Quality of Service Abstractions for Software-defined Networks

Cole Schlesinger
Princeton University

Hitesh Ballani
Microsoft Research

Thomas Karagiannis
Microsoft Research

Dimitrios Vytiniotis
Microsoft Research

ABSTRACT

Software-defined networking (SDN) provides a means of configuring the packet-forwarding behavior of a network from a logically-centralized controller. Expressive, high-level languages have emerged for expressing data-plane configurations, and new tools allow for verifying packet reachability properties in real time. But SDN largely ignores quality of service (QoS) primitives, such as queues, queuing disciplines, and rate limiters, leaving configuration of these elements to be performed out of band in an ad-hoc manner. Not only does this make QoS elements difficult to configure, it also leads to a "try it and see" approach to analysis and verification of QoS properties.

We propose a new language for configuring SDNs with quality of service primitives. Our language comes equipped with a well-defined semantics drawn from the network calculus, which we believe will yield an equational theory for reasoning about network quality of service as well as decision procedures for verifying QoS properties.

1. INTRODUCTION

Software-defined networking (SDN) has seen a great surge in popularity, in large part because of the programmatic control the OpenFlow protocol [26] provides for configuring data-plane forwarding. This has, in turn, led to the development of higher-level policy languages for configuring forwarding behavior, along with compilers targeting OpenFlow [13, 5, 33, 27]. Verification tools have been developed for checking reachability properties of data plane configurations [21, 22, 30], as well as correctness properties of the controller programs that orchestrate network behavior over time [9, 6, 32].

Quality of service (QoS), on the other hand, has received less attention. The OpenFlow protocol provides only limited support for queue selection, and any configuration of QoS elements—queues, rate limiters, *etc.*—must be done out of band. Despite these limitations, several projects have made use of SDNs to offer guarantees about bandwidth as part of network orchestration frameworks [12, 31], and others have employed SDN coupled with traffic monitoring for increasing overall wide-area network utilization [16, 17].

For these projects, it was sufficient to reason about bandwidth allocation with respect to link capacities. However, many datacenter applications are sensitive to network delay—this is especially true for user-facing applications like search and recommendation systems that need to respond to users in a timely fashion and generate short, delay-sensitive flows across the network [4]. Specifically, a datacenter operator may want to know, given a characterization of traffic entering the network, will packets ever be dropped due to congestion, will packets within a flow ever be reordered, what are the minimum queue sizes necessary to avoid packet loss, what is the maximum packet jitter, *etc.*. Given that queuing is a dominant component of network delay in datacenters, we show that many such QoS questions boil down to quantifying queuing delay.

While it is very hard to reason about QoS and queuing delay in general networks, datacenters offer a unique opportunity on this front, as well. Recent proposals argue for enabling per-tenant bandwidth guarantees in datacenters by enforcing rate limits at the end hosts [7, 15, 8, 29, 18], which enables a precise characterization of the traffic entering the datacenter's internal network. This, in turn, makes it possible to apply techniques like network calculus [11, 24, 25, 20] to analyze network queuing delay.

Reasoning about quality of service is fundamentally tied to reasoning about network forwarding behavior, which might add packets, such as when supporting multicast or flooding; remove packets, as in access control; or aggregate or split flows during routing. Hence, we propose a language, called QtKAT,[1] that provides a unified model of forwarding and quality of service.

QoS inherently concerns how packets interact in the network, and so QtKAT operates at the granularity of *flows*, where a flow is a set of packets that share some traffic characteristic. Thus, QtKAT models the network

[1]Pronounced "cutie cat," of course.

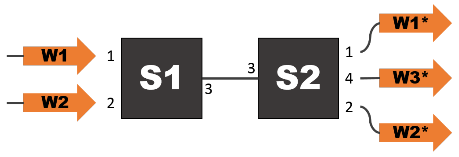

Figure 1: A simple network.

as a function that consumes a set of input flows and produces a set of output flows that reflect the changes imposed on packets by the forwarding configuration, as well as changes wrought on each flow's traffic characteristics by rate limiters, bandwidth constraints, queuing disciplines, and interference with other flows.

