Fairness Issues in Software Virtual Routers

Norbert Egi
Computing Dept.
Lancaster University
Lancaster, UK
n.egi@lancaster.ac.uk

Adam Greenhalgh
Dept. of Computer Science
University College London
London, UK
a.greenhalgh@cs.ucl.ac.uk

Mark Handley
Dept. of Computer Science
University College London
London, UK
m.handley@cs.ucl.ac.uk

Mickael Hoerdt
Computing Dept.
Lancaster University
Lancaster, UK
m.hoerdt@lancaster.ac.uk

Felipe Huici
NEC Europe Ltd
Heidelberg, Germany
felipe.huici@nw.neclab.eu

Laurent Mathy
Computing Dept.
Lancaster University
Lancaster, UK
l.mathy@lancaster.ac.uk

ABSTRACT

In this paper we investigate the building of a virtual router platform that ensures isolation and fairness between concurrent virtual routers. Recent developments in commodity x86 hardware enable us to take advantage of the flexibility and wealth of resources available to a software router in order to build a virtual router platform. Using commodity x86 hardware we show that it is viable to run highly experimental and untrusted router systems along side a production router on the same hardware platform without sacrificing performance. We investigate the extent to which we can isolate a virtual router running experimental code from other virtual routers.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Internetworking

General Terms

Design, Experimentation, Performance

Keywords

Virtualization, Routers, Commodity Hardware

1. INTRODUCTION

There has been renewed interest recently in re-evaluating the Internet architecture. However, for any non-trivial change to be successful it will have to run concurrently with existing protocols, firstly in test networks and eventually in production. In Bavier et al. [**?**], the authors advocate the use of virtualised routers to aid such experimentation and deployment; virtual routers, where one box behaves as several independent logical routers, are one approach to enable such innovation. Alternative uses for virtual routers include enabling ISPs to offer each customer control of their edge router, or as a platform to allow the in-network deployment of new applications or appliances.

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.
PRESTO'08, August 22, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-60558-181-1/08/08 ...$5.00.

For our purposes, we define a virtual router to be a router instance able to run concurrently with other instances using the same physical resources of a virtual router platform. In this work we investigate the ability of a software virtual router platform to support a number of virtual routers concurrently, and the extent to which it is possible to isolate virtual routers from each other. In section 2, we outline the underlying limitations of commodity x86 hardware and its impact on the ability of a virtual router platform to share and isolate virtual routers. In section 3, we investigate three different virtual router configurations that increasingly trade isolation for performance. Section 4 concludes this paper.

2. VIRTUAL ROUTERS

The idea of virtualising a router is not new; indeed, major router vendors offer variations on this theme that share a common binary control and data plane that filters packets to their corresponding virtual router instances. Unfortunately, this approach does not have the flexibility we desire, as such routers usually have no or very little programmability.

For research, there is a fine balance between performance and flexibility; specialist hardware is clearly needed for very high performance forwarding, but modern x86 hardware is actually remarkably capable and in principle provides arbitrary flexibility for different virtual routers to use completely different stacks. In this paper we focus on such commodity hardware, and aim to cast light on the trade-off between performance, flexibility, isolation and fairness.

To investigate these issues we start with the Xen virtualisation platform [**?**] running Linux and use the Click modular router package [**?**] for forwarding. Click's modular model is based around the composition of simple elements that perform basic packet-processing functions and allows for flexible configuration of different routers. At the same time, Click yields very close to optimal forwarding performance and supports multi-threading, enabling us to investigate the potential of modern, multi-core x86 CPUs.

In order to provide the functionality that users might want from a virtual router platform, we envisage three scenarios. In the first scenario, all of the virtual routers on the platform have identical forwarding paths. In the second scenario, each virtual router can have a different forwarding plane, but each must be built from a set of trusted and provided Click elements. Finally, in the third scenario, this last constraint is removed, allowing users to build

forwarding planes from untrusted elements[1]. For a virtual router platform to be viable it should be able to concurrently run all three scenarios while maintaining reasonable performance and providing isolation. We discuss the complexity associated with each scenario in section 3.

