SecureNoC

(Team Colonel Panic): Xiaoming Guo, Sijia He, Amlan Nayak, Jay Zhang

December 3, 2015

1 Problem Statement

Networks on Chip (NoCs) have been steadily investigated from reliability, efficiency, and performance perspectives.However, little effort has been directed towards the security of complex on-chip networks. These networks are prone to attacks similar to those perpetrated against large scale networks such as datacenters and the internet. If a single core within an NoC is compromised, it can be used as a vehicle to contaminate other cores on the network, or even the entire network itself.

2 Importance

Datacenter security is of utmost importance given the explosive growth of big-data centric applications. The amount of data that a single user generates on a diurnal rhythm is staggering and thus high performance computing with large numbers of integrated cores are now a necessity. Since a vast portion of the data being generated needs to remain private, secure handling of data by data center applications and hardware is paramount. Though mechanisms exist to thwart attacks on a large scale network of servers, little attention has been paid to the security of on-die NoCs.

3 Solution

Two important security vulnerabilities of NoCs are Denial of Service (DoS) attacks and Extraction of Secret Information attacks. Now, in order to ensure secure communication between cores on an NoC, we propose a novel scheme. We augment each router within the network with a traffic monitoring unit and a mechanism to block the communication between local port and router. These modules carry out the following objectives :

- Monitor injection rate in order to detect DoS attacks and halt an attack in progress
- Support secure exchange of priority packets between different routers.

4 Progress

- We have implemented a threshold based DoS attack detection mechanism both in Booksim and in SystemVerilog. The system can detect an abnormally high flit injection rate from any node in a given epoch. Once this abnormal behavior is detected, the system will temporally stop accepting more packets from the compromised node for two epochs. This will allow the "suspiciously compromised" core to re-schedule the packet injection process. After the stalling period, if the core keeps injecting with high rate, it will be shut down permanently, as shown in Figure 1; otherwise, we allow the core to send packets normally, which is illustrated in Figure 2. This mechanism will help eliminating false positive.

