

# Why Can Some Advanced Ethernet NICs Cause Packet Reordering?

Wenji Wu, Phil DeMar, and Matt Crawford

**Abstract**—The Intel Ethernet Flow Director is an advanced network interface card (NIC) technology. It provides the benefits of parallel receive processing in multiprocessing environments and can automatically steer incoming network data to the same core on which its application process resides. However, our analysis and experiments show that Flow Director can cause packet reordering in multiprocessing environments. In this paper, we use a simplified model to analyze why Flow Director can cause packet reordering. Our experiments verify our analysis.

**Index Terms**—Packet Reordering, NIC, TCP, Flow Director.

## I. INTRODUCTION

COMPUTING is now shifting towards multiprocessing. The fundamental goal of multiprocessing is improved performance through the introduction of additional hardware threads, CPUs, or cores (all of which will be referred to as cores for simplicity). The emergence of multiprocessing has brought both opportunities and challenges for TCP/IP performance optimization in such environments. Modern network stacks can exploit parallel cores to allow either message-based parallelism or connection-based parallelism as a means of enhancing performance. To date, major network stacks like Windows and Linux have been redesigned and parallelized to better utilize additional cores. While existing OSes exploit parallelism by allowing multiple threads to carry out network operations concurrently in the kernel, supporting this parallelism carries significant costs, particularly in the context of contention for shared resources, software synchronization, and poor cache efficiencies. However, investigations [1] indicate that CPU core affinity on network processing in multiprocessing environment can significantly reduce contention for shared resources, minimize software synchronization overheads, and enhance cache efficiency.

Core affinity on network processing has the following goals:

(1) *Interrupt affinity*: Network interrupts of the same type should be directed to a single core. Redistributing network interrupts in either a random or round-robin fashion to different cores has undesirable side effects. (2) *Flow affinity*: Packets belonging to a specific TCP flow should be processed by the same core. TCP has a large and frequently accessed state that must be shared and protected when packets from the same connection are processed. Ensuring that all packets in a TCP flow are processed by a single core reduces contention for shared resources, minimizes software synchronization, and

enhances cache efficiency. (3) *Network data affinity*: Incoming network data should be steered to the same core on which its application process resides. This is becoming more important with the advent of Direct Cache Access (DCA). Network data affinity maximizes cache efficiency and reduces core-to-core synchronization.

The emergence of parallel network stacks and the necessity of core affinity on network processing in multiprocessing environment require new NIC designs. A NIC should not only provide mechanisms to allow parallel receive processing to better utilize parallel network stacks, but also to facilitate core affinity on network processing in multiprocessing environments. Receive Side Scaling (RSS) [2] is a NIC technology that steps toward that direction. It supports multiple receive queues and integrates a hashing function in the NIC. The NIC computes a hash value for each incoming packet. Based on hash values, NIC assigns packets of the same data flow to a single queue and evenly distributes traffic flows across queues. With Message Signal Interrupt (MSI/MSI-X) support, each receive queue is assigned a dedicated interrupt and RSS steers interrupts on a per-queue basis. RSS provides the benefits of parallel receive processing in multiprocessing environments. OSes like Windows and Linux now support interrupt affinity. When an RSS receive queue is tied to a specific core, packets from the same flow are steered to that core (Flow pinning). This ensures flow affinity on most OSes. However, RSS has a limitation: it cannot steer incoming network data to the same core where its application process resides. The reason is simple: the existing RSS-enabled NICs do not maintain the relationship “Traffic Flows→Network applications→Cores” in the NIC. Since network applications run on cores, we simply put it as “Traffic Flows→Cores (Applications).” This is symptomatic of a broader disconnect between existing software architecture and multicore hardware. With OSes like Windows and Linux, if an application is running on one core, while RSS has scheduled received traffic to be processed on a different core, poor cache efficiency and significant core-to-core synchronization overheads will result. The overall system efficiency may be severely degraded. To remedy the RSS limitation, the Intel Ethernet Flow Director technology [3] has been introduced. The basic idea is simple: Flow Director maintains the relationship “Traffic Flows→Cores (Applications)” in the NIC. OSes are correspondingly enhanced to support such capability. Flow Director not only provides the benefits of parallel receive processing, it also can automatically steer packets of a specific data flow to the same core on which its application process resides, which facilitates core affinity on network processing. However, our analysis and experiments

Manuscript received August 16, 2010. The associate editor coordinating the review of this letter and approving it for publication was Prof. Vinod Vokkarane.

