Computing Journal manuscript No.
(will be inserted by the editor)

Adaptive Scheduling Algorithm for Media-Optimized Traffic Management in Software Defined Networks

Florin Pop · Ciprian Dobre · Dragos Comaneci · Joanna Kolodziej

Received: date / Accepted: date

Abstract Multi-policy resource management have been considered as an efficient methodology for delivering ready-to-use media-optimized applications in Software-Defined Networks (SDNs). Prioritized flow scheduling ensures high-speed communication in SDNs under large-scale distribution, heterogeneity of network resources, and exponential distribution of the flows granularity. The effectiveness of priority-based approaches depends usually on the control mechanism of the resource management. In this paper we improve the resource utilization by developing a novel adaptive scheduling strategy. We came with an effecting scheduling strategy to determine what resource to be allocated to a set of flows keeping their priority, increasing the average utilization of resources and, most importantly, establishing a virtual circuit for a specific flow over a network. Our theoretical remarks and extensive simulation results show that the proposed scheduling strategies can achieve the described goals and has better performance than its classical approach.

Keywords Scheduling · Flow Management · Adaptive Methods · Media-Optimized Applications · Software-Defined Networking

PACS Distribution theory, 02.50.Ng

Florin Pop - Principal Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: florin.pop@cs.pub.ro

Ciprian Dobre - Corresponding Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: ciprian.dobre@cs.pub.ro

Dragos Comaneci
PhD Student, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: dragos@comaneci.ro

Joanna Kolodziej, PhD in Computer Science (with tenure), Associate Professor,
Institute of Computer Science, Cracow University of Technology, Poland,
E-mail: jokolodziej@pk.edu.pl

Mathematics Subject Classification (2000) 68M20 · 68M14 · 68U20

1 Introduction

In packet-switched networks, message files are divided into small data packets that can be sent through the most efficient neighbor routers as paths become available. There is a simple selection mechanism embedded in each routing or switching device which is responsible for dropping some data packages, and processing the other ones. However, it is important to avoid high packet loss rates in the Internet. When a packet is dropped before it reaches its destination, all of the resources it has consumed in transit are wasted. In extreme cases, this situation can lead to congestion collapse [27]. In the last years, a lot of research results on dynamic Internet resource management methodologies have been developed. The most popular are Weighted fair queue (WFQ) [11] and Stochastic Fairness Queueing (SFQ) [31], to Random Early Detection (RED) [15] and BLUE [12], to Explicit Congestion Notification (ECN) [14], eXplicit Control Protocol (XCP) [29] and Rate Control Protocol (RCP) [41], to Data Center TCP (DCTCP) [2], Preemptive Distributed Quick (PDQ) [23], Controlled Delay (CoDel) [33], and pFabric [3]. The main aim of such models is to make the Active Queue Management (AQM) process automatic, or at least semi-automatic. Although, the general problem is not so novel, there is still a need of the development of new "one-size-fits-all" ideal AQM algorithms. In particular, in switch scheduling, the consensus today appears to be to implement some variant of weighted fair queueing [11]. More recently, authors such as [38] brought evidence that the AQM policy, in fact, depends crucially on the requiements coming from upper-layer applications. For example, the throughput of the computer backup application are different from an interactive web page load times, which in turn differ from a videoconference application. The multimedia application is able to cope with the loss of few packets, but the delivery rate has to be maintained high. For the backup application, instead, packet losts are not easily tolerated. In fact, even when the applications want the same thing (e.g., throughput), the answers depend on how the endpoint's transport protocol reacts to congestion signals; the right answer for TCP NewReno may well be different from TCP CUBIC [20] or Compound TCP [39]. New network applications are certain to be developed, and new queueing strategies, scheduling techniques, and endpoint control protocols will be invented. New applications will require different objectives, e.g., a different trade-off between throughput and delay, an emphasis on throughput or delay variation above other metrics, and so on. So how should the switch designer respond to these concerns about the diversity of objectives and mechanisms?

Software Defined Networks (SDNs) [6] seem to be today the promising solutions for solving the complex large-scale network control problems. SDNs clearly separate the the network control functions from the network switching elements. By moving the control plane out from the network elements

into stand-alone servers, the switching elements can remain simple, general-purpose, and cost-effective and at the same time the control plane can rely on design principles of distributed systems in its implementation instead of being confined to distributed routing protocols.

There are many examples of split-architecture networks with the separation mechanisms of network control from data plane, which is based on the *general principle* of policy/mechanism separation specified in [24]. In the past, the internal components of networks – switches, routers, firewalls, traffic management, and so on – have generally evolved in a separate software and hardware ecosystem from end systems. Hence, while commodity price, performance and production of smart phones, desktops, servers and whole data center computing has largely benefited from exactly the same advances, and scale of use of computer science, the infrastructure of the Internet has moved more and more into being a set of niche businesses generating specialised technologies. The newer approaches work to reverse this trend, by moving large fractions of the control plane of the Internet, and even some parts of the data plane into the arena of main stream computer science. Such approaches, however, do not introduce yet sufficiently powerful abstractions to realize full convergence of the very diverse set of network and service control interfaces inherent to today's carrier networks. As carrier networks *become more integrated with services such as Internet Protocol television (IPTV)*, customer data hosting, and general infrastructure as a service, there is increasing demand for a unified control plane and management interface for the network. An SDN approach to this problem would introduce complex distributed systems problems of control state management. Nevertheless, SDN presents an opportunity toward unifying today's complex tangle of network and service control management.

The main aim of our work presented in this paper is to extend the general SDN model through an integration with a novel software-controlled scheduler for flow queuing over the data plane (i.e., abstract the packet queue policy within the software layer). We try to reduce within this scheduling strategy the complexity of the control state management. We argue that end-point switches in Data Centers, enabled with support for OpenFlow and other realizations of Software-Defined Networking, but also enriched with fine-grained flow-oriented multi-policy scheduling capabilities, controlled at the software level, have the power to support advanced levels of configurability. Such an approach could allow network operators to align queueing and scheduling behavior with a wide-range of media-rich application objectives. We focus on the SDN multi-mechanism paradigms widely discussed at the "Future Internet" seminar organized in Dagstuhl on March 25-27, 2013 [9]. The term "multi-mechanism" usually refers to a set of mechanisms that serve the same functional purpose and hence may be interchangeable with each other (the mechanisms may nevertheless differ in non-functional qualities such as throughput, delay, reliability, or security).

