Performance Isolation in Network and Computing Systems with Multiple Inputs

Jack Brassil
HP Laboratories, Princeton, NJ, USA
Email: jack.brassil@hp.com

***Abstract*— The growing use of virtualization technologies in settings such as multi-tenant compute and storage clouds challenges us to both specify and enforce the isolation of clients sharing network and compute resources. In this paper we propose a novel analytical measure of performance isolation for shared resource systems serving multiple traffic flows or computing workloads. By basing our isolation measure on well-known results from network calculus, we demonstrate how isolation metrics can be calculated for flows traversing both individual system elements and networks of those elements. We argue that this measure facilitates the design of systems capable of ensuring that clients can realize a specified isolation target. We present illustrative examples of how our quantitative isolation measure can be used to compare the isolation properties of alternative instantiations of resource sharing systems ranging from experimental testbeds to dynamically-instantiated compute clouds. Finally, we show how next generation resource allocators can be designed to preserve the isolation of clients by either routing newly arriving flows, or re-arrange existing flows.**

***Index Terms*— network calculus, deterministic queueing theory, security and privacy, QoS, fairness**

I. INTRODUCTION AND MOTIVATION

Consider a heavily utilized experimental network testbed (e.g., Emulab [1], PlanetLab [2]) supporting multiple client experiments that must dynamically instantiate a networked computing system based on a resource request from a newly arriving client. Suppose that sufficient communications and computation resources are available such that the testbed's resource allocation system identifies two feasible candidate system instantiations satisfying the client's resource demands. One candidate instantiation shares certain resources (e.g., link bandwidth, virtual machines within a physical compute server) with one set of existing test facility clients, while the second candidate implementation shares resources with another set of existing clients. Assuming that all client workloads and traffic demands are heterogeneous and unknown, how does the resource allocator determine which of the implementations will jointly offer *all* clients the maximum possible performance isolation from each other?

Network systems researchers ability to answer this question has been hampered by both the lack of a formal definition of performance isolation, as well as lack of consensus on how performance isolation should be measured.

Manuscript received August 14, 2011; revised October 5, 2011; accepted October 7, 2011.

Yet there is immediate urgency to confront this problem; elastic computing in emerging systems such as multi-tenant compute clouds require system designers to balance the requirement to isolate clients and the desire to share resources efficiently. By developing a method to specify and enforce experiment isolation, clients of future network testbeds such as GENI [3] can enjoy a greater degree of experiment repeatability, a highly desirable feature often not found in current generation testbeds.

A shared resource system should ideally provide each client a measure of its isolation *fidelity*, and meet a specific quality-of-isolation guarantee. Further, in the absence of mis-configuration or other failure, this system isolation property should be demonstrable, if not provable. The demand for stronger, measurable isolation is growing most notably in virtualized settings (e.g., large-scale experimental testbeds, compute and storage clouds) where co-resident clients may have no pre-established trust relationship, and may even have no knowledge of the existence or identities of other clients. In such a setting, the risk of isolation failure is unknowable, and this risk is a barrier to acceptance of these new technologies.

The isolation properties of a system (or a system element) describe the relative 'apartness' or 'separation' that the system offers to two (or more) concurrent system workloads. *Performance isolation* characterizes how effectively a system maintains the performance separation of workloads while serving those workloads. An immediate goal of our work is to draw attention to performance isolation as a crucial, first-class property of resource sharing systems, deserving of the same close attention the research community has paid to other resource sharing system properties such as *fairness*.

The absence of a theoretical framework to study the isolation properties of systems has left the evaluation of isolation to be largely an empirical science. This current state of isolation metrology is troubling for system design; there exists no formal approach to analyzing isolation in a complex system. As a result isolation can not be increased systematically, nor can the isolation of comparable systems implementations be compared in a rigorous way. We seek a theoretical framework for studying the isolation properties of complex systems that would compliment the valuable empirical work which is performed, to form a complete isolation characterization.

In this paper we propose such an analytical framework for studying performance isolation. We do not seek

doi:10.4304/jcm.7.1.39-51

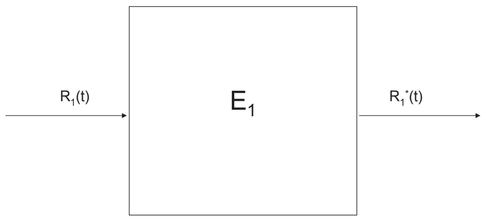

Figure 1. A network element with a single arriving and departing flow.

to consider the general problem of categorizing overall system isolation here – nor isolation in the presence of system failures – but instead focus on the specific problem of performance isolation. We build on results presented in earlier work [4] where we introduced a measure of isolation for network systems with contending flows. Here we extend that work to systems with an arbitrary number of contending workloads, and consider how our analytical measures can be exploited by resource allocators in experimental testbeds and clouds. We aspire not just to categorize the performance isolation property of systems, but influence the system design process to incorporate design guidelines enabling a network system developer to realize a desired 'quality-of-isolation'.

This paper is organized as follows. The next section introduces the reader to some of the basic conventions, approaches and results of the study of network calculus. We make no attempt to extend the theory itself, but instead show how the theory of deterministic queues can be used to measure performance isolation not just in networks, but more generally for complex computing and communication resource sharing systems. Section III introduces our proposed measure of performance isolation for simple network elements with N input flows. Next we summarize how the measure can be applied element-by-element to calculate an isolation measure for complex systems. Sections V and VI present some challenging examples of the application of this approach to experiment isolation in actual systems. Here we argue that resource allocators can make informed decisions about the relative isolation provided by alternative instantiations of a requested experiment topology. The final sections summarize our contribution, discuss related work, and highlight some opportunities for future work.

II. Analytical Isolation Metric

To develop an analytical measure of performance isolation we begin by recalling some elementary results from network calculus; readers seeking additional background are referred to the textbook by Leboudec and Thiran [5] – whose conventions we adopt here – or early work by Cruz [6] [7].

Figure 1 depicts a system element E_1 with a single arriving and departing flow $R(t)$ and $R^*(t)$. For the extent of this manuscript we limit our discussion to elements that are lossless, well-behaved, and work-conserving; we will return to discuss these initial assumptions in Sections VII and VIII. Such a system element could model a communication or network element, where arriving and departing flows correspond to bit or packet streams (i.e., network traffic), or a computing element where workloads describe newly arriving and completed tasks (i.e., compute jobs). For the moment, we will suppose that we are modeling a network element – and refer to flows and traffic – while recognizing that the approach also models the behavior of shared computing elements. Unless otherwise noted, we will restrict our attention to flows that are non-decreasing and described by a fluid model (i.e., continuous functions in continuous time).

Suppose that flow $R(t)$ is constrained by an arrival curve $\alpha(t)$ such that for all time $s \leq t$ we have

$$R(t) - R(s) \leq \alpha(t-s). \tag{1}$$

The arrival curve serves as an envelope for the admitted traffic; we say that $R(t)$ is constrained by an arrival curve $\alpha(t)$ iff $R \leq R \otimes \alpha$, where the operator $\otimes$ is the min-plus convolution given by

$$R \otimes \alpha(t) \triangleq \inf_{0 \leq s \leq t} \{R(t-s) + \alpha(s)\}. \tag{2}$$

We will frequently use the common and important set of affine arrival curves given by

$$\alpha(t) = \gamma_{r,b}(t) \triangleq \begin{cases} rt + b & t \geq 0 \\ 0 & otherwise. \end{cases} \tag{3}$$

Flows that have been regulated by the well-known leaky bucket traffic shaper with drain rate r and bucket size b follow the arrival curve $\gamma_{r,b}(t)$ (see Proposition 1.2.3 of [5], pg. 11).