Specifically, QtKAT extends NetKAT [5], a language for modeling network forwarding, by adding a notion of *traffic models* to characterize a flow's traffic and *service models* to capture how a switch services traffic. QtKAT's semantics rely on four basic properties of traffic and service models that, in turn, are satisfied by techniques from network calculus (both deterministic and stochastic). This allows QtKAT to remain oblivious of the choice of network calculus model, so long as it satisfies each property.

Through a series of examples, we show how to phrase and verify interesting QoS questions with QtKAT. We conclude by discussing open questions regarding the language's properties (like yielding an equational theory for network QoS) and the systems' implications (like the ability to verify properties in a scalable and timely fashion).

The scope of this work. It bears noting that this work does not contribute any new algorithms for providing stronger quality of service guarantees. Nor does it contribute new network calculus techniques. Rather, it provides a unifying model for configuring and verifying routing and quality of service properties. We found the composition to be nontrivial.

2. QtKAT BY EXAMPLE

We introduce QtKAT by example—Figure 1 depicts a simple topology with two switches (s_1 and s_2) aggregating two flows (w_1 and w_2) over a single link. For simplicity, we focus on traffic flowing from left to right, but characterizing bidirectional traffic (and larger networks) is straightforward. Figures 2 and 3 present the language syntax and semantics (detailed in Section 3).

Modeling flows and switches. Recent approaches to analyze network forwarding behavior aim to do the following: Given a packet at a particular switch/port, produce the (possibly empty) set of packets the switch will emit. In this paper, we extend this to reason about packets' quality of service (QoS). The challenge is that while it is feasible to reason about forwarding of individual packets in isolation, QoS inherently concerns how packets interact with each other at network switches.

To capture this, we use the term *flow* to refer to a set of packets that share some traffic characteristic. Specifically, we model a flow as a pair (a, f) comprising a predicate and a function. The predicate a is a boolean predicate on packets (defined in Figure 2) and declares which packets are included in a flow. The function f is a traffic model that captures how the flow's packets arrive at a location. Existing approaches for QoS analysis like network calculus and stochastic network calculus use deterministic and stochastic traffic models respectively. QtKAT uses an abstract traffic model that can be instantiated using such existing models (details in Section 4). For the example in this section, we use a deterministic model f to bound the aggregate number of bits transmitted in the flow as a function of time.

$$\begin{aligned} w_1 &\triangleq (\mathsf{src} = 1; \mathsf{pt} = 1, g_1(t) = t) \\ w_2 &\triangleq (\mathsf{src} = 2; \mathsf{pt} = 2, g_2(t) = 2t) \end{aligned}$$

For example, w_1 describes a flow of packets at port 1 with a source address of 1 (shorthand for, say, 10.0.0.1). The flow's traffic model g_1 says that one bit for the flow arrives per time interval.

To model the forwarding *and* QoS behavior of the network, we model each switch as follows. Given a set of flows, a switch will produce a new set of flows, perhaps modifying the packets in them (i.e. changing the flows' predicates), and perhaps altering their traffic model (i.e. changing the flows' traffic functions). In other words, the network as a whole can be viewed as a function that takes a set of flows and produces a new set of flows with new traffic characteristics. Thus, QtKAT policies are functions on *sets of flows*. We write $[\![p]\!]$ to be the function denoted by a policy p.

Routing flows. An SDN controller is responsible for configuring switches s_1 and s_2 to forward packets appropriately. Indeed, we might configure s_1 with the following policy p, which states that packets arriving on ports 1 or 2 should be emitted from port 3.

$$p \triangleq (\mathsf{pt} = 1 + \mathsf{pt} = 2); \mathsf{pt} \leftarrow 3$$

We write $\mathsf{pt} = 1 + \mathsf{pt} = 2$ as a predicate on packets—here, $\mathsf{f} = v$ is a basic test on the value of packet header fields, and $(+)$ is disjunction. The predicate is sequenced with $\mathsf{pt} \leftarrow 3$, which assigns 3 to the "port" field and thus, the packet is emitted from port 3.

