UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

MIGUEL CARDOSO NEVES

# On Time-based Strategies for Optimizing Flow Tables in SDN

Graduation Work presented in partial fulfillment of the requirements for the degree of Computer Engineer

Prof. Dr. Marinho Pilla Barcellos
Advisor

Porto Alegre, December 2014

**CIP – CATALOGING-IN-PUBLICATION**

Neves, Miguel Cardoso

On Time-based Strategies for Optimizing Flow Tables in SDN / Miguel Cardoso Neves. – Porto Alegre: UFRGS, 2014.

51 f.: il.

Final Report (Master) – Universidade Federal do Rio Grande do Sul. Engenharia de Computação, Porto Alegre, BR–RS, 2014. Advisor: Marinho Pilla Barcellos.

1. SDN. 2. Flow Table. 3. Forwarding rule. 4. Durability. 5. Timeout. 6. Efficiency. 7. Evaluation. I. Barcellos, Marinho Pilla.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitora de Graduação: Prof. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do curso: Prof. Marcelo Goetz
Bibliotecário-Chefe do Instituto de Informática: Alexsander Borges Ribeiro

*“If you don’t love something,*
*you’re not going to go extra mile,*
*work the extra weekend,*
*challenge the* status quo *as much.”*
— Steve Jobs

*“The greatest glory in living*
*lies not in never falling,*
*but in rising every time we fall.”*
— Nelson Mandela

# AGRADECIMENTOS

Este trabalho de graduação representa muitas coisas para mim: a conclusão de uma etapa da minha vida, a realização de um sonho, um sinal de crescimento pessoal e intelectual, a satisfação de ajudar a construir conhecimento, uma confirmação do prazer que sinto em pesquisar. Sua materialização foi possível graças às pessoas que por algum momento, longo ou curto, estiveram comigo no caminho percorrido. Dedico este trabalho a essas pessoas.

*À família.* Agradeço aos meus pais, Rosane e Paulo, e aos meus familiares, que incentivaram a buscar os meus sonhos desde a infância. Agradeço por fazerem-se sempre presentes, apesar da distância. Obrigado aos meus tios, Sandra, Paulo e Tereza, por oferecerem um porto seguro, e a minha namorada, pelo carinho, paciência, apoio e compreensão recebidos.

*Aos amigos.* Agradeço aos meus amigos, que cruzaram o meu caminho pelos diferentes lugares onde passei, por estarem presentes nas horas boas e ruins. Amigos de SAP, do futebol, do bairro, das viagens de ônibus, da faculdade. Em especial, agradeço aos amigos que fiz dentro do grupo de redes de computadores, pelo companheirismo que excedia o ambiente de trabalho. Agradeço a esses também pelas discussões acadêmicas, pela ajuda no desenvolvimento desta pesquisa e pelos momentos de descontração que permeavam a sua realização.

*Aos professores.* Agradeço aos meus professores, do jardim de infância à universidade, por terem me proporcionado uma formação profissional de qualidade. Mais do que mestres, foram e ainda são verdadeiros amigos. Em especial, agradeço aos professores Luciano Gaspary, Luciana Buriol e Lisandro Granville, por contribuírem diretamente com este trabalho.

*Ao orientador.* Agradeço ao meu orientador, professor Marinho Barcellos, pelo exemplo de competência, persistência e dedicação. Agradeço também por incentivar o meu interesse pela pesquisa. Obrigado pelos conselhos, pelas cobranças, pelas reflexões, e por mostrar o valor da determinação.

*À universidade.* Agradeço à UFRGS e, em especial, ao Instituto de Informática por todo o suporte necessário para realizar este trabalho. Obrigado aos funcionários (da secretaria, biblioteca e laboratórios) pela competência com que propiciam aos alunos um ambiente de aprendizagem altamente qualificado.

# CONTENTS

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| CDF | Cumulative Distribution Function |
| LRU | Least Recently Used |
| POF | Protocol Oblivious Forwarding |
| SDN | Software Define Networking |
| TCAM | Ternary Content Adressable Memory |

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

Software Defined Networking (SDN) has allowed network operators to greatly improve traffic control through finer-grained network policies. However, finer-grained policies often increase the number of forwarding rules in flow tables, thus aggravating resource scarcity and performance. To tackle current limitations, a promising approach is to use time-based strategies for flow table control. Such strategies attempt to optimize the duration of rules to improve performance. In this work, we characterize the key properties of time-based strategies and discuss their implications on existing SDN networks. In particular, we investigate the effects on the number of rule reinstallations, a key concern that has major influence on flow completion times. Extensive experimentation is performed using representative workloads and accurate parameter sensitivity analysis. Our main findings indicate that strategies proposed in literature are still far from optimal and that their performance depends heavily on a precise parameterization. Additionally, important insights are provided to help understanding the benefits and limitations of current time-based strategies. These insights are useful for designing new forwarding devices, and provide the foundations for novel flow table optimization strategies to be investigated.

**Keywords:** SDN, Flow Table, forwarding rule, durability, timeout, efficiency, evaluation.

**Análise de Estratégias Temporais para Otimização de Tabelas de Fluxos em Redes Definidas por Software**

# RESUMO

As Redes Definidas por Software (SDN) permitiram um aperfeiçoamento no controle de tráfego em rede de computadores através do suporte à implantação de políticas de rede com alta granularidade. No entanto, tais políticas causam um aumento expressivo no número de regras de encaminhamento que devem ser instaladas nas Tabelas de Fluxos, agravando problemas de desempenho e escassez de recursos. Para enfrentar essas limitações, uma alternativa é utilizar estratégias temporais para controlar o uso dessas tabelas. Essas estratégias tentam otimizar a duração das regras de encaminhamento, reduzindo o número de regras presentes nas tabelas ao longo do tempo. Neste trabalho, nós caracterizamos as principais propriedades das estratégias temporais e discutimos suas implicações sobre as SDNs. Em particular, nós investigamos o efeito sobre o número de reinstalações de regras, fator determinante no tempo de conclusão dos fluxos. Com base em extensa experimentação feita sobre cargas de trabalho representativas e uma análise acurada de sensibilidade de parâmetros, nossos resultados indicam que as estratégias propostas na literatura estão longe do cenário ótimo e que o seu desempenho depende fortemente de uma parametrização adequada. Ademais, nossas conclusões ajudam a entender os benefícios e limitações das estratégias atuais, sendo úteis no desenvolvimento de novos dispositivos de encaminhamento bem como na construção dos fundamentos para que novas estratégias de otimização das tabelas de fluxos sejam investigadas.

**Palavras-chave:** SDN, Tabela de Fluxos, regra de encaminhamento, durabilidade, timeout, eficiência, avaliação.

# 1 INTRODUCTION

Network configuration and management nowadays requires individually programming vendor-specific forwarding equipment (e.g. switches, routers, etc). As a consequence, operating and maintaining a computer network becomes an arduous task to network administrators, who must be "masters of complexity". Besides, the set of services and functionality that can be offered become restricted. In this context, Software Defined Networking (SDN) emerged as a paradigm to facilitate innovation and simplify administration tasks in networks (KREUTZ et al., 2014).

In SDN, a logically centrilized entity (i.e. the network controller) has global view of the network and enables administrators to flexibly and dynamically configure networking devices using an open, standard interface (such as OpenFlow (MCKEOWN et al., 2008)). To configure a device, networking policies are installed as forwarding rules in entries of flow tables. These tables allow fine-grained traffic control, but require a large number of rules need to be installed (MOSHREF et al., 2013). Unfortunately, table size is limited due to cost and performance constraints on the device resource capacity. In a general sense, the smaller the flow table, the better the performance of the forwarding device (KUZNIAR; PERESINI; KOSTIC, 2014).

To deal with this limitation, a number of alternatives have recently been proposed in the literature (KANG et al., 2012; KANIZO; HAY; KESLASSY, 2013; KOGAN et al., 2014). Time-based strategies, in particular, constitute a promising class of solutions. They adjust the duration of rules in tables (ZAREK, 2012) so that fewer entries are occupied and the number of rule reinstallations is reduced. In this work, we characterize the key properties of time-based strategies and discuss their implications on existing SDN networks. More specifically, we investigate the effects on the number of rule reinstallations, a key concern that has major influence on flow completion times.

We perform an extensive experimentation using representative workloads and an accurate parameter sensitivity analysis. Our main findings provide important insights to help understanding the benefits and limitations of current time-based approaches. These insights are useful for designing new forwarding devices and provide the foundations for novel flow table optimization strategies to be investigated.

The remainder of this work is organized as follows. Chapter 2 introduces the SDN forwarding model, motivates this work, and gives a general overview of related work. Chapter 3 describes properties and principles of the main time-based strategies, a largely unexplored class of alternatives. Experimental findings are presented in Chapter 4, while Chapter 5 summarizes the main conclusions and discusses possibilities for future work.

# 2 BACKGROUND: ON FLOW TABLES AND THEIR UTILIZATION

This chapter provides the fundamental concepts related to this work. We begin by introducing the SDN forwarding model. Then, we describe the main motivation for minimizing the number of entries in flow tables. Finally, we discuss the state-of-the-art, presenting strategies that could be jointly applied with the ones evaluated in this work.

## 2.1 SDN forwarding model

Although the idea of programmable networks and decoupled control logic has been around for many years, the strong support from both industry and academia to the OpenFlow SDN implementation made it a singular success case and a *de facto* standard (FEAMSTER; REXFORD; ZEGURA, 2013; NUNES et al., 2014). Therefore, the discussion presented in this document will be based on the OpenFlow use case and arguments will be generalized whenever necessary.

An OpenFlow switch, in its current model, contains a rich set of forwarding abstractions (see Figure 2.1). The core of this model is the flow table. An SDN device can have multiple flow tables, typically organized in a pipeline. Each table contains one or more packet handling rules. Depending on the rules installed by a controller application, an OpenFlow switch can behave like a router, switch, firewall or middlebox. Each rule specifies matches for a subset of the traffic, actions to be performed on matching packets, and counters to keep statistics of interest (e.g. number of bytes or packets matched) (OPENFLOW SPECIFICATION, 2014).

Matches can be based on different combinations of packet headers. The set of possible headers, which ultimately determines the size of the rules, is on the increase since the initial versions of OpenFlow (dated from 2008), from around 12 to over 40 elements. This number tends to grow due to mechanisms like extensible matchings and protocol oblivious forwarding (POF) (BOSSHART et al., 2013; SONG, 2013). As a consequence, a single rule tends to consume more capacity (i.e. processing and storage) of the forwarding device. In addition, matchings can also be exact or wildcard-based, being this characteristic determinant to the granularity of the traffic control.

Actions, on the other hand, have a more restricted set, and can involve forward the packet to outgoing port(s), encapsulate and forward it to the controller, drop it, and send it to the next table in the pipeline or to special tables such as group and meter tables. The network controller can manage flow tables (and the SDN forwarding model as a whole) through a determined set of operations, defined at the OpenFlow specification. These operations include adding, deleting and modifying forwarding rules

(OPENFLOW SPECIFICATION, 2014).

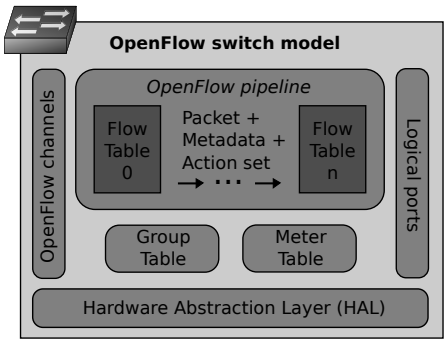



Figure 2.1: Current OpenFlow switch model.

## 2.2 Motivating examples

The performance of the SDN forwarding model is tightly dependent on the device architecture in which it is implemented. At the present time, there are two popular architectures for implementing SDN forwarding devices: i) general-purpose CPU + RAM and ii) dedicated hardware (ASIC, network processor, etc) + TCAM (KATTA et al., 2014). Each one has its own strengths and limitations, but both share a common principle: the smaller the size of the flow tables the better. In what follows, we provide two motivating examples explaining why this is a valid principle.

**Energy consumption.** Due to their low-latency lookup performance - $O(1)$ and consequent high-throughput packet processing, Ternary Content Addressable Memories (TCAMs) have become the common choice to hold flow tables. However, is spite of their advantages, TCAMs are typically small in size in current packet forwarding elements. The main reason, as pointed in the literature, is because they have a high power consumption (easily achieving tens or hundreds of Watts in a core router), normally representing a major power drain to the devices (AGRAWAL; SHERWOOD, 2006; KREUTZ et al., 2014; BANERJEE; KANNAN, 2014; NARLIKAR; BASU; ZANE, 2003).

According to (AGRAWAL; SHERWOOD, 2006), the excessive power consumption is due to the fact that all active memory elements are used in each lookup task. Hence, the consumption is proportional to the number of entries active and the frequency of lookups. Note that, in this scenario, many lookups in a memory with little active entries can consume less power than few lookups in a memory with many active entries. As the demand for devices containing larger TCAMs tends to grow in the coming years, pushed mainly by an increasing in the number and width of forwarding rules, looking for alternatives that reduce the size of the flow tables without loss of precision to the traffic control is an inevitable and mandatory task.

**Packet processing delay.** An alternative to the high power-consuming TCAM implementation of flow tables is to implement them algorithmically using variations of RAM memories (e.g. DRAM or SRAM). In this case, despite their low cost (up to 400x lower per Mbit than the TCAM one) and power consumption (up to 100x lower), packet processing times suffer from greater (typically three orders of magnitude greater than in TCAMs) and varying latencies (LAZARIS et al., 2014; KATTA et al., 2014). According to (QI et al., 2009), latency in this case varies with the number and dimension (i.e.

amount of fields) of the entries stored in the table. As a consequence, we fall into the same challenge of reducing the size of flow tables presented for TCAMs.

## 2.3 Space-based strategies

This section overviews space-based strategies for optimizing flow tables. They are based on the principle of reducing the amount of resources needed to represent a set of forwarding rules at a given moment (i.e. not acting over the duration of the rules). Note that these strategies could be used in conjunction with time-based ones, which we expose in the next chapter.