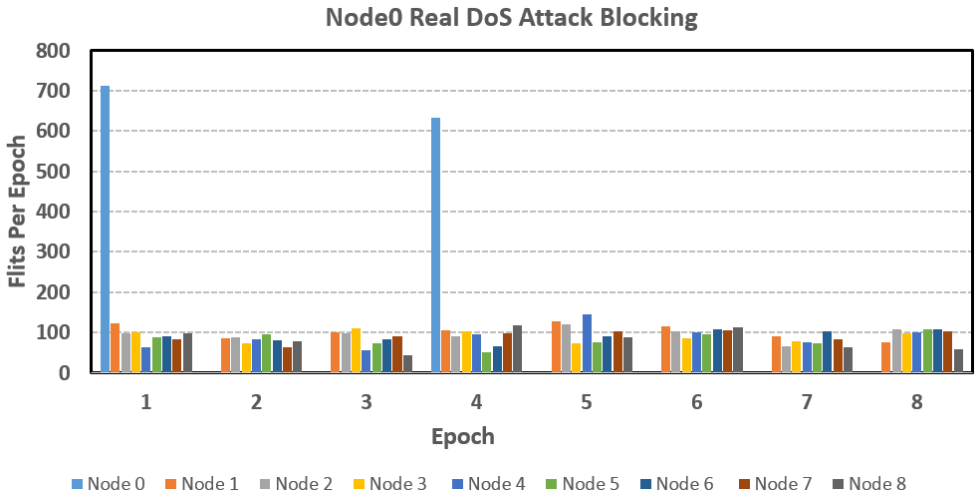

Figure 1: Read attack

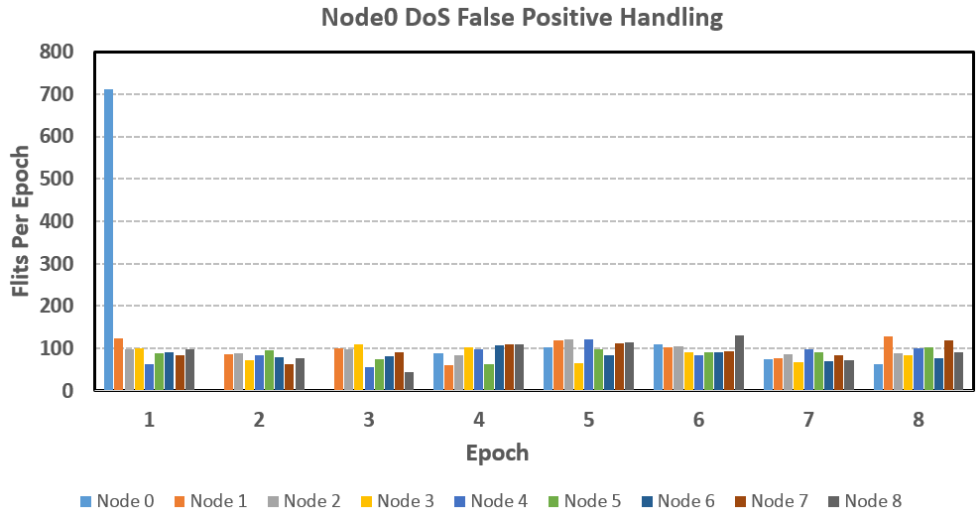

Figure 2: False positive handling

- We justify the threshold based mechanism from the observation that the average packet latency will shoot up once the injection rate for a given node goes beyond a certain threshold value. And this threshold value shows little dependency on the injection rate of other nodes (as long as they are working normally), number of VCs, etc. This discovery is shown in Figure 3, where the threshold is around 0.3 for all different configurations.

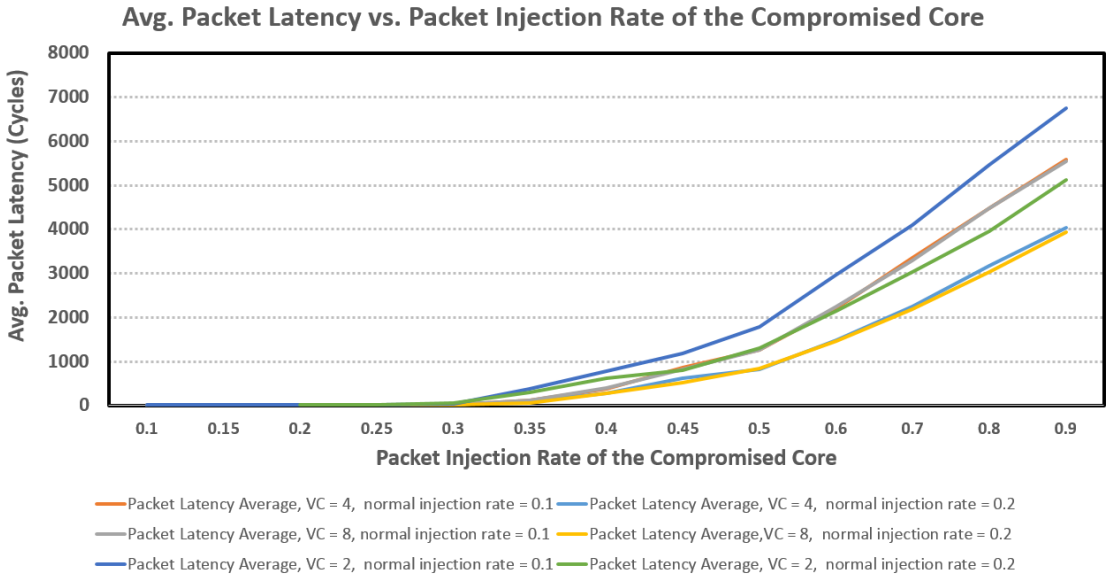

Figure 3: Packet latency vs. injection rate

- We augmented the RTL model from Prof. Dally's group to incorporate our DoS attack detection mechanism. We synthesized our implementation and compared the data with the baseline. The results are given in Table 1.

Table 1: Baseline vs. DoS Detection

	clock period (min)	**area**
Baseline	2.9ns	1896527.2
DoS Detection	2.9ns	1912835.5
Overhead	0	0.86%

- We came up with **S**ecure**P**acket**E**xch**A**nge**R** (**SPEAR**), a mechanism that can support the secure exchange of priority packets between different routers. It can be used to exchange keys for encryption, private information, etc. Figure 4 shows how this mechanism works. Secure packets with a high privilege level (indicated by a bit during packet injection) are assigned strictly to a single VC. VC0 is reserved for such packets. Once the head filt of a privileged packet arrives at a router, the router blocks all its VCs except for reserved VC0. In addition, the router cuts off the connection between local core and the router, effectively preventing the local core from snooping the incoming

secured packets. Once the tail flit of a privileged packet has arrived at a router, all VCs of the router are re-enabled and the connection between local core and the router is re-established.