And so, how does s_1, configured with p, handle flows w_1 and w_2? The result of applying p to $\{w_1, w_2\}$ is given by $[\![p]\!]\, \{w_1, w_2\}$.

$$[\![p]\!]\, \{w_1, w_2\} = \left\{ \begin{array}{l} (\mathsf{src} = 1; \mathsf{pt} = 3, g_1(t) = t), \\ (\mathsf{src} = 2; \mathsf{pt} = 3, g_2(t) = 2t) \end{array} \right\}$$

What started as two flows in different locations simply becomes two flows in the same location. We have not yet accounted for the interaction between flows.

Aggregating and bounding flows. As of yet, the policy p says nothing about quality of service. But the switch s_1 is bound by physical constraints on how quickly it can service packets arriving in flows w_1 and w_2. A standard practice in the deterministic network calculus is to model the service s_1 can provide as a wide-sense increasing function on time $s(t)$ which denotes the aggregate number of bits serviced by time t. For example, $s(t) = 4t$ means the switch can service 4 bits per time interval. Drawing from network calculus, we write $r \leftarrow s$ to mean, "apply service curve s."

Hence, we can conjoin our policy p with a service curve attached to port 3: $(p; \mathsf{pt} = 3; r \leftarrow s)$. This models the physical limitations the switch faces—packets leaving from port 3 are constrained by s. Applying this combined policy to the flows w_1 and w_2 results in a single aggregate outgoing flow w_3.

$$[\![p; \mathsf{pt} = 3; r \leftarrow s]\!]\, \{w_1, w_2\} = w_3$$
$$\text{where } w_3 \triangleq (\mathsf{src} = 1|2; \mathsf{pt} = 3, (g_1 + g_2) \otimes s)$$

The outgoing flow w_3 has a source address of 1 or 2, and its traffic function is the combined traffic function of the incoming flows $(g_1 + g_2)$ when processed by the service function s. This is represented by $(g_1 + g_2) \otimes s$. We explain the convolution operator $\otimes$ in Section 4—the intuition is that the service function upper-bounds the traffic function of the outgoing flow. With $s(t) = 4t$, the service function is strictly greater than the combined incoming traffic function $((g_1 + g_2) = 3t)$, so the traffic function for the outgoing flow is simply $3t$.

Determining the aggregate flow w_3 is a straightforward application of network calculus. However, for the combined forwarding and QoS analysis, it is useful to retain information about each individual flow, in part to retain precision if the flows are later split downstream (as we will see in the "Splitting flows" paragraph below). Thus, instead of generating an aggregate flow w_3, we retain the set of outgoing flows $\{w_1', w_2'\}$. Here flows w_1' and w_2' are the result of applying $p; \mathsf{pt} = 3; r \leftarrow s$.

$$[\![p; \mathsf{pt} = 3; r \leftarrow s]\!]\, \{w_1, w_2\} = \{w_1', w_2'\}$$
$$\begin{aligned} \text{where } w_1' &\triangleq (\mathsf{src} = 1; \mathsf{pt} = 3, \amalg(g_1, g_2, s)) \\ w_2' &\triangleq (\mathsf{src} = 2; \mathsf{pt} = 3, \amalg(g_2, g_1, s)) \end{aligned}$$

The formula $\amalg(f, g, s)$ represents the traffic characteristic of f after being processed by a service curve s subject to interference by traffic from g.

Modeling the topology. As we saw in the policy p, by convention packet headers include special fields holding the current switch and port at which they are located. By encoding location as part of the packets, we can in turn model the topology as a special policy that simply changes the location of the packet.

$$l \triangleq \mathsf{sw} = s_1; \mathsf{pt} = 3; \mathsf{sw} \leftarrow s_2 + \mathsf{sw} = s_2; \mathsf{pt} = 3; \mathsf{sw} \leftarrow s_1$$

The link l between $s1$ and $s2$ comprises two parts: if a packet resides at port 3 on switch s_1, rewrite its switch field sw with s_2; similarly, if it resides on switch s_2, assign it s_1. Furthermore, we can use the bounding operator to model link capacities. Suppose l had a capacity of C. Bounding flows across the link by $s(t) = Ct$ effectively characterizes the maximum link bandwidth: $l; r \leftarrow (s(t) = Ct)$.