### 2.3.1 Transforming network policies in forwarding rules

Nowadays, configuring SDN switches requires that a programmer explicitly creates and manages flow rule patterns and priorities. This is usually a complex, redundant and error prone process and it is not uncommon that network operators waste precious time and network (forwarding) resources codifying complex policies into their respective sets of rules (KANG et al., 2012; VOELLMY et al., 2013). Trying to optimize the forwarding resources utilization, in this case, is closely related to the need of simplifying SDN programming. Therefore, there has been a strong effort of the SDN community to create high level abstractions (languages, data structures, etc) to facilitate the management of the low level details in the switch programming process (FOSTER et al., 2013).

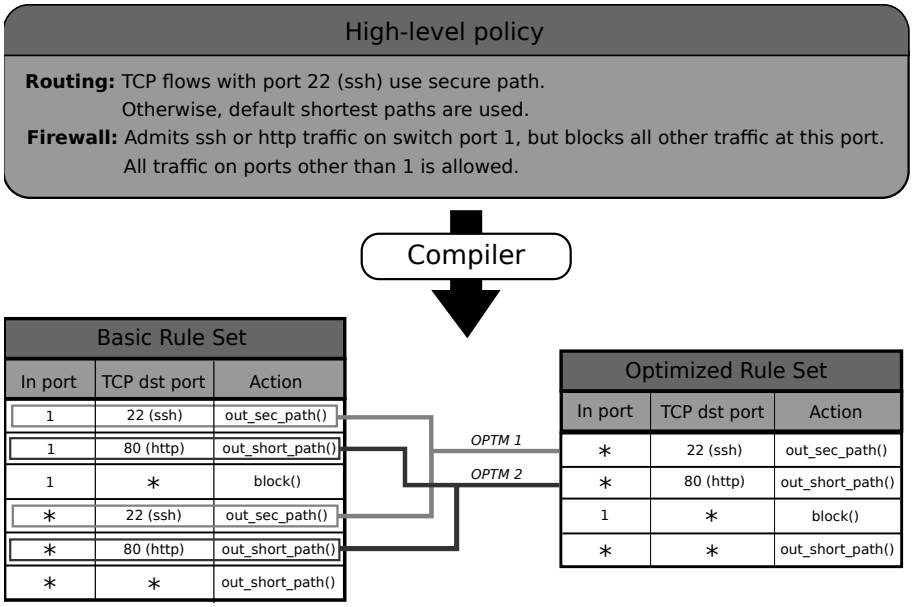



Figure 2.2: Translation of a network-wide policy in its respective rule set.

Figure 2.2 illustrates the general ideas behind major related proposals. This figure depicts a basic network-wide policy. For the sake of simplicity, and to keep the explanation agnostic of a specific proposal, we used natural language to represent it. Nevertheless, in practice more restricted languages are actually applied (typically specific of each proposal). Resembling a conventional computing system, a compiler is in charge of translating the high-level policy into a set of forwarding rules. In the process, it attempts to reduce the number of rules generated, thus saving entries in the flow tables. In the exam-

ple, a basic rule set (one rule for each possible and relevant combination of type of traffic and switch port) is presented in the left table. On the right, we represent the respective optimized rule set, where rules regarding ssh and http traffic were joined (*OPTM 1* and *OPTM 2*, respectively) without changing the semantic of the policy.

Priorities are implicit in the tables (lowest-priority rule is at the bottom) and are typically used in the optimization process (VOELLMY et al., 2013). Note that although the way of representing the rules changes (through the use of wildcards), it is only considered information regarding the semantic of the policy to apply the optimizations. This strategy is fundamentally different from the one employed by proposals described in Section 2.3.3, where just information about the structure of the rules are considered to apply the optimizations.

Another important consideration relates to the high-level abstraction expressiveness. There are many different types of policies (e.g. access control, measurement, routing or firewall policies), and in some cases their peculiarities can be better expressed with mechanisms from specific languages (e.g. support for algorithmic (VOELLMY et al., 2013), declarative (FOSTER et al., 2011), hierarchical (FERGUSON et al., 2012) or pipelined policies (SCHLESINGER; GREENBERG; WALKER, 2014), logic programming (KATTA; REXFORD; WALKER, 2012), etc). Moreover, different compilers can apply different optimization mechanisms, affecting rule set size depending on the policy characteristics. This implies that choosing the high-level abstraction is an important step towards the efficient utilization of the network forwarding resources and hence must be carefully done.

### 2.3.2 Placing rules in the network

Depending on the set of policies to apply, even with the utilization of a highly efficient translation mechanism (i.e. a compiler capable of generating the minimum number of forwarding rules), this number can still be large. As an example, consider a datacenter operator who wishes to apply a bandwidth allocation policy to the communication among the VMs of each tenant. In this case, at least one rule will be required for each pair of communicating VMs (MOSHREF et al., 2013).

The typically large number of forwarding rules to be installed in the networks, and potentially complex interactions among these rules (e.g. rule dependencies), have motivated attempts to improve rule placement (NGUYEN et al., 2014; KANG et al., 2013; COHEN et al., 2014). Figure 2.3 illustrates a basic placement scenario, in which a simple topology (switches $S1, ..., S3$) connects four independent systems ($N1, ..., N4$). Each switch in the topology has a flow table whose capacity is represented inside brackets (e.g. table $T2$ at switch $S2$ supports 4 rules). Furthermore, each link has a limited amount of bandwidth, not represented.

Now consider three distinct flows are to be sent through the infrastructure described above. Each flow has its own set of forwarding rules, which is presented in the left part of the figure. Generally, the rule placement problem consists in positioning the rules in the network to allow the related flows to be allocated while respecting all the table and link capacity constraints (COHEN et al., 2014). A possible placement is presented in the figure.

It is common to separate rules regarding routing and endpoint policies when considering their placement. This happens because routing policy rules typically need to be present in all devices along a flow path, while the endpoint rules must be present just one time and can be spread along it. In this sense, the total number of rules can vary depend-

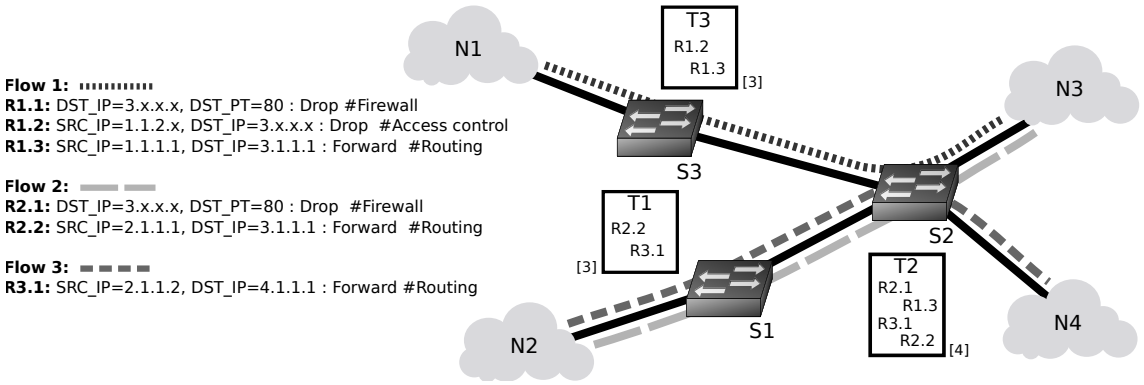



Figure 2.3: Placement of rules in the network infrastructure.

ing on how endpoint policy rules are spread along paths (KANIZO; HAY; KESLASSY, 2013; KANG et al., 2013). As an example, suppose we had placed rule $R1.2$ instead of $R2.1$ at switch $S2$. In this case, rules $R1.1$ and $R2.1$ would be necessary at switches $S3$ and $S1$ respectively, thus increasing the total number of rules. Regarding paths, these can be predefined (input to the placement algorithms) or timely calculated (NGUYEN et al., 2014). Moreover, it is also possible to apply objectives other than minimizing the number of rules to the placement strategies (e.g. to minimize the energy consumption looking for a minimal set of active links in the infrastructure (GIROIRE; MOULIERAC; PHAN, 2014)).

### 2.3.3 Changing the way of representing rules

The SDN forwarding model is, in essence, based on legacy packet classifiers, where tuples are stored in tables and compared against packet header fields. In this sense, efficient packet classification is a core concern to provide high-performance network services (KOGAN et al., 2014). However, the SDN environment brings a significant stress over conventional packet classification mechanisms, mainly due to its flexible and fine-grained nature.

Different from classical 5-tuple IP lookup structures, SDN rules can require the analysis of many more fields and the length of a rule can exceed 1000 bits in latest OpenFlow versions (OPENFLOW SPECIFICATION, 2014). Since switch memory resources are fundamentally limited, and rule manipulations (e.g. lookup, update, etc) get slower when flow table occupancy gets higher (KUZNIAR; PERESINI; KOSTIC, 2014), a bunch of alternatives that change the way of representing and processing rules can be found in the literature (e.g. (MA; BANERJEE, 2012), (KOGAN et al., 2014) and (AFEK; BREMLER-BARR; SCHIFF, 2014)). In this sense, Figure 2.4 illustrates their relevant aspects.

In the figure, the same rule set is represented in three different forms (using 1, 2 and 4 pipelined tables). Note that the number of entries in each table as well as the total number of entries varies according to the representation. In addition, note that although the number of entries is lower in the 4-table structure, a greater number of lookups is required to process a specific rule compared to the 1-table structure. This increases the packet classification time and shows the existence of a trade-off between space and speed that depends on the representation strategy (KESSELMAN et al., 2013). In this sense, the rep-

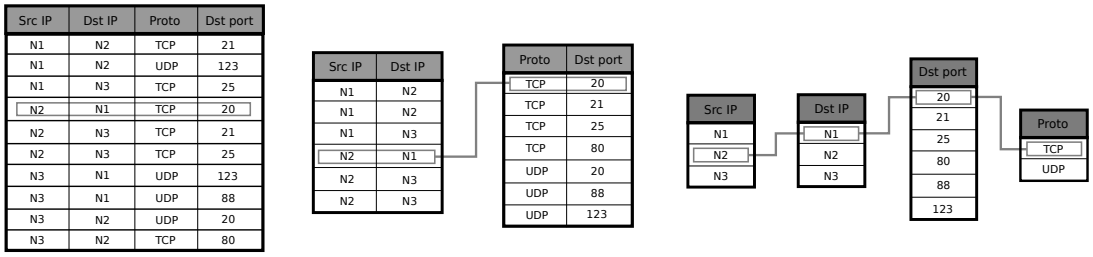



Figure 2.4: Three different forms of representing rules.

resentation structure is normally derived from the hardware capabilities of the forwarding devices (i.e. number of tables supported, CPU and memory capacities, etc) and performance requirements of network programmers (e.g. latency and throughput of packet classification).

Despite the extensive use of exact match rules in the example, it is common to apply field wildcards to represent a desired forwarding behavior, which greatly reduces the number of rules needed. Moreover, it is also possible to compact rules through data codification and minimization schemes (e.g. boolean minimization) (KOGAN et al., 2014; ROTTENSTREICH et al., 2014). However, the former (wildcarding) reduces the accuracy of the traffic control due to the coarse-grained processing, and the latter typically requires a packet pre-processing before its classification.

# 3 TIME-BASED STRATEGIES

In the previous chapter, we presented the background for a better understanding of this work. Now, in this one, we focus our attention on the time-based approaches, as they are our main object of study. Their common principle is to act setting the duration of the rules for minimizing the number of entries at flow tables along the time, while avoiding rule reinstallations. We begin in Section 3.1 with a description of the main properties that characterize them. Then, in Section 3.2, we present the strategies.

## 3.1 Properties

We can characterize time-based strategies for optimizing flow tables according to three main properties: the rule caching algorithm they apply when the table is full (i.e. has reached the switch capacity), the nature of their rule timeout mechanism and the depth of their knowledge about the state of the flows. We provide more details about each property below.

*Rule caching algorithm.* The rule caching algorithm determines the behavior of the time-based strategies when the number of active (i.e. used) entries in the flow table has reached the switch capacity. At this operational state, before installing a new rule, the caching algorithm needs to select an entry and evict the rule from there. Rule caching algorithms have distinct processing requirements and overheads, which means that the rule caching efficiency varies with different operational conditions of the network. The rule caching algorithms adopted in this work are the same ones employed by the time based flow table optimization strategies studied. Although more sophisticated algorithms exist (YU et al., 2010; KATTA et al., 2014; LEE; KANAGAVELU; AUNG, 2013; YAN et al., 2014), using the same ones is necessary to keep the results comparable.

*Timeout nature.* Most of the current SDN implementations (i.e. OpenFlow based ones) offers two types of rule timeouts: inactivity (idle) and hard timeouts. Idle timeouts cause the flow entry to be removed if no packet matches it within a certain time interval. Hard timeouts, on the other hand, cause rule removals after a given period, regardless of how many packets it has matched along the time (LEE; KANAGAVELU; AUNG, 2013). All of the strategies presented in this work use idle timeouts as part of their rule removal mechanism and we are not aware of any work using hard timeouts up to now. We believe the main reason is that a hard timeout can cause a rule removal during the transmission of a burst of packets, which increases latency and damages network performance.

In addition to the idle and hard timeout categorization, we can classify rule timeout mechanisms into static or dynamic. A static timeout mechanism sets a fixed timeout value to the rules independent of the evolution of the flows (i.e. a rule is set with the same timeout no matter it has already experienced a reinstallation event, for example).

Dynamic timeout mechanisms, on the other hand, adjust timeout values according to the behavior of the flows (e.g. the timeout varies with the number of reinstallations a rule has already experienced).

