

# Evaluating the Suitability of Server Network Cards for Software Routers

Maziar Manesh<sup>§</sup> Katerina Argyraki<sup>‡</sup> Mihai Dobrescu<sup>‡</sup> Norbert Egi<sup>\*</sup>  
Kevin Fall<sup>§</sup> Gianluca Iannaccone<sup>§</sup> Eddie Kohler<sup>†</sup> Sylvia Ratnasamy<sup>§</sup>  
<sup>‡</sup>EPFL, <sup>†</sup>UCLA, <sup>§</sup>Intel Labs Berkeley, <sup>\*</sup>Lancaster Univ.

## 1. INTRODUCTION

The advent of multicore CPUs has led to renewed interest in software routers built from commodity PC hardware[6, 5, 8, 7, 3, 4]. The typical approach to scaling network processing in these systems is to distribute packets, or rather flows of packets, across multiple cores that process them in parallel. However, the traffic arriving (departing) on an incoming (outgoing) link at a router is inherently serial and hence we need a mechanism that appropriately demultiplexes (multiplexes) the traffic between a serial link and a set of cores. I.e., multiple cores can parallelize the processing of a traffic stream but to fully exploit the parallelism due to multiple cores we must first be able to parallelize the *delivery* of packets to and from cores. Moreover, this parallelization must be achieved in a manner that is: (i) *efficient*, ensuring that the splitting/merging of traffic isn't the bottleneck along a packet's processing path and (ii) *well balanced* such that input processing load can be well balanced across available cores for a range of input traffic workloads (*e.g.*, diverse flow counts, flow sizes, packet processing applications, and so forth). If either of these requirements is not met, the parallelism due to multiple cores might well be moot. In other words, achieving parallelism in packet delivery is critical for any software routing system that exploits multiple cores.

Recent efforts point to modern server network interface cards (NICs) as offering the required mux/demux capability through new hardware classification features[5, 8, 3, 7]. In a nutshell,<sup>1</sup> these NICs offer the option of classifying packets that arrive at a port into one of mul-

<sup>1</sup>We discuss current NIC architectures and features in greater detail in Section 3.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM PRESTO 2010, November 30, 2010, Philadelphia, USA.

Copyright 2010 ACM 978-1-4503-0467-2/10/11 ...\$10.00.

iple hardware queues on the NIC and this set of queues can then be partitioned across multiple cores; each core thus processes the subset of traffic that arrives at its assigned queues. Effectively, this NIC-level classification capability serves to parallelize the path from a NIC to the cores, and vice versa. Building on this observation, recent efforts leverage “multi-Q” NICs in their prototype systems[5, 8, 3, 6] and show that enabling NIC-level classification significantly improves a server's packet processing capability ([5] reports a 3.3x increase in forwarding throughput through NIC-level classification).

However there has been little evaluation to stress test the *extent* to which modern NICs and their classification capabilities match the requirements of software routing from the standpoint of both, performance and functionality. For example, as we discuss in the following section, there has been little evaluation of the performance impact of scaling NIC-based classification to large numbers of queues (*i.e.*, a high mux/demux fanout) or whether current classification options can balance load under varied input traffic workloads. It is perhaps worth emphasizing that high-speed packet processing is not<sup>2</sup> a common server application and hence testing server NICs under routing workloads is not, to our knowledge, on the “routine” checklist for NIC designers; as such, it isn't obvious that server NICs would fare well when scrutinized through the lens of a router designer.

This paper takes a first step towards such an evaluation. We focus on the latest generation of widely-used server NICs and experimentally compare its performance characteristics to that of an ideal “parallel NIC” and a “serial only” NIC. We show that although commodity NICs do improve on serial-only NICs (with 3x higher throughput on typical workloads) they lag an ideal parallel NIC (achieving 30% lower throughput than an ideal parallel NIC). We find similar gaps in the classification features these NICs offer. We thus conclude with recommendations for NIC modifications that we believe would improve their suitability for software

<sup>2</sup>yet!

routers.

The remainder of this paper is organized as follows: in Section 2, we define the problem of parallelizing packet delivery in software routers and the goals for an ideal parallel NIC. Section 3 explains the methodology for our experimental evaluation which we present in Section 4. We conclude in Section 5.