Splitting flows. Suppose switch s_2 redirects SSH traffic from *both* flows out port 4 for monitoring; leaving the remaining traffic in flows w_1 and w_2 to be emitted from ports 1 and 2, respectively.

$$q \triangleq \left(\begin{array}{l} \mathsf{if\ typ} = ssh\ \mathsf{then\ pt} \leftarrow 4 \\ \mathsf{else\ if\ src} = 1\ \mathsf{then\ pt} \leftarrow 1 \\ \mathsf{else\ if\ src} = 2\ \mathsf{then\ pt} \leftarrow 2 \end{array} \right)$$

Suppose the policy q configures switch s_2 and we apply $[\![q]\!]$ to flows w_1' and w_2'. For simplicity, we will omit the service curve for this switch.

$$[\![q]\!]\, \{w_1', w_2'\} = \{w_1^*, w_2^*, w_3^*\}$$

What are the traffic characteristics of the output flows w_1^*, w_2^*, and w_3^*?

First, we know that $w_1^* \leq w_1'$ and $w_2^* \leq w_2'$, where $f \leq g$ means $f(t) \leq g(t)$ for all $0 \leq t$—after all, w_1 and w_2 may contain no SSH traffic. But to account for the fraction of traffic that may, in fact, be SSH traffic, we introduce *symbolic constants* α and β. Intuitively, when a flow is split, α and β represent the unknown fraction of traffic in each split sub-flow. For example, w_1 may be split into its SSH and non-SSH components,

$$\begin{aligned} w_1^* &\triangleq (\mathsf{src} = 1; \mathsf{pt} = 3, \alpha_1 \amalg (g_1, g_2, s)) \\ w_1^{ssh} &\triangleq (\mathsf{src} = 1; \mathsf{pt} = 3, \beta_1 \amalg (g_2, g_1, s)) \end{aligned}$$

where $0 \leq \alpha_1, \beta_1 \leq 1$ and $\alpha_1 + \beta_1 = 1$, and $\alpha \amalg (\cdot, \cdot, \cdot)$ is point-wise multiplication of α (resp. β) on the function produced by $\amalg(\cdot, \cdot, \cdot)$.

Splitting flows is useful for two reasons. First, it maintains a tight bound on traffic across sub-flows. And second, analyzing the resulting α/β pairs can provide insight into improving the granularity of traffic shaping. In this example, imposing two rate limiters at each flow source—to separately control SSH and non-SSH traffic—will allow for more fine-grained control over traffic to the monitor.

3. THE QtKAT MODEL

Figure 2 presents the core operations of our language. QtKAT is defined using an abstract set of fields f, making it suitable to reason about existing and future protocols. Packets are records of field/value pairs,[2] and traffic and server models f and s are left abstract. An interesting result is that f and s, and operations on them,

[2]See [14] for a more sophisticated packet model that accounts for field dependencies.

fields	f	$::=$	$\mathsf{f}_1 \mid \cdots \mid \mathsf{f}_k$	
packets	pk	$::=$	$\{\mathsf{f}_1 = v_1, \cdots, \mathsf{f}_k = v_k\}$	
traffic models	f			
service models	s			
predicates				
a, b		$::=$	id	*identity*
		$\mid$	drop	*drop*
		$\mid$	$\mathsf{f} = v$	*test*
		$\mid$	$a + b$	*disjunction*
		$\mid$	$a; b$	*conjunction*
		$\mid$	$\neg a$	*negation*
policies				
p, q		$::=$	a	*predicate*
		$\mid$	$\mathsf{f} \leftarrow v$	*modification*
		$\mid$	$p + q$	*union*
		$\mid$	$p; q$	*sequencing*
		$\mid$	p^*	*Kleene star*
		$\mid$	$\mathsf{if}\, a\, \mathsf{then}\, p\, \mathsf{else}\, q$	*if statement*
		$\mid$	$r \leftarrow s$	*rate limiting*
flows	$w \in \mathsf{W}$	$::=$	(a, f)	

Figure 2: Syntax.

can be instantiated using different underlying models from either the deterministic or stochastic network calculi. (Section 4 defines these operations formally.)