*Knowledge about flow state.* The depth of the knowledge about the state of the flows (i.e. their duration, packet inter-arrival times, number of bytes sent, etc) can impact the performance of the caching and timeout mechanisms applied by time-based strategies. Intuitively, the greater the knowledge about the flow state, the more accurate is the decision about the duration of its forwarding rules. However, to acquire this state is often a costly process that involves monitoring traffic and/or collecting statistics. It is out of the scope of this work to evaluate the cost of acquiring the state of the flows in an SDN, and we state that there is a lot of work focusing on optimizing this process (SEKAR et al., 2008; MANN; VISHNOI; BIDKAR, 2013). Instead, we concentrate our analysis on the effects of knowing the state of the flows over the performance of the time-based strategies as they were originally proposed (i.e. without modifying their state acquisition mechanism).

## 3.2 Strategies

In this section we describe the main time-based strategies found in the literature for optimizing flow tables in SDN. We focus on presenting their principles and characterizing them according to the properties introduced in the previous section. More details can be found at the references. Furthermore, at the end of the section we present Table 3.3, which summarizes their relevant information.

### 3.2.1 ST - Static timeout

This is the baseline. The main idea is to use the same fixed timeout value for every rule installed in the table. Therefore, this is the unique parameter of this strategy, which we will call $t_{ST}$ hereafter. In this sense, (ZAREK, 2012) state that it is possible to find a timeout value from which any increasing cause insignificant reduction in the number of reinstallations and unnecessary growing in the resource utilization, named as *Break-even point*. We also use this principle when evaluating this strategy. Moreover, as there is no specific rule caching algorithm tied to this approach, we consider the same applied by (VISHNOI et al., 2014) in their experiments (i.e. a random solution).

### 3.2.2 PEV - Probabilistic Early Eviction.

This strategy is based on the proposals of (ZAREK, 2012) and (KANNAN; BANERJEE, 2014). Its basic principles are: i) to set a long and static timeout when installing forwarding rules and ii) to monitor the state of the flows for proactively deleting unnecessary rules before their actual expiration.

We have found two different mechanisms in the literature for determining the best moment of deleting a rule. The first one looks for flags that describe the flow state (e.g. SYN, ACK and FIN of TCP connections) inspecting packets. The second one, on the other hand, tries to delete rules in periodic intervals based on the rule inactivity time and probabilities determined from traffic characteristics (e.g. flow duration, packet inter-arrival times, etc). We decided to implement the second mechanism in this work, since it is more generic (i.e. independent of the traffic type).

Note that the deletion process, in this case, requires either an extra message from the controller or a special mechanism at the forwarding device to sinalize the rule eviction. When the table is full, the authors suggest the application of a least recently used (LRU)

caching algorithm, evicting the rule that is longest inactive to install the new one. Table 3.1 summarizes the parameters involved in this strategy.

| Parameter | Description |
|---|---|
| $t_{PEV}$ | Timeout set when installing rules |
| $DI$ | Periodic interval for trying to delete rules |
| $P(i)$ | Probability of deleting a rule inactive during the last $i$ time units ($i < t_{PEV}$) |

Table 3.1: Probabilistic Early Eviction - Description of parameters.

### 3.2.3 AI - Adaptative increasing

This strategy was developed by (VISHNOI et al., 2014). It is based on the principle of increasing the timeout every time its respective rule needs to be reinstalled. As a consequence, there are three intrinsic parameters involved: an initial and an upper bound to the timeout value, called "*min timeout*" and "*max timeout*", respectively, and an increasing function, that determines the values the timeout can take. When the flow table becomes full, the authors suggest the use of a random rule caching algorithm. Moreover, there is no process of gathering information about the state of the flows from the network in this strategy, which reduces the cost of managing the flow table (i.e. the communication overhead in the control channel). Table 3.2 summarizes the parameters of this strategy.

| Parameter | Description |
|---|---|
| $t_{AI}^{min}$ | *min timeout* - Initial value that the timeout assumes |
| $t_{AI}^{max}$ | *max timeout* - Upper bound to the timeout |
| $f(n\_reinstall)$ | Timeout increasing function, dependent on the number of reinstallations the rule has already experienced |

Table 3.2: Adaptative increasing - Description of parameters.

To finish this chapter, we summarize the most important information about each strategy in the Table 3.3.

| Strategy | Proposals | Timeout nature | Rule caching algorithm | Knowledge about flow state | Main principle |
|---|---|---|---|---|---|
| ST | - | Static | Random | Low | Determine an adequate timeout value which is applied to all rules |
| PEV | (KANNAN; BANERJEE, 2014) (ZAREK, 2012) | Static | LRU | High | Use a long timeout value and periodically evict inactive rules |
| AI | (VISHNOI et al., 2014) | Dynamic | Random | Low | Increase the timeout value when a rule is reinstalled |

Table 3.3: Time-based strategies - Summary.

# 4 EVALUATION

This chapter presents the experiments used to measure the performance of the time-based strategies as well as the results we obtained. We have two main objectives in this evaluation: i) to compare the performances of the strategies, identifying characteristics and scenarios that directly affect them and ii) to quantify how near the strategies are from the optimal considering both the number of reinstallations and the resource utilization, analysing the feasibility and necessity of possible improvements.

## 4.1 Methodology

**Testbed.** Experiments are performed on Mininet (version 2.0.0), a network emulation framework based on Linux containers. The framework was executed on an Intel i7 with 4 cores of 3.1 GHz and 16 GB of RAM running Ubuntu 13.04. We use the minimal topology with two hosts connected by a single switch described in Figure 4.1. To leverage the population of the flow table at the switch, we run socket-based sender and receiver programs as applications at the hosts. Each application triggered in the testbed randomly selects one flow description from the workload acording to a uniform distribution and replays it.

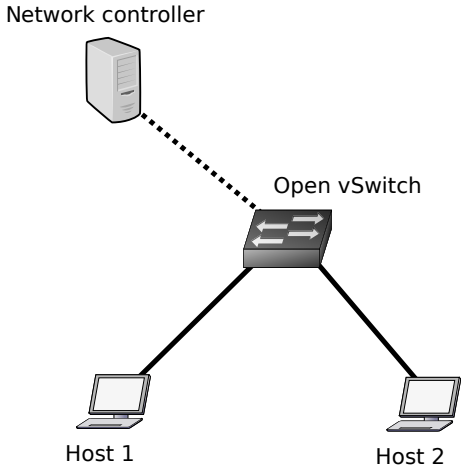



Figure 4.1: Topology used in the experiments.

Each experiment lasts 10 minutes and we set the number of active flows to 300 (equivalent to 600 forwarding rules as we used TCP connections) along the whole execution, with a new application (flow) initiating immediately after another one finishes. We have

implemented the strategies as applications in the POX controller. Furthermore, the controller is also responsible for enforcing a limitation on the size of the flow table at the switch, evicting rules according to the rule caching algorithm of each strategy (application) whenever a desired capacity is achieved. All rules have exact matches.

**Workloads.** We use two workloads in our experiments. Each one is composed of a set of flows which are described by their duration and packet inter-arrival times. Due to Mininet performance constraints and current OpenFlow timeout granularity (in the order of seconds), we model a burst of packets by its first occurence and consider just durations and inter-arrival times greater than 1 second. The first workload, named "*Workload 1*", has packet inter-arrival times according to a Poisson process while the second one, named "*Workload 2*", follows a Weibull distribution. The flow duration of both workloads follows a Lognormal process. This choice of setting was based on the literature (BENSON; AKELLA; MALTZ, 2010; KARAGIANNIS et al., 2004) and is summarized in Table 4.1, plus the parameterization of the distributions.

| **Workload** | **Flow Duration** | **Packet inter-arrival times** |
|---|---|---|
| *Workload 1* | Lognormal - $\mu = 4$, $\sigma = 1$ | Poisson - $\lambda = 10$ |
| *Workload 2* | Lognormal - $\mu = 4$, $\sigma = 1$ | Weibull - $\alpha = 4$, $\beta = 0.5$ |

Table 4.1: Description of the workloads.

**Metrics.** We evaluate the proposals using two metrics. The *number of rule reinstallations per flow* captures the impact of removing a rule from the flow table before the end of the flow. In practice, an early removal tends to increase the flow completion time, an undesired effect that can damage the user experience. Moreover, we also measure the *utilization of the switch capacity*, an indicative of how well the strategies are using the forwarding resources for a given load. For all metrics, in the context of this work, the smaller the value, the better.

**Comparison parameter.** We use the size of the flow table (defined as *switch capacity*), in number of entries, as comparison parameter. This definition is based on the literature (ZAREK, 2012; VISHNOI et al., 2014; KANNAN; BANERJEE, 2014). We use values ranging from 300 to 1000, which enables us to evaluate the impact of the duration of the rules considering different expected table occupancies, as we have fixed the number of active flows.

## 4.2 Basic settings

Before effectively comparing the strategies, in this section we first define the best possible values to their parameters, allowing us to obtain a precise evaluation of their performance. The parameters were first defined empirically, and then tunned until there was no sensitive improvement in the quality of the solutions. Tables 4.2, 4.3 and 4.4 show the values tested for ST, PEV and AI, respectively. Bold values indicate the best configuration achieved. Furthermore, we considered a switch capacity of 600 entries and used *Workload 1* in these experiments. All timing values are in seconds.

| Parameter | Tested values |
|---|---|
| $t_{ST}$ | 5, 10, **20**, 30, 60 |

Table 4.2: Static Timeout - Parameter tunning.

| Parameter | Tested values |
|---|---|
| $t_{PEV}$ | 5, 10, **20**, 30, 60 |
| $DI$ | 5, 20, **60** |
| $P(i)$ | CDF of the duration of the flows<br>**CDF of the packet inter-arrival times** |

Table 4.3: Probabilistic Early Eviction - Parameter tunning.

| Parameter | Tested values |
|---|---|
| $t_{AI}^{min}$ | 1, 2, 5, 10, **20**, 30 |
| $t_{AI}^{max}$ | 30, **60**, 120 |
| $f(n\_reinstall)$ | Linear - $t_{AI}^{min} \times n\_reinstall$<br>**Exponential** - $t_{AI}^{min} \times 2^{n\_reinstall}$ |

Table 4.4: Adaptative Increasing - Parameter tunning.

Note that, according to the best configuration achieved for each strategy, $t_{ST} = t_{PEV} = t_{AI}^{min}$. Hereafter, we consider this set of parameters ($t_{ST}$, $t_{PEV}$ and $t_{AI}^{min}$) a "*reference timeout*" and base our discussion in variations of this value.

## 4.3 Results

This section presents results obtained in the evaluation through the analysis of the metrics described in Section 4.1. All results were obtained considering the *Workload 1*, except when mentioned contrary.

### 4.3.1 Resource-constrained switches

For smaller switches, where our workload overloads the resources (i.e. size of the flow table supported) in the forwarding device, it does not make sense to anticipate rule removals looking for a reduction in the size of the flow table. In other words, it would be possible to apply short timeouts or to hastily delete rules trying to reduce the resource utilization, but it would certainly lead to many more reinstallations. Hence, despite the use of timeout mechanisms, the major component in the performance of time-based strategies in these scenarios are their rule caching algorithms, as capacity utilization is constantly around 100% (see Table 4.5 for results obtained in our experiments).

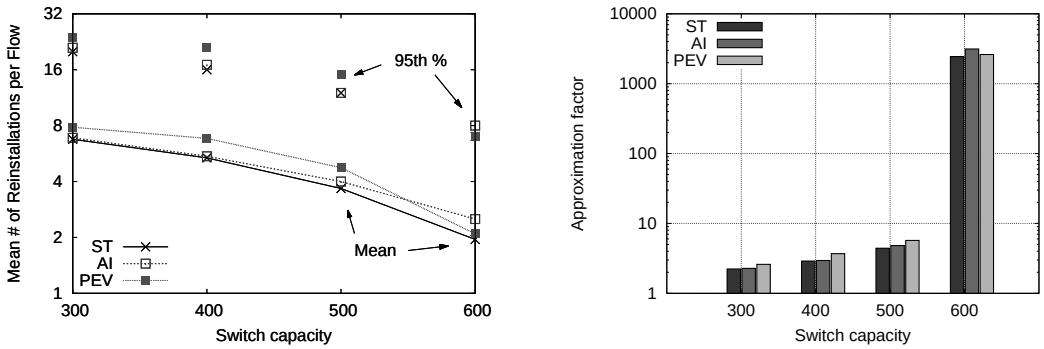
Considering the evaluated strategies, the algorithm of PEV (LRU) performed worse compared to the other one (both ST and AI use a random selection algorithm to evict a rule from the flow table when it is full), as can be seen in Figure 4.2. Note that there is an

| Statistic | Switch capacity | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 300 | | | 400 | | | 500 | | | 600 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| Avg. | 96.9 | 93.0 | 94.0 | 97.3 | 94.4 | 95.6 | 98.0 | 96.2 | 97.2 | 98.3 | 97.3 | 98.0 |
| Std. dev. | 4.4 | 8.2 | 7.7 | 3.6 | 6.2 | 5.5 | 3.6 | 4.4 | 4.1 | 3.8 | 3.9 | 4.7 |

Table 4.5: Average utilization considering resource constrained switches.

inversion in the behavior when the switch capacity approximates to the workload demand, but in this case timeout mechanisms and early evictions can also influence the results, as pointed in (ZAREK, 2012). Additionally, note the tail behavior regarding the number of reinstallations (high 95th percentile shown in Figure 4.2(a)), indicating that special mechanisms are needed in situations where deadline constrained flows are present, for example.

To motivate the development of better rule caching algorithms for this scenario, we compared the performance of the ones applied in current time-based strategies against an optimal rule caching solution. As the objective is to minimize the number of rule reinstallations, the optimal solution evicts the flow which next packet is farthest in the future. To do this, it considers the exact flow duration and packet inter-arrival times (information that can be estimated from traffic in practice). Figure 4.2(b) presents the results. As we can see, all strategies perform more than 2x worse (the smaller the approximation factor, the better) than the optimal solution in the evaluated scenarios. Note that the approximation factor increases when the device capacity tends to the workload demand, mostly because the optimal solution knows the duration of the flows and hence proactively evicts a rule exactly when its respective flow finishes.




