

# Resource Virtualisation of Network Routers

Ross McIlroy

Department of Computing Science  
University of Glasgow, UK  
Email: ross@dcs.gla.ac.uk

Joe Sventek

Department of Computing Science  
University of Glasgow, UK  
Email: joe@dcs.gla.ac.uk

**Abstract**—There is now considerable interest in applications that transport time-sensitive data across the best-effort Internet. We present a novel network router architecture, which has the potential to improve the Quality of Service guarantees provided to such flows. This router architecture makes use of virtual machine techniques, to assign an individual virtual *routelet* to each network flow requiring QoS guarantees. We describe a prototype of this virtual routelet architecture, and evaluate its effectiveness. Experimental results of the performance and flow partitioning of this prototype, compared with a standard software router, suggest promise in the virtual routelet architecture.

## I. INTRODUCTION

As a best-effort service, the Internet delivers packets as quickly and reliably as possible; however, there is no guarantee as to how long a packet will be in transit, or even whether it will be delivered. There is increasing interest in networks that can provide Quality of Service (QoS) guarantees. Such networks would be able to effectively transport isochronous datastreams, such as audio or video, since each stream could have guaranteed bounds on latency and throughput.

There has been considerable research into the overall infrastructure required for a QoS network [1], however, the actual allocation of routers' resources to QoS streams has received less attention. A means of partitioning a network router's resources between QoS streams is required, so that the traffic of one stream is minimally affected by the traffic on other streams. Commonly used commercial methods include over-provisioning routers or using a process sharing queueing scheme, such as Weighted Fair Queueing (WFQ) [2], to guarantee bounds on transmission latency.

We propose a router architecture based on virtual machine techniques, which can increase partitioning between different network flows, thus providing better QoS guarantees. This *virtual routelet* architecture consists of a number of virtual machines, each of which is allocated a proportion of the underlying physical machine's resources (such as CPU time, or network bandwidth). Each such virtual machine is called a *QoS Routelet*, and is assigned to route a single QoS flow (set up using existing QoS signalling techniques, such as RSVP [3]). The underlying virtual machine monitor ensures that each QoS routelet can only access its allocated resources, and so guarantees that no other flow can interfere with the routelet's processing of its flow. A final virtual machine routes all best-effort traffic, and controls the router overall.

This router resource virtualisation could offer a number of advantages over previous approaches to router resource allo-

cation. It allows partitioning of a router's resources between streams, without expensive or inflexible over-provisioning. It also provides the ability to support application-specific queueing within network flows, unsupported by WFQ. E.g., if a router servicing an MPEG stream becomes overloaded, an application-specific queueing scheme could choose to drop packets containing difference, rather than the key frames, ensuring gradual degradation in streaming video performance.

## II. RELATED WORK

A number of programmable network projects have attempted to address the issue of router resource partitioning between network flows. Programmable networks [4] dynamically deploy network services based on the demands of users (either through active networking or open signalling). Many programmable networks focus on easing network management, however, resource partitioning between flows is important to prevent Denial of Service attacks by packets requesting excessive network services.

Some programmable networks have focused on providing QoS directly. For example, the Darwin project [5] is an attempt to create a set of customisable resource management mechanisms that can support value added services which have specific QoS requirements, such as video and voice data streams. However, the focus of the Darwin project is on providing a middleware environment for value added network services, not on the underlying mechanisms needed to guarantee quality of service flows within a network. As such, it simplifies the aspect of resource partitioning between QoS flows. Similarly, NodeOS [6] provides kernel interface abstractions for accounting and scheduling of resources required by network flows. This framework allows resources to be allocated to QoS flows as required; however, the NodeOS platform is simply a kernel interface, and does not provide details on how the resources used by flows should be partitioned.

