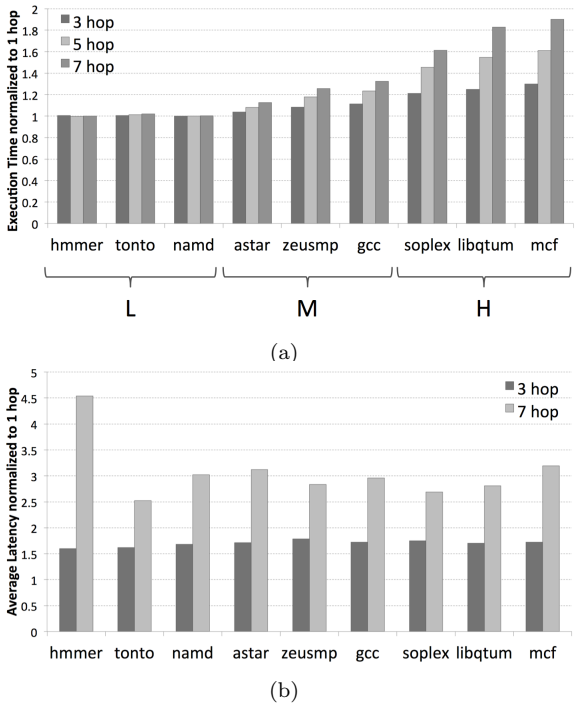

Figure 13: Results from roundabout attack including (a) performance degradation and (b) average network latency.

that for the HMMER, latency has significant degradation with 7-hop roundabout attack (by 4.5×), higher than other workloads, but the performance impact is negligible. Since HMMER is non-memory intensive, the number packets in the procesor-interconnect network are relatively small and thus, the variability from non-workload (background process) packets likely results in high latency value but has minimal impact on the workload performance.

To evaluate the hotlink attack, we assume a flow of traffic from node 6 (R6) accessing the memory in node 0 (R0) as shown earlier in Figure 10. For simplicity, we show the harmonic mean result for each group of workload (L, M, H). Since the hotlink attacks the bandwidth aspect of the interconnect, we measure the link utilization from the performance counters and average them across the different workloads within each category. The results of hotlink attack are shown in Figure 14(a) and the results are normalized to the baseline with no routing table modification. In the initial comparison, we add the results of *intra-node* hotlink where malicious codes are added to the *same* node. In our target system, each node consists of four cores so in addition to the application of interest, we run a very memory intensive workload (i.e., mcf) on the other cores to utilize the same hotlink channel bandwidth.

As shown in Figure 14(a), the performance degradation increases as the number of flows is increased since the hotlink becomes more of a performance bottleneck with more flows sharing the hotlink. In addition, the memory intensive workload performance is further degraded with the hotlink attack. For the H workloads with 6 flows in the hotlink attack, it results in approximately 3.8× performance degradation. Figure 14(b) shows the link utilization for the hotlink attack. Because of the limitations of the Hypertransport performance counters, we focus on the relative values. For the

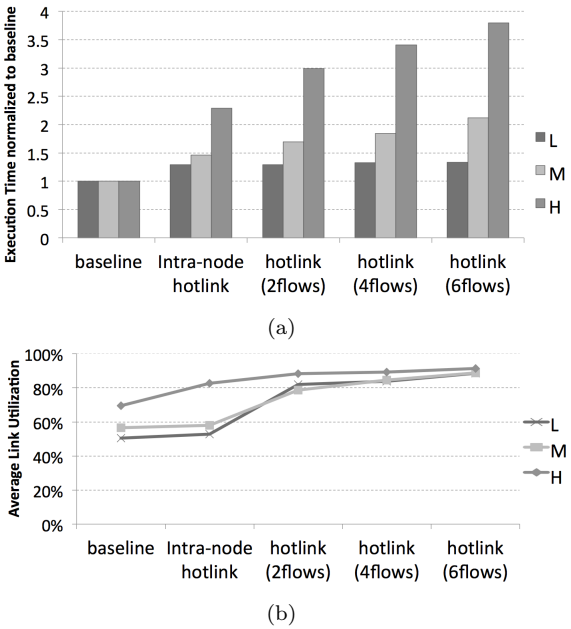

Figure 14: Hotlink attack evaluation results including (a) performance degradation and (b) average link utilization.

memory intensive workloads, the link utilization values are high even for baseline and continue to remain highly utilized (or slightly increase) as the hotlink becomes saturated. For the memory non-intensive workloads, the utilization is low in the baseline but increases as additional flows are added to the link and thus, there is performance degradation for memory non-intensive workloads, approximately 50% degradation with 6 flows in the hotlink attack.