## 2. PROBLEM DEFINITION

The high level question we aim to address in this paper is quite simple: what is the most efficient way to split  $R$  bps of incoming traffic across  $n$  cores?

Despite its simplicity, addressing this question represents a key challenge for the feasibility of high speed software routers. In fact, spreading the traffic load across multiple cores is typically unavoidable since a single core may not be able to handle the line rate of one port. To date, the best reported per core forwarding rate falls short of 10Gbps in the case of basic IPv4 routing (approx. 5 Gbps/core with 64B packets according to [8]). Given that the current trends for processor architectures is to increase the total number of cores per processor rather than the single core performance, it is clear that traffic must be split across multiple cores to handle 10Gbps or above rates or any kind of more advanced packet processing.

Once we split traffic across cores, we still need to specify what properties make a design “efficient”. To a first approximation, efficiency can be characterized along two dimensions: *performance* and *control*. The ideal performance is such that if one core is capable of processing  $X$  bps, then  $n$  cores should handle  $nX$  bps. In reality, there are many different reasons why this ideal scaling may not be achieved in general (e.g., contention for memory, cache or shared data structures). However, in this paper we are interested in understanding the overhead, if any, due just to mux/demux traffic from the network interface to the cores. As we will see in Section 4 the challenge is in the careful design of packet processing experiments that allow us to isolate the impact of the NIC on overall system performance.

By “control” we mean the ability to define which core receives what subset of packets. This is important in the context of IP forwarding to avoid introducing packet reordering, for example. Other forms of fine grained control on the manner in which traffic is split may include assigning priorities to traffic streams or making sure that a given core processes traffic of a specific customer.

In this paper we investigate two approaches available today to split traffic: the first relies on hardware support in the form of independent queues and a packet classification engine on the network interface; the second is instead implemented in software with a single core dedicated to the task of splitting traffic across the

other cores.

## 3. OVERVIEW OF CURRENT NETWORK INTERFACES

To evaluate and understand the performance of current off-the-shelf server NICs, we consider a simplified view of the server components. Figure 1 shows a high level view of the components that participate in packet processing. Current high-end Intel Ethernet NICs<sup>3</sup> use a single controller that can handle two 10Gbps Ethernet ports. The NIC communicates to the rest of the system via the I/O Hub that terminates the PCIe lanes (8 lanes are required for this type of network card). Finally the I/O Hub transfers the packets to memory using Intel Quick Path Interconnect links (or similar technologies for other processor manufacturers – e.g. Hypertransport in AMD systems).

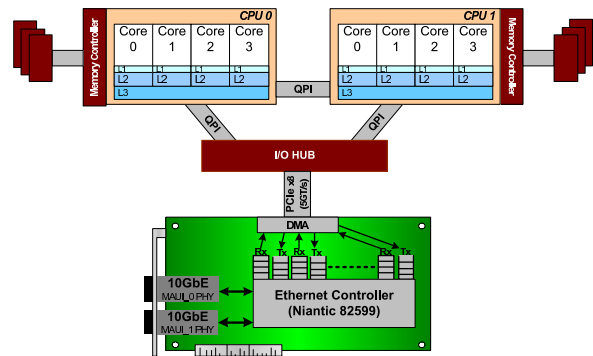


Figure 1: Server components

To avoid contention when multiple cores access the same 10Gbps port, server NICs can give the illusion of a dedicated 10Gbps port per core using multiple hardware queues. This way, each core accesses the hardware queue independently (i.e. no need for any synchronization with other cores) while the NIC controller is in charge of classifying packets and placing them in one of the many queues.

Today’s NIC provide a small set of classification algorithms on the receive side [2]:

- **hash-based** (such as Receive Side Scaling, RSS) where a hash function is applied to the header fields of the incoming packets and its output is used to select one of the hardware queues. This is used to make sure each queue receives an equal portion of the incoming traffic.
- **address-based** (such as VMDq) where each queue is associated with a different ethernet address and

<sup>3</sup>All the results and discussions on this paper refer to the Intel 10Gbps 82599EB controller [2] (codename “Niantic”).

the NIC accepts packets destined for any of the ethernet addresses. This classification is commonly used in conjunction with virtualization to give each guest VM the abstraction of a dedicated interface.