The authors are with the Fermi National Accelerator Laboratory, Batavia, IL 60510, USA (e-mail: {wenji, demar, crawdad}@fnal.gov).

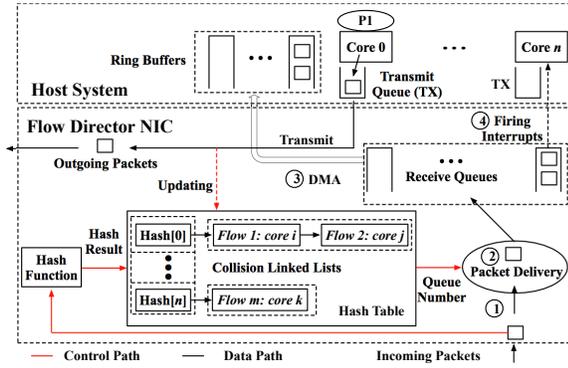


Fig. 1. Flow Director Mechanism

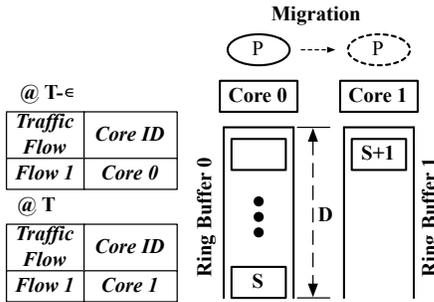


Fig. 2. Packet Reordering Analysis Model

show that Flow Director can cause packet reordering in multiprocessing environments. TCP performance suffers in the event of severe packet reordering. In this paper, we use a simplified model to analyze why Flow Director can cause packet reordering. Our experiments verify our analysis.

## II. WHY CAN FLOW DIRECTOR CAUSE PACKET REORDERING?

Intel Ethernet Flow Director supports multiple receive queues in the NIC, up to the number of cores in the system. With MSI/MSI-X and Flow-Pinning support, each receive queue has a dedicated interrupt and is tied to a specific core; each core in the system is assigned a specific receive queue. The NIC device driver allocates and maintains a ring buffer in system memory for each receive queue. For packet reception, a ring buffer must be initialized and pre-allocated with empty packet buffers. The ring buffer size is device and driver-dependent. For transport-layer traffic, Flow Director maintains a “Traffic Flow→Core” table with a single entry per flow. Each entry tracks the core to which a flow should be assigned. Flow Director makes use of the 5-tuple  $\{src\_addr, dst\_addr, protocol, src\_port, dst\_port\}$  in the receive direction to identify a flow in the table, and a core ID to specify the core to which the flow should be assigned. Entries within the “Traffic Flow→Core” table are updated by outgoing packets. To support Flow Director, the OS must be multiple TX queue capable [4]. For an outgoing transport-layer packet, the OS records its processing core ID and passes it to the NIC to generate or update the corresponding entry

within the table. The passed core ID advertises the core on which a network application resides. For example, a network application that resides on core  $i$  sends an outgoing packet with the header  $\{(src\_addr: x), (dst\_addr: y), (protocol: z), (src\_port: p), (dst\_port: q)\}$ . The OS records its processing core ID  $i$  and passes it to the NIC. The NIC generates or updates the corresponding flow entry in the table as  $\{(src\_addr: y), (dst\_addr: x), (protocol: z), (src\_port: q), (dst\_port: p), (core\_id: i)\}$ . A flow entry is deleted from the table after a configurable period of time has elapsed without traffic.

Figure 1 illustrates packet receive-processing for transport-layer packets with Flow Director. (1) When incoming packets arrive, the hash function is applied to the header to produce a hash result. Based on the hash result, the NIC identifies the core and hence, the associated receive queue. (2) The NIC assigns the incoming packets to the corresponding receive queues. (3) The NIC deposits via direct memory access (DMA) the received packets into the corresponding ring buffers in system memory. (4) The NIC sends interrupts to the cores associated with the non-empty queues. Subsequently, the cores respond to the network interrupts and process the received packets up through the network stack from the corresponding ring buffers one by one.