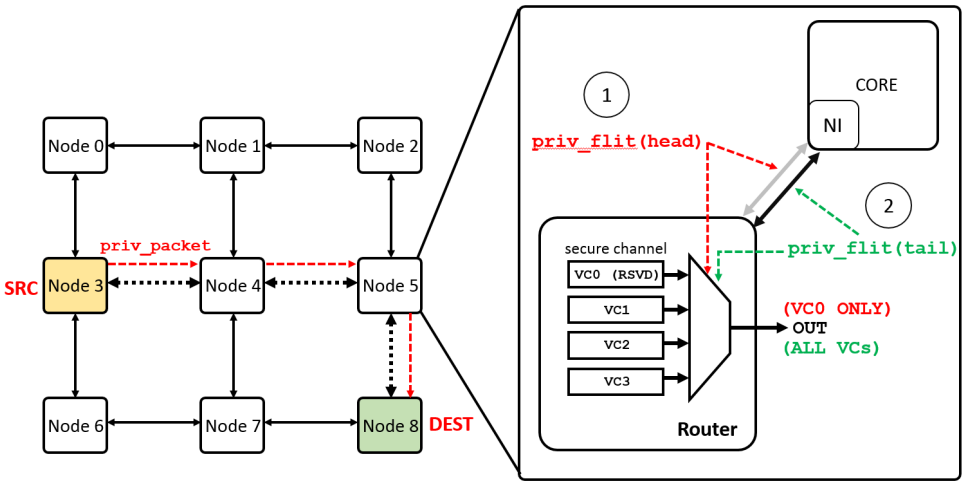

Figure 4: **S**ecure **P**acket **E**xch**A**nge**R** **(SPEAR)**

- Since we stop the injection of new packets for the routers that are transferring privileged packets, the whole system tends to inject fewer packets than normal operation. To study this effect, we set our expected packet injection at 0.15, which is about the maximum injection rate that our system can hold, and then change the percentage of privileged packets to see the actual injection rate of the system. Figure 5 shows that the actual injection rate is not affected as long as the percentage of privileged packets is smaller than 30%.

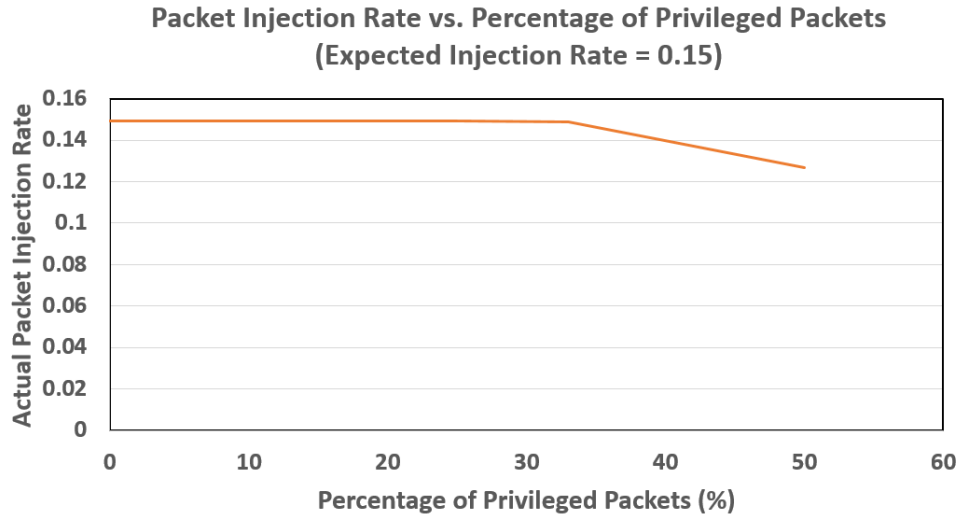

Figure 5: Packet injection rate vs. % of privileged packets @ expected injection rate=0.15

- We evaluated the performance of our approach by monitoring average packet latency in the network with different privileged packet injection frequencies, shown below in Figure 6. When one priority packet appears in every 50 packets or every 100 packets, packet latency in the network is smaller than packet latency with no priority packets. This can be attributed to the fact that priority packets are routed immediately and are able to arrive at their destination without having to wait for allocation. Thus, their latencies are small, leading to an overall smaller average. However, when one in every 10 packets is privileged, then the routers must block their VCs more frequently, resulting in a slow down of the non-privileged packets, which increases the average packet latency in the network. In real applications, we expect the frequency of privileged packets will be less than 10 percent. Therefore, our SPEAR mechanism will not negatively affect performance.