We say that the element E_1 offers a service curve β to the arriving flow iff the departing flow satisfies $R^* \geq R \otimes \beta$. A common service curve is the rate-latency curve $\beta_{R,T}$ given by

$$\beta_{R,T}(t) \triangleq \begin{cases} R\ (t-T) & t \geq T \\ 0 & otherwise. \end{cases} \tag{4}$$

Two key results we will exploit from network calculus are the well known upper bounds on the backlog and delay of any α-constrained flow arriving to a lossless system providing a service curve β. The backlog for all t is given by

$$R(t) - R^*(t) \leq \sup_{s \geq 0} \{\alpha(s) - \beta(s)\}. \tag{5}$$

The delay through the system at any time t is bounded by

$$\delta(t) \leq \sup_{s \geq 0} \{\inf_{\tau \geq 0} \{\tau \colon \alpha(s) \leq \beta(s+\tau)\}\} \tag{6}$$

The following example helps make these results more accessible and intuitive. First, if an α-constrained arrival process arrives to a network element providing a minimum guaranteed service, then clearly the backlog

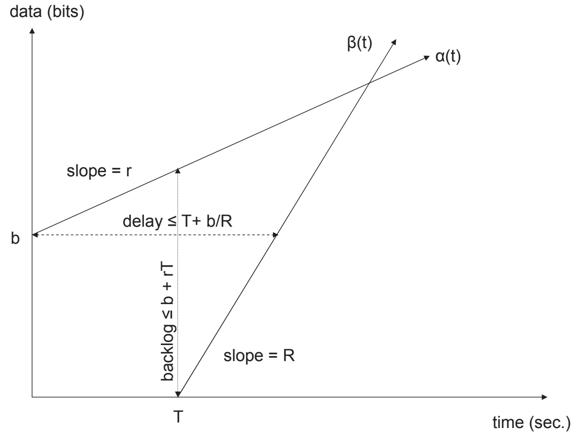

Figure 2. The largest horizontal and vertical deviations between arrival and service curves establish worst case delay and backlog.

and worst case delay will both be bounded. Suppose, for instance, that a flow constrained by an arrival curve following $\gamma_{r,b}(t)$ passes though a network element (e.g., leaky bucket) with service curve $\beta_{R,T}(t)$, with $R \geq r$. Figure 2 depicts such a result, and shows the graphical representation of the arrival and service curves.

While not shown in the figure, the arrival process is guaranteed to stay beneath the envelope α, and the departure process above the envelope $R \otimes \beta$. The backlog (Eq. 5) corresponds to the *vertical deviation* between the arrival and service curves, with a maximum backlog of

$$b + RT \qquad \textit{bits.} \tag{7}$$

The delay corresponds to the horizontal deviation between the arrival and service curves, which is bounded (Eq. 6) by

$$T + \frac{b}{R} \qquad \textit{seconds.} \tag{8}$$

We pause to reinforce the critical idea we apply throughout this manuscript. Informally stated, Eq. 6 states tha given a system element that provides a minimum guaranteed service to a properly constrained arriving flow, we can readily write an upper bound on the delay suffered by any packet passing through the element. In some cases such as Eq. 8, the delay bound can be both tight and simple to write analytically. We will use this delay bound as our preferred measure of system performance, and show how to derive from it an isolation measure for systems serving two or more workloads.

III. Performance Isolation with Multiple Inputs

Let's begin by considering the general case of the performance isolation of a single workload from a set of other concurrent workloads. Consider a network element E_N with N arriving flows $\vec{R} = \{R_1, R_2, ..., R_N\}$. Let $I_{R_i,\vec{R_x}}$ be the measure of isolation that the element offers

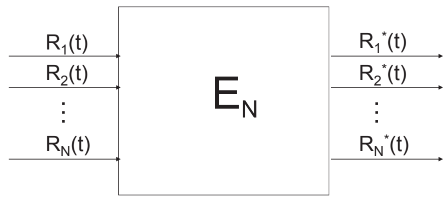

Figure 3. A network element with N arriving and departing flows.

flow $R_i(t)$ from a subset of other flows $\vec{R_x} \subset \vec{R^-}$, where $\vec{R^-}$ is the set of all arriving flows other than the i^{th} flow (i.e., $\vec{R} = \vec{R^-} \cup \{R_i\}$).

Let $D_{R_i,\vec{R_x}}$ ($D_{R_i,\vec{R_x}=\vec{0}}$) be the maximum delay suffered by the i^{th} flow given the presence (absence) of the flows in $\vec{R_x}$. Define the *performance isolation* of R_i from the subset of other flows $\vec{R_x}$ as

$$I_{R_i,\vec{R_x}} \triangleq 1 - \frac{D_{R_i,\vec{R_x}} - D_{R_i,\vec{R_x}=\vec{0}}}{D_{R_i,\vec{R_x}}} \tag{9}$$

$$= \frac{D_{R_i,\vec{R_x}=\vec{0}}}{D_{R_i,\vec{R_x}}}. \tag{10}$$

Hence, our performance isolation metric is based on the difference in worst case delay seen by a marked flow in the presence and absence of a set of concurrent, interfering flows. In a work conserving system we have

$$D_{R_i,\vec{R_x}} \geq D_{R_i,\vec{R_x}=\vec{0}} \geq 0 \tag{11}$$

so clearly we have

$$0 \leq I_{R_i,\vec{R_x}} \leq 1. \tag{12}$$

Alternately, we define the *exposure* [1] of R_i to $\vec{R_x}$ as the complement of the isolation, i.e.

$$E_{R_i,\vec{R_x}} \triangleq 1 - I_{R_i,\vec{R_x}}. \tag{13}$$

In general, completely reporting the performance isolation metrics of flow i traversing element E_N with $N-1$ concurrent flows requires specifying an $(N-1)!$-tuple of isolation measures against each possible subset of flows. Fortunately, we will rarely find that level of detail necessary; in practice we are typically interested in a flow's isolation from either *all* other flows, or a specific concurrent flow.

Since we will be examining the case of N concurrent flows let's take a moment to clarify how we treat flows that are concurrent with the flows on which we are measuring isolation. In a system with 3 or more inputs we seek to measure the isolation of the i^{th} flow against a proper subset of the remaining flows; that is, we are *not*

[1] Using *exposure* helps us overcome the common but inconvenient use of the expression *lack of isolation*.

logically turning off all other flows (i.e., $|\vec{R_x}| < |\vec{R^-}|$). Hence, since some inputs are active, they will naturally affect the measure of system isolation provided to the other inputs. This will be made more clear with an example in a moment.

Discussion: Eq. 10 is inspired by the ad hoc, empirical isolation measures where a performance measure (e.g., throughput) of one workload is taken in the presence and absence of one (or more) other workloads, with these typically chosen to degrade the performance of the first workload to the largest degree. As we mentioned in Section I, a difficulty with this approach is the inability to know that the chosen interfering workload is in fact the one that would degrade performance the most. Indeed, even if such a workload was correctly chosen, actually applying it in a system setting might be difficult or impractical; simpler, easily generated workloads are often substituted in practice instead. Here, however, we make no attempt to hypothesize what such a workload would be, and instead use an analytically derived bound on worst case delay for *all* competing workloads conforming to a specified arrival curve.

Let's pause to immediately address a potential issue with our isolation measure. Many readers are aware that a common objection to the use of delay bounds derived from network calculus is that – regardless of whether the bounds are tight – they are *pessimistic*. Here, however, our isolation measure relies on a ratio of delay bounds, and Eq. 12 establishes that the ratio is normalized. Perhaps more importantly for the study of isolation, exploiting network calculus enables our isolation measure to hold under a wide class of inputs. In contrast, our ability to generalize results from empirical assessments is often quite limited.