Figure 1 shows an example of all three scenarios and an alternative option for the third scenario. A Xen driver domain (dom0) hosts a Click forwarding plane composed of standard forwarding elements for three virtual routers, two are only forwarding via dom0 (scenario 1&2), while the third one is also running custom forwarding software in its domU (scenario 3). The alternative scenario 3 is the fifth Xen domain which has its own private network interfaces directly mapped, and does its own forwarding by monopolising these interfaces.

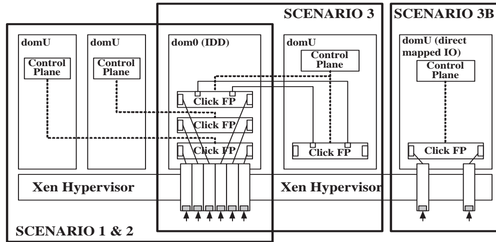

Figure 1: Virtual Router Architecture.

2.1 Baseline Performance

Before we can consider how to assign resources to virtual routers, we must understand the system's baseline performance. We chose Dell PowerEdge 2950 servers for these experiments, as they allow us to tease apart SMP performance issues. By serendipity, these have two 2.66 GHz quad-core Intel X5355 CPUs. However, these are not true quad-core, as they are effectively two dual-core CPUs in one package, where each pair shares a 4MB L2 cache. This combination allows us to investigate the issues when packets switch CPUs, switch cores within the same package, or switch cores that share a common cache. These systems have a uniform memory architecture, with 8GB of 667MHz DDR2 memory connected to the north bridge via four 5.3 GB/s channels. Networking is handled by three Intel gigabit quad-port cards each using a PCIe x4 channel, for a total of 12 gigabit ports.

2.1.1 Forwarding Performance

When using Click in a simple bridge configuration, the Dell 2950 has a maximum packet forwarding rate of 7.1 million packets per second (Mp/s) for minimum-sized, 64-byte packets (equivalent to 3.64 Gb/s). For larger, 1024-byte packets, the rate is 1.43 Mp/s, or 11.72 Gb/s, which is the theoretical maximum bidirectional rate for all 12 gigabit ports. For minimum-sized packets, however, the rate is well below the theoretical maximum, indicating a performance bottleneck.

As it turns out, the reason for this is rather complex and is primarily due to memory latency, exacerbated by numerous transfers over the PCIe bus and FSB per packet forwarded to main memory. Indeed, the transfers of the small packets between a NIC and memory, as well as the associated packet descriptor book-keeping, result in several short read and write memory operations, one cache line or less, both by the NIC and the CPU. This is compounded by the fact that with 12 network interfaces concurrently transferring packets to main memory via DMA, the Memory Controller Hub has to multiplex these short transfers to and from memory, resulting in a sequence of short memory accesses exhibiting poor localisation. This, in turn, causes poor performance because of the time required to continually change the memory address lines, thus preventing the memory controller from entering the more efficient "burst" mode.

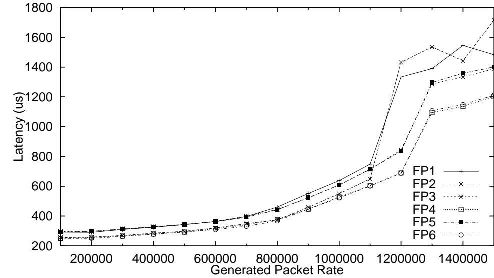

Figure 2: Mean latency induced by our system per forwarding path

Latency is an important characteristic of forwarding performance, in Figure 2 we show that the latency induced by the system on six static forwarding paths each allocated to a separate core. The average latency of each forwarding path stays below 700μs up to the point where the machine gets saturated (i.e. it reaches the 7Mp/s limit, which happens when each forwarding path is forwarding 1.2Mp/s) and after that the latencies still stay below 1.5ms. Note, the buffer size on our NICs is 48KB, that induces on its own 573μs delay when full (i.e. in overload conditions), showing that the delay induced by forwarding a packet from the input to the output port is acceptably low.