Flow Director provides the benefits of parallel receive processing; it also facilitates core affinity on network processing. However, our analysis shows that Flow Director can cause packet reordering. TCP performs poorly with severe packet reordering [5]. We use a simplified model to analyze why Flow Director can cause packet reordering.

As shown in Figure 2, at time  $T - \epsilon$ , *Flow 1*'s flow entry maps to Core 0 in the “Traffic Flow→Core” table. At this instant, packet  $S$  of *Flow 1* arrives; based on the table, it is assigned to Core 0. At time  $T$ , due to process migration, *Flow 1*'s flow entry is updated and maps to Core 1. At  $T + \epsilon$ , Packet  $S + 1$  of *Flow 1* arrives and is assigned to the new core, namely Core 1. As described above, after assigning received packets to the corresponding receive queues, the NIC copies them into system memory via DMA and fires network interrupts, if necessary. When a core responds to a network interrupt, it processes received packets up through the network stack from the corresponding ring buffer one by one. In our case, Core 0 processes packet  $S$  from Ring Buffer 0, and Core 1 services packet  $S + 1$  from Ring Buffer 1. Let  $T_{svc}(S)$  and  $T_{svc}(S + 1)$  be the times at which the network stack starts to service packets  $S$  and  $S + 1$ , respectively. If  $T_{svc}(S) > T_{svc}(S + 1)$ , the network stack would receive packet  $S + 1$  earlier than packet  $S$ , resulting in packet reordering. Let  $D$  be the ring buffer size and let the network stacks packet service rate be  $R_{svc}$  (packets/s). Assume there are  $n$  packets ahead of  $S$  in Ring Buffer 0 and  $m$  packets ahead of  $S + 1$  in Ring Buffer 1. Then  $T_{svc}(S) = T - \epsilon + n/R_{svc}$  and  $T_{svc}(S + 1) = T + \epsilon + m/R_{svc}$ .

If  $\epsilon$  is small and  $n > m$ , the condition of  $T_{svc}(S) > T_{svc}(S + 1)$  would easily hold and lead to packet reordering. Since the ring buffer size is  $D$ , the worst case is  $n = D - 1$  and  $m = 0$ , giving  $T_{svc}(S) = T - \epsilon + (D - 1)/R_{svc}$  and  $T_{svc}(S + 1) = T + \epsilon$ . The ring buffer size  $D$  is a design parameter for the NIC and driver. For example, the Myricom

10Gb/s NIC has  $D = 512$ , and Intels 1Gb/s NIC has  $D = 256$ .

In a multicore system, a general-purpose OS scheduler tries to use all core resources in parallel as much as possible, distributing and adjusting the load among the cores. Process migration across cores occurs frequently. Flow Director can cause packet reordering in these conditions.

### III. EXPERIMENTS

To validate our analysis, we ran data transmission experiments over an isolated network. A sender was directly connected to a receiver via a physical 10Gbps link.

**Sender:** Dell R-805; 2 Quad Core AMD Opteron 2346HE, 1.8GHz; Myricom 10G Ethernet NIC; Linux 2.6.28.

**Receiver:** SuperMicro Server; 2 Intel Xeon CPUs (4 cores total), 2.66GHz; Intel X520 Server Adapter with Flow Director enabled (configured with suggested default parameters [4]), 10Gbps, MTU 1500; Linux 2.6.34, Multiple TX Queue Capable.

In our experiments, iperf is used to send  $n$  parallel TCP streams from sender to receiver for 100 seconds. Iperf is a multi-threaded network application. With multiple parallel TCP data streams, a dedicated child thread is spawned for each stream. In our first experiment, iperf was not pinned to a specific core in the receiver. In our second experiment, we pinned iperfs with different ports to core 0, 1, 2, and 3, respectively, in the receiver and each core is sent with  $n/4$  TCP streams. In both experiments, Linux was configured to run in multicore peak performance mode. The receiver was instrumented to record out-of-order packets and we calculated packet reordering ratios. We also calculated the overall throughputs. The experiment results with a 95% confidence interval are shown in Table I and II.