Representing system behavior through arrival and service curves as depicted in Fig. 2 enables us to arrive at a powerful, graphical interpretation of flow isolation. Fig. 4 graphically illustrates how our isolation metric is derived as a consequence of what is effectively a change in the element's service curve. Under a lightened workload the shape of the element's service curve β deforms to β^*. For an affine curve, the minimum delay may reduce from T to T^* and the service rate may increase from R to R^*. The second term of Eq. 14 is simply a normalized measure of the resulting decrease in worst-case delay, as labeled Δ in the figure.

Before moving on to examples of the analysis of systems with N flows, let's first focus on the simplest case of an element with 2 inputs, i.e., $\vec{R} = \{R_1, R_2\}$. Eq. 10 is simply a normalized measure of the decrease in a flow's worst case delay bound that can be achieved by removing the second flow from the system. If there is no decrease in delay, we say that the first workload is (performance) isolated from the 2^{nd} workload, i.e., $I_{R_1,R_2} = 1$. If we have $I_{R_2,R_1} = 1$ and $I_{R_1,R_2} = 1$, we say that the two workloads are isolated from each other. In practice, achieving such a result is impossible in a practical system sharing resources efficiently, though we strive to design systems with the largest possible isolation metric for each workload.

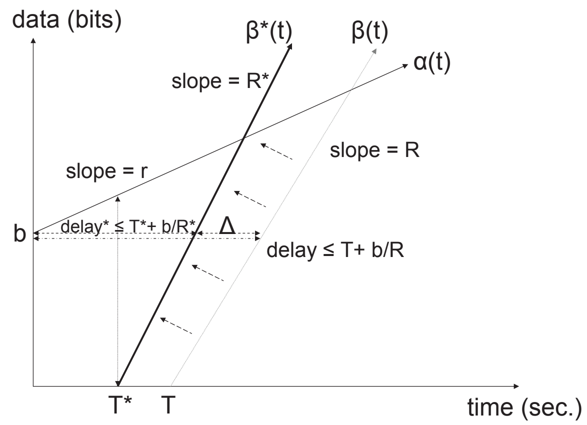

Figure 4. Under a lightened workload the element's service curve may effectively shift. The reduction in worst delay Δ is the basis for our isolation metric.

From Eq. 10 we write the *performance isolation* of R_1 from R_2 as

$$I_{R_1,R_2} \triangleq 1 - \frac{D_{R_1,R_2} - D_{R_1,0}}{D_{R_1,R_2}} \tag{14}$$

$$= \frac{D_{R_1,0}}{D_{R_1,R_2}} . \tag{15}$$

Here we take D_{R_1,R_2} to be the bound on the delay of R_1 in the presence of R_2, and $D_{R_1,0}$ to be the bound on delay given the absence of the second input ($R_2 = 0$). Hence, our performance isolation metric is the difference in worst case delay seen by a flow in the presence and absence of the interfering flow.

Completely reporting the performance isolation properties of element E_2 requires specifying both I_{R_1,R_2} and I_{R_2,R_1}, where

$$I_{R_2,R_1} = \frac{D_{R_2,0}}{D_{R_2,R_1}} \tag{16}$$

describes the isolation of flow R_2 from flow R_1.

A. Priority Queueing System

Now let's consider an example of determining the analytical performance isolation of flows in a system with 3 inputs. Suppose we have a non-preemptive priority node with constant rate C serving 3 input flows – high, medium and low priority flows labeled R_H, R_M and R_L. Service for the high priority flow takes precedence; a packet in the middle priority queue is only serviced if the higher priority queue is empty, and a packet in the low priority queue is only serviced if both higher priority queues are empty. Suppose that l_{MAX} is the largest packet size for any flow, and that the high, medium and low priority flows are constrained to be $\gamma_{r_h,b_h}(t)$, $\gamma_{r_m,b_m}(t)$ and $\gamma_{r_l,b_l}(t)$.

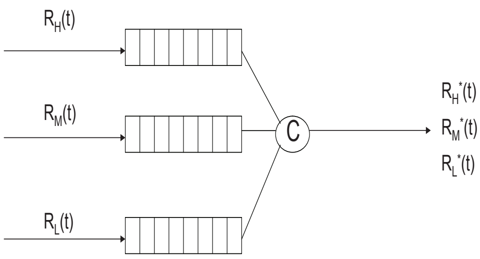

Figure 5. A non-preemptive priority queuing system with 3 inputs.

Following the proof of the 2 flow non-preemptive priority node (see Proposition 1.3.4 of [5], pg. 21), the appendix sketches a proof that the high, medium and low priority flows are guaranteed rate-latency curves of

$$\beta_{C,\frac{l_{MAX}}{C}}, \qquad \textit{high priority} \tag{17}$$

$$\beta_{C-r_h,\frac{l_{MAX}+b_h}{C-r_h}}, \textit{and} \qquad \textit{medium priority} \tag{18}$$

$$\beta_{C-r_h-r_m,\frac{b_h+b_m}{C-r_h-r_m}}. \qquad \textit{low priority}. \tag{19}$$

Hence the delays seen by each flow are given by Eq. 8 as

$$D_{R_H,\{R_M,R_L\}} = \frac{l_{MAX}}{C} + \frac{b_h}{C} \tag{20}$$

$$= \frac{l_{MAX}+b_h}{C}. \tag{21}$$

$$D_{R_M,\{R_H,R_L\}} = \frac{l_{MAX}+b_h}{C-r_h} + \frac{b_m}{C-r_h} \tag{22}$$

$$= \frac{l_{MAX}+b_h+b_m}{C-r_h}. \tag{23}$$

$$D_{R_L,\{R_H,R_M\}} = \frac{b_h+b_m}{C-r_h-r_m} + \frac{b_l}{C-r_h-r_m} \tag{24}$$

$$= \frac{b_h+b_m+b_l}{C-r_h-r_m}. \tag{25}$$

Let's now write two isolation measures for this system. Suppose we seek to find the isolation of the medium priority flow from *both* the high and low priority flows (i.e., $I_{R_M,\vec{R_x}} = I_{R_M,\{R_H,R_L\}}$). In the absence of the higher priority and lower priority flows, the delay of the medium priority flow is given by setting $l_{MAX} \to 0$, $b_h \to 0$, and $r_h \to 0$ in Eq. 22, yielding

$$D_{R_M,\vec{R_x}=\vec{0}} = \frac{b_m}{C}. \tag{26}$$

The isolation of the medium priority flow from the 2 other flows is found by inserting Eqs. 22 26 into Eq. 10 to find

$$I_{R_M,\{R_H,R_L\}} = \frac{C-r_h}{C}\,\frac{b_m}{l_{MAX}+b_h+b_m}. \tag{27}$$

Note that the isolation of the medium priority flow is heavily dependent on the rate of the high priority flow (r_h), as we expect. Though difficult to see in this expression, the medium priority flow is strongly isolated from the lower priority flow, depending only on that flow's maximum packet length.

Now let's consider the isolation of the low priority flow from just the medium priority flow. From our definition in Eq. 10 and Eq. 24 we have

$$I_{R_L,\{R_M\}} = \frac{C-r_h-r_m}{C-r_h}\,\frac{b_h+b_l}{b_h+b_m+b_l}. \tag{28}$$

Note that the isolation of the low priority flow from the medium priority flow is heavily dependent on *both* higher priority flows, through the medium priority flow's exposure to the high priority flow.