Our contribution in this paper is the following:

- First, we develop a multi-policy scheduling strategy in order *to transfer* media-optimized traffic over SDNs. We acknowledge that there is no solutions that could fit all kinds of communication patterns. Therefore, our scheduling strategy is based on the adaptive internal mechanism, where flows are prioritized considering both the heterogeneity of network resources, and application flows. The effectiveness of the utilization of the network resources is determined by the efficiency of the allocation strategy of the resources.
- We provide a comprehensive theoretical and empirical analysis of the effectiveness of the resource management in SDNs under different scheduling policies and conditions. Our simulation experiments show that we can establish an upper bound imposed by the condition to have the maximum probability of arriving flows in the systems in a steady state.

The rest of the paper is organized as follows. Sec. 2 presents the background of the proposed solution introducing the mathematical model, queues management, fluid-flow Markov chain model. In Sec. 4 we define the scheduling algorithms for traffic flow in SDNs. Sec. 5 presents the results of a simple simulation study for our proposed solution considering different scenarios. The paper ends with conclusions and future work presented in Sec. 6.

2 Background and System Description

SDNs together with OpenFlow [32], NOX [19], DevoFlow [10], and other software frameworks, have eased the management of enterprise networks in the last few years. SDNs are not a revolutionary technology, but rather an organizing principal, a new way of looking at networking using abstractions. It all started around 2004, with projects such as RCP [5], 4D [17], SANE [7], and others, that proposed new ways to look at managing networks. In approximately 2008, the SDN elements were formed, with the NOX Network Operating System [19] and the OpenFlow switch interface [32]. Three years later, the Open Networking Foundation was formed [1] - a non-profit organization founded by Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo! to improve networking through software-defined networking (SDN). New SDN frameworks were developed and will continue to appear since then on this topic [34]. The rapid-adoption of SDN principles (and technologies) by most major networking vendors and Cloud providers alike has to do with the main advantage that SDN brings: simplified networking management. Previously, network engineers were forced to deal with complex large-scale systems, to understand hardware-dependent mechanisms and instructions for a wide-variety of switches and networks, to deal with many operating systems and network protocols, both at control- and data-planes. Such a monolithic approach, where a core of (highly-trained) people (sometimes, one network administrator alone, trained in all

[1] https://www.opennetworking.org/

possible networking technologies) has to deal with all-kind of networking complexities, was no longer feasible for complex Data Centers. SDNs, on the other hand, propose the adaption of software modularity and abstraction principles into the networking world. In software, modularity provides code reusability, flexibility of implementation, and/or conceptual separation of concerns. These are the aspects that SDN bring to the networking management (i.e, within the control plane). A SDN offers and separates the networking management into (see Figure 1) 1) a *control program*, where a network operator can express (in software) goals on an abstract view, 2) a virtualization layer, which provides a clean separation between the abstract view and the complexities of network topologies, and 3) a network operating systems (NOS), responsible for the actual translation from software commands (driven by network state abstractions) into actual commands understood by the hardware switches and routers within the network (similar to how a compiler translates higher-level instructions into machine-level instructions - in this case, forwarding abstractions). The control program sees a very simple network, it tells the network what it wants to have happened (i.e., blocking traffic), the virtualization layer converts that to the global network view (i.e, adapts the request to the actual network topology), which it hands off to the network OS, which in turns talk to the physical switches. Both the virtualization and NOS are doing an automated job, through frameworks such as NOX and OpenFlow. What is missing is actual automation at the Control Program, and we present in this paper such a construction.

Flow scheduling is essential in software-defined network (SDN) systems. In [30] is proposed a prioritize algorithms called Iterative Parallel Grouping Algorithm (IPGA) for prioritized flow-scheduling problem. The case study presented in this approach was a CUDA system within a SDN controller. Our solution is similar considering a multi-queuing system.

Hedera is a scalable and dynamic flow scheduling system for a multi-stage switching fabric [1]. Hedera considers commodity switches and unmodified hosts and the results obtained for dynamic scheduling shoes an optimal bandwidth allocation around 96% than static load-balancing methods. [1]. We obtained with our proposed solution similar system efficiency, but we consider a practical approach for heterogeneous system (Poisson arriving queues with exponentially processing time).

An API for applications to control a software-defined network (SDN) is presented in [13]. The scheduling model is OpenFlow controller that delegates flows over different networks.

LocalFlow is an optimal (under standard assumptions), scalable, and practical switch-local algorithm for aggregated flow per destination and allowing slack in distributing this flow [35]. LocalFlow acts independently on each switch being highly scalable. Our solution consider a unified approach with a multi-queuing system and priorities for each flow.

SDNs, largely directed at influencing or centralizing control-plane decisions, have allowed network operators to configure their networking-based production systems [28] more efficiently and realize new behaviors not otherwise available

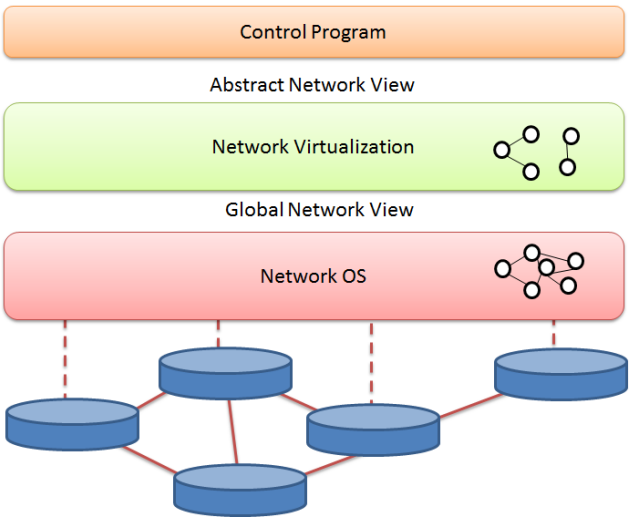

Fig. 1 SDN: Layers for the Control Plane.