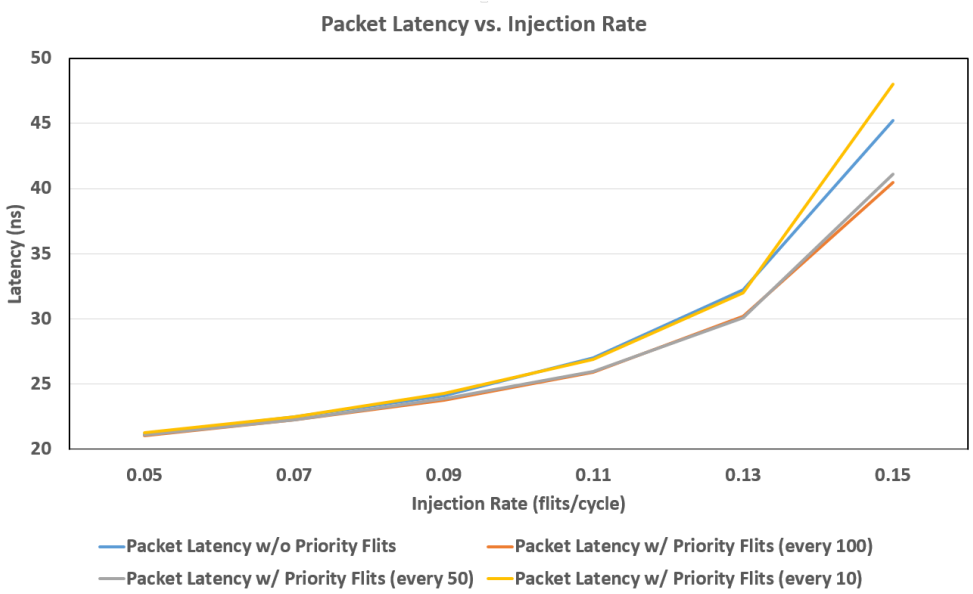

Figure 6: Packet latency with increasing injection rate for different priority packet frequencies