B. PGPS Queueing System

Let's next look at the isolation performance of a network element that promises to offer similar isolation to each flow. Due to its importance in assuring quality-of-service and achieving fair bandwidth sharing, consider a node with fixed service rate C employing a Packetized General Processing Scheduling (PGPS) discipline. Let the maximum packet length of any flow be L packets. Suppose that the i^{th} arriving flow is allocated a rate $c_i = C\frac{\phi_i}{\sum_j \phi_j}\ :\ i = 1,2...,N$. Then the element offers each arriving flow a service curve $\beta_{c_i,\frac{L}{C}}$ ([5], p. 68). If the i^{th} arriving flow is constrained by γ_{r_i,b_i}, then we can employ Eq. 8 to find that the maximum delay the i^{th} flow suffers is given by

$$\frac{L}{C} + \frac{b_i}{c_i} \quad . \tag{29}$$

Suppose that there are a fixed number $N > 2$ flows traversing the element, and each is backlogged and granted an identical rate $c_i = \frac{C}{N}$, with arriving flow rates satisfying $r_i < c_i$. Then the isolation offered between flows i and j is

$$\frac{\frac{L}{C} + \frac{b_i}{\frac{C}{N-1}}}{\frac{L}{C} + \frac{b_i}{\frac{C}{N}}} \approx \frac{N-1}{N}. \tag{30}$$

Hence, the isolation a PGPS node offers each individual flow from any other single flow approaches unity (total isolation) as the (fixed) number of flows increases.

Since the PGPS server shares bandwidth among backlogged flows, we observe that the isolation of flows i and j is affected by the presence of other flows. Suppose that at any instant there are k active flows, $k < N$, each with an identical bandwidth share (i.e., $\phi_i \equiv \phi = \frac{C}{k},\quad k = 1,2,...,N$). First, let's mark one flow for consideration, the m^{th} flow. Suppose that the flow initially passes through the server with $N-1$ other flows, and $N-k-1$ flows vanish.[2] The marked flow initially receives a rate of $\frac{C}{N}$, but then receives a higher rate $\frac{C}{k}$. Though still

[2]We have not formally defined what it means for a flow to vanish. To be consistent with our definition, the set $R_{\vec{x}}$ remains unchanged, but $N-k-1$ flows are no longer receiving service.

receiving a 'fair share' of server resources, the resource assigned to the flow increases, and clearly the flow is not isolated from other flows. By our measure the system offers the marked flow an isolation of

$$\frac{\frac{L}{C} + \frac{b_m}{\frac{C}{k}}}{\frac{L}{C} + \frac{b_m}{\frac{C}{N}}} \approx \frac{k}{N}. \tag{31}$$

This result, where changes to a PGPS node to improve service clearly worsen the isolation between flows, vividly depicts the tradeoff to be made between performance isolation and system performance in shared resource systems.

IV. Applications to Networked Computing Systems

Up to this point we have focused on defining a measure of performance isolation, and applying this measure to simple system elements in isolation with two or more applied demands. But the strength of a metric based on network calculus lies in its applicability to the analysis of systems comprising many such elements [7].

In particular, the *node concatenation theorem* [5] tells us that a series connection of two elements E_x and E_y with service curves β_x and β_y offers an aggregate service curve of $\beta_x \otimes \beta_y$. If the aggregate service curve is known, then the isolation provided by the aggregate element can be written directly from Eq. 10. But what if the aggregate's service curve is unknown? It follows that the worst-case delay of a flow passing through a sequence of elements – and hence the isolation – can be upper bounded by the sum of the delay (or isolation) of the individual elements. Note, however, that the sum of the isolation metrics of the two elements may represent a larger value than if the isolation metric could be written directly from the aggregate's service curve.

The concatenation theorem tells us that a bound on isolation of the aggregate system does not depend on the order in which the elements are connected. Hence, the concatenation theorem lays the groundwork for analyzing the isolation performance of systems, and the first result available to us is that the order of elements in an aggregate can be interchanged without affecting the overall isolation property. Though a seemingly simple observation, this represents the first *design rule* we can apply to the study of system isolation.

Two other techniques we borrow from the study of deterministic queueing systems will also be invaluable. First, for the purposes of calculating the isolation of a system, we can logically replace a nonlinear element with service curve β_{NL} with a linear element β_L with a *dominating* service curve, which we define to be a service curve satisfying

$$\int_0^s \beta_L \otimes R(t)\, dt \geq \int_0^s \beta_{NL} \otimes R(t)\, dt, \tag{32}$$

for all time s. Second, when we encounter an element with *unknown* service curve, we can logically replace it with an element with a known but dominating service curve for the purposes of calculating system isolation.

The final tools required in our system performance isolation calculation toolkit is the identification of the specified *workflow* of interest, and the selection of an aggregation algorithm. The workflow is the set of computing and communication elements servicing the workload. For example, consider a single client-server transaction (e.g., a file upload). The corresponding workflow might comprise task execution on a multi-tasking client computing 'element', followed by a flow traversal of multiple network elements with potentially interfering cross-traffic, followed by task execution at a web server also occupied with numerous other transactions. Suppose that for each of N elements visited by the workflow we can calculate the isolation $I_{R_i,\vec{R_{y_i}}}$ of the specified flow R_i (arriving to the i^{th} element) relative to some distinct subset of other arriving workloads $\vec{R}_y$. We then identify a measure of the workflow's end-to-end isolation by combining or aggregating individual measures at each element traversed. For example, one possible metric is the linear combination of the exposure of the specified workflow at each element visited, i.e.,

$$\sum_{i=1}^{N} a_i \cdot (1 - I_{R_i,\vec{R_{y_i}}}), \tag{33}$$

where $0 \leq a_i \leq 1, \quad i = 1, 2..., N$. That is, we take as our measure of system isolation the weighted sum of isolation measures of the workload passing through each visited element. Hence, when comparing the isolation of two alternative system implementations, the implementation offering the lowest composite exposure would be preferred.

Note that the decision to form a composite 'figure of merit' by using Eq. 33 is flexible and can be determined by application needs; any alternate composition algorithm might be appropriate. For example, another natural approach would be to aggregate the exposure at compute elements and communications elements separately, and compare similar systems using both measures. Irrespective of the aggregation approach selected, obtaining an end-to-end isolation measure allows us to focus on a design intended to minimize exposure. Indeed, we can now identify an *isolation budget* for a system, and consider the selection of elements that differ in certain properties (e.g., cost, flexibility) that meet our overall isolation budget. In sum, we are now armed with means of designing systems to meet a desired isolation fidelity.

Let's next consider the isolation of experiments on complex systems such as shared resource experimental testbeds. In the following examples, we highlight the power of an isolation metric to influence overall system design to maximize the isolation enjoyed by system users.

V. Using Isolation Measures in Resource Allocators

Resource allocation systems in experimental network computing systems testbeds such as Emulab and PlanetLab receive asynchronous client requests to instantiate experimental system topologies. To simplify allocator implementation and speed the identification of a feasible set of compute and network resources, a client's resource specification (*rspec*) has historically been kept lean. Clients often lack controls (or hints) to specifically force the allocator to include or reject certain components, or indicate preferences for details such as VM placement. For example, Amazon's EC2 [34] technical literature argues that a desire to simplify system instantiation warrants a client's lack of VM placement control. Further, today's allocators (e.g., Emulab's *assign*) use admission control liberally; inability to map a resource request to available hardware in a congested sytem leads to rejection of the client's entire request. Designers of allocation systems for future testbeds (e.g., GENI) envision richer and more expressive client specifications, with perhaps the ability for clients to negotiate partial or less favorable instantiations. For example, a client might be amenable to an incomplete system request fulfillment (e.g., during a development or debugging phase) rather than suffer a complete rejection.

On the other hand, Emulab and emulab-based systems (e.g., DETER [8]) enable experimenters to achieve a relatively high degree of inter-experiment isolation through a combination of admission control, permitting the allocation of dedicated end computing systems to a single experiment, and carefully limiting the use of shared resources to effectively achieve an acceptable 'quality-of-isolation.' The most common example of the latter is the way that Emulab's control infrastructure dynamically creates multiple VLANs to enable one or more ethernet switches to segregate traffic associated with different experiments.