2.1.2 Sharing CPUs

Given the memory bottleneck, a key goal when allocating cores to virtual routers is to try to keep a packet on the same core throughout its processing to improve cache locality and thus reduce memory accesses. To analyse this effect we measured the performance of two gigabit forwarding paths with 64-byte packets. With each forwarding path handled in its entirety by a separate core so that packets do not change cores, the line-rate of 2.9 Mp/s is achieved. If, on the other hand, the inbound half of the path is handled on one core and the outbound half on a different core that shares an L2 cache, then the rate falls to approximately 1.9 Mp/s. Finally, if we force packets to switch to cores that do not share an L2 cache, the rate drops to 1.2 Mp/s.

2.1.3 Sharing Interfaces

If a NIC is shared between more than one virtual router, then packets need to be classified on input so they are handled by the right router; this classification can be either done in hardware or in software. Hardware classification is undertaken on the NIC and presents the system with a separate queue per virtual NIC thereby allowing the system to poll packets from each queue only when the associated virtual router is allocated resources and greatly simplifying scheduling. Intel's VMDq [**?**] provides such NIC-based virtualisation, but unfortunately it is still very new and not yet fully supported by the network driver, Click or Xen.

The alternative is software classification, where packets are polled from the NIC prior to classifying them into separate queues for each virtual router. Unfortunately, to avoid livelock under overload conditions, we want to discard unwanted packets directly from the NIC without wasting memory bandwidth. In the case of software classification this is not possible, greatly complicating fair scheduling because the scheduler cannot control the resources used prior to classification.

We examine this issue by comparing a standard single-queue

[1] In each of these scenarios the router control plane for each virtual router instance would run in its own virtual system.

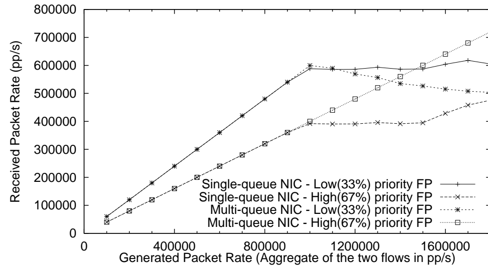

Figure 3: Per-flow forwarding rate on single-queue and (emulated) multi-queue NICs with forwarding path priorities (1:2)

NIC with an emulated multi-queue NIC. The emulation is achieved by using two separate NICs and having the switch they are attached to separate the traffic stream between them. In the standard NIC we use software classification to separate the traffic stream. In both single and multi-queue scenarios we destined traffic for two forwarding paths, one high priority with 67% of the resources and one low priority with 33% of the resources. The resource allocation scheme is discussed later in section 3.3.1.

Figure 3 shows the forwarding rate of the forwarding paths in both the single and multi-queue NIC cases. We generated twice as many packets per second for the low priority forwarding path than for the high priority one to illustrate the fair-sharing property of both scenarios. The difference can be clearly seen in the overload condition for generated packet rates of over 0.9Mp/s, where the forwarding performances in the single-queue NIC case are determined by the arriving traffic pattern and not by the allocated resources since we have no control over which VR's packet to poll in next. In contrast, in the multi-queue NIC the performance of the high priority path continues to increase as the CPU becomes saturated and the low priority path becomes throttled accordingly.

3. EVALUATION OF THE FORWARDING PLANES

In this section we examine the problems of fair resource sharing in the different virtualised forwarding-plane scenarios introduced in Section 2. We focus on CPU cycles and memory accesses as these are the key factors in determining forwarding performance. For each of the three scenarios, we will propose and evaluate a possible solution for CPU cycle scheduling and suggest extensions for memory scheduling.

Using virtual multi-queueing is a prerequisite for fair resource sharing in a virtualised forwarding path environment. Hence, in all of our scenarios we assume that the NICs are capable of filtering and demultiplexing the arriving packets into the multiple queues present on the NIC; each belonging to a separate virtual router. Due to the lack of such hardware we emulated a VMDq NIC with multiple, single-queued NICs.