from commercial vendors. Several switch vendors today support OpenFlow - a framework that support fine-grained, flow-level control of Ethernet switching. OpenFlow is used today for traffic management in data centers around the world [1][22]. The kind of control enabled by SDN is desirable because it enables (1) correct enforcement of flexible policies without carefully crafting switch-by-switch configurations, (2) visibility over all flows, allowing for near optimal management of network traffic, and (3) simple and future-proof switch design. And all is offered free of hardware details - for the first time network operators are able to deal with network management, regardless of the actual hardware and how the nodes are connected. But, even if SND enables flexibility in the control plane, the obstacles toward widespread deployment of congestion-control algorithms and protocols, such as XCP [29], RCP [41], or CoDel [33] speak of the consequences of today's data-plane rigidity (at the actual switch level, the mechanisms to decide which link and queue to use, where and when to send each packets, are decisions yet to be tackled by researchers). We propose to extend SDN's flexibility to cover queueing and scheduling decisions made traditionally at the hardware data plane, with a flow-dependent adaptive software scheduler capable to take decisions at the control plane (i.e., within the edge software switch).

3 Generic System Model

In order to define the generic model of the whole system, let us denote by $\{F_i\}_{1 \leq i \leq n}$ a set of the package flows. We assume that all packages in flow F_i have the same size D_i. Let us also assume that all flows form a global

queue (Q), which can be divided into sub-queues $\{Q_j\}_{1 \leq j \leq m}$[2] by the internal classification mechanism. This queue decomposition process is based on the flows priorities (which can be for example the sizes of the packages in the flows). The scheduler allocates ports to all queued flows and updates transmission bandwidth for all ports according with flows' package size.

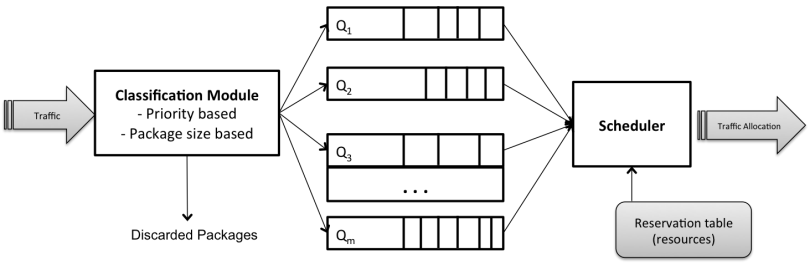

Fig. 2 System architecture based on Queue Management

The main concept of the model is presented in Fig. 2. A *Reservation Table (RT)* is used by the scheduler for a static allocation to maintain the resources workload (W_p). A resource in our model is defined as a switch port (P). We assume, for simplicity, that all resources are of the same capacity, and each flow F_i requires a unit of bandwidth. We consider that the flows arrive according with a Poisson distribution with the same parameter λ. The flows size has a exponentially distributed processing time with average equal to $1/\mu$. The notation used in the system model is presented in Table 1.

3.1 Queue Management

Let us consider a multiple channel queues with $N(t) = n$ flows generated at time t. Let us denote by $p_n(t)$ a probability of the generation of n flows at time t. From the Poisson distribution we can compute the transition probabilities as following:

$$P(N(t + \Delta t) = n + 1 | N(t) = n) = \lambda \Delta t, \tag{1}$$

$$P(N(t + \Delta t) = n - 1 | N(t) = n) = \mu_n \Delta t \tag{2}$$

The equilibrium state in the system of n flows is time independent and can be computed in the following way:

$$\lambda p_n = \mu_{n+1} p_{n+1}, \tag{3}$$

It means that all new flows are allocated in the resources defined as the system equilibrium state. Based on that, we can compute the probability to

[2] The value of the parameter m can be different for different Q_j.

Table 1 Notations

Notation	Description
$n; n_{max}$	The number of Flows; The maximum number of Flows.
m	The number of resources (ports).
F_i	The Flow.
D_i	The size of a package in Flow F_i.
Q_i	The sub-Queue of a global queue (Q).
W_i	The resource workload (used bandwidth).
λ	Variance of a Poisson distribution.
μ	The parameter of the exponential distribution, often called the *rate parameter*.
μ_n	The exponential rate parameter for a specific number of Flows, n.
$t; \Delta t$	Time; A short interval of time denoting a transition step.
$N(t)$	Number of Flows at moment t.
$p_n(t)$	Probability to have n Flows at moment t.
p_n	Probability to have n Flows in a steady state (at equilibrium).
$E(F)$	Efficiency metric for a set of Flows (F).
ρ	The amount of processing-time per unit time (λ/μ) divided by the processing capabilities per unit time (number of resources, m).
T_L	Lower throughput.
T_U	Upper throughput.
T_M	Utilization rate.

have n Flows in a steady state (at equilibrium) as follow:

$$p_n = p_0 \frac{\lambda^n}{\prod_{i=1}^{n} \mu_i}, \tag{4}$$

where p_0 is the probability of generation an 'empty' network (no switches) and p_0 is computed based on the following relation:

$$p_0 = 1 - \sum_{i=1}^{n} p_i; \ \sum_{i=0}^{n} p_i = 1, \tag{5}$$

Let's consider m identical resources, which means that we have common queues. This supposition is based on the fact that all ports in a switch are identical. The processing time for a flow is exponentially distributed with rate parameter μ_n depending on the number of flows. We consider that $\mu_n = n\mu$ if $n \leq m$ and $\mu_n = m\mu$ when $n > m$, which simplify the proposed model (μ is a constant used for an homogeneous system). The second consideration describes the possibility to drop a flow, by limiting the average processing time to $1/(m\mu)$.