In addition to the hotlink and roundabout attacks, the two can be combined to further increase the performance degradation, as shown earlier in Figure 11. The bandwidth contention for the hotlink will further degrade the latency on top of the roundabout attack and thus, further increase the remote memory access. Although the number of flows for the hotlink can be significantly higher, we only evaluate roundabout attack combined with hotlink attack that consists of only 1 and 3 flows. Higher number of flows resulted in more significant performance degradation and became infeasible to evaluate them appropriately. The results for the combined attack are shown in Figure 15 – and shows result where only 1 thread is being executed on a node and where 4 threads (i.e., intra-node hotlink) is being executed. Results show that with only 1 thread, the combined attack results in approximately 58× degradation in performance while with 4 threads, the combined attack results in approximately 250× performance degradation.

6. DISCUSSION

In this section, we discuss other possible performance attacks based on modifying the routing tables, in addition to what was discussed earlier in Section 4. We then discuss some possible solutions to the routing table vulnerability in processor-interconnect systems and their trade-off.

6.1 Other Performance Attacks

Coherence Message: In this work, the modification of the routing tables focused on request and response packets.

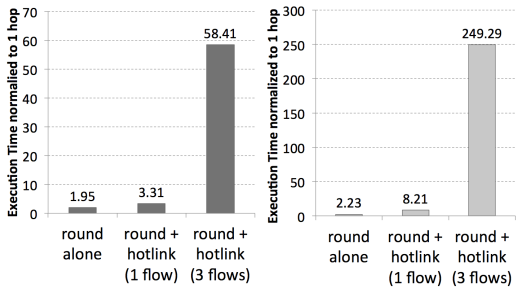

Figure 15: Results from combined performance attack (roundabout and hotlink attacks) for (a) 1 thread and (b) 4 threads per core.

However, another important class of messages in processor-interconnect is coherence messages and in the Hypertransport-based system, the corresponding broadcast messages. By modifying the coherence messages (especially for workload that generate a lot of coherence messages), both the bandwidth and the latency can be impacted by altering the coherence message routing table entries.

Topology Modification: The roundabout attack modified the hop count for a particular source-destination communication. However, this attack can be generalized such that the effective topology of the processor-interconnect is modified and further degrade overall performance. In particular, the diameter of the topology has significant impact on overall performance. By generalizing the roundabout attack to all of the nodes, the effective diameter of the network can be increased and further degrade overall performance.

I/O Bandwidth: Another shared resource in these multisocket server is the I/O bandwidth and the I/O devices are not necessarily connected to all of the nodes/routers but to a few limited nodes. As a result, for I/O intensive workloads, the I/O bandwidth can be the bottleneck but by modifying the routing table, the hotlink attack (combined with the limited I/O bandwidth) can result in another form of performance attack.

6.2 Alternative Solutions

The different possible solutions for the routing table vulnerability are described in Table 2.

Software Configurable Routing Table: This is the current solution provided in the Hypertransport routing table. While it provides high flexibility and minimal hardware cost, it introduces the vulnerability that we have described in this work.

Hardware Configurable Routing Table: Instead of a software approach, the routing table can still support flexible (configurable) routing table with the hardware responsible for updating the routing table. This would introduce significant hardware complexity but similar approaches have been used in SPIDER SGI router chip [12]. The vulnerability described in this work can be removed at additional hardware cost.