Syntax. The syntax of QtKAT is divided into predicates (a, b) and policies (p, q). Predicates form a Boolean algebra and include the constants id (0) and drop (1), tests $(\mathsf{f} = v)$, disjunction $(a+b)$, conjunction $(a; b)$, and negation $(\neg a)$. All predicates are valid policies, and policies additionally include field modification $(\mathsf{f} \leftarrow v)$, union $(p + q)$, sequential composition $(p; q)$, iteration (p^*), if statements $(\mathsf{if}\, a\, \mathsf{then}\, p\, \mathsf{else}\, q)$, and rate limiting $(r \leftarrow s)$. By convention, $(^*)$ binds more tightly than $(+)$, which binds more tightly than $(;)$.

Semantics. The semantics, listed in Figure 3, is similar in spirit to the symbolic evaluation underlying Header Space Analysis (HSA), a technique for analyzing reachability properties [21]. Indeed, when evaluating policies that exclude the rate limiting operator $(r \leftarrow s)$, our semantics roughly coincide with theirs. But whereas HSA symbolically represents sets of packets that share common forwarding behavior, our semantics also accounts for sets of packets belonging to the same *flow* and sharing common traffic characteristics.

We define flows to be pairs (a, f), where the predicate a indicates which packets belong to this flow, and the traffic model f describes its transmission characteristics. We write $[\![p]\!]$ as the denotational interpretation of policies as functions on sets of flows—intuitively, a policy thus interpreted consumes a set of input flows and produces a set of output flows that reflect the routing and interference each encountered within the network.

Predicates and modifications only affect the predicate portion of flows. When a flow (a, f) is processed by the policy b, packets in the flow not matching b will be dropped; hence, the resulting flow only includes packets matching $\lfloor a; b \rfloor$. Similarly, the output flow processed by the policy $\mathsf{f} \leftarrow v$ will include the packets matched by a, except with the field f assigned the value v, written $a[\mathsf{f} := v]$. We write $\lfloor a \rfloor$ to be a normal form as a technical convenience, which ensures that if two predicates are equivalent $(a \equiv b)$, then their normal forms are syntactically equal—an important property for developing an equational theory (Section 7).

The union operator $(p + q)$ duplicates the set of input flows, processes one copy with p and the other with q, and merges the resulting output sets. The sequencing operator $(p; q)$ is modeled as function composition, capturing the output of p piped as the input to q. Following [5], Kleene star (p^*) is modeled as the infinitary sum $(\mathsf{id} + p + p; p + \ldots)$.

The if-statement $(\mathsf{if}\, a\, \mathsf{then}\, p\, \mathsf{else}\, q)$ is a special case that combines union and sequential composition to retain more precise information about traffic characteristics. The definition resembles $[\![a; p + \neg a; q]\!]$, but when each flow is duplicated (as per $[\![+]\!]$), the copies are scaled by symbolic constants α_i and β_i. Each α_i/β_i pair is unique to an input flow and captures the fact that no new traffic is generated; rather, every packet in each flow is either processed by p or q, but not both.

Finally, the rate limiting operator $(r \leftarrow s)$ has the effect of aggregating its input flows and applying a per-flow output constraint that captures the interference between flows under the bounding constraint of the service curve s. The auxiliary function $\texttt{sum\_others}(ws)$ computes the summation of traffic models in ws, excluding the flow w, where $f + g$ is an abstract flow aggregation operation. The resulting per-flow service-bounded traffic model for a flow $w = (a, f) \in ws$ is computed by the abstract function $\Pi(f, \texttt{sum\_others}(w, ws), s)$, defined the next section.

4. NETWORK CALCULUS

Network calculus was developed as a more tractable alternative to queuing theory for analyzing the performance of high-speed packet networks [11, 10, 24]. It has been developed in both deterministic (DNC) and stochastic (SNC) models,[3] and the choice of model influences the properties that can be verified. DNC places worst-case bounds on network behavior and is suitable for providing absolute (but conservative) service guarantees, whereas SNC accounts for statistical multiplexing gains when some violations of the deterministic bounds are acceptable.