- **flow-based** (e.g., Flow Director) where each hardware queue is associated to a flow that can be defined using any sequence of bytes in the packet header. This allows for a very flexible flow definition and can support several thousands of concurrent flows.

For the purpose of this paper we will look only at the first classification method (hash-based) where the hash is computed on the classical 5-tuple made of protocol, source and destination IP addresses and port numbers if present (this is the same approach followed in [5] and [8]). The reason behind this decision is that the address-based and flow-based classification algorithms are not well suited to routing workloads as they both limit the total number of flows that are allowed in the system.

On the transmit side, the NIC is in charge of merging the traffic from several hardware queues to the FIFO buffer that feeds into the ethernet transmitter. The hardware queues are served in a round-robin fashion with the possibility of rate limiting each queue independently or assigning traffic priorities following the IEEE 802.1p standard [1].

## 4. EVALUATION

In order to understand how off-the-shelf network cards can support the requirements of high speed software routers we focus on two questions:

1. How well does the packet forwarding performance scale *with the number of cores*?
2. How well does the packet forwarding performance scale *with the number of queues*?

Addressing the first question is really akin to a classical black-box approach to system performance evaluation. As such, it measures the NIC and system performance as well as the overhead introduced by the software. The second question instead attempts at isolating the impact of multiqueue support in the NIC from that of other system components. Our intent there is to directly measure the performance degradation due to multiplexing traffic across cores.

### 4.1 Experimental setup

For our study, we chose an Intel Xeon 5560 (Nehalem) server with two sockets, each with four 2.8GHz cores and an 8MB L3 cache. The system uses one 10Gbps NIC with two ports installed in a PCIe2.0 x8 slot. Our server runs Linux 2.6.24 with Click[9] in polling mode—i.e., the CPUs poll for incoming packets rather than being interrupted.

The input traffic is made of 64 byte back-to-back packets—the worst-case traffic scenario and it is generated by a separate 8-core server. We consider only one type of simple packet processing that we call *minimal forwarding*. With minimal forwarding traffic arriving at one of the NIC ports is always forwarded to the other physical port of the same NIC. We chose this simple test because we are interested in measuring the raw performance of the network card without introducing any side effects due to the packet processing software.

With this setup, our primary performance metric is the maximum forwarding rate we achieve, reported in terms of packets-per-second (pps).

### 4.2 Forwarding Performance

To address the first question, we performed a set of experiments using a single 10GbE port. These experiments involve increasing both the numbers of CPU cores dedicated to packet processing as well as the number of hardware queues allocated on the NIC. In all experiments the number of cores is equal to the number of queues and each core has a dedicated receive and transmit queue pair. This guarantees that there is no contention across cores or shared memory data structures. This setup is the one commonly recommended when using modern server NICs.

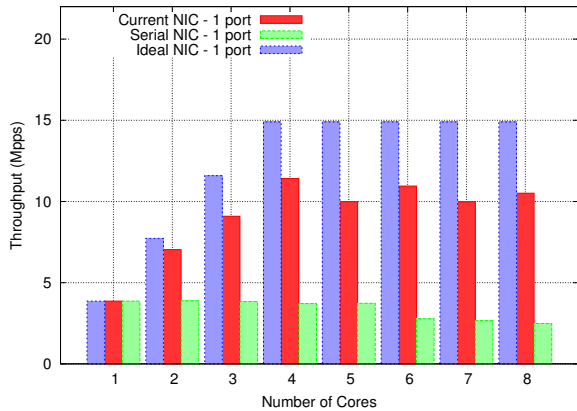
We then compare the performance achieved using multiple hardware queues against two natural alternatives: *i*) a purely software solution (“serial NIC”) that assumes that the NIC has no multiple queue support; *ii*) an “ideal NIC” scenario where each core has completely dedicated access to the 10GE port (i.e., no traffic splitting/merging necessary). This is an idealized view of the system as it assumes there is no overhead in running an application in a multicore system compared to a single core server.

The serial NIC does away with the use of the NIC hardware queues and performs traffic splitting and merging in software. In all such experiments, one core is in charge of the receive path, one core is in charge of the transmit path and the remaining cores perform the actual packet processing. In the case of one core experiments, the same core handles both the receive and transmit side. The intent here is to understand the role hardware queues in modern NICs play in high speed software routers. We call this set of experiments “serial NIC” given that the NIC behaves as a simple FIFO packet buffer.