Fixed Routing Table: A simple solution is to provide fixed topology routing table. This removes any routing table vulnerability and is simplest in hardware implementation – and thus, can result in the best router performance as described earlier in Section 2.1. However, this removes all flexibility of the system design and thus, only one topology for a given system can be supported. Any node or link failure would result in breaking the system.

Routing Table Encryption: The fundamental problem with the routing table security vulnerability is that the routing table is mapped to the memory address space and it is visible to the user. To avoid this security vulnerability, the visibility of the routing table can be minimized by encrypting the routing table with existing technologies (e.g., TPM [33]). While this minimizes the vulnerability of the routing table, this can have significant impact on performance of the processor-interconnect. Routing table lookup needs to be done at each router for each packet. Thus, the decryption overhead of the routing table lookup will have significant impact on interconnect latency and have significant impact on overall performance.

7. RELATED WORK

Security of processor- and memory-interconnects: Huang [14] introduces bus probing attacks targeting the processor-interconnects of an XBox gaming console. Huang shows critical data can be extracted using a custom FPGA device because XBox transfer data without encryption. Moscibroda and Mutlu [28] describes a denial-of-service (DoS) attack between applications by exploiting a shared memory controller with First Ready First-Come-First-Served (FR-FCFS) scheduling. They show that an application having streaming access pattern can be slowed down significantly when co-scheduled with another application having random access pattern. The authors propose a fair memory scheduling algorithm to thwart this attack by the scheduler monitoring the relative slowdown of each application.

Although an AMD's Hypertransport-based system is used throughout this paper, Intel-based systems also have similar programmable routing tables for their processor interconnects [16, 17]. Hence, the routing table attack is (very likely to be) applicable to Intel processors as well. Recently, decoy routing is introduced [21, 31] to combat Internet (IP network) censorship. The decoy router circumvents the original routing path to enable the client to reach to a blocked destination server.

The routing table in the network-on-chip (NoC) can also have similar vulnerability. If a simple routing algorithm (such as a dimension-ordered routing [8]) is used, the routing can be implemented in hardware and there would no vulnerability. However, even with such simple routing, some programmability is likely required to provide fault tolerance and expansibility. The NoC from Tilera [3] is one such example as the number of cores can an vary from 9 nodes to over 100 nodes. With a strictly hardware-only approach, a different routing logic needs to be implemented for each network size. Instead, each router has a memory-mapped configurable register [1] that is used to identify the node register (x and the y coordinates) and is used to decide the next hop. Hence, one may exploit this to devise a similar attack.

The large-scale networks designed for supercomputers [4, 32] can also be vulnerable. The routing tables for such networks differ from the processor-interconnect described in this work. For example, the routing table often exists for each port in the large-scale network routers and the initial values of the routing tables are not loaded from the BIOS but computed by the software. The service processor attached to the routers enables access to the routing table, similar

		Vulnerability	Flexibility	Hardware Complexity	Routing Performance
Configurability	S/W Configurable Routing Table	-	+	+	o
	H/W Configurable Routing Table	+	+	-	o
	Fixed Topology Routing Table	+	-	+	+
Visibility	Routing Table Encryption	o	+	+	-

Table 2: Alternative solutions to avoid routing table security vulnerability. + : good, o : moderate, - : poor

to what the device driver provided in the Hypertransport-based system. As a result, the configurablility of the routing tables in these networks is also vulnerable to similar attacks.

Secure network-on-a-chip's (NoCs): Wang and Suh [35] introduce a possibility of timing channel attack for network-on-chip. They show network interference among applications leads to timing variation because applications interfere with each other on the network. Then, a malicious application can infer critical data for target application from the timing information. To remove the timing channel, they suggest a one-way information leak protection mechanism in which applications can be categorized into multiple secure domains according to security level and a router eliminates the network interference from more secure domain to less secure domain with priority-based arbitration. But applications in higher secure domain can have excessive network latency as the traffic increase. Wassel and Gao [36] argue only static scheduling, such as time-division multiplexing (TDM), can guarantee complete non-interference among secure domains. However, time-multiplexed scheduling introduces additional latency since a packet needs to wait for its turn. They introduce SurfNoC to reduce the extra latency in which a packet in a domain is routed in a dimension pipelined manner as if a packet surfs the waves.