The system efficiency $E(F)$ is measured in this work as a flow-time processing and is defined as follows:

$$E(F) = \frac{1}{\lambda} \sum_{n=0}^{\infty} n p_n. \tag{6}$$

The computation of the $E(F)$ values can be realized in the following steps:

– Bootstrap case: $n = 0, \ldots, m$ and $\mu_n = n\mu$:

$$p_n = \frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n p_0 \tag{7}$$

– Dropped packages case: $n = m+1, m+2, \ldots$ and $\mu_n = m\mu$:

$$p_n = \frac{m^{m-n}}{m!}\left(\frac{\lambda}{\mu}\right)^n p_0 \tag{8}$$

– Empty system with no flows, $n = 0$:

$$p_0 = \left[\sum_{n=0}^{m-1}\frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n + \frac{1}{m!}\left(\frac{\lambda}{\mu}\right)^m \frac{m\mu}{m\mu-\lambda}\right]^{-1} \tag{9}$$

Let us finally introduce a parameter ρ, which is defined as the amount of processing-time per unit time (λ/μ) divided by the processing capabilities per unit time (number of resources), that is to say:

$$\rho = \frac{\lambda}{m\mu}. \tag{10}$$

The parameter ρ can be used for an estimation of the average number of resource used per unit time for flows processing:

$$\rho = \frac{1}{m}\left(\sum_{n=0}^{m} np_n + m\sum_{n=m+1}^{\infty} p_n\right) \tag{11}$$

Therefore $E(F)$ can be finally expressed as follows:

$$E(F) = \frac{1}{\lambda}\sum_{n=0}^{\infty} np_n = \frac{1}{m!}\left(\frac{\lambda}{\mu}\right)^m \frac{m\mu}{(m\mu-\lambda)^2}p_0 + \frac{1}{\mu} \tag{12}$$

3.2 Fluid Flow Markov Chain Model and Queue State Estimation Model

The system schedules all flows by allocating one flow on a resource (port) or dropping flows when all resources are busy. This process can be defined based on a fluid flow Markov chain model presented in [8]. This model is used for the estimation of the probability used to establish the upper bound for number of tasks in a queue in the developed system [37]. The model is defined by the following fluid flow equations:

$$\frac{dp_0(t)}{dt} = \mu p_1(t) - \lambda p_0(t), \tag{13}$$

$$\frac{dp_n(t)}{dt} = \lambda p_{n-1}(t) + \mu p_{n+1}(t) - (\lambda+\mu)p_n(t). \tag{14}$$

where $p_n(t)$ is the probability to have n flows in queue at moment t.

A Markov state is defined by the number of flow in a system's queue. The processing time follows exponential distribution with average $1/\mu$ and the flows arrive in a queue by a Poisson distribution with parameter λ. We analyze one queue, so $m = 1$ and $\rho = \lambda/\mu$.

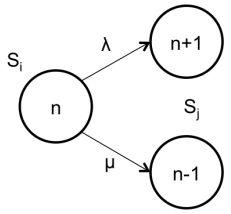

Fig. 3 Possible transitions from state S_i to state S_j.

The space of the states can be defined based on transitions presented in Figure 3 and the transition probability (which can be interpreted the probability to process a flow from a queue during h time unit) is:

$$p_{ij}(h) = \begin{cases} \lambda h & \text{if } j = i+1 \\ \mu h & \text{if } j = i-1 \\ 1-(\lambda+\mu)h & \text{if } j = i \\ 0 & otherwise. \end{cases} \tag{15}$$

The Markov chain that models arrival process is defined considering hidden transitions and observable transitions. Based on above equations, we can define a transition rate matrix for a continuous-time Markov chain as:

$$S = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & \dots \\ \mu & -(\lambda+\mu) & \lambda & 0 & \dots \\ 0 & \mu & -(\lambda+\mu) & \lambda & \dots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix} \tag{16}$$

We can remark that $S = [s_{ij}] = P'(0) = [p'_{ij}(0)]$, and by using Chapman-Kolmogorov equations [16]:

$$p_{ij}(h+t) = \sum_k p_{ik}(h) p_{kj}(t) \tag{17}$$

and differentiate with respect to h, and then setting $h = 0$ we have:

$$p'_{ij}(h+t) = \sum_k p'_{ik}(h) p_{kj}(t) \tag{18}$$

$$p'_{ij}(t) = \sum_k p'_{ik}(0) p_{kj}(t) = \sum_k s_{ik} p_{kj}(t), \tag{19}$$

which gives the following differential equation:

$$P'(t) = SP(t). \tag{20}$$

This result gives the following form for the probability matrix:

$$P(t) = e^{St} = \sum_{n=0}^{\infty} \frac{S^n t^n}{n!} = I + \sum_{n=1}^{\infty} \frac{S^n t^n}{n!}. \tag{21}$$

Based on previous results we can conclude that the defined chain is a time-homogeneous Markov chains (or stationary Markov chains) because the processes have the following properties:

$$\Pr(N(t+\Delta t) = n+1 | N(t) = n) = \Pr(N(t) = n+1 | N(t-\Delta t) = n), \tag{22}$$
$$\Pr(N(t+\Delta t) = n-1 | N(t) = n) = \Pr(N(t) = n-1 | N(t-\Delta t) = n), \tag{23}$$

$\forall n$, which means that the probability of the transition is independent of n.

At equilibrium, for proposed fluid flow, we have a steady state defined from equations 20 by:

$$SP(t) = 0. \tag{24}$$

Considering equations 13 and 14 we have:

$$\frac{dp_n(t)}{dt} = 0, \forall n \tag{25}$$

and the new equations produced the following result: $p_n = \rho p_{n-1}$ and $p_n = \rho^n p_0$. Based on the assumption $\sum_{k=0}^{n} p_n = 1$, we can compute $p_0 \sum_{k=0}^{n} \rho^n = 1$ and $p_0 = (1-\rho)/(1-\rho^{n+1}) \approx (1-\rho)$. The final result is:

$$p_n = \rho^n (1-\rho). \tag{26}$$

We can establish the maximum number of flows by considering $p_n = p_n(\rho)$ and computing $p'_n(\rho) = 0$, which means $n\rho^{n-1} - (n+1)\rho^n = 0$ and:

$$n_{max} = \left[\frac{\rho}{1-\rho} \right] + 1 \tag{27}$$

This Markov chain is *ergodic*, cause there exists a finite number N such that any state can be reached from any other state in exactly N steps [36] (for example FCFS strategy has $N = 1$, which means that a flow is processed immediate when it arrives). This property is necessary to prove the correctness of proposed scheduling algorithms, which consider that exist a schedule solution any system state.

