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 releavant 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 $\mathcal{V}_p$.

*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.

$$[\![x \leftarrow \omega]\!](\rho,\ pk :: h) \quad = \quad \begin{cases} \{\rho,\ pk[\omega/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[\omega/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases} \tag{1}$$

$$[\![x = \omega]\!](\rho,\ pk :: h) \quad = \quad \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = \omega \\ & \text{or if } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = \omega \\ \emptyset & \text{otherwise} \end{cases} \tag{2}$$

$$[\![y \leftarrow (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) \quad = \quad \begin{cases} \{\rho,\ pk[r'/x] :: h\} & \text{if } x \in \mathcal{V}_p \\ \{\rho(v)[r'/x],\ pk :: h\} & \text{if } x \in \mathcal{V}_s \text{ and } pk(sw) = v \end{cases} \tag{3}$$
where $r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$

$$[\![y = (\Sigma_{y' \in \mathcal{V}'} y' + r)]\!](\rho,\ pk :: h) \quad = \quad \begin{cases} \{\rho,\ pk :: h\} & \text{if } x \in \mathcal{V}_p \text{ and } pk(x) = r' \\ & \text{or } x \in \mathcal{V}_s, pk(sw) = v \text{ and } \rho(v, x) = r' \\ \emptyset & \text{otherwise} \end{cases} \tag{4}$$
where $r' = \Sigma_{y_p \in \mathcal{V}' \cap \mathcal{V}_p} pk(y_p) + \Sigma_{y_s \in \mathcal{V}' \cap \mathcal{V}_q} \rho(v, y_s) + r$

Table 1: Semantics of *WNetKAT*: (1) and (2) describe the semantics for the non-quantitative variable assignment and test respectively. When the variable is a packet variable, the semantics are defined like in NetKAT. When the variable is a switch variable, the semantics of assignment changes the value of the variable in ρ rather of the head packet, and the test compares the value of the variable in ρ. (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 ρ need to be calculated for the related values.

- *WNetKAT* also includes a set of *switch-variables*, denoted by $\mathcal{V}_s$, 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 $x \in \mathcal{V}_q$, $\mathcal{V}' \subseteq \mathcal{V}_q$, $\delta \in \mathbb{N}$, $\bowtie \in \{>, <, \leq, \geq, =\}$:

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

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

A *WNetKAT* expression denotes a function $[\![\ ]\!] : \rho \times H \to 2^H$, where H is the set of packet histories. The semantics of *WNetKAT* is defined in Table 1, where $x \in \mathcal{V}_n, y \in \mathcal{V}_q$, $\delta \in \mathbb{N}$ 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.*

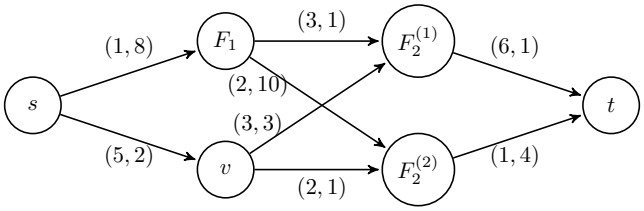

Figure 1: *Example:* A network hosting two (virtualized) functions F_1 and F_2. Function F_2 is allocated twice. The functions F_1 and F_2 may change the traffic rate.

$$
\begin{aligned}
t ::= \quad & sw = s; (sw \leftarrow F_1; co \leftarrow co + 1; ca \leftarrow \min\{ca, 8\} \\
& \qquad \& \; sw \leftarrow v; co \leftarrow co + 5; ca \leftarrow \min\{ca, 2\}) \\
& \& \; sw = F_1; \\
& \qquad (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 1\} \\
& \qquad \& \; sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 10\}) \\
& \& \; sw = v; (sw \leftarrow F_2^{(1)}; co \leftarrow co + 3; ca \leftarrow \min\{ca, 3\} \\
& \qquad \& \; sw \leftarrow F_2^{(2)}; co \leftarrow co + 2; ca \leftarrow \min\{ca, 1\}) \\
& \& \; sw = F_2^{(1)}; sw \leftarrow t; co \leftarrow co + 6; ca \leftarrow \min\{ca, 1\} \\
& \& \; sw = F_2^{(2)}; sw \leftarrow t; co \leftarrow co + 1; ca \leftarrow \min\{ca, 4\}
\end{aligned}
$$

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

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 Σ. 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 *WNetKAT*.

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

References

[1] Bosshart et al. P4: Programming protocol-independent packet processors. *SIGCOMM CCR*, 2014.

[2] C. Anderson et al. Netkat: Semantic foundations for networks. *SIGPLAN Not.*, 49(1), January 2014.

[3] G. Bianchi et al. Openstate: Programming platform-independent stateful openflow applications inside the switch. *SIGCOMM CCR*, 2014.

[4] K. Larsen et al. Wnetkat: A weighted sdn programming and verification language. *arXiv*, 2016.

[5] L. Schiff et al. In-band synchronization for distributed sdn control planes. *SIGCOMM CCR*, 2016.