(a) Rule reinstallation.

(b) Approximation factor.[1]

Figure 4.2: Results obtained for resource-constrained switches.

### 4.3.2 Large capacity switches

For large switches, or in situations where the workload does not overload the resources of the forwarding device, to anticipate the removal of unnecessary rules can greatly improve the efficiency in the resource utilization (i.e. the ability of using just necessary

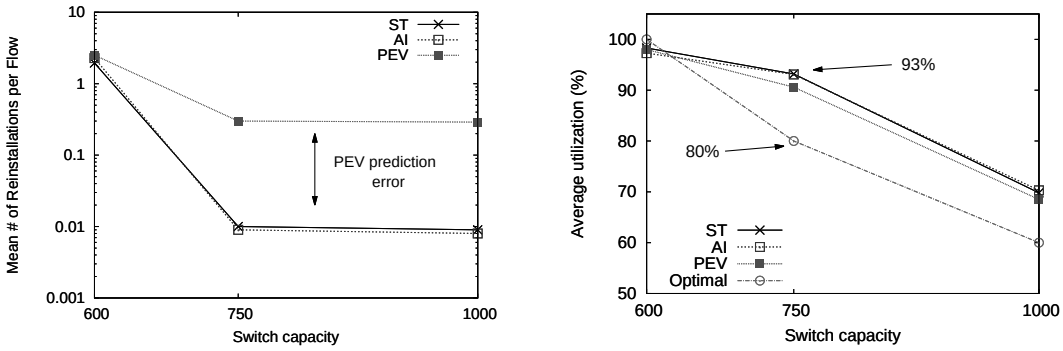
[1]We evaluated the optimal solution in the 300, 400, 500 and 599 switch capacity scenarios, since there is no necessity of applying the cache policy when the switch capacity is equal to the workload demand.

resources). Ultimately, this action has the potential for improving performance and/or saving capital (CAPEX) and operational (OPEX) expenditures of the network as a whole, as described in Section 2.2. In this section, we evaluate the performance of the time-based strategies considering large-capacity switches. Section 4.3.2.1 focuses on results obtained considering a tunned reference timeout, while Section 4.3.2.2 shows what happens when the reference timeout can not be precisely defined (i.e. when it is necessary to use estimated values to the reference timeout). Finally, Section 4.3.2.3 presents a proof of concept. Note that, at the optimal scenario, there is no rule reinstallation when the capacity of the devices is greater than the workload demand.

#### *4.3.2.1 Tunned reference timeout*

Figure 4.3 shows the results for a tunned reference timeout, achieved through the process described in Section 4.2. As we can see, all strategies achieved less than one reinstallation per flow at the average, indicating that a portion of the flows does not suffer reinstallations. Note that the PEV strategy has an intrinsic prediction error, which leads to worse results compared to the other strategies. On the other hand, it also leads to lower resource utilization (about 10% considering the 5th percentile). Nevertheless, all strategies use a considerable amount of resources more than necessary (about 13% at the average), as can be seen in Figure 4.3(b).

Tables 4.6 and 4.7 show the 95th percentile and maximal of the number of reinstallations and the standard deviation and 5th percentile of the resource utilization, respectively. As can be seen, the tail behavior of the number of reinstallations almost disappear when the amount of resources is sufficiently large (95th % tends to zero). However, resource utilization is constantly inneficient since the 5th % is near or greater than the optimal utilization for all strategies, which means that the utilization was greater than the optimal during 95% of the evaluation time.




(a) Rule reinstallation

(b) Average utilization

Figure 4.3: Results considering a tunned reference timeout.

| Statistic | Switch capacity | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| 95th % | 7 | 8 | 7 | 0 | 0 | 2 | 0 | 0 | 2 |
| Max | 17 | 23 | 17 | 3 | 2 | 7 | 2 | 2 | 6 |

Table 4.6: Tunned reference timeout - Dispersion data of the number of reinstallations.

| Statistic | Switch capacity | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| Std. dev. | 3.8 | 3.9 | 4.7 | 5.2 | 5.7 | 6.7 | 4.0 | 4.1 | 5.3 |
| 5th % | 95.6 | 92.3 | 92.8 | 90.4 | 89.6 | 77.8 | 67.1 | 67.8 | 58.3 |

Table 4.7: Tunned reference timeout - Dispersion data of the resource utilization.

#### *4.3.2.2 Estimated reference timeout*

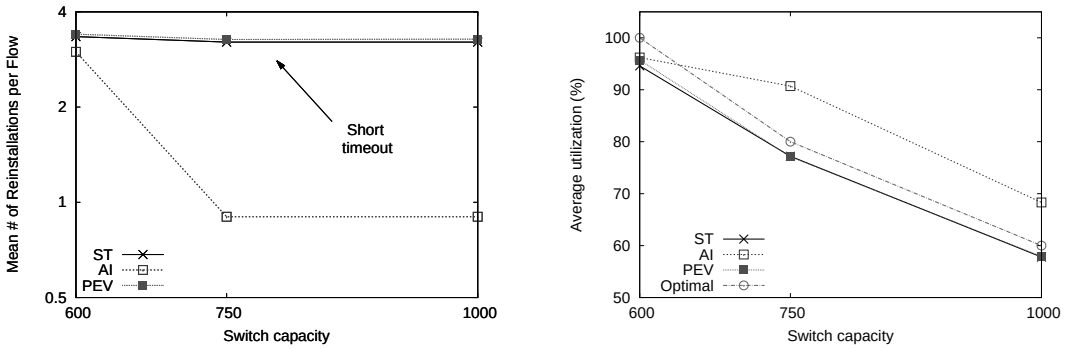
Determining the best reference timeout can be a challenging task due to two main reasons: i) it usually demands a deep examination of the network traffic for obtaining a reasonable timeout value and ii) the dynamic nature of the network traffic may require the reference timeout to be recalculated many times in a short period. Considering these scenarios, we also evaluate the impact of using estimated timeouts on the performance of the time-based strategies. Results are explained below.

*Underestimated reference timeout.* To evaluate the performance of the strategies in an underestimated scenario, we consider a reference timeout equals to 50% of the cumulative distribution function (CDF) of the workload packet inter-arrival time distribution (around 10 seconds for the *Workload 1*). Figure 4.4 shows that the number of reinstallations experiences a dramatic increasing for all strategies in this scenario compared to the tunned one (more than 300x for ST depending on the device capacity). Furthermore, as expected, AI strategy presents the best results for this case, since the timeout can grow until it is no longer adversely short. However, as a result, its average resource utilization is considerably greater than the optimal (Figure 4.4(b)), showing that it is important to establish a reasonable *max_timeout* ($t_{AI}^{max}$) value as well.

Completing this analysis, Table 4.8 shows dispersion data related to the number of rule reinstallations. As we can see, a tail behavior is present in most of the strategies. Table 4.9, on the other hand, presents dispersion data related to the device resource utilization, and shows that resources are adversely underutilized for both ST and PEV (5th percentile is lower than 80 and 60 for switch capacities of 750 and 1000 entries, respectively).

| Statistic | Switch capacity | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| 95th % | 11 | 9 | 11 | 11 | 2 | 10 | 11 | 2 | 11 |
| Max | 24 | 26 | 31 | 27 | 3 | 27 | 28 | 3 | 27 |

Table 4.8: Underestimated reference timeout - Dispersion data of the number of reinstallations.




(a) Number of reinstallations

(b) Average utilization

Figure 4.4: Results considering an underestimated reference timeout.

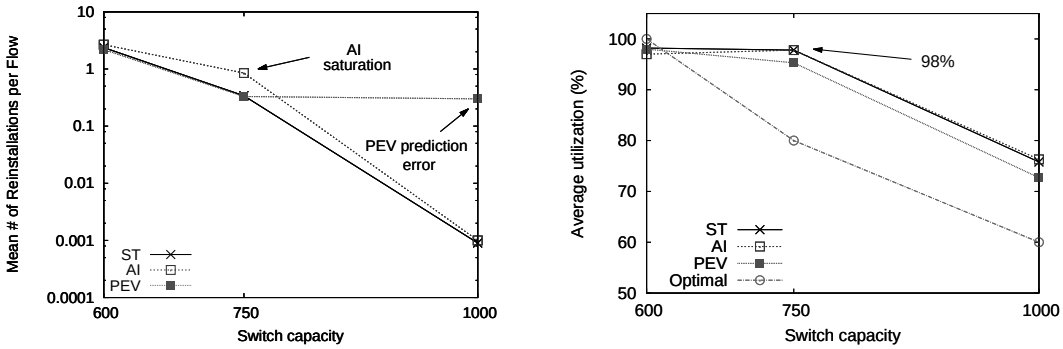
| **Statistic** | **Switch capacity** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| Std. dev. | 4.5 | 4.8 | 4.7 | 3.8 | 5.7 | 3.8 | 2.9 | 4.2 | 3.2 |
| 5th % | 90.0 | 90.5 | 90.1 | 73.3 | 84.8 | 72.0 | 55.0 | 63.4 | 53.8 |

Table 4.9: Underestimated reference timeout - Dispersion data of the resource utilization.

*Overestimated reference timeout.* We also study the performance of the strategies in an overestimated reference timeout scenario. For this scenario, we consider a timeout value greater than the tunned one and symmetric to the value of the underestimated case, which give us around 30 seconds for the *Workload 1*. Figure 4.5(a) shows two interesting effects observed. The first one is intuitive and relates to the presence of the PEV prediction error even for very long timeouts. This happen because the error is intrinsic to the probabilities set, not to the timeout value. Moreover, it acts as a bound to the performance of the strategy.

The second effect, on the other hand, is more subtle and relates to the timeout increasing of the AI strategy. We call it *AI saturation* because it happens always the timeouts become sufficiently long to saturate the resource utilization (as the case shown in Figure 4.5(b), where the utilization achieved 98% at the average for a capacity of 750 entries). In this case, the mechanism of rule caching will be triggered and as it can be unprecise when evicting rules (i.e. deleting rules from still alive flows), the number of reinstallations tends to grow. Note that this effect disappear when the resource capacity is sufficiently large to host all the rules. Moreover, it can also be contained with a more restrictive *max_timeout* value. We let the evaluation of this alternative for a future work.

As in the other scenarios, we also present the dispersion values for both metrics (Tables 4.10 and 4.11 for rule reinstallations and average utilization, respectively). Note the presence of the tail behavior in the AI strategy for a switch capacity of 750 entries, as well as the reduced values of the 5th percentile in the PEV strategy, the last not always converted in a more efficient utilization.




(a) Number of reinstallations

(b) Average utilization

Figure 4.5: Results considering an overestimated reference timeout.

| **Statistic** | **Switch capacity** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| 95th % | 8 | 9 | 7 | 2 | 4 | 2 | 0 | 0 | 2 |
| Max | 18 | 22 | 17 | 6 | 13 | 8 | 1 | 1 | 6 |

Table 4.10: Overestimated reference timeout - Dispersion data of the number of reinstallations.

| **Statistic** | **Switch capacity** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| Std. dev. | 4.5 | 4.6 | 4.7 | 5.6 | 5.6 | 8.2 | 4.9 | 5.1 | 7.3 |
| 5th % | 95.3 | 92.1 | 92.6 | 95.5 | 95.0 | 78.2 | 72.3 | 72.3 | 58.5 |

Table 4.11: Overestimated reference timeout - Dispersion data of the resource utilization.

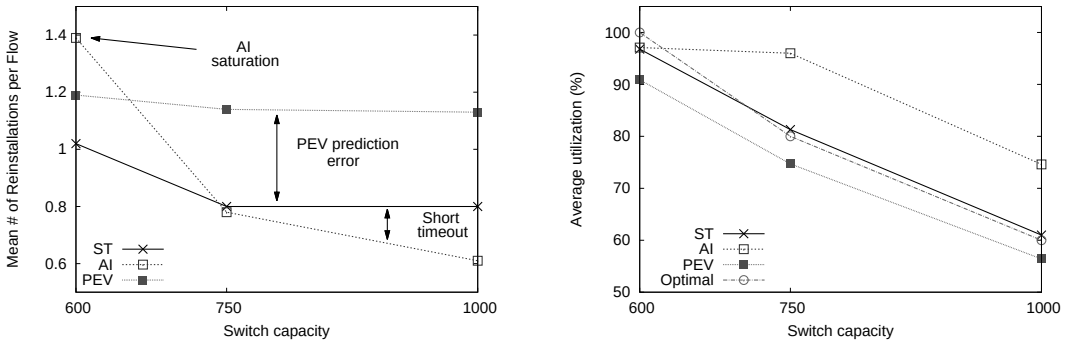
#### *4.3.2.3 Proof of concept*

To study the behavior of the strategies under variations in the nature of the network traffic and as a proof of concept, we also evaluate their performance considering a different workload. In this set of experiments, we used the tunned parameter configuration obtained for *Workload 1*, but applied *Workload 2* (described in Section 4.1) to the system.

As can be seen in Figure 4.6, there is a similarity with results obtained considering an underestimated reference timeout scenario for *Workload 1*. As in that case, all strategies present a significant number of rule reinstallations (typically more than one per flow) and AI performs better for larger switch capacities, the last certainly a consequence of the timeout increasing. Note that AI uses more resources as a counterpart for reducing rule reinstallations. Note also that, differently of the results from Section 4.3.2.2, we can perceive the presence of the AI saturation effect when we consider lower capacity devices. This is probably a signal that the timeouts are achieving greater values, which causes rules to remain installed for longer periods (sufficiently long to saturate the resource utilization and trigger rule caching mechanisms).

Tables 4.12 and 4.13 present the dispersion data for the number of reinstallations and the average utilization, respectively. Note the consistent resource overutilization of the AI strategy (higher values of the 5th percentile) and the presence of a tail behavior (not as long as in Section 4.3.2.2) when we consider the number of reinstallations.