Zec [7] describes an implementation of clonable network stacks which is in some respects similar to this work. Clonable network stacks enable groups of user applications to independently access network resources through *virtual images*, in effect creating pseudo virtual machines. This provides a degree of isolation between applications, partitioning virtual images from a management, as well as a performance point of view. However, clonable network stacks require substantial changes to an operating system and are not directly applicable

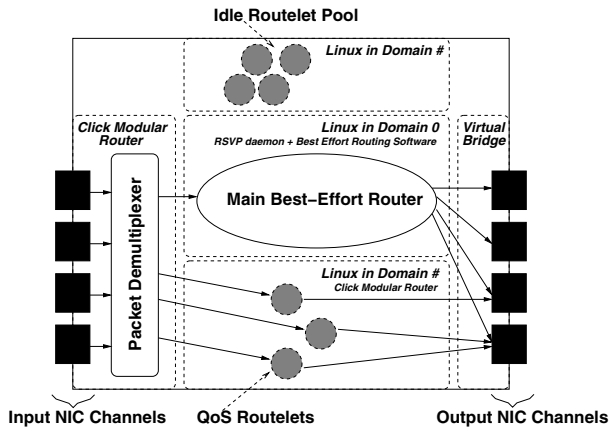


Fig. 1. A view of the components used by the QuaSAR architecture.

to resource partitioning within a router, since each network stack requires its own IP address.

However, the key difference between this paper and previous work in this area is the degree of partitioning which can be exploited by considering each routelet as a completely separate virtual machine. This provides more flexibility in scheduling packet processing of different flows (i.e. the router can preempt the processing of a low priority flow for a higher priority flow at any point, without having to consider reentrant code or locked data structures), as well as allowing independent algorithms for the scheduling of packets within flows.

### III. QUASAR ARCHITECTURE

In order to evaluate the effectiveness of the virtual router architecture, a prototype router (known hereafter as *QuaSAR* - Quality of Service Aware Router) was built which incorporated these design features. QuaSAR [8] uses virtual routelets to route MPLS (Multi-Protocol Label Switching) traffic flows which specify QoS constraints when they are created. We chose MPLS [9] for QoS traffic because of its relatively wide industry acceptance, its support for QoS routing aspects such as traffic engineering and its relatively simple processing requirements. QuaSAR can also route best-effort IP and MPLS traffic through normal routing methods. This prototype was implemented as a software router on commodity x86 hardware.

The overall architecture (Figure 1) of the QuaSAR router consists of multiple guest Operating Systems (OSs) running on top of a single virtual machine manager. Each of these guest OSs runs software to route incoming packets. One guest OS controls QuaSAR overall, as well as routing all the best-effort, non-QoS traffic. The other guest OSs are known as *routelets*, and are available for the routing of QoS traffic flows.

Virtualisation software is used to create multiple virtual machines (VMs) simultaneously on the same physical machine. Each VM contains a routelet, running on its own independent OS, thus partitioning its resource usage. For QuaSAR to cope with the demands of high-throughput packet routing, this virtualisation software must have little overhead. We, therefore, decided to use a para-virtualisation system, rather than a fully emulated virtualisation system. Para-virtualisation

presents an idealised machine interface to virtual machines, rather than fully emulating the machine's instruction set, removing virtualisation unfriendly features. This increases scalability and performance, at the expense of having to port OSs to the idealised architecture. Both the Xen [10] and Denali [11] para-virtualisation systems were considered; we chose Xen for the QuaSAR prototype, since, unlike Denali, ports of existing OSs (such as Linux) are available for its virtual machine architecture. This allowed pre-existing routing software to be used as a basis for QuaSAR's implementation.

The first guest OS to start in a machine running Xen (domain 0)<sup>1</sup> has special privileges, such as starting new guest OS domains and configuring their resources. QuaSAR's main router is therefore started in domain 0, so that it can start, stop and control the resource access of the QoS routelets as new QoS flows arrive, or as their requirements change.