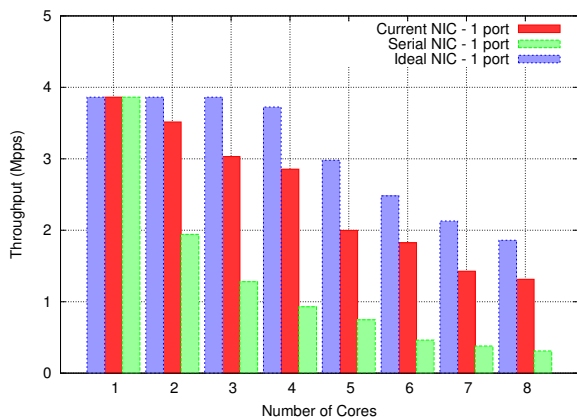
The “ideal NIC” scenarios – where no hardware or software overhead in splitting traffic is assumed – is also drawn for reference purposes only. The performance of the ideal NIC is a computed, not measured, value—we consider the maximum forwarding rate the system can reach with just one core and then multiply that rate by the number of cores until we reach the line rate for a

10Gbps Ethernet port, i.e., approximately 14.9 Mpps.<sup>4</sup>

Figure 2 shows the aggregate forwarding rate in million of packets per second with an increasing number of cores. Figure 3 plots the same data normalized by the number of cores.



**Figure 2: Aggregate forwarding rate with increasing numbers of cores and queues**



**Figure 3: Per-core forwarding rate with increasing numbers of cores and queues**

From the figures we can derive two conclusions. On the one hand, multiple hardware queue support in the NICs is a clear improvement over software-only solutions. The performance of current NICs is more than 3x higher than software-only serial NICs.

On the other hand, the overhead of current NICs in handling multiple queues is significant. The per-core performance of current NICs is 30% lower than what

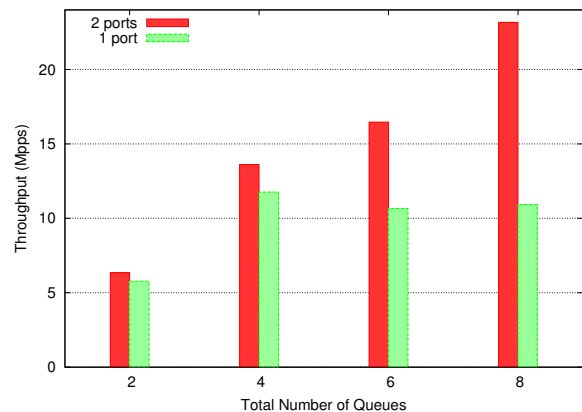
<sup>4</sup>Note that the maximum packet rate on a 10Gbps Ethernet port is not 19.5 Mpps (i.e.,  $10^{10}/(8 * 64)$ ) because the standard requires a 20 byte long spacing between back-to-back packets.

an ideal NIC would predict. Yet, these experiments do not allow us to identify the source of the overhead. It could be in the NIC itself or in other system resources (e.g., memory, PCI bridge, etc.) that are shared across the cores.

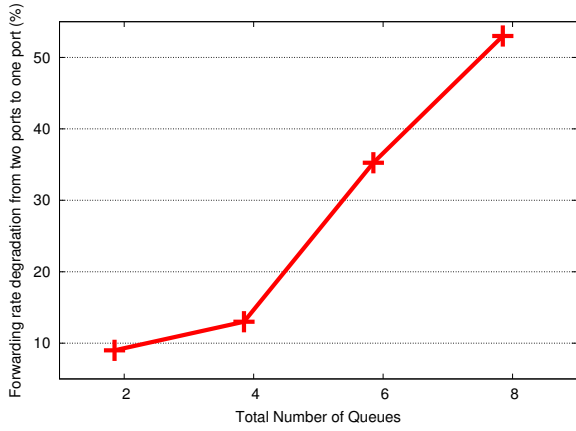
In order to isolate the impact of the NIC alone, we should measure the packet forwarding performance when all system components are kept fixed (i.e. number of cores, PCIe lanes, controllers, etc.) and the only change is in the number of hardware queues. Specifically, this means that we would ideally use only one controller and one PCIe slot (8 lanes) while varying the mapping between hardware queues and physical 10Gbps Ethernet ports (e.g., 8 queues on 1 port, 4 queues on 2 ports, 2 queues on 4 ports, etc.)