As a general result of this proof of concept, we show that the behavior of the strategies will probably be similar to one of the cases described in the previous sections (tunned, underestimated or overestimated reference timeout), independent of the nature of the traffic. Moreover, we show that it is difficult to keep the reference timeout adjusted in a dynamic environment, where characteristics of the traffic vary, considering current time-based strategies. This happen mostly because they depend on an accurate parameterization process.




(a) Number of reinstallations

(b) Average utilization

Figure 4.6: Results considering variations in the nature of the traffic.

| **Statistic** | **Switch capacity** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| 95th % | 4 | 4 | 4 | 3 | 2 | 4 | 3 | 2 | 4 |
| Max | 12 | 14 | 14 | 8 | 9 | 10 | 8 | 4 | 11 |

Table 4.12: Proof of concept - Dispersion data of the number of reinstallations.

| **Statistic** | **Switch capacity** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 600 | | | 750 | | | 1000 | | |
| | ST | AI | PEV | ST | AI | PEV | ST | AI | PEV |
| Std. dev. | 3.6 | 4.7 | 15.2 | 4.3 | 6.5 | 13.5 | 3.4 | 6.0 | 10.0 |
| 5th % | 93.6 | 92.3 | 53.6 | 76.5 | 85.7 | 42.0 | 57.5 | 64.1 | 32.7 |

Table 4.13: Proof of concept - Dispersion data of the resource utilization.

# 5 CONCLUSION

Software Defined Networking has been proposed as a means to simplify network configuration and management. In this sense, one of its potential advantages is to allow the application of fine-grained networking policies (i.e. at the level of flows) by network administrators. However, these policies, that are stored as forwarding rules in entries of flow tables, aggravate resource scarcity and performance constraints at forwarding devices. In this work, we compare the performances of time-based strategies for optimizing flow tables in SDN, identifying characteristics and scenarios that directly affect them and quantifying how near they are from the optimal considering both the number of rule reinstallations and the resource utilization. Our key empirical findings are the following:

- All strategies performed far from optimal in most of the evaluated scenarios, considering both the number of rule reinstallations and the efficiency in the resource utilization. Moreover, their performance depends heavily on a precise parameterization, which is difficult to achieve and maintain in practice due to traffic variations.

- All strategies presented a tail behavior considering the number of rule reinstallations, which can adversely increase flow completion times.

- For resource-constrained switches, the performance of the time-based strategies is mainly determined by their rule caching algorithm. In this case, the nature of the rule timeout (i.e. static or dynamic) has little influence on the results. Furthermore, all algorithms we have evaluated performed at least 2x far from optimal.

- For large capacity switches, rule timeouts are determinant for performance. In a general sense, higher timeout values reduce the number of reinstallations. However, they also reduce the efficiency in the resource utilization, as the number of unnecessary rules intalled at the forwarding device in a given moment tends to grow. Too long timeouts are also undesired, since they typically require the use of rule chaching mechanisms, which can be inefficient.

- Early evictions tend to increase the efficiency in the resource utilization, but an intrinsic error due to their probabilistic nature also increases the number of rule reinstallations. This error acts as a bound to the performance of the strategy, resulting in no performance improvement even when both the duration of the rule and the capacity of the device grow.

- Adaptatively increasing the duration of the rules from short values tends to perform better (in terms of both the number of reinstallations and the resource utilization) than trying to decrease them from longer ones or keeping them fixed. However, it

is also necessary to establish precise growing paces and upper limits to the process in order to prevent effects as the saturation of the resource utilization.

As future work, we aim to investigate mechanisms for improving the performance of the time-based strategies, mainly regarding their tail behavior related to the number of rule reinstallations. Additionally, we intend to analyze their effective benefits over packet processing latency and energy consumption in real network topologies (i.e. not just a single forwarding device). Lastly, we also plan to study potential advantages and challenges of combining space and time-based strategies for optimizing flow tables in SDN.

# REFERENCES

AFEK, Y.; BREMLER-BARR, A.; SCHIFF, L. Cross-Entrance Consistent Range Classifier with OpenFlow. In: PRESENTED AS PART OF THE OPEN NETWORKING SUMMIT 2014 (ONS 2014), Santa Clara, CA. **Proceedings...** USENIX, 2014.

AGRAWAL, B.; SHERWOOD, T. Modeling TCAM power for next generation network devices. In: ISPASS. **Proceedings...** [S.l.: s.n.], 2006. p.120–129.

BANERJEE, S.; KANNAN, K. Tag-In-Tag: efficient flow table management in sdn switches. , [S.l.], 2014.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Network Traffic Characteristics of Data Centers in the Wild. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT, 10., New York, NY, USA. **Proceedings** ACM, 2010. p.267–280. (IMC '10).

BOSSHART, P.; DALY, D.; IZZARD, M.; MCKEOWN, N.; REXFORD, J.; TALAYCO, D.; VAHDAT, A.; VARGHESE, G.; WALKER, D. Programming Protocol-Independent Packet Processors. **CoRR**, [S.l.], v.abs/1312.1719, 2013.

COHEN, R.; LEWIN-EYTAN, L.; NAOR, J.; RAZ, D. On the effect of forwarding table size on SDN network utilization. In: INFOCOM, 2014 PROCEEDINGS IEEE. **Proceedings...** [S.l.: s.n.], 2014. p.1734–1742.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN. **Queue**, New York, NY, USA, v.11, n.12, p.20:20–20:40, Dec. 2013.

FERGUSON, A. D.; GUHA, A.; LIANG, C.; FONSECA, R.; KRISHNAMURTHI, S. Hierarchical Policies for Software Defined Networks. In: FIRST WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS, New York, NY, USA. **Proceedings** ACM, 2012. p.37–42. (HotSDN '12).

FOSTER, N.; GUHA, A.; REITBLATT, M.; STORY, A.; FREEDMAN, M.; KATTA, N.; MONSANTO, C.; REICH, J.; REXFORD, J.; SCHLESINGER, C.; WALKER, D.; HARRISON, R. Languages for software-defined networks. **Communications Magazine, IEEE**, [S.l.], v.51, n.2, p.128–134, February 2013.

FOSTER, N.; HARRISON, R.; FREEDMAN, M. J.; MONSANTO, C.; REXFORD, J.; STORY, A.; WALKER, D. Frenetic: a network programming language. In: ACM SIGPLAN INTERNATIONAL CONFERENCE ON FUNCTIONAL PROGRAMMING, 16., New York, NY, USA. **Proceedings** ACM, 2011. p.279–291. (ICFP '11).

GIROIRE, F.; MOULIERAC, J.; PHAN, T. K. **Optimizing Rule Placement in Software-Defined Networks for Energy-aware Routing**. [S.l.]: INRIA, 2014. Rapport de recherche, Available at: http://hal.inria.fr/hal-00996780. (RR-8537).

KANG, N.; LIU, Z.; REXFORD, J.; WALKER, D. Optimizing the "One Big Switch" Abstraction in Software-defined Networks. In: NINTH ACM CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES, New York, NY, USA. **Proceedings** ACM, 2013. p.13–24. (CoNEXT '13).

KANG, N.; REICH, J.; REXFORD, J.; WALKER, D. Policy Transformation in Software Defined Networks. In: ACM SIGCOMM 2012 CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATION, New York, NY, USA. **Proceedings** ACM, 2012. p.309–310. (SIGCOMM '12).

KANIZO, Y.; HAY, D.; KESLASSY, I. Palette: distributing tables in software-defined networks. In: INFOCOM, 2013 PROCEEDINGS IEEE. **Proceedings...** [S.l.: s.n.], 2013. p.545–549.

KANNAN, K.; BANERJEE, S. FlowMaster: early eviction of dead flow on sdn switches. In: CHATTERJEE, M.; CAO, J.-n.; KOTHAPALLI, K.; RAJSBAUM, S. (Ed.). **Distributed Computing and Networking**. [S.l.]: Springer Berlin Heidelberg, 2014. p.484–498. (Lecture Notes in Computer Science, v.8314).

KARAGIANNIS, T.; MOLLE, M.; FALOUTSOS, M.; BROIDO, A. A nonstationary Poisson view of Internet traffic. In: INFOCOM 2004. TWENTY-THIRD ANNUALJOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. **Proceedings...** [S.l.: s.n.], 2004. v.3, p.1558–1569 vol.3.

KATTA, N.; ALIPOURFARD, O.; REXFORD, J.; WALKER, D. Infinite CacheFlow in Software-defined Networks. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, New York, NY, USA. **Proceedings** ACM, 2014. p.175–180. (HotSDN '14).

KATTA, N. P.; REXFORD, J.; WALKER, D. **Logic Programming for Software-Defined Networks**.

KESSELMAN, A.; KOGAN, K.; NEMZER, S.; SEGAL, M. Space and Speed Tradeoffs in TCAM Hierarchical Packet Classification. **J. Comput. Syst. Sci.**, Orlando, FL, USA, v.79, n.1, p.111–121, Feb. 2013.

KOGAN, K.; NIKOLENKO, S.; ROTTENSTREICH, O.; CULHANE, W.; EUGSTER, P. SAX-PAC (Scalable And eXpressive PAcket Classification). In: ACM CONFERENCE ON SIGCOMM, 2014., New York, NY, USA. **Proceedings** ACM, 2014. p.15–26. (SIGCOMM '14).

KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-Defined Networking: A comprehensive survey. **CoRR**, [S.l.], v.abs/1406.0440, 2014.

KUZNIAR, M.; PERESINI, P.; KOSTIC, D. **What you need to know about SDN control and data planes**. [S.l.: s.n.], 2014.

LAZARIS, A.; TAHARA, D.; HUANG, X.; LI, L. E.; VOELLMY, A.; YANG, Y. R.; YU, M. Tango: simplifying sdn control with automatic switch property inference, abstraction, and optimization. , [S.l.], 2014.

LEE, B.-S.; KANAGAVELU, R.; AUNG, K. An efficient flow cache algorithm with improved fairness in Software-Defined Data Center Networks. In: CLOUD NETWORKING (CLOUDNET), 2013 IEEE 2ND INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2013. p.18–24.

MA, Y.; BANERJEE, S. A Smart Pre-classifier to Reduce Power Consumption of TCAMs for Multi-dimensional Packet Classification. In: ACM SIGCOMM 2012 CONFERENCE ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATION, New York, NY, USA. **Proceedings** ACM, 2012. p.335–346. (SIGCOMM '12).

MANN, V.; VISHNOI, A.; BIDKAR, S. Living on the edge: monitoring network flows at the edge in cloud data centers. In: COMMUNICATION SYSTEMS AND NETWORKS (COMSNETS), 2013 FIFTH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2013. p.1–9.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v.38, n.2, p.69–74, Mar. 2008.

MOSHREF, M.; YU, M.; SHARMA, A.; GOVINDAN, R. Scalable Rule Management for Data Centers. In: PRESENTED AS PART OF THE 10TH USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI 13), Lombard, IL. **Proceedings...** USENIX, 2013. p.157–170.

NARLIKAR, G. J.; BASU, A.; ZANE, F. CoolCAMs: power-efficient tcams for forwarding engines. In: INFOCOM. **Proceedings...** [S.l.: s.n.], 2003.

NGUYEN, X.-N.; SAUCEZ, D.; BARAKAT, C.; TURLETTI, T. Optimizing Rules Placement in OpenFlow Networks: trading routing for better efficiency. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, New York, NY, USA. **Proceedings** ACM, 2014. p.127–132. (HotSDN '14).

NUNES, B.; MENDONCA, M.; NGUYEN, X.-N.; OBRACZKA, K.; TURLETTI, T. A Survey of Software-Defined Networking: past, present, and future of programmable networks. **Communications Surveys Tutorials, IEEE**, [S.l.], v.16, n.3, p.1617–1634, Third 2014.

OPENFLOW specification. <https://www.opennetworking.org>. Access in October, 2014.

QI, Y.; XU, L.; YANG, B.; XUE, Y.; LI, J. Packet Classification Algorithms: from theory to practice. In: INFOCOM 2009, IEEE. **Proceedings...** [S.l.: s.n.], 2009. p.648–656.

ROTTENSTREICH, O.; RADAN, M.; CASSUTO, Y.; KESLASSY, I.; ARAD, C.; MIZRAHI, T.; REVAH, Y.; HASSIDIM, A. Compressing Forwarding Tables for Datacenter Scalability. **Selected Areas in Communications, IEEE Journal on**, [S.l.], v.32, n.1, p.138–151, January 2014.

SCHLESINGER, C.; GREENBERG, M.; WALKER, D. Concurrent NetCore: from policies to pipelines. In: ACM SIGPLAN INTERNATIONAL CONFERENCE ON FUNCTIONAL PROGRAMMING, 19., New York, NY, USA. **Proceedings** ACM, 2014. p.11–24. (ICFP '14).

SEKAR, V.; REITER, M. K.; WILLINGER, W.; ZHANG, H.; KOMPELLA, R. R.; ANDERSEN, D. G. cSamp: a system for network-wide flow monitoring. In: USENIX NSDI, 5., San Francisco, CA. **Proceedings...** [S.l.: s.n.], 2008.

SONG, H. Protocol-oblivious Forwarding: unleash the power of sdn through a future-proof forwarding plane. In: SECOND ACM SIGCOMM WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, New York, NY, USA. **Proceedings** ACM, 2013. p.127–132. (HotSDN '13).

VISHNOI, A.; PODDAR, R.; MANN, V.; BHATTACHARYA, S. Effective Switch Memory Management in OpenFlow Networks. In: ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED EVENT-BASED SYSTEMS, 8., New York, NY, USA. **Proceedings** ACM, 2014. p.177–188. (DEBS '14).

VOELLMY, A.; WANG, J.; YANG, Y. R.; FORD, B.; HUDAK, P. Maple: simplifying sdn programming using algorithmic policies. In: ACM SIGCOMM 2013 CONFERENCE ON SIGCOMM, New York, NY, USA. **Proceedings** ACM, 2013. p.87–98. (SIGCOMM '13).