New MPLS label switched paths (LSPs) are created using RSVP-TE (ReSerVation Protocol - Traffic Extensions) in QuaSAR. A *flowspec* can be included in the RSVP-TE messages to define the QoS requirements of the LSP being created. When the QuaSAR router receives an RSVP message specifying the creation of a new QoS flow, it calculates the necessary resources (e.g. CPU time, throughput) required to support the flow's requirements<sup>2</sup>. If the necessary resources are available a routelet is given access to a certain proportion of the router's resources, such that it can meet the flow's requirements and is assigned to this new flow. A pool of routelets is created initially by QuaSAR, to remove the overhead of starting a new routelet (a relatively heavyweight operation) every time a QoS flow arrives.

### IV. QOS ROUTELET

QoS routelets are simplified routing engines for individual QoS flows in QuaSAR. Each routelet must use as little of the overall system resources as possible, so that QuaSAR can support a reasonable number of network flows at any one time. In the QuaSAR prototype, each routelet runs in a *guest* Linux operating system in its own VM. This reduced the prototype's development time by allowing reuse of Linux networking software, and by forstalling the creation of a custom minimal routelet OS. However, the Linux operating system contains many features which are not required by QuaSAR routelets. To alleviate this problem, a conscious effort was made to reduce the resource footprint of each routelet. A viable solution would use a minimal OS for routelets, or employ copy-on-write paging [12], to increase the router's scalability.

As well as using a minimal build of the Linux Kernel and a very small Linux distribution, the resource requirements of routelets are further reduced by the use of specialised routing software. Each routelet only routes packets from a single, unidirectional network flow (a single MPLS Label Switched Path), therefore it need only provide static routing and simple

<sup>1</sup>Xen uses the terminology of *domains* to refer to guest virtual machines.

<sup>2</sup>This mapping is currently carried out statically. Future research is needed into the resource requirements necessary to support particular QoS parameters before this mapping can be made fully automatic.

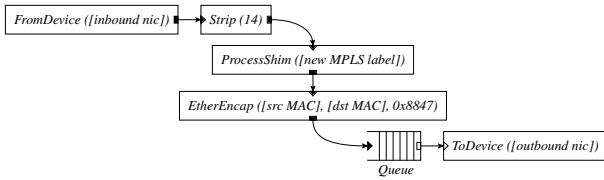


Fig. 2. The Click architecture used by QuaSAR routelets to process packets.

packet processing. When a network flow’s route changes, or its QoS requirements change, the main QuaSAR router will simply reconfigure the routelet. A routelet, therefore, has no need for routing tables or other dynamic routing features, however, it must support dynamic reconfiguration. With these requirements in mind, custom packet forwarding software was created for routelets, using the Click Modular Router [13].

### A. Routelet Forwarding Engine

Click is a modular routing framework, which enables *elements* (each of which performs a simple, well-defined function - e.g. header stripping) to be chained together to create a bespoke router. Packet forwarding in a routelet consists of packet retrieval from a static inbound network interface, processing of these packets, and transmission on a static outbound interface. This maps well to a sequence of Click elements.

QuaSAR routelets process packets at the MPLS layer, therefore they are responsible for the processing of both the data-link (in this case Ethernet) header and the MPLS shim header. Figure 2 outlines the Click configuration used by each routelet to perform this packet processing. The routelet first retrieves packets directly from the network device driver, using the *FromDevice* element (preventing Linux from further processing the packet through its networking stack). Each packet is processed by stripping off the outermost (old) Ethernet header, processing the MPLS shim as required for the next network hop, and encapsulating this packet in a new Ethernet header. Finally, the packet is queued, before being transmitted directly to the outgoing network device’s driver.

Click elements can be parameterised by different values when they are created (shown by the bracketed values in Figure 2). For example, *Strip(14)* strips 14 bytes (the Ethernet header) off the packet. The parameters designated by square brackets are not known until the time of flow creation. They are also subject to changes during the lifetime of a flow, for example, dynamic routing changes may modify the next hop of this flow, necessitating a change in the *dst MAC* parameter. These parameters must, therefore, be dynamically modifiable at runtime. QuaSAR’s main router can modify these parameters by manipulating a virtual filesystem, created on each routelet by Click, through a virtual network control link.