In the first experiment, the rate of packet reordering is significant. At  $n = 200$ , packet reordering ratio reaches as high as 0.897%. The experiment results validated our analysis. In the first experiments, since iperf is not pinned to a specific core, when the scheduler tries to distribute the load equally among the cores, it will lead to frequent process migration. Flow Director can easily cause packet reordering in these conditions. In the second experiment, no packet reordering was observed. Each flow was always attached to a core, with no process migration. As the condition  $T_{svc}(S) > T_{svc}(S+1)$  was not met, Flow Director did not cause packet reordering. Therefore, reducing or preventing process migration is a way to help Flow director to cause less or no packet reordering. Process migration in the first experiment is not only degrading performance, but also causing packet reordering. In the second experiment, since each flow was tied to a core, there was strict core affinity on network processing (interrupt, flow, and network data affinity). Also, because there was no packet reordering, the negative effects of reordering [5] do not exist. Therefore, the throughputs in the second experiment are higher than those in the first experiment. In the experiments, we noticed that it is the Intel X520 10Gb NIC, not the CPUs, that limits the overall throughput. The Intel X520 10Gb NIC cannot reach 10Gbps with the flow director feature enabled. Otherwise, the throughputs in the second experiments would be even higher than those in the first experiments.

TABLE I  
PACKET REORDERING RATIO

$n$	First	Second
100	0.705% $\pm$ 0.042%	0
200	0.897% $\pm$ 0.038%	0
500	0.635% $\pm$ 0.154%	0
1000	0.409% $\pm$ 0.009%	0
2000	0.129% $\pm$ 0.003%	0

TABLE II  
THROUGHPUTS

$n$	First(Gb/s)	Second(Gb/s)	Improvement
100	6.877 $\pm$ 0.006	6.941 $\pm$ 0.004	0.90% $\pm$ 0.097%
200	6.649 $\pm$ 0.006	6.791 $\pm$ 0.003	2.14% $\pm$ 0.120%
500	6.202 $\pm$ 0.003	6.445 $\pm$ 0.002	3.91% $\pm$ 0.012%
1000	5.943 $\pm$ 0.005	6.172 $\pm$ 0.004	3.86% $\pm$ 0.154%
2000	5.898 $\pm$ 0.008	5.909 $\pm$ 0.002	0.20% $\pm$ 0.010%

### IV. CONCLUSION & DISCUSSION

We use a simplified model to analyze why Flow Director can cause packet reordering in multiprocessing environments. Our experiments validated our analysis. The finding of packet reordering and our model can be applied to any other software that directs packets to the core where the application resides in a multiprocessing environment. The root cause of the packet reordering is that Flow Director lacks mechanisms to ensure the satisfaction of the inequality constraint  $T_{svc}(S) < T_{svc}(S+1)$  when it steers packets across cores. Reducing or preventing process migration is a way to help Flow director to cause less or no packet reordering. However, this is not a true solution for the reordering problem. We believe that a better solution is to add extra mechanisms within the NIC to ensure the satisfaction of the ordering constraint  $T_{svc}(S) < T_{svc}(S+1)$  when packets are steered across cores.

### REFERENCES

- [1] J. D. Salehi, J. F. Kurose, and D. Towsley, "The effectiveness of affinity-based scheduling in multiprocessor network protocol processing (extended version)," *IEEE/ACM Trans. Netw.*, vol. 4, pp. 516–530, August 1996.
- [2] Microsoft Corporation. (2008, Nov.) Receive-side scaling enhancements in windows server 2008. [Online]. Available: [http://download.microsoft.com/download/a/d/f/adf1347d-08dc-41a4-9084-623b1194d4b2/RSS\\_Server2008.docx](http://download.microsoft.com/download/a/d/f/adf1347d-08dc-41a4-9084-623b1194d4b2/RSS_Server2008.docx)
- [3] Intel Corporation. (2010, Nov.) Intel 82599 10gbe controller datasheet. [Online]. Available: [http://download.intel.com/design/network/datashts/82599\\_datasheet.pdf](http://download.intel.com/design/network/datashts/82599_datasheet.pdf)
- [4] ——. (2010, Nov.) Ixgbe device driver readme. [Online]. Available: <http://downloadmirror.intel.com/14687/eng/README.txt>
- [5] W. Wu, P. Demar, and M. Crawford, "Sorting reordered packets with interrupt coalescing," *Computer Networks*, vol. 53, no. 15, pp. 2646 – 2662, 2009.