Our next goal is to explore how isolation measures can enhance the articulation of client preferences in rspecs, and enable resource allocators to evaluate candidate instantiations offering varying measures of performance isolation. By carefully creating models of testbed elements, we can examine the degree of isolation between experiments, and in particular assess the relative degree of isolation offered in comparable but different implementations.

A. Flow Placement

Let's begin by considering the simplest example of how isolation measures can be used for the selection of a single, preferred shared element in a testbed. Consider a new experiment request requiring the use of a shared network element (e.g., a trunked VLAN link, a router egress link). Suppose that exactly two such shared network elements are available for assignment by the resource allocator; switch 1 has an egress link capacity C_1, and switch 2 has an egress link capacity C_2. Assume that each switch is being partially used by exactly one pre-existing flow, R_1 and R_2, with each flow associated with a different experiment.

Suppose that our model for the shared network link in each experiment is a non-preemptive priority queue; this would be an acceptable model of an element such as a switch or router port that is QoS-enabled. Let's say that the pre-existing flows are assigned as the high-priority flows in each system, and the new flow we seek to assign to either switch will be a low-priority flow. Let the flows R_1 and R_2 be constrained as γ_{r_1,b_1} and γ_{r_2,b_2}. Suppose that the flow assigned by the new arrival is R_N that follows γ_{r_N,b_N}. For convenience let us assume that all flows have the same maximum packet length l.

The *flow placement* problem is as follows: to which switch should we assign the newly arriving flow to minimize its isolation from pre-existing flows? Figure 6 depicts the placement problem. Clearly the new flow is only exposed to the flow in the element chosen (i.e., not exposed to the other switch's flow). It is easy to show that for a 2-flow non-preemptive priority element the high priority flow is isolated from the low priority flow as

$$I_{R_H,R_L} = \frac{b_h}{l+b_h}, \tag{34}$$

while the low priority flow is isolated from the high priority flow as

$$I_{R_L,R_H} = \frac{b_l}{b_l+b_h}\frac{C-r_h}{C} \ . \tag{35}$$

Eq. 35 tells us that the newly arriving flow will minimize its isolation from existing flows by being assigned to switch 1 if

$$\frac{b_N}{b_1+b_N}\frac{C_1-r_1}{C_1} < \frac{b_N}{b_2+b_N}\frac{C_2-r_2}{C_2}, \tag{36}$$

and switch 2 otherwise. In the simple case where both pre-existing flows have the same burstiness and each of the two shared links have identical capacity, Eq. 36 confirms our intuition that isolation of the new flow is minimized by assigning it to the switch whose existing flow has the lowest rate. Hence, this simple example demonstrates that future resource allocators can calculate isolation metrics for both current and newly arriving experiments, and instantiate a system – where feasible – to meet or maintain a client's desired quality-of-isolation.

B. Flow Re-arrangement

As experiments are de-instantiated in experimental testbeds, resources are returned to an available resource pool. In principle these resources are allocatable to both new resource requests associated with experiments to arrive in the future, as well as to existing experiments. Hence it is possible to engineer resource allocators to modify existing instantiations of experiments to reduce isolation. Consider the following simple example; two clients each use a virtual machine on a single physical processor. If a second physical processor becomes available, a VM could be migrated to that newly available, unused processor, lowering the exposure of the experiments' workloads to each other.

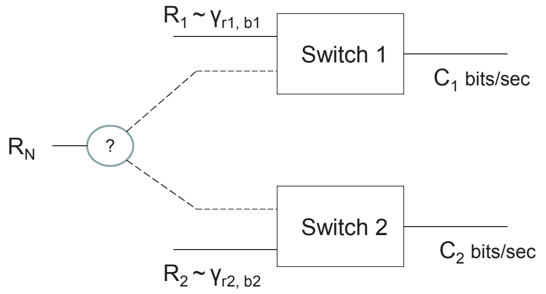

Figure 6. An envisioned resource allocation system uses isolation measures to assign a flow to a resource to minimize the newly arriving client's exposure to existing flows.

In practice such actions are rarely taken; implementing such a resource allocator is difficult, and re-instantiation of experiments risks disrupting a running experiment. Hence, re-instantiations are generally limited to those cases where an experimenter explicitly *modifies* an existing experiment, an action typically done while the experiment is halted.

A potentially less disruptive approach to maintaining network isolation does not involve a complete experiment re-instantiation, but the simpler step of rearranging flows to exchange shared resources with other already instantiated resources that promise less exposure to existing workloads. In some cases such a modification might involve only reconfiguring existing VLANs, an operation that can be completed with minimal disruption.

Let's consider a simple example. Let 2 identical switches each represent a resource to be shared by 6 flows associated with 6 distinct experiments. Suppose that each switch is well-modeled by a PGPS queueing system, and each flow is identically constrained and seeks an identical bandwidth and quality-of-isolation from a shared switch. If 3 flows are initially assigned to each switch, then each flow on each switch is identically isolated from the other 2 flows. Now suppose that 2 flows on one switch vanish. While this leaves the remaining flow fully isolated from the flows on the other switch, those flows suffer relatively worse quality-of-isolation. Re-routing one flow from the switch with 3 flows to the switch with 1 flow clearly rebalances the system load, returning all flows to an identical isolation profile.

Though trivial, this example illustrates that flow rearrangements can be performed with minimal disruption in complex systems, where a resource allocator calculates the effect of rebalancing flows such that isolation can be preserved even as the system's future workloads evolve.

VI. Applications to Experimental Testbeds

We next develop isolation models for very simple but useful elements such as VLANs and traffic shapers, and show how these models can be combined to make *isolation-aware* resource allocation decisions in simple systems. Consider a resource allocator for an experimental testbed (e.g., Emulab) that seeks to instantiate 2 experiments using common switching and compute resources (e.g., VLANs on shared switches, VMs on shared physical servers). Now suppose the resource allocator seeks to compare the isolation offered to each client by two alternative, feasible instantiations of the experiments labeled A and B, with measures $\vec{I}^A$ and $\vec{I}^B$. How should these be compared? In principle each experimenter might indicate as part of her system resource specification (*rspec*) a request for a desired system isolation profile. For instance, such a specification might indicate a preference for maintaining stronger isolation in computing elements rather than in networking elements. Ultimately, of course, the preferred approach to satisfying client isolation requirements would likely be domain-specific, and ideally negotiated with each client.

Let's next consider how we can apply our isolation measure to practical network elements, and an elementary network of those elements.

A. Modeling a VLAN-enabled Switch

Many experimental testbeds (e.g., Emulab, DETER [8]) enable experimenters to achieve a relatively high degree of inter-experiment isolation by a combination of admission control (i.e., rejecting new requests when resources are unavailable), permitting the allocation of dedicated end computing systems to a single experiment, and carefully limiting the use of shared resources to effectively achieve an acceptable 'quality-of-isolation.' The most common example of the latter mechanism is the way that Emulab's control infrastructure dynamically creates multiple VLANs to enable one or more ethernet switches to segregate traffic associated with different experiments.

Let's begin by considering the simplest of Emulab shared resource systems; Figure 7 depicts two experiments each with two identical, dedicated computing systems. Each experiment has a trivial network topology – a single duplex link connecting the 2 dedicated computers in each experiment. Each experimental topology is instantiated as a separate VLAN on a single, shared ethernet switch, with each computing system connected directly to the switch with a link of equal transmission rate. Most designers of large scale systems simply accept that this configuration offers *total* performance isolation between experiments (i.e., no performance degradation of one workload due to the presence of the second workload on the switch); certainly any degradation is immaterial in most situations. Here we will show, however, that in certain cases a closer examination of VLAN performance isolation is merited.