Unfortunately current NIC hardware does not provide enough flexibility to explore the entire parameter space. The Niantic controller can handle only two physical 10Gbps ports. That allows us to experiment only with 8 queues or 4 queues per port (so that the number of cores remains constant).

To overcome this limitation and still be able to understand the impact of a growing number of hardware queues, we compare the case of using one physical port and  $Q$  queues (where  $Q$  ranges from 2 to 8) to the case of using two physical ports and  $Q/2$  hardware queues per port. In both cases the number of cores is constant and equal to  $Q$ , and only one ethernet controller is used. We will then measure the performance degradation (if any) in using a larger number of queues per physical port — if hardware queues were to present no overhead, there should be no performance degradation between using two ports and  $Q/2$  queues per port versus one port and  $Q$  queues.



**Figure 4: Comparing the aggregate forwarding rate varying the number of hardware queues per port. The number of cores is equal to the total number of queues so that one core always has one and only one dedicated hardware queue.**



**Figure 5: Ratio of the forwarding rate varying the number of hardware queues per port. The number of cores is equal to the total number of queues so that one core always has one and only one dedicated hardware queue.**

Figure 4 and 5 show the forwarding performance with varying the number of hardware queues per port ( $Q = 2, 4, 6, 8$ ). The graphs clearly indicate that a smaller number of queues lead to better performance. The forwarding performance scales almost perfectly across multiple cores when using two ports: 6.3 Mpps with 2 cores, 23.2 Mpps with 8 cores. Using a single 10G port (but the same controller) the forwarding rate peaks at 11.7 Mpps with 4 queues (and 4 cores) and then degrades a little (by less than 1 Mpps) as the number of queues grows.

We can derive a few conclusions from these two figures:

- the NIC controller, the cores and the PCIe 8-lane slot can handle 23.2 Mpps: well above the line rate of a 10 Gbps interface. The performance bottleneck that we observed in Figure 2 is therefore not within any of those components.
- each 10Gbps physical port cannot run at the maximum packet rate. The maximum rate we have been able to measure in any experiment is indeed 11.7 Mpps below 14.9 Mpps, the maximum packet rate of a 10 Gbps Ethernet port. Two physical ports can run at about 23.4 Mpps (i.e. 2x 11.7 Mpps).
- there is an overhead in adding more hardware queues as highlighted by Figure 5 for 2 and 4 hardware queues. However, this overhead is relatively small leading to approx 10% loss in forwarding rate.

In summary, we have been able to measure the overhead of multiple queues and isolate where the bottleneck is in the system: between the NIC controller and

the physical port. At this time we do not have access to the detailed design of the NIC controller to PHY connection but we conjecture that handling of the FIFO buffer that accommodates the packets right off the wire could be the reason for the loss in performance.

It is important to note that this could also be a deliberate design decision given that the NIC under study is designed for server platforms. Its design is evaluated by how efficiently it can *terminate* TCP or UDP network streams. In our experiments we have always used a worst case workload (64 byte long, back to back packets, little or no payload). That is a traditional benchmark for routing but of very little interest for traditional server applications. Reaching 11.7 Mpps means that at an average packet size of 86 bytes this system could fully saturate a 10Gbps Ethernet port. However, following the same rationale, it is foreseeable that packet forwarding may become a workload of interest for NIC designers as software routers become a reality in the telecommunication industry.

### 4.3 Functionality

We focus now on whether commodity NICs are well suited for software routing at the *functional* level. We start by summarizing the functionality NICs support today and then introduce a “wishlist” of new features we believe would further improve the suitability of these NICs for packet processing.

#### *NIC functionality today.*

Beyond multiple hardware queues, current NICs implement a plethora of mechanisms that are specifically designed to offload the cores of some of the most basic packet processing operations. As we said before, however, most of these mechanisms are intended for server applications and as such focus on TCP stream processing (e.g., TCP Checksum offload, TCP Receive Side Coalescing, TCP timers, etc.) or for virtualization (e.g., VT-d, virtual machine device queues, etc.).