[3]Our treatment is drawn primarily from [25] and [20].

$$
\begin{array}{rcl}
[\![p]\!] & \in & \mathcal{P}(\mathsf{W}) \to \mathcal{P}(\mathsf{W}) \\
[\![a]\!]\,ws & \triangleq & \{(\lfloor a;b \rfloor, f) \mid (b,f) \in ws \wedge \lfloor a;b \rfloor \not\equiv \mathsf{drop}\} \\
[\![\mathsf{f} \leftarrow v]\!]\,ws & \triangleq & \{(\lfloor a[\mathsf{f} := v] \rfloor, f) \mid (a,f) \in ws \wedge \lfloor a[\mathsf{f} := v] \rfloor \not\equiv \mathsf{drop}\} \\
[\![p+q]\!]\,ws & \triangleq & [\![p]\!]\,ws \bigcup [\![q]\!]\,ws \\
[\![p;q]\!]\,ws & \triangleq & [\![q]\!] \circ [\![p]\!]\,ws \\
[\![p^*]\!]\ \ ws & \triangleq & \bigcup_{i\in\mathbb{N}} F^i\ ws, \text{ where } F^0\ ws \triangleq ws \text{ and } F^{i+1}\ ws\ \triangleq\ ([\![p]\!] \circ F^i)\ ws \\
[\![\mathsf{if}\ a\ \mathsf{then}\ p\ \mathsf{else}\ q\ ]\!]\ \ ws & \triangleq & ([\![a;p]\!]\,\{(b,\alpha_i f) \mid (b,f) \in ws\}) \bigcup ([\![\neg a;q]\!]\,\{(b,\beta_i f) \mid (b,f) \in ws\}) \\
 & & \text{such that } \forall\ 0 \le i < |ws|\ .\ 0 \le \alpha_i, \beta_i \le 1 \text{ and } \alpha_i + \beta_i = 1 \\
[\![r \leftarrow s]\!]\ \ ws & \triangleq & \mathsf{let}\ \mathtt{sum\_others}(w, ws) = (\sum \{g \mid (b,g) \in ws \setminus \{w\}\})\ \mathsf{in} \\
 & & \{(a, \amalg(f, \mathtt{sum\_others}(w, ws), s)) \mid (a,f) = w \in ws\}
\end{array}
$$

Figure 3: Semantics.

Rather than "hard-code" one network calculus model in our language semantics, QtKAT is parameterized by abstract traffic and service models (f and s), as well as four properties (described below) that relate them [19]. Hence, QtKAT can be instantiated with any model that supports these properties, including existing deterministic and stochastic models, as well as potentially more accurate models developed in the future.

- (P1) **Aggregation** | $f_1 + f_2$**.** Two traffic models can be combined to produce a model of their aggregate traffic;
- (P2) **Output characterization** | $f \otimes s$**.** A traffic model can be constrained by a service model to produce a new traffic model characterizing the resulting output flow;
- (P3) **Per-flow output** | $\amalg(f_1, f_2, s)$**.** When an aggregate traffic model is constrained, new traffic models can be produced to characterize the output of each flow in the aggregate individually. We write $\amalg(f_1, f_2, s)$ to denote the output traffic model for f_1 when the aggregate $(f_1 + f_2)$ is constrained by s.
- (P4) **Service guarantees** | $b(t)$, $d(t)$**.** Stochastic backlog and delay guarantees can be derived.

The remainder of this section sketches partial models that support these properties in the deterministic and stochastic settings.

Deterministic network calculus. The deterministic network calculus models both traffic and service as monotonic functions on time; the former bounds the maximum bits transmitted by time t, while the latter bounds the minimum bits serviced by time t. For example, affine functions model traffic governed by token buckets, where R is the rate at which tokens refresh and B is the burst size: $f(t) = Rt+B$. In this setting, f_1+f_2 is pointwise addition, $f \otimes s$ is min-plus convolution, and per-flow output can be calculated as in [24].

Stochastic network calculus. Unfortunately, we have not yet discovered a stochastic model that satisfies all four properties. Kurose [24] presents a model that satisfies (P1), (P2), and (P4) [19], as well as a deterministic approach to (P3). In this work, traffic from a source i is bounded over an interval of time t by a discrete random variable $\mathbf{R}_t^i$ if $\mathbf{R}_t^i$ is stochastically larger than the number of packets generated over any interval of t time units by source i.

Going forward, we hope to incorporate the deterministic approach to (P3) into the stochastic model and quantify how conservative it is in practice.