### B. Routelet Virtual Network Devices

Xen virtual network device interfaces (VIFs) are used by QuaSAR to provide control communication, and packet passing between the main router and routelets. VIFs form a point-to-point link between a back-end driver within a privileged domain (e.g. the QuaSAR’s main router), and a

front-end driver within a guest domain (e.g. the routelet). These VIFs are necessary because standard network devices cannot be controlled by multiple operating systems<sup>3</sup>. Instead, a single domain (the main QuaSAR router) controls the physical network devices, and provides routelets with access to the network through these VIFs.

Each routelet is configured with a number of VIFs, each of which corresponds to an Ethernet device in the underlying QuaSAR router. The back-end of each VIF is connected to the corresponding physical Ethernet device using a virtual Ethernet bridge in the main router. This means that any packets, sent out of a VIF within a routelet, are sent out of the corresponding physical device. When QuaSAR demultiplexes packets destined for a routelet (see Section V-A), it sends these packets to the routelet’s VIF which corresponds to the physical device on which the packet arrived. These VIFs therefore appear like the corresponding physical Ethernet devices to the routelet, but with only the traffic destined for that routelet.

## V. QUASAR’S MAIN ROUTER

QuaSAR’s Main Router has to perform a number of functions, including routelet control and best-effort traffic routing<sup>4</sup>. It also controls physical network devices and demultiplexing of QoS packets to the appropriate routelet for processing.

### A. Packet Demultiplexing

Xen’s mechanisms for sending network packets to the appropriate guest OS are too heavyweight to be used in a router aimed at providing QoS guarantees. Therefore, a lightweight architecture was created to demultiplex packets to the appropriate routelet using the Click Modular Router. Packet classification occurs in two stages - packets are first classified as either MPLS or non-MPLS packets, then the label of MPLS packets is examined to discover which QoS routelet should process it, or whether it is best effort traffic. Figure 3 gives an example of the demultiplexing architecture where two network interfaces are being shared between two QoS routelets (QuaSAR dynamically creates this architecture based upon the number of routelets and network interfaces).

The *MplsSwitch* element examines MPLS packet labels to discover to which (if any) QoS flow each packet belongs. It compares this label to a table of MPLS label / output port pairs and sends the packet to the appropriate port. Since each port is connected to the backend VIF of a certain routelet, this passes the packet to the correct routelet for processing. The pairs are stored in a hash table, indexed by the hashed MPLS label to provide expected O(1) lookup complexity. Label/port pairs can be added and removed from this table dynamically by the main router as new flows are set-up / torn down. Packets which don’t match any of the pairs in the table are sent out of a default port, which returns the packet to the main router’s network stack for best-effort routing.

<sup>3</sup>User-accessible network line cards, such as the Arsenic Ethernet Card [14], could allow multiple OSs direct access to the physical network device.

<sup>4</sup>Best-effort packets are routed using standard Linux routing software in the main router, and will not be discussed further.

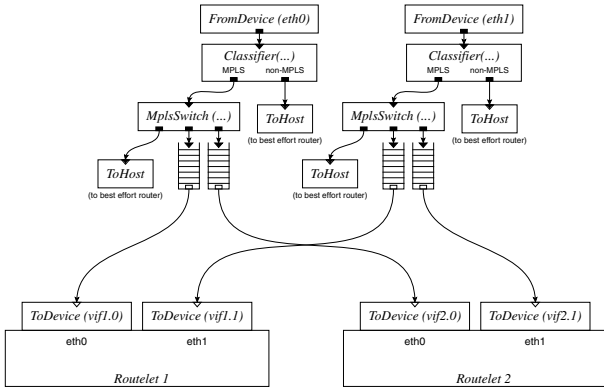


Fig. 3. The Click architecture used to demultiplex packets between routelets.