3.1 Scheduling elements

Scheduling multiple CPUs across multiple forwarding paths (FPs) is a complex operation that can be decoupled into two steps:

1. We have to assign all forwarding paths of each of the virtual routers to the available CPU cores. To do this, we need to know the cost of each schedulable element and the resource entitlement of each virtual router. With this information it is possible to calculate the most effective forwarding path-to-core allocation scheme. This operation needs to be carried out after setting up the forwarding engine, whenever the cost of a virtual router's forwarding path changes significantly, or whenever resource entitlement is changed.
2. After the first step has been done, we have to deal with scheduling each core. If there is more than one schedulable element assigned to a core we ensure fairness is maintained within the boundaries of that core; this is more complex than step 1 and needs to be carried out continuously.

The first step is fairly simple (at least to a rough approximation), so in this paper we focus on the second, more interesting step.

Scheduling memory accesses in a virtual router is not as simple as scheduling CPU cycles. However, it can be performed indirectly by extending the CPU scheduler to factor in the average number of memory accesses (measured in CPU cycles) of the tasks being scheduled. Modern processors provide very low overhead hardware counters that allow us to measure among other things, statistics related to memory bus operations and L2/L3 cache events (i.e. cache hits, misses, etc. that help infer the number of memory accesses needed per forwarding path per packet). This way we can sample and then predict the number of memory accesses required to forward a packet on a forwarding path. In scenarios where the memory has become the bottleneck, we can adjust the CPU scheduler's parameters so as to ensure a fair memory access rate by controlling the forwarding paths' CPU resources for example if a forwarding path is scheduled to use CPU resources less often it will access the memory less often too.

3.2 Scenario 1: Static virtual forwarding plane

The simplest virtual router consists of identical concurrent forwarding paths as shown in Figure 4. In this case, all virtual routers are identical except perhaps for the number of interfaces allocated to each, their addresses, and the entries in their forwarding tables. The cost of processing a packet by any of the forwarding paths is very similar, with slight differences due to variation of packet sizes and forwarding table look-ups. However, these differences are not expected to have a significant effect on the number of CPU cycles and the number of memory accesses needed to process a packet.[2]

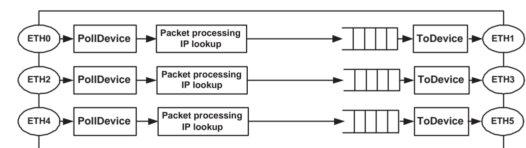

Figure 4: Three identical forwarding paths sharing a CPU core.

In the case where all the forwarding paths need a similar amount of resources to process a packet, a Round-Robin scheduler is sufficient to provide an equal share of the available CPU resources and, as a consequence, of the memory resources. Such a simple scheduler ensures that each of the forwarding paths gets equal access to the CPU and, because they use a similar number of CPU and memory cycles each time they are scheduled, fair sharing among the virtual routers is achieved.

Click's default CPU scheduler is based on proportional-share stride scheduling [**?**], and schedules equally weighed tasks (i.e. for-

[2]If forwarding paths do need a significantly different number of CPU cycles and memory accesses per packet, then the slightly more complex solution proposed for the second scenario (Section 3.3) can be used.

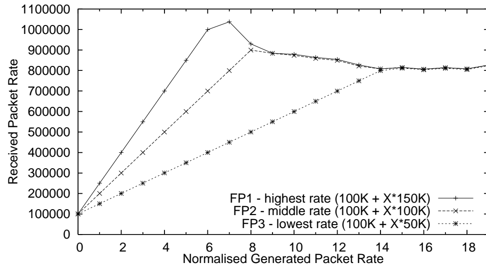

Figure 5: Illustrating fair scheduling when packets are arriving at different rates per forwarding path

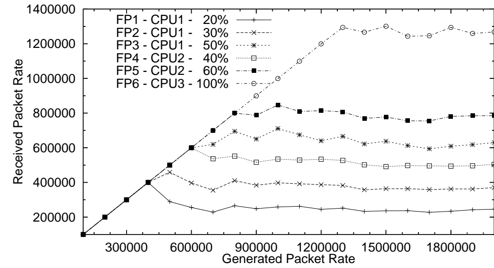

Figure 6: Different weight scheduling of 6 forwarding paths on 3 cores.

warding paths) assigned to the same core in a Round-Robin manner. If different weights need to be given to different forwarding paths to allow some customers more resources than others, this can be done fairly simply by extending the forwarding path scheduler to a Weighted Round-Robin scheduler; our extension to the Click scheduler discussed in more detail in Section 3.3.1 tackles this requirement as well.