5. NETWORK CONFIGURATION

To be more than just a modeling language, QtKAT must be able to implement its semantics by configuring forwarding and QoS elements. For forwarding, the Frenetic SDN controller platform employs NetKAT as its configuration language [2]. Converting QtKAT to NetKAT for forwarding configuration is straightforward.

The OpenFlow protocol lacks comprehensive support for configuring QoS elements like token buckets, priority markers and priority queues at switches and end hosts [3]. However, recent work on datacenter network QoS [7, 18, 28] includes control protocols for configuring such QoS elements. QtKAT's rate limiting operator can be directly mapped to the configuration of token buckets. QtKAT can also support priority queue forwarding as the modification of metadata fields—just as modifying the pt field of a packet indicates which port a switch should emit it from, modifying a distinguished queue field can indicate where a packet should be queued.

6. QUALITY OF SERVICE PROPERTIES

In this section, we show how interesting questions regarding network bandwidth, delay and even placement of QoS elements can be phrased using QtKAT.

Bandwidth. Reasoning about bandwidth constraints is often straightforward. Nevertheless, QtKAT is able to capture bandwidth properties. Consider the problem of ensuring per-tenant hose model bandwidth guarantees [7, 29, 18]. Each host i belonging to a tenant is guaranteed bandwidth B_i. This can be expressed as a simple bandwidth constraint. Suppose a tenant's hosts $h_1, \ldots, h_n$ are connected via a topology t and policy p, and the service model $s \triangleq s_1 + \ldots + s_j$ denotes the sum of the service models of links representing the min cut connecting hosts $h_1, \ldots, h_k$ and $h_{k+1}, \ldots, h_n$. To satisfy the tenant's guarantee, the aggregate service capacity B across the min cut must be greater than the total bandwidth guaranteed to the smaller group of hosts. Thus, it suffices to determine whether $s \geq f(t) = Bt$, where $(\geq)$ is defined point-wise on functions.

Queuing delay. We considered a few non-bandwidth QoS questions that datacenter administrators may ask.

- (Q1) Given a topology and per-host rate limits, will there be losses?
- (Q2) How many failures can the topology absorb before there will be (congestion) losses?
- (Q3) What is the maximum one-way delay?
- (Q4) What is the maximum jitter for a tenant?

We find that such QoS questions can be reduced to constraints on network delay. Further, since queuing dominates the network delay in intra-datacenter settings, we can express these as constraints on queuing delay. We explain this below. Suppose we have a term p that represents the whole of the network, including the topology and the policies of each switch; and suppose we have a set of flows ws from hosts hs, where the traffic curve of each flow reflects the per-host rate limits. We can calculate the output flows $ws^* = [\![p]\!]\, ws$, which, in turn, informs the following.

- (A1) There will be losses if $\lim_{t \to \infty} d(t) = \infty$ for any pair of input/output flows on the same packets.
- (A2) Iteratively removing links and repeating this analysis will indicate which failures the topology can tolerate before there will be congestion losses.
- (A3) The maximum one-way delay is the largest maximum delay for pairs of input/output flows in f and f^*, where maximum delay is defined as $\sup_{0 \leq t} d(t)$.
- (A4) . . . which is also the maximum jitter.

Refactoring. Consider a carefully configured network where the administrator has verified the properties described thus far for the specified traffic model. But there are multiple options for enforcing these traffic models; for instance, rate limiting could be done at the hosts [7] or at the switches [28]. Further, hardware constraints need to be accounted for. For example, the administrator may place a rate limiter on a switch for traffic shaping, but due to constraints on the number of flows the switch can rate limit, some rate limiting may need to done at the hosts. Ideally, the administrator would like to produce an equivalent configuration, but with the switch-based rate limiter instead divided into (potentially) many end-host rate limiters.

An *equational theory* provides a set of equations and inequalities that relate equivalent configurations, which makes it well-suited for guiding and reasoning about refactoring. For example, the equation

$$r \leftarrow s; (p + q) \equiv (r \leftarrow s; p) + (r \leftarrow s; q)$$

states that applying rate limiting before duplicating flows with $(+)$ is the same as applying two rate limiters after the duplication, one to each copy, which in turn indicates a valid program transformation for rate limiters. Section 7 discusses the challenges and benefits of developing an equational theory in general.