Ideally, packet demultiplexing would be performed in an independent domain, or by the virtual machine monitor itself, however, the version of Xen used by QuaSAR did not allow access to physical devices (such as the network interface cards) from any domain other than domain 0. QuaSAR’s main router must run in domain 0 to allow it control over other routelets, therefore, the physical devices’ control/demultiplexing needs to share its resource usage with the main router.

### B. Routelet Control

The main router has control over: starting and stopping routelets; configuring routelets for newly arrived flows; assigning appropriate resources to routelets and enforcing those resource limits; and controlling the packet demultiplexing architecture as new QoS flows arrive.

When the QuaSAR router starts, it creates an idle pool of routelets (in a paused state to reduce their CPU usage). The idle pool can be grown or shrunk as necessary, however this is an expensive operation, requiring booting or halting of multiple guest OSs. Normal practise involves extracting routelets from the idle pool as new QoS flows arrive and returning them to the idle pool as QoS flows are torn down.

An RSVP-TE daemon, running on the main router, processes RSVP messages. When a new QoS flow is signalled, it extracts a routelet from the idle pool and assigns it appropriate resources to meet this flow’s QoS guarantees. The daemon then configures this routelet by filling in appropriate parameters for this flow (e.g. next hop MAC address, next hop MPLS label, etc.) in the routelet’s forwarding engine, using the virtual control network. Finally, the demultiplexing architecture is modified by the RSVP daemon, so that packets with this flow’s MPLS label are directed to the routelet just assigned.

### C. Routelet Resource Management

The purpose of routing QoS network flows through routelets is to guarantee an allocation of the router’s resources to each QoS flow. Routelet management tools are used by the main router to assign resources, such as CPU time and maximum network transmission rate, to each routelet, depending on the

QoS required by the network flow it is servicing<sup>5</sup>.

1) *CPU Time Allocation*: CPU time is assigned to a domain, and therefore to a routelet, by the Xen virtual machine monitor’s scheduler. Xen can support multiple schedulers with different scheduling policies. Therefore, a virtual routelet architecture could assign CPU time to routelets based on algorithms with different trade-offs in terms of partitioning, granularity, fairness and slack stealing depending upon the router’s application. Unfortunately, when QuaSAR was being built, there was only one Xen scheduler which was stable - the Borrowed Virtual Time (BVT) scheduler. This scheduler does not provide any guarantees of how often, or for how long, a domain will be scheduled; instead attempting to providing a proportional fair share of CPU time to each domain. This prevents absolute guarantees on packet latency, or other QoS parameters, being made. A pre-release soft real-time Xen scheduler was added to QuaSAR to rectify this shortcoming, however, it was not stable enough to provide accurate experimental results.

2) *Network Transmission Rate Allocation*: It is important that each routelet can be guaranteed a minimum network transmission rate, so that it can provide the required QoS to the flow it is servicing. The transmission rate of a physical network card is finite, therefore, the network transmission rate of each routelet must be limited, to prevent one routelet from overwhelming a network card. To achieve this, bandwidth limiting was implemented on Xen’s virtual network devices (VIFs). This limiting function prevents a routelet from sending more than a certain (adjustable) number of bytes through a VIF (and thus through a physical network device) within a certain time period. This shapes the traffic of each routelet to ensure it does not exceed its allocated network bandwidth. This could be extended to allow routelets access to any spare capacity, thus preserving the benefits of statistical multiplexing while providing guarantees on minimum throughput.

## VI. EVALUATION

The QuaSAR prototype was evaluated to discover the potential of the virtual routelet architecture in providing resource partitioning between QoS flows when routing diverse network traffic. Evaluation of QuaSAR focused on two main factors: the overheads introduced by virtualisation; and the partitioning which routelets can provide between network flows.

The Experimental testbed consisted of a routing machine, acting either as a QuaSAR router or a standard router. This machine had a 1Ghz Pentium 3 processor, 1Gb of RAM and three 100Mbit/s Ethernet cards. This approximates a typical software routing machine at the time of this project, however, with more RAM to support a large number of routelets running simultaneously, and fewer line cards due to a lack of PCI slots in the desktop machine. A number of desktop PCs were used to inject traffic into the network. All machines ran SUSE Linux 9.2 with the 2.6.9 Linux Kernel and used the same versions of

<sup>5</sup>Other resources such as memory or disk access rate do not affect routelet’s performance significantly, due to their simple packet processing requirements, therefore, these resources are assigned statically at routelet creation.