Figures 5 and 6 illustrate the fair-sharing property of the original CPU scheduler algorithm. In the first case, we created a configuration with three forwarding paths (FPs) representing three virtual routers that share the resources of a CPU core. The FPs were identical, but the incoming traffic to each of them was different. More specifically, we started to generate minimum-sized packets at a rate of 100Kp/s for all the three FPs and increased the rates of each flow in increments of 150Kp/s, 100Kp/s, and 50Kp/s, respectively.

Figure 5 shows that the CPU gets saturated after the 6th step, causing the forwarded packet rate of FP1 (the one receiving the highest generated packet rate) to drop; the rates of FP2 and FP3 increased further as they were using less than their CPU cycle allocation. After the 8th step the forwarded packet rate of FP2 starts to drop too since it exceeded its fair share, while the rate of FP3 was still rising. This phenomenon continues until the generated packet rates for all the FPs have exceeded one third of the overall forwarding capability of the core in question.

In the case of Figure 6, we created a configuration with six forwarding paths representing six virtual routers that share the resources of three CPU cores. The FPs were completely identical again, but in nearly full compliance with the standards of a unicast IP router [**?**] (hence the lower overall forwarding rate per core). The resources were allocated as shown in the figure's key. We can see that the overall throughput of each core is roughly 1.3 Mp/s, and that this rate is shared by the multiple FPs in accordance with their allocation ratio.

3.3 Scenario 2: Configurable virtual forwarding plane

Even though the previous scenario provides fairly straightforward resource management, it does not have any flexibility for users to design their own forwarding path. In this section, we focus on custom forwarding paths built only from a large range of approved Click elements. In this case, the forwarding paths can vary significantly both in structure as well as in cost. As each element is trusted *a priori* by the system, they can share the same forwarding domain (see Figure 1), but as the number of elements in the forwarding path and the resource consumption (i.e. cost) of each element can vary depending on various parameters (e.g. length of packet being forwarded), providing fairness reduces to:

- Preventing a packet from hogging the CPU for too long.
- Monitoring the total resource consumption of each forwarding path, then tuning the packet scheduling parameters. Rather than forwarding an equal number of packets, unequal numbers of packets must be scheduled so as to equalise resource consumption between competing virtual routers.

The first requirement is essential for responsiveness of all virtual routers sharing a CPU and can be addressed with a simple watchdog mechanism, whereby a timer is triggered every time a packet enters a forwarding path, and cleared when the packet exits it or gets dropped. On watchdog expiry, a forwarding path can be declared as non-compliant and removed.

3.3.1 Extended CPU Scheduler

In order to guarantee that all the forwarding paths get access to the amount of resources they are entitled to and no more, the total cost of each forwarding path must be monitored and the CPU scheduling parameters tuned accordingly. Note that we need to monitor the cost of each forwarding path regularly to ensure that the current CPU scheduling adjustments reflect the recent resource usage of all the forwarding paths, including the ones where the per-packet processing cost can vary significantly depending on the character of the packet in question. To this end, we extended Click's original CPU scheduler by adding the ability to measure the cost of each forwarding path and adjusting the weight of each of them according to their costs, thus achieving fair CPU scheduling[3]. In addition, we can also change the quantity of CPU resources that the given FPs are entitled to. In an attempt to mitigate the differing forwarding costs associated with different packets, whilst the sampling frequency is tunable, we sampled at a rate of one in every twenty packets to balance the sampling overhead against the fidelity of the measurements.