An illustrating example can be defined as follows: for $\rho = 0.8$ we have $n_{max} = 5$ flows, for $\rho = 0.9$ we have $n_{max} = 10$ flows and for $\rho = 0.99$ we have $n_{max} = 100$ flows accepted in a queue.

Another significance for ρ considers the upper throughput T_U and lower throughput T_L for a resource allocated with a flow. The utilization rate can be defined using $T_M = T_L/T_U$ rate as:

$$\rho = \frac{T_U - T_L}{T_U} = 1 - \frac{T_L}{T_U} = 1 - \frac{1}{T_M}. \tag{28}$$

The probability can be re-computed now as:

$$p_n = \rho^n p_0 = \rho^n \frac{1-\rho}{1-\rho^{n+1}} = \left(1 - \frac{1}{T_M}\right)^n \frac{1 - \left(1 - \frac{1}{T_M}\right)}{1 - \left(1 - \frac{1}{T_M}\right)^{n+1}} \approx \frac{1 - \frac{n}{T_M}}{1+n}. \tag{29}$$

Here, as a simple remark we have $p_n \geq 0$, so $1 - \frac{n}{T_M} \geq 0$ and we obtain $n \leq T_M$, so we have the following result:

$$n_{max} = [T_M], \tag{30}$$

which is the new bound for maximum number of packages in a queue. In the experimental section we will analyze the variation of maximum number of messages in a queue and the system efficiency defined in this section.

4 Proposed SDN Scheduling Algorithms

For providing an optimal allocation process, the scheduler must find a near-optimal solution for the resource utilization problem under the assumption that the flow F_i mapped to the resource P_j will not be rescheduled to other resource. In this context, the scheduler considers the strict priority queue policy for flow processing and high priority flows are allocated first.

The resource allocation process is provided in three different scenarios based on the following minimal bandwidth definitions: (1) the minimal bandwidth is equal to the transmission window time for the flow of the size D_i *the (basic scenario*); (2) the minimal bandwidth is grater than the sum of all allocated flows plus the minimal bandwidth (the *worst scenario*); and (3) the minimal bandwidth is less than sum of all allocated flows plus the minimum bandwidth (the *"typical" scenario*).

We have developed three versions of the scheduling algorithms based the flows granularity, namely (i) *fixed priority scheduling strategy*, where specific selection strategy (first come first served, random, etc.) is applied; (ii) *balanced scheduling strategy* similar to Max-Min-Max heuristic [26]; and (iii) *adaptive cost scheduling strategy*, which is adapted to the current resource utilization.

Proposed SDN Scheduling algorithms consider minimum processing time for a flow as a metric for resource allocation. This means that, taking into consideration the priority, a flow will be completed earliest according with given priority. Alg. 1 considers different strategies for each queue and process all queues starting with highest priority queue.

Algorithm 1 Fixed Priority Scheduling using Specific Selection Strategy

```
1: procedure FixedPriorityScheduling(Q, strategy)
2:     for all ports, p do
3:         W_p ← 0;
4:     end for
5:     Q ← Set of Flows;
6:     while Q ≠ Φ do
7:         for pr = HighestPriority ... LowestPriority do
8:             Q_pr ← extractFlows(Q, pr);
9:             Q ← Q \ Q_pr;
10:            while Q_pr ≠ Φ do
11:                Select flow F_q ∈ Q_pr using specified strategy;
12:                                          ▷ strategy can be: FCFS, random, etc.
13:                W_p ← min_{j=1,m} {W_j};
14:                if W_p < W_MAX then
15:                    Allocate Flow F_q on Port p; W_p ← W_p + D_q; Q_pr ← Q_pr \ F_q;
16:                else
17:                    Consider other Port.
18:                    if all Ports are used at maximum capacity then
19:                        Drop all Flows from Q_pr.
20:                    end if
21:                end if
22:            end while
23:        end for
24:    end while
25:    Return W;                                  ▷ Array with Ports' workload.
26: end procedure
```

Alg. 2 presents a strategy for balancing the flow scheduling by considering a longest flow then s shortest flow. This mixture ensure e good balancing of queues processing in the proposed system. After each allocation of a flow to a port, the workload of each port will be updated.

The Adaptive Cost Scheduling strategy presented in Alg. 4 consider a normalized cost computed for the shortest flow and for best allocation of it. The intuition behind this is that we select pair flow and port from the first step that the port can process its corresponding flow effectively with a lower processing time in comparison with other ports without overlap the maximum workload.

Scheduling policies used in presented algorithms are based on priorities and ports workload. Each strategy consider a forwarding and dropping policy. When an port is loaded at its maximum capacity we will try another port with respect to the algorithm strategy. This is the forwarding policy. When all ports are loaded to maximum capacity, all flows are dropped.

5 Simulation Study

Data Center (DC) is usually constructed as a large, dedicated cluster of computers that is owned and operated by a single organization. On one hand, world leading universities and private enterprises are increasingly consolidating their IT services within on-site data centers containing a few hundred to a

Algorithm 2 Balanced Scheduling

```
1: procedure BalancedScheduling(Q)
2:     for all ports, p do
3:         W_p ← 0;
4:     end for
5:     Q ← Set of Flows; n ← |Q|;
6:     m ← Number of Ports;
7:     if n ≤ m then                    ▷ Bootstrap Strategy (Allocate each Flow on a Port).
8:         for i = 1, n do
9:             Allocate Flow F_i on Port i; W_i ← W_i + D_i;
10:        end for
11:    else
12:        for k = 1, m do                          ▷ Allocate a Port for a larger Flow.
13:            Costs = ComputeMinAllocationCosts(Q, W);
14:            Select C_pq ← max{C_ji}, C_ji ∈ Costs;
15:            if W_p < W_MAX then
16:                Allocate Flow F_q on Port p; W_p ← W_p + D_q; Q ← Q \ F_q;
17:            else
18:                Consider other Port.
19:                if all Ports are used at maximum capacity then
20:                    Drop all Flows from Q.
21:                end if
22:            end if
23:        end for
24:        while Q ≠ Φ do
25:            Costs = ComputeMinAllocationCosts(Q, W);
26:            Select C_pq ← min{C_ji}, C_ji ∈ Costs;
27:            if W_p < W_MAX then
28:                Allocate Flow F_q on Port p; W_p ← W_p + D_q; Q ← Q \ F_q;
29:            else
30:                Consider other Port.
31:                if all Ports are used at maximum capacity then
32:                    Drop all Flows from Q.
33:                end if
34:            end if
35:            if Q == Φ then
36:                exit;
37:            end if
38:            Costs = ComputeMinAllocationCosts(Q, W);
39:            Select C_pq ← max{C_ji}, C_ji ∈ Costs;
40:            if W_p < W_MAX then
41:                Allocate Flow F_q on Port p; W_p ← W_p + D_q; Q ← Q \ F_q;
42:            else
43:                Consider other Port.
44:                if all Ports are used at maximum capacity then
45:                    Drop all Flows from Q.
46:                end if
47:            end if
48:        end while
49:    end if
50:    Return W;                                ▷ Array with Ports' workload.
51: end procedure
```