YAN, B.; XU, Y.; XING, H.; XI, K.; CHAO, H. J. CAB: a reactive wildcard rule caching system for software-defined networks. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, New York, NY, USA. **Proceedings** ACM, 2014. p.163–168. (HotSDN '14).

YU, M.; REXFORD, J.; FREEDMAN, M. J.; WANG, J. Scalable Flow-based Networking with DIFANE. In: ACM SIGCOMM 2010 CONFERENCE, New York, NY, USA. **Proceedings** ACM, 2010. p.351–362. (SIGCOMM '10).

ZAREK, A. **OpenFlow Timeouts Demystified**. University of Toronto, Toronto, Ontario, Canada.

# APPENDIX A: GRADUATION WORK I

# Utilização Eficiente de Tabelas de Fluxos em Redes Definidas por Software

**Miguel C. Neves**[1], **Marinho P. Barcellos**[1]

[1]Instituto de Informática - Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 - Porto Alegre, RS - Brasil

{mcneves,marinho}@inf.ufrgs.br

***Abstract.*** *Software Defined Networking (SDN) is considered one of the main results of recent research efforts to simplify the configuration and management of computer networks. In this paradigm, a logically centralized controller dynamically programs the network installing forwarding rules in Flow Tables. These tables allow administrators to specify network policies with fine granularity, but they have an extremely limited amount of resources. This paper presents a study about SDN and the main techniques used to achieve an efficient utilization of Flow Tables, analysing them under performance and implementation aspects. This study represents the theoretical basis for the Graduation Work 2.*

***Resumo.*** *As Redes Definidas por Software (SDNs) representam o resultado de esforços de pesquisa recentes para simplificar a configuração e a gerência de redes. Nesse paradigma, um controlador logicamente centralizado programa dinamicamente a rede através da implantação de regras de encaminhamento em Tabelas de Fluxos. Tais tabelas permitem a especificação de políticas de rede com alta granularidade, mas possuem uma quantidade limitada de recursos. Este artigo apresenta um estudo sobre SDN e as principais técnicas utilizadas para permitir o uso eficiente das Tabelas de Fluxos, analisando-as sob aspectos de desempenho e implementação. O estudo apresentado servirá de embasamento teórico para a realização do Trabalho de Graduação 2.*

## 1. Introdução

As Redes Definidas por Software constituem um paradigma de redes de computadores desenvolvido com o objetivo de facilitar a inovação e siplificar as atividades de controle numa rede. Nesse paradigma, a separação entre o plano de dados da rede (presente em cada dispositivo de encaminhamento) e a sua lógica de controle (concentrada num controlador de rede) habilita avanços como o rápido desenvolvimento de novos protocolos e a gerência flexível de novas políticas. O controlador de rede, nesse caso, atua como uma entidade logicamente centralizada que possui visão global da rede e gerencia uma coleção distribuída e programável de *switches* e roteadores [Astuto et al. 2013].

A interação entre o controlador de rede e os dispositivos de encaminhamento nessa arquitetura utiliza dois componentes principais: i) uma interface de comunicação padrão, tipicamente aberta (*p.ex.* OpenFlow [McKeown et al. 2008]) e ii) uma estrutura para o armazenamento em cada dispositivo de parte ou todo o estado de controle da rede. Essa estrutura, denominada Tabela de Fluxos, é utilizada pelo plano de dados para encaminhar pacotes e permite a especificação de políticas de rede com alta granularidade. Contudo,

costuma possuir uma quantidade bastante limitada de recursos [Liu et al. 2010, Ma and Banerjee 2012].

Normalmente as tabelas de fluxos são implementadas por meio de memórias TCAM (*Ternary Content Addressable Memory*). Esse tipo de memória possibilita a verificação simultânea dos pacotes de um fluxo frente a todas as regras armazenadas na tabela, o que por sua vez permite o encaminhamento desses pacotes com um alto desempenho. As memórias TCAM, no entanto, tem como grande desvantagem o alto consumo energético. Uma TCAM típica com 18 Mbit de armazenamento pode consumir até 15 Watts de potência (cerca de 30% do consumo de um *switch* de médio-grande porte atual) quando todas as suas entradas são verificadas [Ma and Banerjee 2012, Congdon et al. 2013]. Em última instância, isso restringe fortemente o número de entradas disponíveis nas tabelas de fluxos.

Uma saída encontrada para esse problema foi o desenvolvimento de soluções híbridas baseadas em *hardware* e *software* (*p.ex.* abordagens de árvores de decisão sobre memórias RAM [Wang et al. 2013]). Porém, apesar de suportar tabelas maiores, essas abordagens são dezenas de vezes mais lentas e insuficientes para atender as demandas de redes de grande porte. Diante desse contexto, o estudo e desenvolvimento de técnicas e mecanismos que buscam a utilização eficiente das tabelas de fluxos em SDNs torna-se extremamente relevante.

Nesse sentido, o objetivo deste trabalho é investigar procedimentos que permitam o uso eficiente das tabelas de fluxos em SDNs, priorizando o desempenho e o consumo energético da rede como um todo. Neste primeiro artigo, é feito um levantamento das principais abordagens propostas visando o entendimento do problema e a identificação dos desafios em aberto. No Trabalho de Graduação 2, serão projetadas, implementadas e avaliadas técnicas que levem ao aumento da eficiência no uso das tabelas de fluxos, buscando solucionar um ou mais dos desafios identificados.

O restante deste artigo está organizado como segue. Na Seção 2, nós propomos uma taxonomia para guiar o estudo do estado da arte e facilitar a sua compreensão. Na sequência, nós examinamos os principais trabalhos da área em maiores detalhes na Seção 3, de acordo com a taxonomia proposta. A Seção 4 delineia os principais desafios e oportunidades com relação ao uso eficiente de tabelas de fluxos. Já na Seção 5, a proposta do Trabalho de Graduação 2 é apresentada, seguida da metodologia adotada e do cronograma de atividades a serem realizadas. Por fim, a Seção 6 conclui o documento.

## 2. Taxonomia

Há uma grande quantidade de trabalhos recentes voltados ao desenvolvimento de técnicas visando a utilização eficiente de tabelas de fluxos em SDNs. Por esse motivo, nós propomos uma taxonomia para categorizar os principais trabalhos existentes e facilitar o entendimento dos esforços na área. Essa taxonomia baseia-se em quatro níveis principais de atuação dentro da arquitetura SDN: transformação de políticas, posicionamento, duração e representação de regras de encaminhamento na rede. A Figura 1 mostra a organização da taxonomia proposta dentro da arquitetura SDN.

A transformação de políticas de rede em regras de encaminhamento envolve os níveis de abstração mais altos, ou próximos das aplicações de controle, dentro da arquitetura SDN. Apesar de possibilitarem a programação da rede, as abstrações existentes ainda

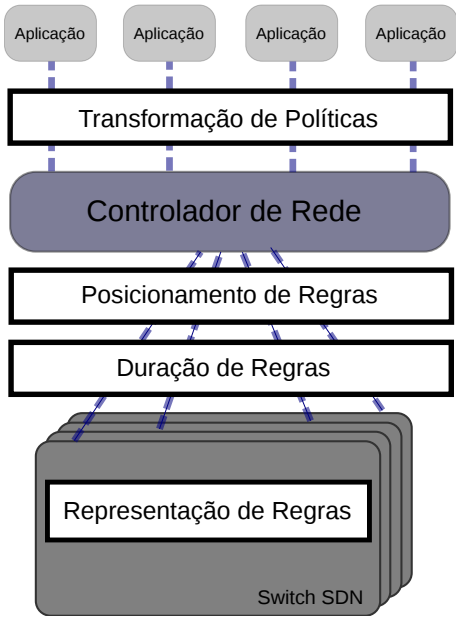



**Figura 1. Organização da taxonomia proposta dentro da arquitetura SDN.**

não permitem que isso seja feito de forma robusta e eficiente. Nesse contexto, os controladores de rede atuais se limitam a oferecer APIs de baixo nível que buscam refletir os recursos de *hardware* dos *switches*. O resultado é um número elevado de regras de encaminhamento mesmo para representar políticas simples de rede [Foster et al. 2013]. Para atingir todo o potencial das SDNs, é preciso identificar as abstrações de alto nível certas sendo este o principal objetivo dos trabalhos deste grupo.

Um segundo grupo de trabalhos emprega técnicas de posicionamento de regras na infraestrutura para aumentar a eficiência no uso das tabelas de fluxos. A ideia básica é intuitiva: dada uma topologia de rede, um conjunto de regras e a capacidade (número máximo de regras suportadas) de cada dispositivo de encaminhamento, o controlador deve ser capaz de decidir automaticamente quais regras ficarão armazenadas em quais dispositivos. As técnicas de posicionamento de regras são importantes pois liberam as aplicações de controle da necessidade de tratar questões envolvendo restrições de *hardware* dos dispositivos de rede, bem como interações que possam ocorrer entre diferentes regras. Todavia, inúmeros desafios ainda precisam ser solucionados para que essas técnicas sejam efetivamente utilizadas (*p.ex.* questões de escalabilidade dos algoritmos de posicionamento sobre o número de regras e de dispositivos da rede) [Kang et al. 2013].

A terceira categoria de trabalhos explora o tempo de vida das regras de encaminhamento como forma de otimizar a ocupação das tabelas de fluxos. Se por um lado há poucos benefícios em manter entradas nas tabelas ocupadas com regras para as quais não há fluxos ativos, por outro a remoção prematura de regras pode aumentar o tráfego de controle na rede e a latência dos fluxos [Zarek 2012]. Claramente há um *trade-off* envolvendo essas questões, cujas escolhas de projeto podem afetar largamente o desempenho da rede. Motivados por essas ideias, pesquisadores propuseram algumas abordagens para a otimização temporal do uso das tabelas de fluxos, as quais serão vistas em maiores detalhes na Seção 3.

Por fim, a quarta categoria de trabalhos proposta busca responder uma questão simples: dado um conjunto de regras de encaminhamento a ser instalado numa tabela de fluxos, como é possível minimizar o número de entradas (ou a quantidade de recursos necessários) para representar essas regras? Reduzir o número de entradas utilizadas numa tabela de fluxos possibilita, entre outros benefícios, a redução do consumo de potência e/ou da latência de processamento dos dispositivos de rede. Por outro lado, as propostas desse grupo muitas vezes são dependentes das tecnologias utilizadas na implementação das tabelas [Liu et al. 2010].

É importante notar que essas quatro categorias de trabalhos não são mutuamente exclusivas. De fato, alguns trabalhos integram mais de uma categoria. Por questões de espaço, nos restringimos a uma visão simplificada dos principais trabalhos encontrados. No entanto, é possível ter uma ideia clara dos principais desafios de pesquisa que a indústria e a comunidade acadêmica enfrentam atualmente. A Tabela 1 mostra uma comparação entre os trabalhos.

| **Categorias** | | | **Propostas** |
|---|---|---|---|
| Transformação de Políticas | | | [Ferguson et al. 2012] |
| | | | [Monsanto et al. 2012] |
| | | | [Foster et al. 2013] |
| | | | [Kim et al. 2013] |
| | | | [Monsanto et al. 2013] |
| | | | [Voellmy et al. 2013] |
| Posicionamento de Regras | | Conservação de políticas de roteamento | [Kang et al. 2013] |
| | | | [Kanizo et al. 2013] |
| | | Alteração de políticas de roteamento | [Yu et al. 2010] |
| | | | [Moshref et al. 2013] |
| Duração de Regras | | Detecção do estado dos fluxos | [Zarek 2012] |
| | | | [Kannan and Banerjee 2014] |
| | | Uso de mecanismos de *cache* | [Yu et al. 2010] |
| | | | [Katta et al. 2013] |
| Representação de Regras | TCAM | Minimização de classificadores | [Liu et al. 2008] |
| | | | [Liu et al. 2010] |
| | | | [Meiners et al. 2012] |
| | | Codificação de dados | [Bremler-Barr and Hendler 2007] |
| | | | [Meiners et al. 2009] |
| | | | [Rottenstreich et al. 2013] |
| | | Modificação de *hardware* | [Bosshart et al. 2013] |
| | | Outros | [Meiners et al. 2011] |
| | | | [Ma and Banerjee 2012] |
| | | | [Kesselman et al. 2013] |
| | RAM | Árvores de decisão | [Qi et al. 2009] |
| | | | [Vamanan et al. 2010] |
| | | | [Wang et al. 2013] |
| | | Modificação de *hardware* | [De Carli et al. 2009] |

**Tabela 1. Visão geral das propostas para utilização eficiente de Tabelas de Fluxos em SDNs.**

## 3. Estado da Arte

Na Seção anterior, nós apresentamos a taxonomia proposta neste trabalho e que será utilizada para guiar o estudo do estado da arte ao longo deste artigo. Esse estudo será apresentado nesta Seção, e está organizado da seguinte forma: a Seção 3.1 discute os principais trabalhos voltados à transformação de políticas em SDNs. Na sequência, a Seção

3.2 mostra as soluções propostas para o posicionamento de regras de encaminhamento na rede, enquanto a Seção 3.3 apresenta as abordagens para a determinação do tempo de permanência dessas regras nos dispositivos da infraestrutura. Por fim, a Seção 3.4 resume as principais estratégias para a representação das regras de encaminhamento dentro desses dispositivos.

### 3.1. Transformação de Políticas

De um ponto de vista prático, os trabalhos voltados à transformação de políticas buscam oferecer ao programadores de aplicações SDN (*i.e.* administradores e operadores de redes) abstrações que permitam aos mesmos focar nos aspectos lógicos por trás da programação da infraestrutura, o que facilita o desenvolvimento e a implantação de funções de gerência e controle (*p.ex* monitoramento, engenharia de tráfego, controle de acesso, etc.) mais robustas. Não há uma distinção explícita entre os trabalhos dessa categoria que nos permita dividí-los em subgrupos. Entretanto, é possível identificar traços característicos em cada uma das propostas.