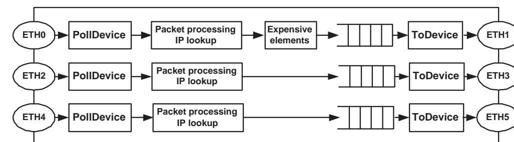

Figure 9: Three FPs sharing a core. Two are identical whilst the third needs more resources to process a packet.

To illustrate the scheduling properties of the configurable virtual forwarding plane, we experimented using a similar configuration to that in Section 3.2, but we changed one of the FPs to include a few expensive Click elements to increase the cost of forwarding a packet (Figure 9). Using the default Click scheduler we achieved

[3]Note that the resource consumption for forwarding a packet can only be measured when the packet exits the forwarding path and control is returned to the Click scheduler.

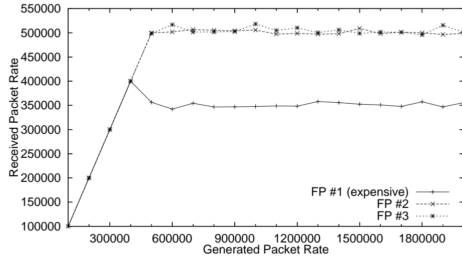

Figure 7: Load balancing a CPU's resources among two identical FPs and a more expensive one.

the same maximum forwarding rate for each of the FPs (approximately 440 Kp/s), meaning that the CPU was not shared equally as the more expensive FP managed to forward the same number of packets per second as the others.

Figure 7 illustrates the throughput of the FPs when our scheduler extension was used to load balance the FPs' CPU usage. The results show that load balancing works properly, since the packet rate of the expensive FP decreased while that of the other two FPs increased; it takes approximately 40% more CPU cycles to forward a packet by FP1 than by FP2 and FP3.

3.4 Scenario 3: Customisable virtual forwarding plane

In the previous scenario the customers already had plenty of flexibility in designing their own forwarding path, but were allowed to use only the functionality provided by elements approved by the system, which might not satisfy their needs. Ideally, we would like to allow customers to build their own packet processing elements performing whatever functions they need. However, as we cannot allow the system to trust these custom-made elements unconditionally, we have to ensure that they are executed in an isolated environment.

A practical solution is to use the domUs running the control plane for the VRs (see Figure 1) to provide an isolated environment for executing the untrusted, potentially unreliable, custom-made forwarding path elements. In such a scenario, packets enter the forwarding path in dom0, are pushed up to the corresponding domU for processing by the custom-made elements, and come back to the same point in the forwarding path in dom0, from where the packets leave the router.

In this case, Xen's (or any other hypervisor being used) CPU scheduler plays a role in providing fair CPU sharing among the VRs. The domUs that perform some functions in the forwarding path must get a bigger share of the CPU resources than the ones assigned to only performing control plane functionalities. Similarly, the CPU share of these forwarding paths within dom0 must be decreased by the equivalent of the extra share granted by the Xen scheduler to the domU.

Using multiple domains for packet processing presents additional challenges. To begin with, we lose the advantage of having relatively easy access to information about CPU and memory usage of the FPs and of having a scheduler in the forwarder domain that is capable of providing the full service of fair scheduling. In addition, we also have to ensure that the packet transfer between the shared forwarder domain and the given domU can happen at high rates. In Xen, the I/O-channel located between dom0 and the domUs is meant to carry out such tasks. Unfortunately, this channel is performance limited [?]. We found, however, that there is no difference between a domU's and dom0's processing power; only the packet rate between domUs is limited.

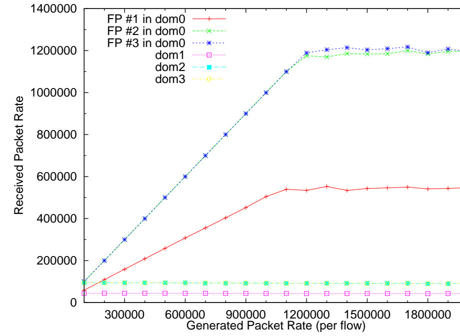

Figure 8: 3 FPs in dom0 and 3 domUs forwarding via Xen's I/O-channel. The first FP in dom0 and the first domU are sharing a core, the others use separate ones.