Since each source and destination computing system is identical and physically separated, they do not contribute to the exposure of workloads to each other. Further, since each experiment has a dedicated switch link, each flow conforms to some affine arrival curve α, for example, with a rate equal to the link transmission rate S. The switch, however, is a shared resource, regardless of how insignificantly it exposes one workload to another.

As long as the switch has adequate internal bandwidth to process the arriving and departing flows, to the first order each VLAN is likely well modeled by an independent element characterized by a server with rate S and a very small, fixed delay T. That is, each VLAN can be modeled by an element with service curve $\beta_{S,T}$ as defined by Eq. 4. Hence, a flow on either VLAN is unaffected by flows on the other VLAN (i.e., $D_{R_1,R_2} = D_{R_1,0} = D_{R_2,0}$) and so the isolation given by Eq. 14 is unity (i.e., complete isolation of R_1 and R_2 from each other). Of course, detailed knowledge of switch implementation and operation could lead to a more sophisticated model of the VLAN element.

Next suppose that the client's request for a duplex link is in fact implemented by the interconnection of two or more switches using a single trunk shared by both VLANs as Fig. 7 shows. This instantiation – while less preferable than that using a single switch – is a relatively common outcome in systems where compute nodes are selected from different racks, and a single experiment spans 2 top-of-rack switches and possible other aggregation switches. Due to the limited capacity of the trunk relative to the ingress access links, a simple fixed delay would no longer be the best model of the switch element, since it potentially fails to capture the interaction of network traffic between experiments sharing the trunked link. That is, the VLAN trunk ensures 'reachability' but not performance isolation. Instead, a preferred element model for a trunked VLAN would be a fixed rate node serving two arriving flows with a scheduling discipline and buffer capacity determined by the switch implementation.

One such model could be 2 queues served by a single server with fixed service rate S and Packetized General Processing Scheduling (PGPS) service discipline. Suppose that the arrival rates R_1 and R_2 satisfy $R_1 < c_1$, $R_2 < c_2$, and $c_1 + c_2 < S$. Let the maximum packet length of any flow be L packets. A PGPS server offers the i^{th} flow in the set of W currently backlogged flows a rate $c_i = S\frac{\phi_i}{\sum_j \phi_j} \;:\; i = 1, 2, ..., W$, and rate 0 to flows with no backlog. The element offers each arriving flow a service curve $\beta_{c_i,\frac{L}{S}}$ ([5], p. 68). If the i^{th} arriving flow is constrained by γ_{r_i,b_i}, then we can employ Eq. 8 to find that the maximum delay the i^{th} flow suffers is

$$D_{R_i,R_j} = \frac{L}{S} + \frac{b_i}{c_i} \quad : \; i = 1, 2 : j = 1, 2 : i \neq j \quad (37)$$

when both flows are present, and

$$D_{R_i,0} = \frac{b_i}{S} \quad : \; i = 1, 2 : j = 1, 2 : i \neq j \quad (38)$$

when only either flow is present. By applying our isolation definition from Eq. 14, a PGPS node offers flow i isolation from a second flow of roughly $\frac{c_i}{S}$, or $1/2$ for two flows equally sharing the entire server bandwidth.

Of course, it is obvious that a system instantiation interconnecting 2 switches with a trunked VLAN potentially offers less performance isolation than one that does not; no model is needed to tell us that. But by assigning an isolation measure we can now *quantify* this potential loss of isolation. As we seek to compare the performance isolation properties of ever more complicated systems, we will find having such a measure imperative.

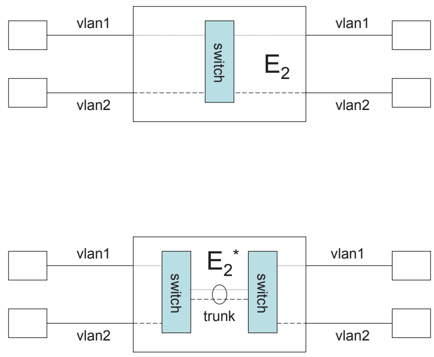

Figure 7. Two experiments each with a separate network implemented with a VLAN on a single shared switch (upper), and two switches interconnected by a single link VLAN trunk (lower).

Now let's turn to the question of how this simple element isolation measure could be used in a resource assignment process. First, note that certain system design rules-of-thumb emerge immediately, such as:

> Given two otherwise identical implementations, a design in which the workflow traverses *more* switches offers relatively less performance isolation, even in the case where each VLAN element is independently characterized by a service curve $\beta_{S,T}$.

This simple fact has implications in the overall design of experimental testbeds such as Emulab. If isolation were the only factor to consider, a system designer might choose to arrange a particular mix of computing systems on each rack (each connected to a 'top-of-rack' switch) to increase the likelihood that the system's resource allocator would contain a higher fraction of experiments to a single rack (and hence single switch), in an effort to minimize system isolation by avoiding experiments traversing multiple switches.

But more interesting is the joint optimization of isolation across multiple experiments. Suppose a resource allocator must assign a newly arriving flow to one of 2 switches, each offering a VLAN trunk to be shared with an existing flow. In one switch the existing flow is an 'elephant', and in the second switch the existing flow is a 'mouse.' How do we assign the flow associated with the newly arriving resource request?

In essence, the penalty for exposing an existing workload to an newly arriving workload is shared by both experiments. For the element modelled by a PGPS node, the isolation loss is greater for the mouse than the elephant. Of course, pairing the new flow with the elephant places the burden of greater exposure on the new flow. The preferred solution depends on the desired quality-of-isolation of each experiment; if both experiments could tolerate the reduction in performance isolation, then admitting this resource assignment might jointly satisfy the needs of the two experimenters.

B. Tne Node Selection Problem

Let's next consider the application of our performance isolation measure to the use of a PlanetLab node by multiple slices (i.e., experiments). The problem of selecting a preferred set of nodes to host a slice is familiar to many researchers. We will next show that if an experimenter is seeking to achieve the highest possible experiment isolation, the selection of the preferred end systems need no longer be arbitrary, or be dictated by greedy overprovisioning.

Suppose the experimenter may choose a subset of end systems from a set of *heterogeneous* end systems with different system clock rates, and NIC transmission speeds, but otherwise identical network interconnection. Given knowledge of the number of active slivers on each node, the experimenter can select the subset of nodes that minimizes the aggregate isolation metric for their experiment. Such an approach offers the potential for improved overall system utilization relative to existing, more ad hoc node selection approaches, such as experimenters always preferring nodes with faster CPUs.

To illustrate, we sketch a rudimentary model of a PL node. PlanetLab uses a collection of technologies to ensure a satisfactory degree of isolation between slivers (i.e., a node's experiment container), including:

- PlanetLab Virtualized Network Access (VNET+) [10] to associate inbound and outbound packets with slices, and
- Linux Hierarchical Token Buffers (HTBs) to rate control outbound traffic associated with each slice, and
- Linux Vservers to provide container-based sliver execution environments.

The PlanetLab CPU scheduler grants each sliver a fair share of the node's available CPU service, while the HTB scheduler provides each sliver a fair bandwidth share (with minimum rate guarantee) for each sliver's outbound packets.

Now let's examine a considerably simplified example motivated by the PlanetLab node selection problem. Suppose an experimenter wishes to deploy a software routing function within the sliver on a prospective node. To calculate the node's performance isolation, we propose the following simple first-order model of a system with 2 network elements. Suppose that the node hosts N active slivers, and that that arrivals to the routing sliver are constrained. Let's choose a queue served at rate S by a PGPS server to model the CPU sharing by the active slivers in the node. Hence we can derive the isolation measure of this first element serving N simultaneous slices from Eqs. 14, 37.