Bypassing Chain of Trust and other performance attacks: The routing table attack introduced in this study is not permanent because the routing table registers containing malicious information are volatile, thus the original table is recovered by rebooting. However, many studies [22, 5] introduce the vulnerabilities for the trusted boot process (e.g. bootloader bug, TPM reset, the Root of Trust for Measurement [33] implementation weakness). Exploiting these vulnerabilities, it is feasible to bypass the Chain of Trust [33] and insert the malicious code into the BIOS to make a routing table attack permanent.

There are other performance attacks previously proposed [38, 34, 15]. This work introduces alternative performance attacks, and, to the best of our knowledge, this is one of the first work to demonstrate the feasibility of performance attack by exploiting the vulnerability of the routing table.

8. CONCLUSION

In this work, we illustrated the architectural vulnerability of router designs in modern processor-interconnect technologies such as QPI and HT. Because of the benefit of providing flexibility, a routing table is commonly is used in the router design but we showed how the routing table entries can be maliciously modified to carry out different attacks in a multisocket server system. We described the procedure required to modify the routing table located in the Northbridge and then, described three different attacks – livelock attack, roundabout attack, and hotlink attack. The livelock attack results in a system hang while the other two (roundabout and hotlink) attacks results in a performance degradation. On an AMD-based system, we performed these attacks to show that the system hangs from a livelock and measured the performance degradation for the other attacks. We also proposed alternative solutions to minimize this security vulnerability that have different performance and complexity (cost) trade-off.

Acknowledgments

We would like to thank the reviewers and our shepherd, Jakub Szefer, for their useful comments. This work was supported by the IT R&D program of MSIP/IITP [10041313, UX-oriented Mobile SW Platform] and [KI001810041244, Smart TV 2.0 Software Platform].

9. REFERENCES

[1] Tile Processor User Architecture Manual. `http://www.tilera.com/scm/docs/UG101-User-Architecture-Reference.pdf`.

[2] D. Anderson and J. Trodden. *Hypertransport System Architecture*. Addison-Wesley Professional, 2003.

[3] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 88–598, SF, CA, USA, Feb 2008.

[4] R. Brightwell, K. Pedretti, K. Underwood, and T. Hudson. SeaStar Interconnect: Balanced Bandwidth for Scalable Performance. *Micro, IEEE*, 26(3):41–57, May 2006.

[5] J. Butterworth, C. Kallenberg, X. Kovah, and A. Herzog. BIOS Chronomancy: Fixing the Core Root of Trust for Measurement. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 25–36, Berlin, Germany, Nov 2013.

[6] P. Conway and B. Hughes. The AMD Opteron Northbridge Architecture. *Micro, IEEE*, 27(2):10–21, Mar 2007.

[7] W. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, Mar 1992.

[8] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[9] Dell. PowerEdge R815 Rack Server. `http://www.dell.com/us/business/p/poweredge-r815/pd`.

[10] A. M. Devices. BIOS and Kernel Developer Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors, 2012.

[11] S. Embleton, S. Sparks, and C. C. Zou. SMM rootkit: a new breed of OS independent malware. *Security and Communication Networks*, 6(12):1590–1605, Dec 2013.

[12] M. B. Galles and R. E. Newhall. System and method for network exploration and access, Oct 1997. US Patent 5,682,479.

[13] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sep 2006.

[14] A. Huang. Keeping secrets in hardware: The microsoft xboxtm case study. In *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 213–227. Springer Berlin Heidelberg, Feb 2003.

[15] Q. Huang and P. P. Lee. An Experimental Study of Cascading Performance Interference in a Virtualized Environment. *SIGMETRICS Perform. Eval. Rev.*, 40(4):43–52, Mar 2013.

[16] Intel. 5520 Chipset and Intel 5520 Chipset. `http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/5520-5500-chipset-ioh-datasheet.pdf`, Mar 2009.

[17] Intel. QuickPath Interconnect Overview. `http://www.hotchips.org/wp-content/uploads/hc_archives/hc21/1_sun/HC21.23.1.SystemInterconnectTutorial-Epub/HC21.23.120.Safranek-Intel-QPI.pdf`, Aug 2012. Hotchips.