**Frenetic.** [Foster et al. 2013] propõem um conjunto de abstrações (*framework*) para a programação de SDNs, permitindo aos programadores escreverem códigos mais simples e robustos. Essas abstrações, utilizadas através de linguagens de programação específicas, estão relacionadas às principais tarefas de gerência de redes: monitoramento de tráfego, especificação e atualização consistente de políticas de encaminhamento. Um compilador é responsável por implementar essas abstrações, transformando efetivamente políticas em regras de encaminhamento e assegurando que os programas sejam executados de forma eficiente.

**NetCore.** [Monsanto et al. 2012] propõem uma linguagem declarativa para a especificação de políticas de encaminhamento em SDNs. Tal linguagem extendeu as linguagens originais do *framework* Frenetic, permitindo a programação de funcionalidades arbitrárias da rede. Essa linguagem é baseada num conjunto de primitivas para a classificação de pacotes e possui uma semântica formal cuja corretude foi provada pelos autores.

**HFT.** [Ferguson et al. 2012] apresentam um *framework* para a implementação de políticas hierárquicas em SDNs. Essas políticas são importantes na resolução de conflitos em controladores de rede com suporte a decisões distribuídas (*p.ex.* PANE [Ferguson et al. 2013]). O *framework* proposto atua em dois estágios. Primeiro, é feita a compilação das políticas hierárquicas de rede para um conjunto de *Network Flow Tables*, abstração proposta pelos autores para a especificação de tabelas de fluxos em alto nível, através de ações lógicas. Em seguida, as *Network Flow Tables* são utilizadas para configurar os dispositivos de rede, por meio da tradução de ações de alto nível (já sem a presença de conflitos) em comandos de configuração específicos de cada dispositivo. A proposta também permite a utilização de diferentes algoritmos para o posicionamento de regras na rede.

**Maple.** [Voellmy et al. 2013] propõe um sistema que simplifica a programação de redes SDN através do uso de políticas *algorítmicas* (ao invés de declarativas). Nesse tipo de política, o programador também utiliza um modelo de programação para definir as políticas de controle da rede (e não apenas implementá-las), através da especificação de um conjunto de funções que são executadas sobre os pacotes da mesma. O sistema utiliza processos de otimização baseados em árvore para reduzir o número de regras de encaminhamento geradas.

**Outros.** [Monsanto et al. 2013] propõe um conjunto de abstrações que permitem o desenvolvimento de aplicações através da composição de múltiplos módulos independentes. Nesse sentido, [Kim et al. 2013] extende a ideia de desenvolvimento em blocos de forma a considerar a adaptação (através da ativação e desativação de módulos) às condições de operação da rede. Isso permite que operadores implementem políticas especificando como o comportamento da rede deve mudar em resposta a eventos pré-determinados.

### 3.2. Posicionamento de Regras

De acordo com [Kang et al. 2013], os trabalhos voltados ao posicionamento de regras na rede podem ser distribuídos em duas categorias, diferenciadas a partir do tratamento dado às políticas de roteamento dos fluxos da rede. De um lado, estão os trabalhos que consideram tais políticas uma "caixa preta", gerenciada exclusivamente pelas aplicações de controle com base em objetivos de alto nível (*p.ex.* implementação de mecanismos de engenharia de tráfego, *firewall*, etc.). Os trabalhos de [Kang et al. 2013] e [Kanizo et al. 2013] são exemplos dessa categoria. De outro lado, estão os trabalhos que aplicam desvios de rota aos fluxos da rede através da implantação de regras específicas, a fim exclusivamente de otimizar o posicionamento das demais regras de encaminhamento segundo algum critério. [Yu et al. 2010] e [Moshref et al. 2013] estão entre as principais propostas contidas nesse grupo.

***One Big Switch.*** [Kang et al. 2013] propõem um algoritmo baseado em programação linear para o posicionamento de regras na rede. Esse algoritmo é dividido em três etapas: decomposição da rede em caminhos, divisão da capacidade dos dispositivos de encaminhamento entre os caminhos que o atravessam e alocação das regras de encaminhamento que implementam uma dada política ao longo de um caminho. O objetivo do algoritmo é minimizar o número de regras necessárias para realizar o conjunto de políticas da rede. Ao mesmo tempo, a capacidade de cada dispositivo deve ser respeitada. Ao contrário dos demais trabalhos, o algoritmo proposto também considera cenários onde a quantidade de regras suportadas por cada dispositivo de encaminhamento não é uniforme, apesar dessa propriedade não ser avaliada nos experimentos.

**Palette.** [Kanizo et al. 2013] propõem um *framework* para a decomposição de grandes tabelas de fluxos em subtabelas, distribuídas entre os elementos da rede de forma a minimizar o tamanho da tabela resultante em cada dispositivo. Em outras palavras, há um balanceamento entre o tamanho das tabelas de fluxos na rede, o que permite às mesmas serem implementadas com maior eficiência. A aplicação de controle, nesse caso, é responsável por indicar quais regras de uma mesma tabela podem ser distribuídas. Já ao *framework*, cabe também assegurar que os pacotes de um fluxo sejam roteados e processados por todas as respectivas partes da tabela original. Os autores utilizaram otimização para modelar o problema, que foi dividido em duas partes ambas NP-difíceis. Algoritmos gulosos foram propostos em busca de soluções viáveis e a sua proximidade dos cenários ótimos demonstrada através de experimentos.

**DIFANE.** [Yu et al. 2010] propõem uma arquitetura hierárquica para o posicionamento de regras na rede. Nessa arquitetura, o controlador particiona e distribui um conjunto de regras entre um grupo dedicado de dispositivos (denominado *authority switches*), que por sua vez distribui tais regras para os demais dispositivos da rede sob demanda. Os *authority switches*, nesse caso, atuam de forma a reduzir tanto o número de pacotes en-

viados ao controlador quanto de regras armazenadas nos demais dispositivos da rede. No entanto, para garantir que os pacotes de um fluxo sejam processados por todas as regras relacionadas ao mesmo, em alguns casos é necessário redirecionar tais pacotes em direção a um determinado *authority switch*, aumentando o caminho percorrido pelo mesmo.

**vCRIB.** [Moshref et al. 2013] propõem um mecanismo, também baseado em partições, para o gerenciamento de regras de encaminhamento em Redes de Datacenters. Os autores consideram o posicionamento de regras tanto em hipervisores quanto em *switches* para atingir um equilíbrio entre o desempenho e a utilização dos recursos da rede. Ao contrário da abordagem de [Yu et al. 2010], onde o número de partições de um conjunto de regras está diretamente relacionado ao número de *authority switches*, vCRIB divide as regras de encaminhamento em um número arbitrário de partições e posiciona uma ou mais partições em cada dispositivo da rede (ou hipervisor). Isso permite considerar objetivos independentes para a partição e o posicionamento das regras.

### 3.3. Duração de Regras

Há dois grupos de trabalhos dentro dessa categoria. O primeiro diz respeito às propostas que buscam determinar o estado de um fluxo na rede para inferir sobre a necessidade de manter uma determinada regra de encaminhamento nas tabelas de fluxos ou não. Nesse caso, se um fluxo é considerado finalizado, suas respectivas regras são automaticamente deletadas das tabelas nas quais estão presentes pelo controlador de rede. Note que essa é uma abordagem *reativa*, uma vez que o controlador toma decisões com base em atividades de monitoramento. Os principais trabalhos desse grupo são os de [Zarek 2012] e [Kannan and Banerjee 2014].

***Flags* TCP.** [Zarek 2012] explora o efeito de variações no mecanismo de *timeout* de regras disponíveis no protocolo OpenFlow sobre a ocupação das tabelas de fluxo e o tráfego de controle na rede. O autor mostra que a ocupação das tabelas cresce de forma aproximadamente linear com o aumento do tempo de *timeout* para o conjunto de *traces* analisados. Com base nos resultados obtidos, é proposto um mecanismo de gerenciamento de regras híbrido, que combina *timeouts* por inatividade de fluxos com ações disparadas pelo controlador de rede para a deleção de regras de encaminhamento. Nesse caso, o controlador infere o estado dos fluxos a partir de campos de cabeçalho dos pacotes (p.ex. flags SYN, FIN e ACK do protocolo TCP) e deleta as regras antes que o seu tempo de *timeout* expire. Resultados mostram que essa abordagem pode reduzir em até 42% a ocupação das tabelas. No entanto, a mesma tem uma série de limitações como a dependência de características específicas de cada protocolo.

O segundo grupo de trabalhos, por sua vez, trata de soluções que utilizam mecanismos de *cache* de regras para tentar manter somente as regras de encaminhamento mais utilizadas instaladas nos dispositivos de rede, fazendo com que regras menos requisitadas sejam mantidas apenas pelo controlador. Um dos grandes desafios dessas abordagens é a necessidade de tratar relações de depedência entre regras, uma vez que implicam em importantes restrições com relação ao seu gerenciamento (*p.ex.* o uso de uma determinada regra pode implicar que um bloco de regras seja efetivamente instalado, o que exige maior número de entradas das tabelas de fluxos). As principais características dessas abordagens podem ser encontradas nos trabalhos de [Katta et al. 2013] e [Yu et al. 2010].

### 3.4. Representação de Regras

Diversas soluções tem sido propostas para aumentar a eficiência no uso dos recursos das tabelas de fluxos através de otimizações em mecanismos de representação de regras. Boa parte dessas soluções é genérica e também pode ser aplicada no contexto de redes convencionais, como é o caso dos trabalhos de [Liu et al. 2010] e [Kesselman et al. 2013]. Nesse caso, da mesma forma que para as redes convencionais, as soluções baseadas na representação de regras em redes SDN também podem ser divididas em dois grandes grupos: *i)* soluções voltadas a memórias TCAM e *ii)* soluções voltadas a memórias RAM [Ma and Banerjee 2012].

De acordo com [Liu et al. 2010], as soluções baseadas em memórias TCAM podem ser divididas em três subgrupos: soluções de minimização de classificadores, codificação de dados e modificação de hardware. A ideia básica das soluções baseadas na minimização de classificadores é converter um dado classificador de pacotes (*i.e.* uma sequência de regras de encaminhamento) em outro semanticamente equivalente, mas que requer menos entradas para implantar uma tabela de fluxos. Essa técnica tem a vantagem de não exigir qualquer modificação no *hardware* dos dispositivos de rede ou preprocessamento dos pacotes de um fluxo. O trabalho de [Liu et al. 2010], além das propostas de [Liu et al. 2008] e [Meiners et al. 2012] fazem parte deste subgrupo.

As soluções baseadas em codificação de dados (*range encoding*), por sua vez, buscam resolver o problema da expansão no número de entradas TCAM (*range expansion*) necessárias para representar intervalos de valores em determinados campos de uma regra de encaminhamento. Para saber mais sobre o problema, é possível consultar os trabalhos de [Liu et al. 2010] e [Rottenstreich et al. 2013]. Entre as soluções deste subgrupo, destacam-se os trabalhos de [Bremler-Barr and Hendler 2007], [Meiners et al. 2009] e [Rottenstreich et al. 2013]. A principal desvantagem dessas abordagens é que tipicamente as mesmas exigem um pré-processamento dos pacotes antes do seu encaminhamento, o que pode inserir atrasos consideráveis na rede.

Por fim, as soluções baseadas em modificações na arquitetura dos dispositivos de rede e seus componentes buscam oferecer opções que permitam a customização das regras de encaminhamento. Isso permite a rápida implantação de novos protocolos e/ou ações de controle sobre a rede. O trabalho de [Bosshart et al. 2013], descrito a seguir, está entre os principais deste subgrupo.

**RMT.** Embora a maioria dos dispositivos de rede atuais e o protocolo OpenFlow considerem o emprego de múltiplas tabelas de fluxo para otimizar o uso dos recursos de *hardware* utilizados na representação de regras, a configuração dessas tabelas ainda é extremamente rígida. Uma vez definida a capacidade e os campos de cada tabela, estes não podem mais ser alterados independente da aplicação à qual o dispositivo se destina (p.ex. roteador IP, switch Ethernet, etc.). Nesse sentido, [Bosshart et al. 2013] introduzem a ideia de um plano de dados flexível, através do uso de tabelas de fluxo reconfiguráveis (RMTs). Os autores apresentam uma arquitetura para a implementação dessas tabelas e advogam que as mesmas permitem *matches* de um conjunto arbitrário de campos, o que possibilita modificações futuras do plano de dados da rede sem a necessidade de alterações no *hardware* dos dispositivos. Não há até o momento uma implementação da arquitetura proposta. Entretanto, os autores fornecem uma série de evidências teóricas que permitem inferir sobre a sua viabilidade e eficiência.

Existem ainda soluções voltadas a memórias TCAM que não se enquadram em nenhum dos três subgrupos descritos. Essas soluções em geral possuem características peculiares (*p.ex.* são destinadas a arquiteturas de memória ou regras de encaminhamento específicas) que restringem a sua aplicabilidade. Exemplos dessas soluções podem ser encontrados nos trabalhos de [Ma and Banerjee 2012], [Kesselman et al. 2013] e [Meiners et al. 2011].

***SmartPC.*** [Ma and Banerjee 2012] fazem uso da organização em blocos de alguns modelos de TCAMs para propor uma solução que diminui o número de entradas verificadas, e consequentemente a energia consumida, a cada encaminhamento. Basicamente os autores propõem um pré-classificador (implementado através da própria TCAM), responsável por distribuir um conjunto de regras de encaminhamento entre os blocos da memória. O encaminhamento de um pacote, nesse caso, ocorre em dois estágios: pré-classificação, que determina qual ou quais blocos da memória podem conter as regras desejadas, e verificação de blocos, que efetivamente determina a regra a ser utilizada no encaminhamento sem a necessidade de verificar todas as entradas ativas da infraestrutura.