Similarly, the rate controller would be well modeled by a network element with a rate-latency curve, with delay determined by the number of communicating slices sharing the interface, and rate determined by the slice's bandwidth cap R (i.e., the slice's HTB *ceiling* rate); this service curve is described by Eq. 4, and the corresponding delay by Eq. 8.

Hence, the performance isolation of a candidate node is represented by 2 element measures. The first metric depends on the CPU performance of the node through S, and the second depends on the NIC speed through R. When considering alternative, heterogeneous nodes, an experimenter (or resource allocation system) may choose to combine these metrics or treat them separately. To perform a node selection based exclusively on a node isolation metric, this pair would be calculated for each prospective node, and a node would be chosen which supports the experimenter's desired isolation profile.

Though the node selection problem is motivated by a resource allocation problem present on systems such as PlanetLab, we stress that considerable further work is needed to apply the results more generally to complex experimental testbeds. For example, our ability to model an uncontrolled network connection between PlanetLab slivers on two separate nodes is limited. Of course, a principal objective of the PlanetLab system is environmental *realism*, not network isolation. The *VINI* [13] system, however, does introduce additional network controls, including 'virtual links' and the ability to introduce and control emulated network elements. Here it seems that it might be possible to calculate end-to-end metrics to express slice isolation.

In closing our discussion of applications, we note that this elementary treatment of complex network systems leaves significant room for additional modeling work. Some more advanced work is available to us; the treatment of time-varying leaky buckets [5] would be helpful in analyzing isolation in highly dynamic elastic systems. But perhaps the greatest challenge we face comes from the fact practical systems are *lossy*, while we have considered only lossless systems. We leave the application of known results for lossy systems ([5], Chapter 9) for future work.

VII. Isolation and Fairness

Performance isolation is closely related to – but distinct from – the property of *fairness* between concurrent flows in networks. A fairness measure expresses how well a given flow realizes a designated target allocation of resources in the presence of other flows; equal sharing of a resource (e.g., bandwidth) is often the target. A flow is said to be treated unfairly if it receives a lesser share than its target.

The preferred objective of bandwidth sharing algorithms has long been debated [18], though *min-max* fairness [19] (i.e., maximizing the rate of the smallest flow under capacity constraints) has arguably been the most common objective. Proportional sharing [20], which seeks to maximize an aggregate utility of a set of rate allocations, is a second common objective. Measuring fairness in dynamic settings – with flows coming and going – continues to be an area of considerable interest.

In contrast, while performance isolation also expresses a relationship between concurrent flows, it measures how one flow is affected by both the presence *and* the absence of another; flow dynamicity is explicitly required for the measure.

A set of flows might receive a fair bandwidth allocation, yet in general will not be isolated from each other. To see this, consider a system in which each of N flows each receive a targeted $1/N^{th}$ share of a single server with fixed capacity. The system would be considered fair (under most measures), and this assessment would be independent of any fixed N. But such a system does not isolate flows (according to our measure); the isolation of any two of the N flows changes as the total number of flows change, while the fairness of the resources they receive does not. While fairness measures establish that the server treats each flow equally, a performance isolation measure captures the fact that the flows share the server at all.

Conversely, a system can isolate flows from one another but not fairly allocate resources. As a practical example, a system that routes each of two flows along disjoint paths isolates them from each other, but may offer no assurance of fair bandwidth sharing if each path presents different cross-traffic. In summary, fairness and performance isolation are distinct system properties, each informing us of a potentially interesting and revealing aspect of a resource sharing system.

VIII. Related Work

The study of isolation as a property in itself has been centered in the systems security community. Carroll [28] discusses 6 broad isolation strategies relevant to computer security – spatial, temporal, cryptographic, selection of processing mode (e.g., online vs. offline processing), privilege restriction, and those isolation capabilities achieved indirectly from system architecture (i.e., design). Roughly stated, Carroll's definition of isolation is that one user should be unable to impart change on a second user's processing.

In earlier work Goguen and Meseguer [32] developed a formal model and analysis of information flows and the concept of *non-interference*, with an emphasis on security policies and models. Bishop [33] discussed isolation in the context of the confinement problem [30] [29], and Lampson introduced a definition of *total isolation*.

In contrast, our work has focused specifically on performance isolation. Work on performance isolation pervades many communities interested in resource sharing, particularly the network systems and computing systems communities. Here a variety of empirical measures are richly used as effective proxies for isolation. Quantitative measures of isolation appear to be most prevalent in the electrical devices and circuits communities. More recently, some research groups have used ideas inspired by network calculus to study the performance isolation provided by algorithms in the presence of misbehaving flows [24]. Wachs, et al [23], have introduced the concept of *performance insulation* in shared storage systems, and have sought to design practical systems that maintain a target throughput efficiency in the presence of competing workloads.

IX. Conclusion

Performance isolation is a property of resource sharing systems. Exposure – the complement of isolation – measures the affect of a given workload (or set of workloads) on other workloads. In this paper we introduced a formal definition of performance isolation, defined a quantitative measure of the performance isolation between workloads, and showed how to apply network calculus as a method for calculating the performance isolation of multiple competing workloads in complex resource sharing systems. We contend that isolation merits study as a first-class property of resource sharing systems alongside other well-studied properties such as fairness.

We have also argued that conventional, empirical measures of an individual element's performance isolation behavior are inadequate for the design of systems capable of meeting a desired quality-of-isolation requirement. Instead we have proposed an analytical measure of an element's isolation capability based on the theory of deterministic queueing systems. By combining isolation metrics of elements forming a system, we have showed that the isolation properties of distinct system implementations can be analyzed and compared, enabling implementors to construct systems that conform to their target isolation budget. Further, the metrics can also be used to address how exposure can be minimized by modifications to a particular system design.

We believe that such a rigorous approach is also needed to address other types of system isolation beyond just performance isolation. For example, the confidentiality of a workflow might be described by creating a measure of the observability of a flow at each element visited (e.g., measured, say, in bits/time from a specified vantage point) and aggregating those metrics along the workflow's path. Given the need to clarify the isolation capabilities of a vast number of new and proposed multi-client computing and communication resource sharing systems (e.g., compute clouds, experimental cyber testbeds), the further study of isolation as a fundamental system property remains critical. Simply adhering to conventional security best practices [27] – while appropriate in relatively static environments – is difficult or impractical to apply in such dynamic settings, and fall short of our need to ground system design in an underlying theory of workflow isolation.

The emergence of the notion that isolation is a first-class system property that needs quantitative measures mirrors early discussions of fairness. In the early 1980's an abundance of fairness metrics were introduced [14] [15] [16]. To be applicable to end-to-end paths, many of the proposed fairness metrics were based on delay, and specifically delay variation. Dissatisfied with these measures, Jain [17] proposed an alternative metric called the *Index of Fairness*, designed to meet a set of properties appropriate for a metric. In commenting on the immaturity of the state of fairness measures, Jain observed:

> ”These papers divide (flow control) algorithms into two categories: fair or unfair. It is not possible to measure the fairness of a particular algorithm.”
>
> ”Quantitative measure of fairness are not well known. Most studies on fairness tend to be either qualitative of too specific to a particular application.”
>
> ”It is clear that fairness implies equal allocation of resources, although there is little agreement among researchers as to what should be equalized. In computer networks, some want delay, others want throughput, and yet others want ...”

Many strikingly similar statements – with the term *isolation* now replacing *fairness* – are being echoed today as the research community wrestles with the challenging problem of defining and quantifying performance isolation.