We tested the CPU scheduling and performance properties of such a split ($dom0 \rightarrow domU \rightarrow dom0$) forwarding path scenario. In the configuration we had 3 forwarding paths in dom0, and packets were pushed up to and received back from 3 domUs in parallel via the I/O-channel. Five cores were used and assigned in the following manner: FP1 in dom0 shared a CPU with dom1, while all the other FPs had their own core.

The forwarding rates of the FPs and domUs reflect this core allocation scheme (see Figure 8). FP1 and dom1 forwarded about half as many packets per second as their counterparts did, as a consequence of having the CPU for only half the time. It is important to note the limited rate of approximately 90 Kp/s for dom2 and dom3, which is similar to the rate the VINI architecture [?] achieves and is caused by the limitations of the I/O channel.

Figure 10 shows the impact of the memory access time limitation on forwarding performance. This figure also shows that mapping interfaces directly to domUs yields the same forwarding performance as when all the forwarding is performed exclusively in dom0, offering plenty of other possibilities for router virtualisation.

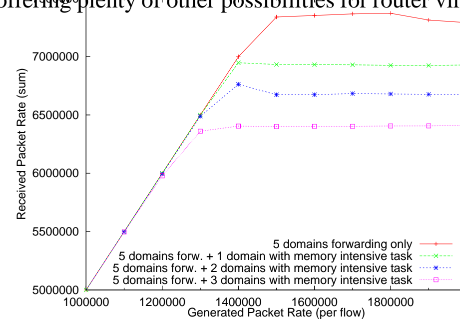

Figure 10: 5 domains were directly mapped each to a pair of interfaces between which they forward packet and memory intensive task(s) were run in different number of other domains.

In this experiment we directly mapped 5 pairs of interfaces to 5 domUs, assigned a separate core to each domU and ran the simplest forwarding code, allowing us to clearly reach the memory limit of approximately 7.3 Mp/s (top-most line on the graph). In addition, we gradually increased the number of domains (up to 8, due to the limit of cores), with separate cores assigned to each domain. In each domain we ran a simulated control plane in the form of a memory-intensive C program that randomly read from and wrote to a large memory space, causing cache misses and therefore high memory access rates. As the results show, increasing the mem-

ory access rates in separate domains can significantly decrease the forwarding performance of other domains. We repeated the same experiment and achieved very similar results when using 5 cores assigned to dom0 to forward packets between 5 pairs of interfaces instead of mapping the pairs of interfaces into 5 separate domains, which shows that the forwarding performance is limited by memory access, and not the domain where the forwarding path runs.

4. CONCLUSION

In this paper we have shown that a virtual router platform based on commodity hardware can forward large packets simultaneously on at least 12 Gigabit interfaces at line rate and 64-byte packets at a very respectable 7 Mp/s. This level of performance demonstrates the feasibility and viability of cost-effective software virtual router platforms at the edge of the network or as an experimental system.

Through experiments, we have demonstrated that forwarding paths composed of trusted elements can be supported with a significant degree of isolation (i.e. not only protection, but also little perceived performance impact) and fairness between the virtual routers, while exhibiting performance equivalent to that of a similar forwarding path running on a single core without virtualisation. Importantly, we have shown this to be true even in the presence of forwarding paths with different costs.

In the presence of untrusted forwarding elements, we have shown that isolation and protection of co-resident trusted forwarding paths can be achieved with very little performance penalty.

These results were achieved through a two-step scheduling process that first statically binds forwarding paths to CPU cores and then aims to co-schedule cores and memory access for maximum performance. However, as the number of cores in a single processor increases (e.g. 16-core CPUs have been announced for 2009) and the total number of cores exceeds the number of virtual routers in the system, sharing the resources of a single core might not be relevant in the future. Albeit, scheduling the memory accesses globally and dynamically allocating the forwarding paths to the available cores based on their current resource usage for maximum performance will become an even more crucial issue. Indeed, for lightweight forwarding paths, where little packet processing is performed, memory turns out to be the main system bottleneck, while, of course, the CPU can become the scarce resource when more involved processing is required. The trouble with scheduling memory access is that it can only be done indirectly, which limits the precision of the access control and constitutes one of the major performance limitations in our proposed system. Nevertheless, the use of hardware architectures with several independent memory controllers (such as in NUMA, the non-uniform memory architecture) offer a promising way to alleviate this problem and achieve increased performance and isolation.