A few features that are directly applicable to software routers include: (i) handling a large number of hardware queues (up to 128) to spread traffic across many cores; (ii) flow-level filters (hash-based or up to 8K flows) that can be used to limit packet reordering; (iii) VLAN support; (iv) rate limiting and priorities (compliant with IEEE 802.1p); (v) IPsec encryption and packet encapsulation.

What follows is a simple list of additional features that are not currently present in server NICs but that we believe could represent a useful initial set of feature for a next generation NIC were packet forwarding and routing is a first class design concern.

#### *Runtime configuration.*

One of the major drawbacks of most of today’s NIC

features is that they cannot be modified quickly while the NIC is processing incoming traffic. Many features even require a NIC reset in order to be reconfigured.

There are instead many network scenarios where it would be desirable to change the configuration at run-time, for example:

- If the current hash function leads to a poor balance of flows across cores, one would like to change the function (maybe even just between a pre-configured set of hash functions). This could be even done automatically by the NIC controller as it realizes that one hardware queue is full or nearly full.
- If the traffic load is low enough that a smaller number of cores could handle all of the packet processing needs, it would be useful to reduce the number of hardware queues. This could lead to a more energy efficient design as one could immediately turn off cores that were assigned to queues that are removed.

#### *More general classification.*

Current NICs support classification on fixed fields (such as port numbers and source/destination IP addresses) or with a list of exact match rules on the ethernet address or on a packet offset (but then limited to a small number of flows).

For new protocols or applications, it might be useful to give the programmer a more fine grained (or even programmable to a certain extent) control on the classification. For example, one could envision packet classification techniques that are designed so that one core only receives packets destined to a subset of the prefixes in the routing table. This way the route lookup operation could be optimized by splitting the routing tables across cores to improve cache locality – this is of particular importance given the current trend towards non-uniform memory architectures (NUMA).

Above we have very briefly listed some of the features that would benefit software routers if present in off-the-shelf NICs. Future work includes exploring efficient hardware implementations for these features and the appropriate abstractions or programming model by which NICs expose the functionality such features offer. The intent is to find the minimal set of features that could benefit software routers without requiring radical changes to current NICs design practices.

## 5. CONCLUSIONS

We presented a study of the performance of modern server NICs in presence of packet forwarding workloads. We highlighted the challenges involved in isolating the

contribution of the different system components to the overall packet processing performance.

Our investigation has shown the advantage that multi-Q NICs present compared to traditional NICs. We have also shown limitations in the design of current NICs both in the form of the maximum rate per port that can be achieved and in performance degradation (though limited) when increasing the number of hardware queues that receive traffic.

In future work, we plan to investigate the feasibility and value of more advanced NIC features. Our goal would be to identify new features that benefit software router workloads without incurring significantly increased manufacturing costs, thereby preserving the high volume, off-the-shelf nature of current server NICs.

## 6. REFERENCES

- [1] Intel 82599 10 GbE Controller Datasheet. [http://download.intel.com/design/network/datashts/82599\\_datasheet.pdf](http://download.intel.com/design/network/datashts/82599_datasheet.pdf).
- [2] Intel 82599EB 10 Gigabit Ethernet Controller. <http://ark.intel.com/Product.aspx?id=32207>.
- [3] R. Bolla and R. Bruschi. PC-based Software Routers: High Performance and Application Service Support. In *Proceedings of the ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Seattle, WA, USA, August 2008.
- [4] R. Bolla, R. Bruschi, G. Lamanna, and A. Ranieri. Drop: An open-source project towards distributed sw router architectures. In *GLOBECOM*, pages 1–6, 2009.
- [5] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2009.
- [6] N. Egi, A. Greenhalgh, mark Handley, M. Hoerdt, F. Huici, and L. Mathy. Fairness Issues in Software Virtual Routers. In *Proceedings of the ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Seattle, WA, USA, August 2008.
- [7] N. Egi, A. Greenhalgh, mark Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards High Performance Virtual Routers on Commodity Hardware. In *Proceedings of the ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, Spain, December 2008.
- [8] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated Software Router. In *Proceedings of the ACM SIGCOMM Conference*, 2010.
- [9] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.