WNetKAT: Programming and Verifying Weighted Software-Defined Networks

Kim G. Larsen, Stefan Schmid, Bingtian Xue
Aalborg University, Denmark
{kgl,schmiste,bingt}@cs.aau.dk

Abstract
Programmability and verifiability lie at the heart of the Software-Defined Networking (SDN) paradigm. This paper presents WNetKAT, a network programming language accounting for the fact that networks are inherently weighted, and communications subject to capacity constraints (e.g., in terms of bandwidth) and costs (e.g., latency or monetary costs). WNetKAT is based on a syntactic and semantic extension of the NetKAT algebra, and comes with relevant applications, including cost- and capacity-aware reachability testing or service chaining. This paper also initiates the discussion of decidability and the relationship to weighted finite automata.

1 WNetKAT
On a high level, a computer network can be described as a set of nodes (hosts or routers) which are interconnected by a set of links, hence defining the network topology. While this high-level view is sufficient for many purposes, for example for reasoning about reachability, in practice, the situation is often more complex: both nodes and links come with capacity constraints (e.g., in terms of buffers, CPU, and bandwidth) and may be attributed with costs (e.g., monetary or in terms of performance). To reason about performance, cost, and fairness aspects, it is therefore important to take these dimensions into account.

The challenge of extending the state-of-the-art SDN language NetKAT [2] to weighted scenarios lies in the fact that in a weighted network, traffic flows can no longer be considered independently, but they may interfere: their packets compete for the shared resource. Moreover, packets of a given flow may not necessarily be propagated along a unique path, but may be split and distributed among multiple paths (in the so-called multi-path routing or splittable flow variant). Accordingly, a weighted extension of NetKAT must be able to deal with “inter-packet states”.

We can think of the network as a weighted (directed) graph $G = (V, E, w)$. Here, $V$ denotes the set of switches (or equivalently routers, and henceforth often simply called nodes), $E$ is the set of links (connected to the switches by ports), and $w$ is a weight function. The weight function $w$ applies to both nodes $V$ as well as links $E$. Moreover, a node and a link may be characterized by a vector of weights and also combine multiple resources: for example, a list of capacities (e.g., CPU and memory on nodes, or bandwidth on links) and a list of costs (e.g., performance, energy, or monetary costs).

We propose a weighted extension of NetKAT:

- WNetKAT includes a set of quantitative packet-variables to specify the quantitative information carried in the packet, in addition to the regular (non-quantitative) packet-variables of NetKAT (called fields in NetKAT): e.g., regular variables are used to describe locations, such as switch and port, or priorities, while quantitative variables are used to specify latency or energy. The set of all packet-variables is denoted by $V_p$.

*Research supported by the Danish VILLUM FONDEN project ReNet. A longer version of this paper is available as a technical report on arXiv [4]. We would like to thank Alexandra Silva, Nate Foster, and Dexter Kozen for many inputs and discussions on WNetKAT.
WNetKAT also allows for non-quantitative switch-variables, similar to those on the packet-variables in NetKAT. Moreover, in NetKAT. Concretely, the variable in $\rho_{\rho}$ rather of the head packet, and the test compares the value of the variable in $\rho$. (3) and (4) are the semantics for quantitative variable assignment and test respectively. In contrast to the non-quantitative variables, here the values of the variables are naturals and the value for updating the head packet or $\rho$ need to be calculated for the related values.

- **WNetKAT** also includes a set of *switch-variables*, denoted by $V_s$, to specify the configurations at the switch. Switch variables can either be quantitative (e.g., counters, meters, meta-rules [1, 5]) or non-quantitative (e.g., location related), as it is the case of the packet-variables.

In addition to introducing quantitative variables, we also need to extend the atomic actions and tests of NetKAT. Concretely, WNetKAT first supports non-quantitative assignments and non-quantitative tests on the non-quantitative switch-variables, similar to those on the packet-variables in NetKAT. Moreover, WNetKAT also allows for *quantitative assignments* and *quantitative tests*, defined as follows, where $x \in V_q$, $V' \subseteq V_q$, $\delta \in \mathbb{N}$, and $\nabla \in \{>, <, =, \geq, \leq\}$:

- **Quantitative Assignment** $x \leftarrow (\Sigma_{x' \in V'} x' + \delta)$: Read the current values of the variables in $V'$ and add them to $\delta$, then assign this result to $x$.