The first step of our scheduling strategy relies on static binding of virtual routers to cores, which can lead to coarse-grain fairness. However, a more fine-grained approach consisting of highly dynamic forwarding path-to-core mappings could result in poor performance, because of packets potentially switching cache memory hierarchies. Hence, while we believe that for most practical purposes coarse-grain fairness is probably acceptable, periodic remapping might be a good compromise when more fine-grained fairness is necessary.

When dealing with untrusted elements, the situation is further complicated by the fact that another level of scheduling is introduced: virtual OS scheduling is indeed necessary to provide basic protection between the various forwarding paths. This, however, seriously undermines the control of memory access needed to maintain high performance (and therefore isolation).

Consequently, while our proposed virtual router platform architecture does exhibit impressive performance and very good fairness, it is clear that further work is needed to break through the current performance plateau. Hardware assists such as virtual queuing on the NICs would improve fairness without losing performance, while novel, advanced OS techniques for intensive network I/O are also necessary from a performance perspective. In our experiments we were using the Click software router, mainly because of its fine-grained modular structure and its multi-threading capability. However, as soon as the Linux kernels supports multi-threading for its IP stack the need for Click may be dissolved.

Our approach differs from that proposed in [**?**, **?**], which relies on OS-level virtualisation for the control planes whilst sharing the same data plane. Because it does not rely on tight integration of the router fast path, our proposal exhibits greater flexibility in terms of the range of forwarding paths that can be concurrently supported. When the different virtual routers have similar forwarding paths, our approach exhibits performance an order of magnitude better.

Another alternative to our proposed architecture would be to follow the approach in [**?**] where network processors are used to achieve high performance and isolation. Although such an approach demonstrates very good performance and isolation, it relies on custom hardware (IXP) that is complex to program and expensive.

In view of this, we conclude that modern commodity hardware architectures constitute a viable platform to support high performance, cost-effective software virtual routers. However, proper mechanisms do need to be put in place to overcome the novel system issues they present before researchers can properly utilize the flexibility and sheer raw processing performance exhibited by modern, many-core CPUs.

5. REFERENCES

[1] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, “In vini veritas: realistic and controlled network experimentation,” *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 3–14, 2006.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R.Neugebauer, I.Pratt, and A. Warfield, “Xen and the art of virtualization,” in *19th ACM Symposium on Operating Systems Principles*. ACM Press, October 2003.

[3] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kasshoek, “The click modular router,” *ACM Transaction on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[4] R. Hiremane, “Intel virtualization technology for directed i/o (intel vt-d),” *Technology@Intel Magazine*, vol. 4, no. 10, May 2007.

[5] C. A. Waldspurger and W. E. Weihl., “Stride scheduling: deterministic proportional-share resource management.” MIT Laboratory for Computer Science, Tech. Rep. MIT/CS/TM-528, June 1995.

[6] F. Baker, “Requirements for IP Version 4 routers,” *Request for Comments 1812*, June 1995, http://ftp.ietf.org/rfc/rfc1812.txt.

[7] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley, “Evaluating xen for virtual routers,” in *PMECT07*, August 2007.

[8] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, “Hosting virtual networks on commodity hardware,” Georgia Tech. University., Tech. Rep. GT-CS-07-10, January 2008.

[9] M. Zec, “Implementing a clonable network stack in the freebsd kernel,” in *Proceedings of the FREENIX Track : 2003 USENIX Annual Technical Conference*. San Antonio, TX, USA: USENIX Association, 2003, pp. 333–346.

[10] J. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiseman, and D. Zar, “Supercharging planetlab - a high performance, multi-application, overlay network platform,” in *Proceedings of SIGCOMM’07*, Kyoto, Japan, August 2007.