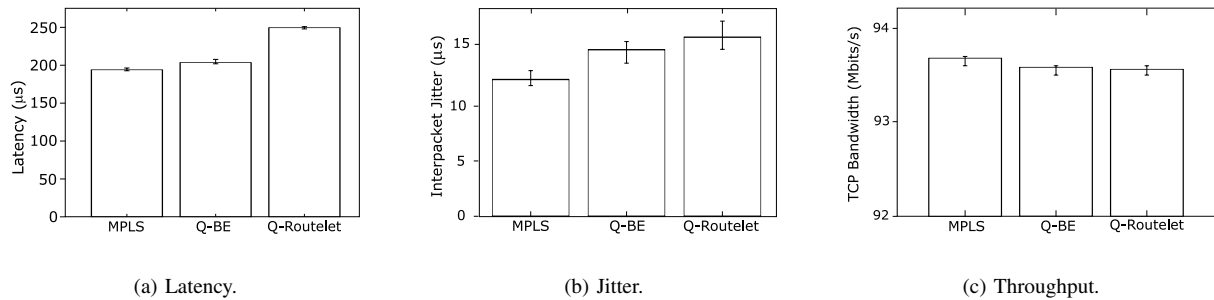


Fig. 4. Overheads incurred by virtualisation on different QoS parameters. The error bars show the minimum and maximum per run averages.

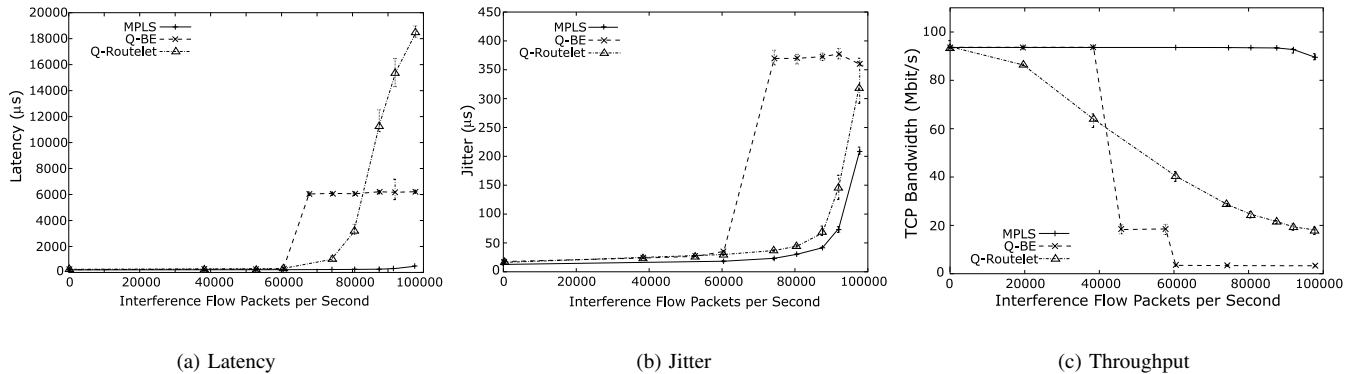


Fig. 5. The effect of increasing cross-flow traffic on the QoS provided by each router.

MPLS Linux and the RSVP-TE daemon (modified to support the virtual routelet architecture in QuaSAR’s case).

All experiments evaluate the performance of three different router setups: A standard MPLS router (MPLS); QuaSAR’s best-effort router (Q-BE), which is the standard MPLS router running within a Xen virtual machine (therefore, evaluating the overhead of Xen’s virtualisation layer); and Quasar’s routelet (Q-Routelet), evaluating the virtual routelet architecture.