- **Quantitative Test** $x \triangleright (\Sigma_{x' \in V'} x' + \delta)$: Read the current value of the variables in $V'$ and add them to $\delta$, then compare this result to the current value of $x$.

Given the set of switches $V$, a *switch-variable valuation* is a partial function $\rho: V \times V_s \rightarrow \mathbb{N} \cup \Omega$. It associates, for each switch and each switch-variable, a integer or a value from $\Omega$. We emphasize that $\rho$ is a partial function, as some variables may not be defined at some switches.

A WNetKAT expression denotes a function $\llbracket \cdot \rrbracket : \rho \times H \rightarrow 2^H$, where $H$ is the set of packet histories. The semantics of WNetKAT is defined in Table 1, where $x \in V_n$, $y \in V_q$, $\delta \in \mathbb{N}$ and $\omega \in \Omega$.

**Example 1.** Consider the network in Figure 1. The topology of the network can be characterized with the following WNetKAT formula $t$, where $sw$ specifies the current location (switch) of the packet, $co$ specifies the cost, and $ca$ specifies the capacity along the links.
Figure 1: Example: A network hosting two (virtualized) functions $F_1$ and $F_2$. Function $F_2$ is allocated twice. The functions $F_1$ and $F_2$ may change the traffic rate.

$$t ::= sw = s; (sw ← F_1; co ← co + 1; ca ← \min\{ca, 8\}) \& (sw ← v; co ← co + 5; ca ← \min\{ca, 2\})$$

$$& sw = F_1; (sw ← F_2^{(1)}; co ← co + 3; ca ← \min\{ca, 1\}) \& (sw ← F_2^{(2)}; co ← co + 2; ca ← \min\{ca, 10\})$$

$$& sw = v; (sw ← F_2^{(1)}; co ← co + 3; ca ← \min\{ca, 3\}) \& (sw ← F_2^{(2)}; co ← co + 2; ca ← \min\{ca, 1\})$$

$$& sw = F_2^{(1)}; sw ← t; co ← co + 6; ca ← \min\{ca, 1\}$$

$$& sw = F_2^{(2)}; sw ← t; co ← co + 1; ca ← \min\{ca, 4\}$$

The variable $co$ accumulates the costs along the path, and the variable $ca$ records the smallest capacity along the path. Notice that $ca$ is just a packet-variable used to record the capacity of the path; it does not represent the capacity used by this packet (the latter is assumed to be negligible).

Assume that function $F_1$ is flow conserving (e.g., a NAT), while $F_2$ increases the flow rate by an additive constant $\gamma \in \mathbb{N}$ (e.g., a security related function, adding a watermark or an IPSec header). The policy of $F_2$ can be specified as:

$$p_{F_2} ::= (sw = F_2^{(1)} \& sw = F_2^{(2)}); ca ← ca + \gamma$$

In our future work, we will investigate the decidability of WNetKAT. In particular, it can be seen that a weighted NetKAT automata is a finite state weighted automaton $A = (S, s, F, \lambda, \mu)$ over a structure $K$ and alphabet $\Sigma$. From the equivalence between WNetKAT and weighted automata, it follows that deciding equivalence of two WNetKAT expressions is impossible. However, we also observe that in many practical scenarios, the above undecidability result is too general and does not apply. For example, many practical applications such as cost reachibility can actually be reduced to test emptiness: we often want to test whether a given WNetKAT expression $e$ equals 0, i.e., whether the corresponding weighted NetKAT automaton is empty.

Finally, we note that we currently witness a trend toward computationally more advanced and stateful packet-processing functionality, e.g., in the context of P4 [1] or OpenState [3]: these platforms are hence interesting compilation targets for \textit{WNetKAT}.

For more details, we refer the reader to the accompanying arXiv report [4].

References