Algorithm 3 Compute Minimum Allocation Costs for a Flow

```
1: procedure ComputeMinAllocationCosts(Q, W)
2:     Costs ← Φ;
3:     for each flow F_i ∈ Q, i = 1, n do
4:         for each port P_j ∈ Ports, j = 1, n do
5:             C_ji ← W_j + D_i;
6:         end for
7:         C_pi ← min {C_ji}, j = 1, m;
8:         Costs ← Costs ∪ C_pi;
9:     end for
10:    Return Costs;
11: end procedure
```

Algorithm 4 Adaptive Cost Scheduling

```
1: procedure AdaptiveCostScheduling(Q)
2:     while Q ≠ Φ do
3:         NormCosts ← Φ;
4:         for each flow F_i ∈ Q, i = 1, n do
5:             for each port P_j ∈ Ports, j = 1, n do
6:                 C_ji ← W_j + D_i;
7:             end for
8:             C_pi ← min {C_ji}, j = 1, m;
9:             D_p ← min {D_i}, i = 1, n;
10:            E_pi = C_pi / D_p;
11:            NormCosts ← NormCosts ∪ E_pi;
12:        end for
13:        Select E_pq ← max {E_ji}, E_ji ∈ NormCosts;
14:        if W_p < W_MAX then
15:            Allocate Flow F_q on Port p;
16:            W_p ← W_p + D_q; Q ← Q \ F_q;
17:        else
18:            Drop all Flows from Q.
19:        end if
20:    end while
21:    Return W;                          ▷ Array with Ports' workload.
22: end procedure
```

few thousand servers. On the other hand, the major online service providers, such as Google, Microsoft, and Amazon, are rapidly building geographically diverse cloud DCs, often containing more than 10K servers, to offer a variety of cloud-based services such as Email, Web servers, storage, search, gaming, and Instant Messaging. These service providers also employ some of their data centers to run large-scale data-intensive tasks, such as indexing Web pages or analyzing large data-sets. A typical DC is designed according to a 3-Tiered architecture (Figure 4) [4]. The three tiers of the data center are the edge tier, which consists of the Top-of-Rack switches that connect the servers to the data center's network fabric; the aggregation tier, which consists of devices that interconnect the ToR switches in the edge layer; and the core tier, which consists of devices that connect the data center to the WAN. In smaller data centers, the core tier and the aggregation tier are collapsed into one tier, resulting in a 2-Tiered data center topology.

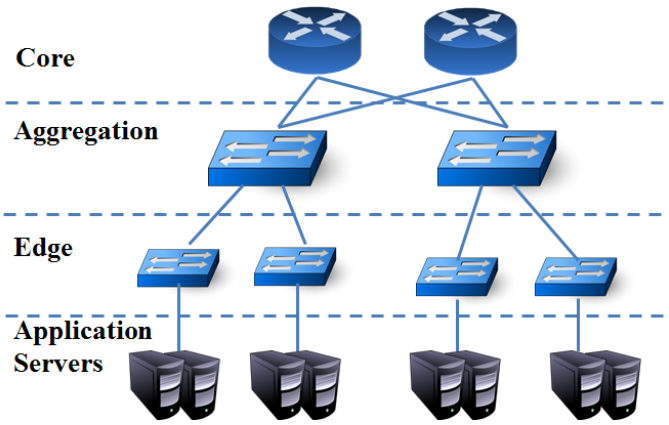

Fig. 4 A Canonical 3-Tier data center topology.

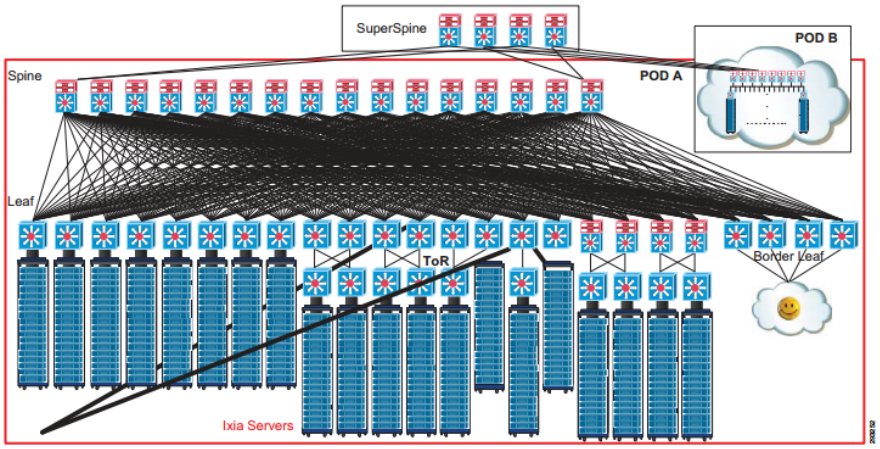

Fig. 5 Topology of Cisco MSDC Design [40].