Assim como as soluções baseadas em memórias TCAM, o grupo de soluções baseadas em memórias RAM também pode ser dividido em subgrupos. Mais especificamente, há dois grandes subgrupos nesse caso: soluções de árvores de decisão e modificação de *hardware*. As soluções de modificação de *hardware*, da mesma forma que o subgrupo voltado a memórias TCAM, busca facilitar a implantação de novas ações de controle sobre a rede na forma de regras de encaminhamento mais flexíveis. No entanto, os trabalhos que adotam o contexto de memórias RAM precisam lidar com escolhas de projeto substancialmente diferentes dos demais (*p.ex.* o uso de memórias RAM tipicamente implica em tabelas de fluxos e tempos de encaminhamento maiores). Nesse sentido, questões de escalabilidade estão entre os grades desafios enfrentados. O trabalho de [De Carli et al. 2009] expressa as principais características das soluções deste subgrupo.

Trabalhos baseados em árvores de decisão, por sua vez, aplicam soluções algorítmicas que normalmente buscam um compromisso entre utilização da memória e latência de processamento. O principal desafio enfrentado pelos trabalhos desse subgrupo é permitir a classificação de pacotes em tempo real em ambientes de rede onde a capacidade de transmissão vem crescendo enormemente (*p.ex* redes ópticas). Entre as alternativas estudadas, está o desenvolvimento de soluções paralelas visando a utilização em arquiteturas de alto desempenho (*p.ex.* processadores de rede com múltiplos núcleos, GPUs, etc) [Qi et al. 2009]. Os trabalhos de [Vamanan et al. 2010], [Wang et al. 2013] e [Qi et al. 2009] estão entre os principais desse subgrupo.

## 4. Desafios e Oportunidades

A pesquisa visando a utilização eficiente de tabelas de fluxos em SDNs ainda está nos seus estágios iniciais, onde existem muitas oportunidades de trabalhos futuros. Além disso, alguns desafios fundamentais desse paradigma ainda permanecem abertos. As SDNs introduziram uma série de conceitos que resultam em condições de operação únicas da rede (*p.ex.* controle de tráfego em nível de fluxos, dinamicidade na reconfiguração do plano de controle, etc). Nesse contexto, o desenvolvimento de soluções que explorem tais condições é fundamentalmente interessante.

A organização do estado da arte desenvolvida neste artigo nos ajuda a entender

sistematicamente o espaço de projeto e os *trade-offs* envolvidos nos desafios enfrentados pela indústria e academia. Individualmente, cada grupo de soluções apresentado possui pontos fortes e fracos, que por razões de espaço não serão discutidos aqui de forma extensiva. Ao invés, preferimos comentar de maneira sucinta os principais tópicos de pesquisa que guiarão os rumos deste trabalho.

### 4.1. Aspectos Temporais *versus* Espaciais

Podemos notar um grande esforço da comunidade em otimizar aspectos *espaciais* de utilização das tabelas de fluxos (p.ex. transformando, distribuindo e representando políticas de rede e suas respectivas regras de encaminhamento eficientemente). No entanto, a consideração dos aspectos *temporais* de uso dessas tabelas se faz igualmente relevante para uma implantação eficiente no contexto de SDNs. Embora tenhamos mostrado que alguns trabalhos também exploram esses aspectos, em muitos casos isso é feito de maneira superficial (*p.ex.* considerando fluxos de protocolos específicos, regras de caráter simplificado - sem dependências, que não possuem *wildcards*, etc). Além disso, a maioria desses trabalhos utiliza abordagens reativas (baseadas em monitoramento) para o gerenciamento de regras, o que tipicamente aumenta a comunicação com o controlador e/ou o processamento efetuado sobre os pacotes da rede.

### 4.2. Atuação em Múltiplos Níveis da Arquitetura SDN

Outro aspecto relevante é a busca de soluções que combinam mais de um nível de atuação dentro da arquitetura SDN. Alguns dos trabalhos apresentados utilizam esse tipo de estratégia para potencializar os resultados obtidos (*p.ex.* transformando políticas de rede enquanto consideram aspectos de posicionamento das regras geradas). Entretanto, há uma série de combinações que ainda são pouco exploradas. Nesse sentido, algumas questões são interessantes. É possível aumentar a eficiência no uso das tabelas de fluxos ao posicionar regras na infraestrutura considerando a duração das mesmas? Ou ainda, quanto podemos otimizar a representação de um conjunto de regras de encaminhamento posicionando-as adequadamente na infraestrutura?

## 5. Objetivos e Organização do Trabalho

No segundo semestre deste Trabalho de Graduação, será feita uma análise dos aspectos temporais para utilização eficiente das tabelas de fluxos em SDNs. Mais especificamente, nosso objetivo é propor um mecanismo robusto, preciso e genérico em relação ao tipo de tráfego e de regras de encaminhamento da rede, que permita otimizar o número de entradas das tabelas de fluxos ao longo do tempo. Além disso, busca-se minimizar a sobrecarga no gerenciamento das regras. Considerando que a latência (para memórias RAM) e a energia consumida (no caso de TCAMs) tipicamente é proporcional ao número de entradas da tabela de fluxos [Liu et al. 2014], o resultado esperado é um aumento no desempenho e eficiência energética da rede como um todo em relação ao estado da arte e ao cenário atual.

### 5.1. Metodologia

A seguinte metodologia está sendo empregada, visando atingir os objetivos propostos.

1. **Estudo:** o principal objetivo desta etapa é melhorar a compreensão do problema. Para isso, a mesma é composta por duas tarefas: (i) estudo aprofundado

do paradigma de Redes Definidas por Software; e (ii) levantamento bibliográfico relacionado a propostas para lidar com aspectos de utilização de tabelas de fluxos em SDNs.
2. **Análise:** com base na etapa anterior, faz-se uma investigação dos mecanismos e técnicas existentes para para a utilização eficiente de tabelas de fluxos em SDNs, procurando identificar os principais desafios enfrentados e delinear o espaço de projeto.
3. **Busca de alternativas:** após a identificação dos desafios e delineação do espaço de projeto na etapa de Análise, busca-se alternativas para aumentar a eficiência na utilização das tabelas de fluxos, aperfeiçoando ou expandindo os mecanismos e técnicas existentes. Adicionalmente, busca-se reduzir os custos de implementação envolvidos.
4. **Implementação:** esta etapa consiste na implementação das alternativas descritas na etapa anterior, utilizando ferramentas para a prototipação de projetos em SDNs (*p.ex.* Mininet, NS-3, etc).
5. **Avaliação dos resultados:** por fim, esta última etapa destina-se à avaliação das alternativas implementadas, comparando-as através de experimentos com as demais propostas do estado da arte e os cenários atuais. Além disso, os resultados obtidos também serão utilizados para observar a viabilidade dessas alternativas através de análises de custo-benefício.

No momento atual, as etapas 1 e 2 foram realizadas e a etapa 3 está em andamento. A compreensão do problema através do estudo e análise do estado da arte deve, contudo, continuar sendo aprofundada durante a realização do Trabalho de Graduação 2.

### 5.2. Cronograma

As tarefas a serem realizadas na segunda etapa do Trabalho de Graduação encontram-se enumeradas a seguir. A Tabela 2 apresenta o cronograma de trabalho.

1. Detalhamento dos aspectos temporais para utilização eficiente de tabelas de fluxos em SDNs.
2. Projeto do mecanismo de otimização do uso temporal das tabelas de fluxos.
3. Implementação do modelo definido.
4. Avaliação experimental, através de experimentos com simulação e/ou emulação do mecanismo em ferramentas de prototipação.
5. Redação do Trabalho de Graduação 2.
6. Apresentação do Trabalho de Graduação 2.

## 6. Considerações Finais

Este artigo apresentou um estudo e análise do estado da arte com relação ao desenvolvimento de técnicas para a utilização eficiente de tabelas de fluxos em SDNs. Como parte do trabalho no qual o mesmo está inserido, nós propomos uma taxonomia para organizar as principais propostas e entender sistematicamente o espaço de projeto e os *trade-offs* envolvidos nos grandes desafios de pesquisa da área. Como próximos passos, nós planejamos explorar os aspectos temporais de uso das tabelas de fluxos, projetando, implementando e avaliando alternativas que aumentem a eficiência no uso das tabelas em relação ao estado da arte e aos cenários atuais. Em particular, nós projetamos o desenvolvimento de soluções com um bom custo-benefício como parte das contribuições do Trabalho de Graduação 2.

| Tarefa | 2014 | | | | | | |
|---|---|---|---|---|---|---|---|
| | Jun | Jul | Ago | Set | Out | Nov | Dez |
| 1 | x | x | | | | | |
| 2 | x | x | | | | | |
| 3 | | x | x | x | | | |
| 4 | | | | x | x | x | |
| 5 | | | | x | x | x | |
| 6 | | | | | | | x |

**Tabela 2. Cronograma para as atividades do Trabalho de Graduação 2.**

## Referências


Astuto, B. N., Mendonça, M., Nguyen, X. N., Obraczka, K., and Turletti, T. (2013). A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. accepted in IEEE Communications Surveys & Tutorials To appear in IEEE Communications Surveys & Tutorials.

Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. (2013). Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA. ACM.

Bremler-Barr, A. and Hendler, D. (2007). Space-efficient tcam-based classification using gray coding. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1388–1396.

Congdon, P., Mohapatra, P., Farrens, M., and Akella, V. (2013). Simultaneously reducing latency and power consumption in openflow switches.

De Carli, L., Pan, Y., Kumar, A., Estan, C., and Sankaralingam, K. (2009). Plug: Flexible lookup modules for rapid deployment of new protocols in high-speed routers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 207–218, New York, NY, USA. ACM.

Ferguson, A. D., Guha, A., Liang, C., Fonseca, R., and Krishnamurthi, S. (2012). Hierarchical policies for software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 37–42, New York, NY, USA. ACM.

Ferguson, A. D., Guha, A., Liang, C., Fonseca, R., and Krishnamurthi, S. (2013). Participatory networking: An api for application control of sdns. *SIGCOMM Comput. Commun. Rev.*, 43(4):327–338.

Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M., Katta, N., Monsanto, C., Reich, J., Rexford, J., Schlesinger, C., Walker, D., and Harrison, R. (2013). Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134.

Kang, N., Liu, Z., Rexford, J., and Walker, D. (2013). Optimizing the "one big switch"abstraction in software-defined networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 13–24, New York, NY, USA. ACM.

Kanizo, Y., Hay, D., and Keslassy, I. (2013). Palette: Distributing tables in software-defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 545–549.

Kannan, K. and Banerjee, S. (2014). Flowmaster: Early eviction of dead flow on sdn switches. In Chatterjee, M., Cao, J.-n., Kothapalli, K., and Rajsbaum, S., editors, *Distributed Computing and Networking*, volume 8314 of *Lecture Notes in Computer Science*, pages 484–498. Springer Berlin Heidelberg.

Katta, N., Rexford, J., and Walker, D. (2013). Infinite cacheflow in software-defined networks. Technical report.

Kesselman, A., Kogan, K., Nemzer, S., and Segal, M. (2013). Space and speed tradeoffs in {TCAM} hierarchical packet classification. *Journal of Computer and System Sciences*, 79(1):111 – 121.

Kim, H., Gupta, A., Shahbaz, M., Reich, J., Feamster, N., and Clark, R. (2013). Simpler network configuration with state-based network policies. Technical report.

Liu, A. X., Meiners, C. R., and Torng, E. (2010). Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Trans. Netw.*, 18(2):490–500.

Liu, A. X., Meiners, C. R., and Torng, E. (2014). Packet classification using binary content addressable memory. To appear in *INFOCOM 2014*.

Liu, A. X., Meiners, C. R., and Zhou, Y. (2008). All-match based complete redundancy removal for packet classifiers in tcams. In *INFOCOM*, pages 111–115.

Ma, Y. and Banerjee, S. (2012). A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 335–346, New York, NY, USA. ACM.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Meiners, C., Liu, A., Torng, E., and Patel, J. (2011). Split: Optimizing space, power, and throughput for tcam-based classification. In *Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on*, pages 200–210.

Meiners, C. R., Liu, A. X., and Torng, E. (2009). Topological transformation approaches to optimizing tcam-based packet classification systems. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 73–84, New York, NY, USA. ACM.

Meiners, C. R., Liu, A. X., and Torng, E. (2012). Bit weaving: A non-prefix approach to compressing packet classifiers in tcams. *IEEE/ACM Trans. Netw.*, 20(2):488–500.

Monsanto, C., Foster, N., Harrison, R., and Walker, D. (2012). A compiler and run-time system for network programming languages. *SIGPLAN Not.*, 47(1):217–230.

Monsanto, C., Reich, J., Foster, N., Rexford, J., and Walker, D. (2013). Composing software defined networks. In *Presented as part of the 10th USENIX Symposium on*

*Networked Systems Design and Implementation (NSDI 13)*, pages 1–13, Lombard, IL. USENIX.

Moshref, M., Yu, M., Sharma, A., and Govindan, R. (2013). Scalable rule management for data centers. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 157–170, Lombard, IL. USENIX.

Qi, Y., Xu, L., Yang, B., Xue, Y., and Li, J. (2009). Packet classification algorithms: From theory to practice. In *INFOCOM 2009, IEEE*, pages 648–656.

Rottenstreich, O., Keslassy, I., Hassidim, A., Kaplan, H., and Porat, E. (2013). On finding an optimal tcam encoding scheme for packet classification. In *INFOCOM*, pages 2049–2057.

Vamanan, B., Voskuilen, G., and Vijaykumar, T. N. (2010). Efficuts: Optimizing packet classification for memory and throughput. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 207–218, New York, NY, USA. ACM.

Voellmy, A., Wang, J., Yang, Y. R., Ford, B., and Hudak, P. (2013). Maple: Simplifying sdn programming using algorithmic policies. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 87–98, New York, NY, USA. ACM.

Wang, X., Chen, C., and Li, J. (2013). Replication free rule grouping for packet classification. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 539–540, New York, NY, USA. ACM.

Yu, M., Rexford, J., Freedman, M. J., and Wang, J. (2010). Scalable flow-based networking with difane. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 351–362, New York, NY, USA. ACM.

Zarek, A. (2012). OpenFlow Timeouts Demystified. University of Toronto, Toronto, Ontario, Canada.