[18] Intel. Xeon processor e7-8800/4800/2800 v2 product family. `http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e7-v2-datasheet-vol-1.pdf`, Feb 2014.

[19] H. P. Intel. Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification. `http://www.intel.com/content/dam/www/public/us/en/documents/articles/acpi-config-power-interface-spec.pdf`, Dec 2005.

[20] H. Jang, C. Kim, and J. W. Lee. Practical Speculative Parallelization of Variable-length Decompression Algorithms. In *Proceedings of the 14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pages 55–64, Seattle, Washington, USA, Jun 2013.

[21] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy routing: Toward unblockable internet communication. In *USENIX Workshop on Free and Open Communications on the Internet*, SF, CA, Aug 2011.

[22] B. Kauer. OSLO: Improving the security of Trusted Computing. In *Proceedings of the 16 USENIX Security Symposium*, Boston, MA, Aug 2007.

[23] D. A. Keck and P. Devriendt. System and method for programming hypertransport routing tables on multiprocessor systems, Jun 2004. US Patent 20,040,122,973.

[24] W. E. Kühnhauser. Root Kits: An Operating Systems Viewpoint. *SIGOPS Oper. Syst. Rev.*, 38(1):12–23, Jan 2004.

[25] P. Kumar, Y. Pan, J. Kim, G. Memik, and A. Choudhary. Exploring concentration and channel slicing in on-chip network router. In *3rd ACM/IEEE International Symposium on Networks-on-Chip (NoCS)*, pages 276–285, San Diego, CA, May 2009.

[26] C. Lameter. Local and remote memory: Memory in a Linux/NUMA system. In *Linux Symposium*, Ottawa, Ontario Canada, Jul 2006.

[27] A. Lineberry. Malicious Code Injection via/dev/mem. *Black Hat Europe*, Mar 2009.

[28] T. Moscibroda and O. Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-core Systems. In *Proceedings of 16th USENIX Security Symposium*, pages 257–274, Boston, MA, Aug 2007.

[29] B. Mutnury, F. Paglia, J. Mobley, G. Singh, and R. Bellomio. QuickPath Interconnect (QPI) design and analysis in high speed servers. In *IEEE 19th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 265–268, Austin, TX, USA, Oct 2010.

[30] S. Niu, J. Mo, Z. Zhang, and Z. Lv. Overview of Linux Vulnerabilities. In *2nd International Conference on Soft Computing in Information Communication Technology*. Atlantis Press, May 2014.

[31] M. Schuchard, J. Geddes, C. Thompson, and N. Hopper. Routing Around Decoys. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 85–96, Raleigh, North Carolina, USA, Oct 2012.

[32] S. Scott, D. Abts, J. Kim, and W. J. Dally. The BlackWidow High-Radix Clos Network. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture(ISCA)*, pages 16–28, Boston, MA, USA, Jun 2006.

[33] N. Sumrall and M. Novoa. Trusted Computing Group (TCG) and the TPM 1.2 Specification. In *Intel Developer Forum*, 2003.

[34] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift. Resource-freeing Attacks: Improve Your Cloud Performance (at Your Neighbor's Expense). In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 281–292, Raleigh, North Carolina, USA, Oct 2012.

[35] Y. Wang and G. Suh. Efficient Timing Channel Protection for On-Chip Networks. In *6th IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 142–151, Lyngby, Denmark, May 2012.

[36] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood. SurfNoC: A Low Latency and Provably Non-interfering Approach to Secure Networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pages 583–594, Tel-Aviv, Israel, Jun 2013.

[37] R. Wojtczuk. Exploiting large memory management vulnerabilities in Xorg server running on Linux. *Invisible Things Lab*, Aug 2010.

[38] Z. Yang, H. Fang, Y. Wu, C. Li, B. Zhao, and H. Huang. Understanding the effects of hypervisor I/O scheduling for virtual machine performance interference. In *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 34–41, Taipei, Taiwan, Dec 2012.