Warehouse-Scale Computers (WSC) is a typical network fabric for Data Centers that provides symmetric bi-section bandwidth between any two nodes that are attached to the fabric. It is built using identical switching platforms and is deployed in a Clos topology with variable Clos stages with short spine layer serving as the switch fabric to an up to 2X longer leaf layer serving as the server access layer. The precursors of this were network architectures such as Monsoon [18], where researchers decoupled traffic from application workload (mainly for load balancing) using mesh-like topologies (among other constructions). The Clos network is a multistage switching network, where connections between a large number of input and output ports can be made by using only small-sized switches. A bipartite matching between the ports can be made by configuring the switches in all stages. Thus, a Clos network is a multistage switching network. It can be shown that a Clos network can be

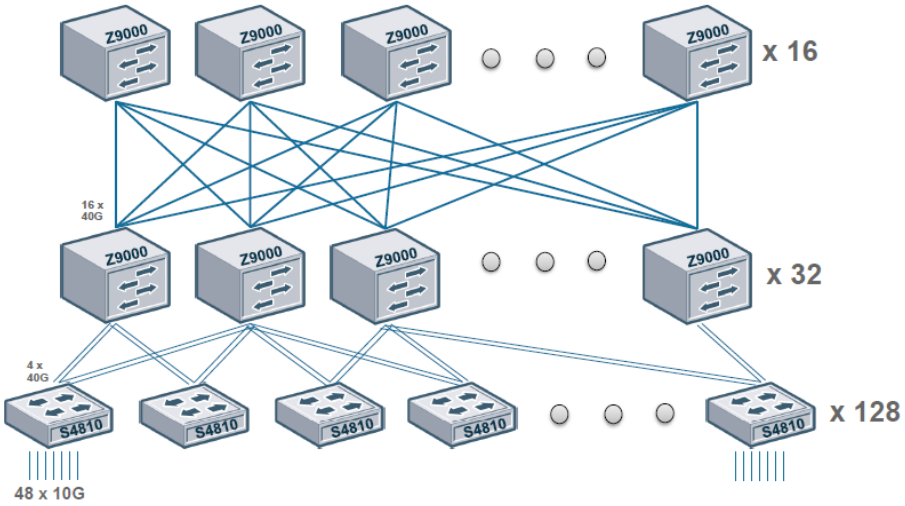

Fig. 6 Sample Folded Clos Dell topology [21].

non-blocking like a cross-bar switch. That is, for each input-output matching we can find an arrangement of paths for connecting the inputs and outputs through middle-stage switches [25]. This is exactly the kind of architecture we'll consider in our experiments next, where theoretically each package could be shipped in the edge switch on any output port.

Theoretically, a three-stage fat tree Clos (i.e., considernig a 16-wide spine) architecture using 32 port switches can connect up to 8192 servers. Considering the universality theorem (for a given number of switches, the most optimal network exists), a network topology design for DCs is optimized for transporting data into and out of a data center by imposing a logical tree topology on a multi-path physical network. DCs use some variant of a multi-rooted topology, which actively manages the multiple paths available between two end points using Equal Cost Multipathing (ECMP) [40]. In addition, the selection of a high port count platform as the spine enables a deployment of folded-clos (or fat-trees) without using additional switching components. Examples of such architectures currently being deployed in DCs all over the world are presented in Figure 5.

A comprehensive study of network design and usage of modern Data Centers is presented in [4]. The authors studied data collected from ten DCs, to shed light and identify properties that can help improve operation of their networking substrate. According to their findings, there are a wide variety of applications across the DCs, ranging from customer-facing applications, such as Web services, file stores, authentication services, Line-of-Business applications, and custom enterprise applications to data intensive applications, such as MapReduce and search indexing. The application placement is non-uniform across racks. Most flows in the Data Centers are small in size ($\leq 10KB$), a

significant fraction of which last under a few hundreds of milliseconds, and the number of active flows per second is under 10,000 per rack across all DCs.

Despite the differences in the size and usage of the DCs, traffic originating from a rack in a DC is ON/OFF in nature with properties that fit heavy-tailed distributions. Also, in the cloud Data Centers, a majority of traffic originated by servers (80%) stays within the rack. For the university and private enterprise DCs, most of the traffic (40-90%) leaves the rack and traverses the network's interconnect. Still, irrespective of the type, in most DCs, link utilization are rather low in all layers but the core. In the core a subset of the core links often experience high utilization. Furthermore, the exact number of highly utilized core links varies over time, but never exceeds 25% of the core links in any DC. Finally, link utilization are subject to time-of-day and day-of-week effects across all DCs. However in many of the cloud DCs, the variations are nearly an order of magnitude more pronounced at core links than at edge and aggregation links.

Our experiments were designed with respect to these findings. In the followings, we accurately mimic real-world networking conditions in Data Centers.

Evaluation of Proposed System. Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important being Flow-Time, which is the sum of finalization times of all the flows existing in the system. An optimal schedule will be the one that minimizes the Flow-Time.

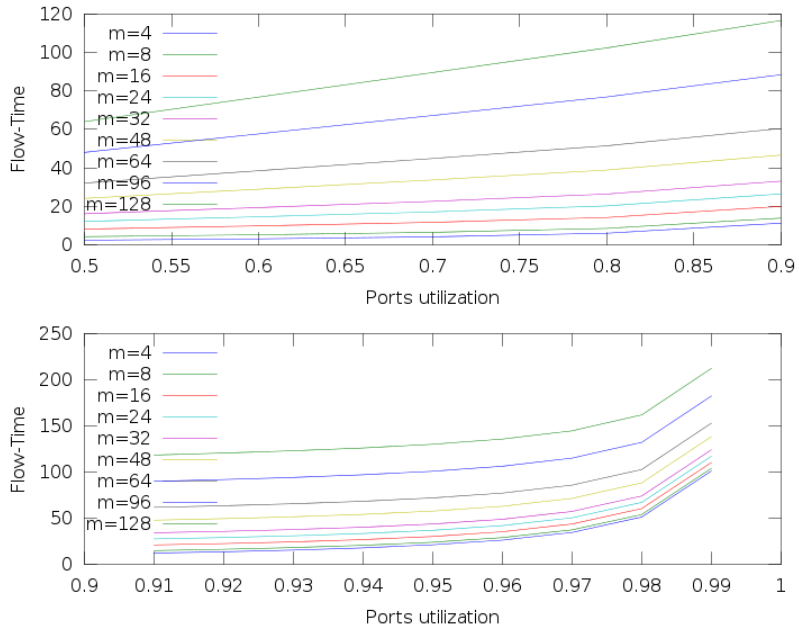

Fig. 7 Expected flow-time for different number of ports (fixed arrival rate).