7. DISCUSSION

QtKAT is still early-stage work. In this section, we discuss several open questions going forward.

Performance and accuracy of automated verification. Section 6 describes how several QoS properties can be phrased with QtKAT. Interpreting each property with respect to the QtKAT semantics essentially generates a set of *verification conditions* in the form of systems of linear equations amenable to a solver, such as CPLEX [1]. It remains to be seen how the verification condition generation and the solver scale as the number of flows and the size and complexity of the network increases.

The accuracy of the model also bears investigation—is the model capable of expressing the traffic and service characteristics of real networks, and do properties verified against the model hold on real networks as well? The extensive work on network calculus gives us hope that the answers to both these questions will be "yes."

Equational theory. As its name implies, we hope that QtKAT will enjoy an equational theory drawn from the Kleene algebra with tests (KAT) [23]. An equational theory gives meaning to a language by relating terms in the language and provides a means of justifying program transformations (*i.e.* program rewriting based on the equational axioms) and verifying correctness properties.

A well-designed equational theory will be sound and complete with respect to the denotational semantics in Figure 3. Soundness implies that the programmer can rely entirely on the equational theory to reason about policies, and completeness implies that there are enough equations to relate all equivalent policies.

8. CONCLUSION

Quality of service analysis and configuration is intrinsically tied to network forwarding behavior. We present QtKAT, a unified language for configuring and verifying QoS and forwarding properties.

9. REFERENCES

[1] CPLEX optimizer. See `http://tinyurl.com/mnh8mlp`.

[2] Frenetic, 2013. See `github.com/frenetic-lang`.

[3] Openflow switch specification 1.4.0, 2013.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). SIGCOMM, 2010.

[5] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *POPL*, January 2014.

[6] T. Ball, N. Bjorner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. Vericon: towards verifying controller programs in software-defined networks. In *PLDI*, 2014.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. SIGCOMM, 2011.

[8] H. Ballani, D. Gunawardena, and T. Karagiannis. Network sharing in multi-tenant datacenters. Technical Report MSR-TR-2012-39, MSR, 2012.

[9] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford. A NICE way to test openflow applications. NSDI, 2012.

[10] C.-S. Chang. Stability, queue length, and delay of deterministic and stochastic queueing networks. *IEEE Transactions on Automatic Control*, 39(5):913–931, 1994.

[11] R. L. Cruz. A calculus for network delay, parts I and II. In *IEEE Transactions on Information Theory*, January 1991.

[12] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory networking: An API for application control of SDNs. In *SIGCOMM*, 2013.

[13] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ICFP*, September 2011.

[14] A. Guha, M. Reitblatt, and N. Foster. Machine-verified network controllers. In *PLDI*, June 2013.

[15] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. November 2010.

[16] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. SIGCOMM, 2013.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[18] V. Jeyakumar, M. Alizadeh, D. MaziÃÍres, B. Prabhakar, and C. Kim. EyeQ: Practical network performance isolation at the edge. In *NSDI*, April 2013.

[19] Y. Jiang. A basic stochastic network calculus. SIGCOMM, 2006.

[20] Y. Jiang and Y. Liu. *Stochastic network calculus*, volume 1. Springer, 2008.

[21] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *NSDI*, 2012.

[22] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying network-wide invariants in real time. In *NSDI*, 2013.

[23] D. Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, May 1997.

[24] J. Kurose. On computing per-session performance bounds in high-speed multi-hop computer networks. SIGMETRICS, 1992.

[25] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer, 2001.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM CCR*, 2008.

[27] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi. Tierless programming and reasoning for software-defined networks. NSDI, 2014.

[28] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. In *SIGCOMM*, August 2012.

[29] L. Popa, P. Yalagandula, S. Banarjee, J. Mogul, Y. Turner, and R. Santos. ElasticSwitch: practical work-conserving bandwidth guarantees for cloud computing. In *SIGCOMM*, August 2013.

[30] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, 2012.

[31] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster. Managing the network with merlin. HotNets, 2013.

[32] G. Stewart. Computational verification of network programs in coq. In *Certified Programs and*

Proofs. 2013.

[33] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak. Maple: Simplifying SDN programming using algorithmic policies. In *SIGCOMM*, 2013.