References

[1] Emulab, http://www.emulab.net
[2] PlanetLab, http://www.planet-lab.org
[3] Global Environment for Network Innovations, http://www.geni.net
[4] J. Brassil, “Analyzing Flow Isolatation in Shared Resource Systems,” Proc. of ICCCN 2011, August, 2011.
[5] J-Y. Le Boudec, P. Thiran, “Network Calculus: A Theory of Deterministic Queuing Systems for the Internet,” Lectures Notes on Computer Science 2050, Springer, New York, 2001. Online edition (v4): http://ica1www.epfl.ch/PS_files/netCalBookv4.pdf
[6] R. L. Cruz, “A Calculus For Network Delay, Part I: Network Elements In Isolation,” *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114-131, Jan. 1991.
[7] R. L. Cruz, “A Calculus for Network Delay, Part II: Network Analysis,” *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132-141, Jan. 1991.
[8] Deterlab, http://www.isi.deterlab.net
[9] K. Sklower and A. Joseph, “Very Large Scale Cooperative Experiments in Emulab-Derived Systems,” *DETER Community Workshop on Cyber Security Experimentation and Test 2007*, Boston, August 2007.
[10] M. Huang, “VNET: PlanetLab Virtualized Network Access,” PlanetLab Design Note PDN-05-029, June 2005, http://www.planet-lab.org/files/pdn/PDN-05-029/pdn-05-029.pdf.
[11] S. Bhatia, “VNET+: Associating Packets with Slices,” *PlanetLab Miscellaneous Documentation*, http://www.cs.princeton.edu/ sapanb/vnet/.
[12] M. Huang,“PlanetLab Bandwidth Limits,” *PlanetLab Miscellaneous Documentation*, http://www.planet-lab.org/doc/BandwidthLimits.
[13] A Bavier, N Feamster, M Huang, L Peterson, J Rexford, “In VINI Veritas: Realistic and Controlled Network Experimentation,” *Proc. of ACM SIGCOMM’06*, 2006.
[14] J. Sauve, J.W. Wong, J. Field, “Fairness in Packet-Switching Networks,” *Proc. of CompCon’80*, Washington DC, pp. 466-470, 1980.
[15] M. Gerla, M. Staskauskos, “Fairness in Flow Controlled Networks,” *Proc. of ICC’81*, Denver CO, 1981.
[16] M. Marsan, M. Gerla, “Fairness in Local Computing Networks,” *Proc. of ICC’82*, Philadelphia PA, 1982.
[17] Raj Jain, Dah-Ming Chiu, William Hawe, “A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems,” *DEC Technical Report 301*, 1984.
[18] L. Massoulie, J. Roberts, “Bandwidth Sharing: Objectives and Algorithms,” *IEEE Trans. on Networking*, vol. 10, no. 3, June 2002.
[19] E. Hahne, “Round-robin Scheduling for max-min Fairness in Data Networks,” IEEE JSAC, vol. 9. pp. 1024-1039, 1991.
[20] F. Kelly, A. Maulloo, D. Tan, “Rate Control for Communication Networks: Shadow Prices, Proportional Fairness, and Stability,” *J. Operations Research Soc.*, vol. 49. pp237-252, 1998.
[21] M. Karlsson, C. Karamanolis, X. Zhu, “Triage: Performance Isolation and Differentiation for Storage Systems,” *Twelfth IEEE International Workshop on Quality of Service (IWQOS 2004)*, June 2004, pp. 67-74.
[22] S.R. Seelam and P.J. Teller , “Fairness and Performance Isolation: an Analysis of Disk Scheduling Algorithms,” *2006 IEEE International Conference on Cluster Computing*, Barcelona, Sept. 2006.
[23] M. Wachs, M. Abd-El-Malek, E. Thereska, G. Ganger, “Argon: Performance Insulation for Shared Storage Servers,” *Proc. 5th USENIX Conf. on File and Storage Technologies (FAST’07)*, Feb. 2007, San Jose, CA.
[24] Ajay Gulati, Arif Merchant, Peter J. Varman, “pClock: an arrival curve based approach for QoS guarantees in shared storage systems,” *Proceedings of ACM SIGMETRICS 2007*, Vol. 35, No. 1, June 2007, pp. 13-24.
[25] R. Farrow, *LAN Isolation*, http://www.spirit.com/Network/net0103.html
[26] D. Pollion, M. Schiffman, “Secure Use of VLANs: An @stake Security Assessment,” *@stake Research Report*, August 2002.
[27] Cisco Systems, “Cisco SAFE Reference Guide,” *Cisco Document OL-19523-01*, August 2009. http://www.cisco.com/en/US/docs/solutions/Enterprise/ Security/SAFE_RG/SAFE_rg.pdf
[28] John Millar Carroll, “Chapter 16: Isolation in Computer Systems,” Computer Security, 3rd Ed., 1996.
[29] B. Lampson, “A Note on the Confinement Problem,” 1973.
[30] B.W. Lampson, “Dynamic protection structures,” *Proc. AFIPS 1969 FJCC*, Vol. 35, AFIPS Press, Niontvale, N.J., pp. 27-38.
[31] Schroeder, NI.D., and J.H. Saitzer, “A Hardware Architecture for Implementing Protection Rings,” *Comm. ACM*, vol. 15, no. 3, (Mar. 1972), pp. 157-170.
[32] J. Goguen, J. Meseguer, “Security Policies and Security Models,” *Proceedings of the 1982 Symposium on Security and Privacy*. Oakland, 1982.
[33] M. Bishop, Computer Security: Art and Science, Pearson Education Co., Boston, 2003.
[34] Amazon Elastic Compute Cloud http://aws.amazon.com/ec2

X. Appendix

We next present a proof of 22; the proofs of 24 and 20 follow similarly.

Consider some time t, and let some earlier time s be the start of a server busy period. Service for the medium priority queue can be delayed by at most a single packet that arrives earlier on the low priority queue, or a conforming burst of packets arriving on the high priority flow. Suppose that packets arrive to the low, then high, then medium priority queues in quick succession at times s, s', and s'', with $s < s' < s''$. In this case a single packet in the low priority queue must complete service, followed by the backlog in the high priority queue, followed finally by the medium priority queue. During the interval $(s, t]$ the server's output is $C(t-s)$. During the interval $(s, t]$ the output of the medium priority queue is

$$R^*_M(t) - R^*_M(s) = C(t-s) - l_{MAX} - R^*_H(t) - R^*_H(s) \tag{39}$$

But at the beginning of the backlog period the total bits on arriving and departing flows are identical, i.e. $R_M(s) = R^*_M(s)$ and $R_H(s) = R^*_H(s)$, so we have

$$R^*_M(t) - R_M(s) = C(t-s) - l_{MAX} - R^*_H(t) - R_H(s) \tag{40}$$

But the high priority flow is constrained to $\gamma_{r_h,b_h}(t)$, so we have

$$R^*_M(t) - R_M(s) \leq [C(t-s) - l_{MAX} - r_h + b_h(t-s)]^+ \tag{41}$$

$$= (C - r_h)(t-s) - l_{MAX} - b_h \tag{42}$$

which is equivalent to $\beta_{C-r_h, \frac{l_{MAX}+b_h}{C-r_h}}$.

Using Eq. 8 with this service curve and the constraint on the arrival flow to the medium priority queue yields the corresponding delay for the delay given by Eq. 22. □

Jack Brassil received the B.S. degree from the Polytechnic Institute of New York in 1981, the M.Eng. degree from Cornell University in 1982, and the Ph.D. degree from the University of California, San Diego, in 1991, all in electrical engineering. Dr. Brassil has been with Hewlett-Packard Laboratories since 1999. He currently is a Research Scientist and Program Manager in Princeton, NJ. Before joining HP he held multiple research positions at Bell Laboratories in Murray Hill and Holmdel, NJ. Dr. Brassil is an IEEE Fellow, a member of the IEEE Communications Society, a member of the ACM, and a member of ACM SIGCOMM.