#### A. Virtual Machine Overhead

To measure the overhead introduced by the virtualisation layer in QuaSAR, its performance was measured against that of a standard router in the key QoS parameters of latency, inter-packet jitter and single flow throughput. Timing measurements (latency and jitter) were performed by timestamping 70 byte packets as they left the source host and as they arrived in the sink host. All measurements were averaged over 5 different runs of at least 3000 packets each.

The average per-packet latency induced by each of the router types is shown by Figure 4(a). A packet takes, on average, 194µs to travel through the network using the standard MPLS router. Xen virtualisation introduces an overhead of 5.2% for packets processed by the Q-BE router. The average latency for packets processed by Q-Routelet is 249µs, 22% greater than Q-BE router. This additional overhead is not unexpected, since these packets must pass through two domains (demultiplexing in the main router, then the routelet), incurring additional domain context switches.

Figure 4(b) shows the average jitter<sup>6</sup> introduced by each

router type. The standard router introduces, on average, 12.4µs of jitter. Closer investigation suggests that this jitter is mainly caused by a timer event occurring roughly every second. When packets pass through the Q-BE router, the average jitter increases to 15.1µs. This increase seems to be caused by increased overhead in the regular 1 second timer events. Packets processed by Q-Routelet incur an average of 16.3µs of jitter. This increase is caused by more packets experiencing the higher jitter level, rather than an increase in the maximum jitter experienced by each packet. This suggests that more timer events occur in this setup, probably due to the increase in the number of executing OSs.

The maximum per-flow throughput supported by each router was investigated using the iPerf network performance measurement tool. The maximum throughput was calculated by sending a large amount of data over a TCP connection<sup>7</sup>, and recording its maximum transfer rate. Figure 4(c) shows the maximum single flow throughput achieved by each type of router, averaged over 5 runs. The results show that virtualisation does not significantly affect the throughput of a single network flow, with the decreases in throughput well within the bounds of experimental uncertainty.

#### B. QoS Flow Partitioning

The main purpose of using virtualisation techniques in QuaSAR was to improve the partitioning between network flows. This was evaluated by measuring the QoS provided to one flow, as a competing flow tried to use an increasing proportion of the router’s resources. The cross-flow traffic

<sup>6</sup>  $Jitter = \frac{\sum_{k=2}^n \sqrt{((received_k - received_{k-1}) - (sent_k - sent_{k-1}))^2}}{n-1}$

<sup>7</sup> The increased latency induced by the QuaSAR routers did not materially affect TCP’s flow control mechanism.

emulates a misbehaving flow which is sending more traffic than it requested resources for beforehand. It consists of an increasing rate of minimum sized packets, sent over different network line cards from the measured flow, to analyse QuaSAR's effectiveness at providing partitioning between the CPU resources required by different flows.

The effect of increasing the cross-flow's traffic on the QoS parameters of the measured flow is shown in Figure 5. Q-Routelet never outperforms a standard MPLS router, due to virtualisation overheads. However, it does perform better than a standard router running on Xen (Q-BE) over most of the range of each experiment. The fact that Q-Routelet outperforms Q-BE, even though it has much higher overheads, indicates that the virtual routelet architecture prevents cross-flow traffic from affecting the measured flow as significantly, thereby providing some partitioning between flows.

## VII. DISCUSSION

The Xen virtualisation software was built to support complex, multi-threaded, multi-address space operating systems, in order to provide *virtual server farms*. Some of Xen's design decisions, therefore, do not suit the virtual routelet design model. This section discusses lessons learned from this project and addresses changes which could reduce the overheads introduced by virtualisation as well as increasing the effectiveness of the virtual routelet router architecture.

1) *VM Context Switch Overhead*: The processing of each QoS packet requires two context switches between domains. Since Xen is designed to support multiple untrusted domains, security is an important design consideration. To this end, each domain runs within its own virtual memory address space, which increases the cost of each context switch because of moves between address spaces (causing overheads such as TLB flushes). QuaSAR routelets run a very small, known set of software and can therefore be trusted or verified. The overhead of these context switches could be substantially reduced by routelets sharing a single address space.