- In Figure 7, we can see that `router_1_0` tries to retire flit 2922 (requesting output 4, which is the local port). However, since there is a privileged packet on-the-fly, the switch allocator keeps granting to VC 0 and avoids sending flit 2922 to the local port. This shows the correct functionality of SPEAR.

```
811 | network_0/router_1_0 | Beginning switch allocation for VC 2 at input 2 (front: 2922).
811 | network_0/router_1_0 |   Requesting output 4.0 (non-spec, pri: 0).
811 | network_0/router_1_0/sw_allocator | Input requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ 3 4 ]  3 -> [ 4 ]  4 -> [ 3 ]  ],
                                          output requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ ]  3 -> [ 2 4 ]  4 -> [ 2 3 ]  ].
811 | network_0/router_1_0/sw_allocator | Input grants = [ 2 -> 3  ], output grants = [ 3 -> 2  ].
811 | network_0/router_1_0 | Switch allocation failed for VC 2 at input 2: Granted to VC 0.
811 | network_0/router_1_0 | Completed switch allocation for VC 2 at input 2 (front: 2922).
811 | network_0/router_1_0 |   No output port allocated.
811 | network_0/router_1_1 | Beginning crossbar traversal for flit 2904 from input 3.0 to output 4.0.
811 | network_0/router_1_1 | Completed crossbar traversal for flit 2904 from input 3.0 to output 4.0.
811 | network_0/router_1_1 | Buffering flit 2904 at output 4.
811 | network_0/router_1_1 | Sending flit 2904 to channel at output 4.
812 | network_0/network_0_fchan_egress4 | Beginning channel traversal for flit 2904 with delay 1.
812 | network_0/router_1_0 | Beginning switch allocation for VC 2 at input 2 (front: 2922).
812 | network_0/router_1_0 |   Requesting output 4.0 (non-spec, pri: 0).
812 | network_0/router_1_0/sw_allocator | Input requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ 3 4 ]  3 -> [ 4 ]  4 -> [ 3 ]  ],
                                          output requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ ]  3 -> [ 2 4 ]  4 -> [ 2 3 ]  ].
812 | network_0/router_1_0/sw_allocator | Input grants = [ 2 -> 3  ], output grants = [ 3 -> 2  ].
812 | network_0/router_1_0 | Switch allocation failed for VC 2 at input 2: Granted to VC 0.
812 | network_0/router_1_0 | Completed switch allocation for VC 2 at input 2 (front: 2922).
812 | network_0/router_1_0 |   No output port allocated.
812 | network_0/network_0_fchan_egress4 | Completed channel traversal for flit 2904.
813 | node4 | Ejecting flit 2904 (packet 653) from VC 1 with privilege 0.
813 | node4 | Injecting credit for VC 1 into subnet 0.
813 | node4 | Retiring flit 2904 (packet 653, src = 1, dest = -1, hops = 2, flat = 14).
813 | node4 | Retiring packet 653 (plat = 19, nlat = 19, frag = 2, src = 1, dest = 4).
813 | network_0/router_1_0 | Beginning switch allocation for VC 2 at input 2 (front: 2922).
813 | network_0/router_1_0 |   Requesting output 4.0 (non-spec, pri: 0).
813 | network_0/router_1_0/sw_allocator | Input requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ 3 4 ]  3 -> [ 4 ]  4 -> [ 3 ]  ],
                                          output requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ ]  3 -> [ 2 4 ]  4 -> [ 2 3 ]  ].
813 | network_0/router_1_0/sw_allocator | Input grants = [ 2 -> 3  ], output grants = [ 3 -> 2  ].
813 | network_0/router_1_0 | Switch allocation failed for VC 2 at input 2: Granted to VC 0.
813 | network_0/router_1_0 | Completed switch allocation for VC 2 at input 2 (front: 2922).
813 | network_0/router_1_0 |   No output port allocated.
814 | network_0/router_1_0 | Beginning switch allocation for VC 2 at input 2 (front: 2922).
814 | network_0/router_1_0 |   Requesting output 4.0 (non-spec, pri: 0).
814 | network_0/router_1_0/sw_allocator | Input requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ 3 4 ]  3 -> [ 4 ]  4 -> [ 3 ]  ],
                                          output requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ ]  3 -> [ 2 4 ]  4 -> [ 2 3 ]  ].
814 | network_0/router_1_0/sw_allocator | Input grants = [ 2 -> 3  ], output grants = [ 3 -> 2  ].
814 | network_0/router_1_0 | Switch allocation failed for VC 2 at input 2: Granted to VC 0.
814 | network_0/router_1_0 | Completed switch allocation for VC 2 at input 2 (front: 2922).
814 | network_0/router_1_0 |   No output port allocated.
815 | network_0/router_1_0 | Beginning switch allocation for VC 2 at input 2 (front: 2922).
815 | network_0/router_1_0 |   Requesting output 4.0 (non-spec, pri: 0).
815 | network_0/router_1_0/sw_allocator | Input requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ 3 4 ]  3 -> [ 4 ]  4 -> [ 3 ]  ],
                                          output requests = [ 0 -> [ ]  1 -> [ ]  2 -> [ ]  3 -> [ 2 4 ]  4 -> [ 2 3 ]  ].
815 | network_0/router_1_0/sw_allocator | Input grants = [ 2 -> 3  ], output grants = [ 3 -> 2  ].
815 | network_0/router_1_0 | Switch allocation failed for VC 2 at input 2: Granted to VC 0.
815 | network_0/router_1_0 | Completed switch allocation for VC 2 at input 2 (front: 2922).
815 | network_0/router_1_0 |   No output port allocated.
```

Figure 7: VC 0 always has priority in switch allocation in privileged mode

5 Issues/Showstoppers

- Does it make sense to measure the average packet latency for only the non-privileged packets?
- We are still debugging the SystemVerilog implementation of SPEAR.