2) *Linux Operating System for each Routelet*: The QoS routelets have very simple requirements, therefore the use of a complex multi-user operating system, such as Linux, decreases the possible scalability of the QuaSAR router. A custom-built, minimal OS would be more appropriate.

3) *Routelet Access to Network Line Card*: Another major problem with the design of the QuaSAR router is that it requires packets to be passed between domains multiple times, due to only a single domain having access to the network device. Although Xen uses page table manipulation to achieve this packet transfer, it still incurs a significant overhead. Routelets could be provided with direct access to the physical network through a user accessible network card [14].

4) *Classifying Packets*: A large proportion of each packet's processing time is spent classifying the packet and deciding which routelet processes that packet's flow. This processing time cannot be assigned to the flow's routelet, because the packet has not yet been assigned to its particular flow. An im-

proved classification method, or using hardware classification built into the network line card, could reduce this problem.

5) *Soft Real Time CPU Scheduler*: The purpose of the QuaSAR router was to evaluate the effectiveness of a virtual routelet approach. The lack of a soft real time CPU scheduler for Xen prevented QuaSAR from fully following the design philosophy of the virtual routelet approach. It is expected that the partitioning performance of QuaSAR would be markedly improved with the use of a stable soft real time scheduler.

## VIII. CONCLUSION

The evaluation of the QuaSAR prototype router suggests that it is indeed feasible to use virtualisation techniques within a network router, without overly compromising the router's ability to route high-throughput, low-latency traffic. Routing through a QuaSAR routelet provided some improvements in partitioning, over the QuaSAR best-effort router, but virtualisation overheads prevented it from ever reaching the level of performance achieved by a standard software router. This suggests that the virtual routelet architecture has some promise, especially with respect to access routers, if these overheads can be reduced. The results of the QuaSAR router experiments provide valuable insight into the areas of virtualisation which could be enhanced to reduce these overheads and improve the effectiveness of the virtual routelet architecture.

## ACKNOWLEDGEMENT

The authors wish to thank Jonathan Paisley for his assistance in setting up the network testbed. We also thank the Xen and Click Modular Router communities for their assistance.

## REFERENCES

- [1] X. Xiao and L. M. Ni, "Internet QoS: A big picture," *IEEE Network*, vol. 13, no. 2, pp. 8–18, March 1999.
- [2] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *IEEE/ACM Trans. Netw.*, vol. 2, no. 2, pp. 137–150, 1994.
- [3] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network Magazine*, 1993.
- [4] A. T. Campbell, H. G. D. Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A survey of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 2, pp. 7–23, 1999.
- [5] P. Chandra, A. Fisher, C. Kosak, T. Ng, P. Steenkiste, E. Takahashi, and H. Zhang, "Darwin: customizable resource management for value-added network services," in *Sixth Int. Conf. on Network Protocols*, 1998.
- [6] L. Peterson, "NodeOS interface specification," Active Networks NodeOS Working Group, Technical Report, February 1999.
- [7] M. Zec, "Implementing a Clonable Network Stack in the FreeBSD Kernel," in *Proc. of USENIX Annual Technical Conference*, June 2003.
- [8] R. McIlroy, "Network Router Resource Virtualisation," Master's thesis, University of Glasgow, 2005.
- [9] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031 (Proposed Standard), Jan. 2001.
- [10] 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 *SOSP*, 2003.
- [11] A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and performance in the Denali isolation kernel," *SIGOPS Operating Systems Review*, 2002.
- [12] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, fidelity, and containment in the potemkin virtual honeyfarm," in *SOSP*, 2005.
- [13] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The Click modular router," in *Symposium on Operating Systems Principles*, 1999.
- [14] I. Pratt and K. Fraser, "Arsenic: A user-accessible gigabit ethernet interface," in *INFOCOM*, 2001, pp. 67–76.