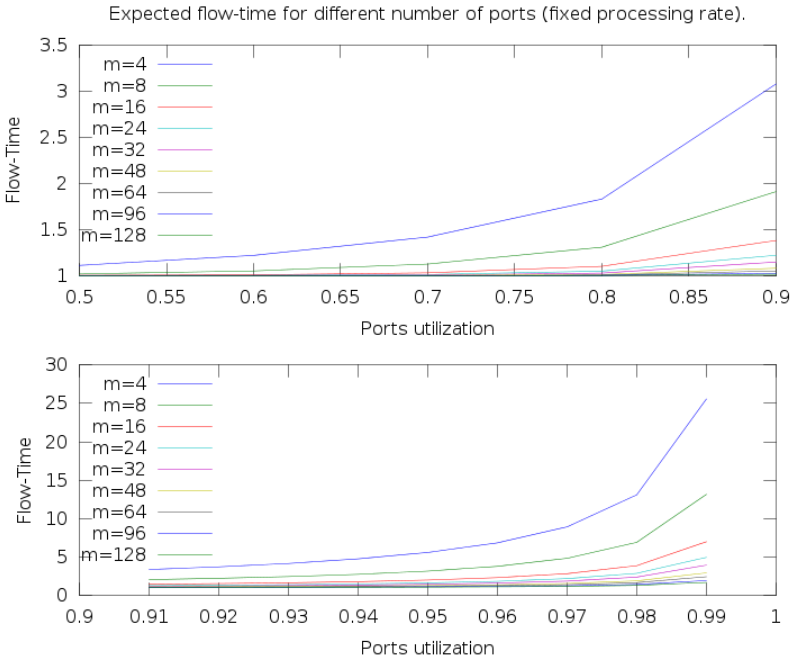

Fig. 8 Expected flow-time for different number of ports (fixed processing rate).

Figure 7 presents system efficiency (flow-time processing) for different number of ports considering fixed arrival rate, $\lambda = 1$ and $\mu = (m\rho)^{-1}$. As we can see, the flow time increases when average utilization (ρ) increases and we can assume that the system must be calibrated for a higher value of ρ, for example $\rho = 0.90 \ldots 0.99$. Here, the blue line, for $m = 4$, has the the lower values, which means that for a fixed arrival rate for flows in the considered systems, the number of ports must be maximal as possible.

Figure 8 presents system efficiency for different number of ports considering fixed processing rate, $\mu = 1$ and $\lambda = m\rho$. As we can see, the flow time increases when average utilization (ρ) increases, like in the previous results, and we have similar conclusion that the system efficiency reach the maximum for higher values of $\rho = 0.90 \ldots 0.99$. But, in this case, the blue line corresponding to $m = 4$, has the the higher values, which means that for a fixed processing rate for flows in the considered systems, the number of ports must be minimal as possible. Here we can consider a uniform processing system, which sustain our conclusion.

Figure 9 presents the dependencies of flows number for different ports utilization ratio starting with $\rho = 0.5$ to $\rho = 0.99$. We have here an upper bound imposed by the condition to have the maximum probability of arriving flows in the systems in a steady state. For this number of flows, in each case, the presented algorithms (Alg. 1, Alg. 2 and Alg. 4) offer the best results for ports allocation.

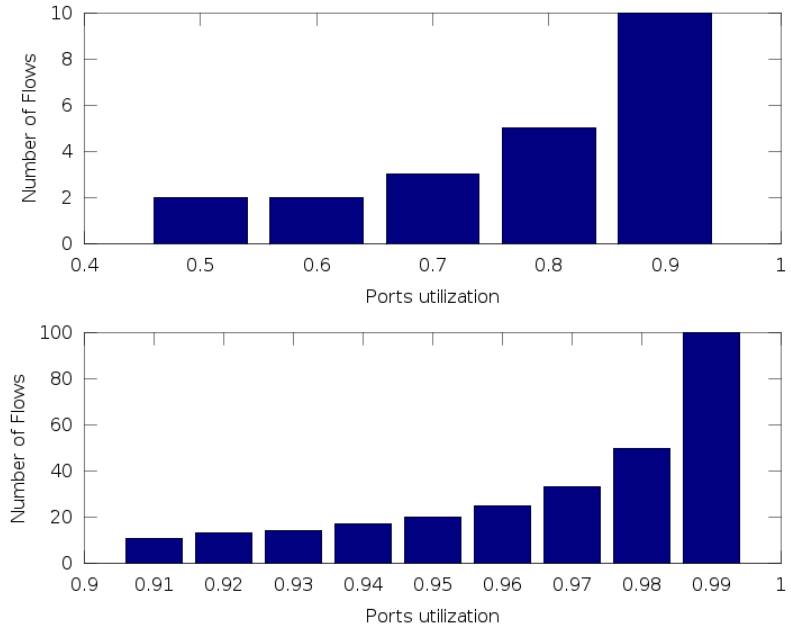

Fig. 9 Maximum Number of Flows.

Finally, Figure 10 presents the probability to have n flows in the considered system. For $n = 2$ flows the maximum probability is reached for ρ around 0.75. For a higher number of packages ρ must take a higher value, around 0.98, which means that for all considered results the value for ρ must be high, so we can set the $\rho_{optimal} = 0.98$ and $n_{optimal} = 50$.

6 Conclusions

We presented in this paper a new approach to control state management in SDN with a software-controlled scheduler for flow queuing over the data plane. Proposed SDN Scheduling algorithms consider minimum processing time for a flow as a metric for resource allocation. The proposed scheduling algorithms are: Fixed Priority Scheduling using Specific Selection Strategy, Balanced Scheduling and Adaptive Cost Scheduling. For these algorithms we proposed a Fluid Flow Markov Chain Model and a Queue State Estimation Model that offer a good estimation of optimal number of ports and of flow-time as efficiency measure for proposed system.

First, we develop a multi-policy scheduling strategy in order *to transfer* media-optimized traffic over SDNs. Our scheduling strategy is based on the adaptive internal mechanism, where flows are prioritized considering both the heterogeneity of network resources, and application flows. Second, we provide

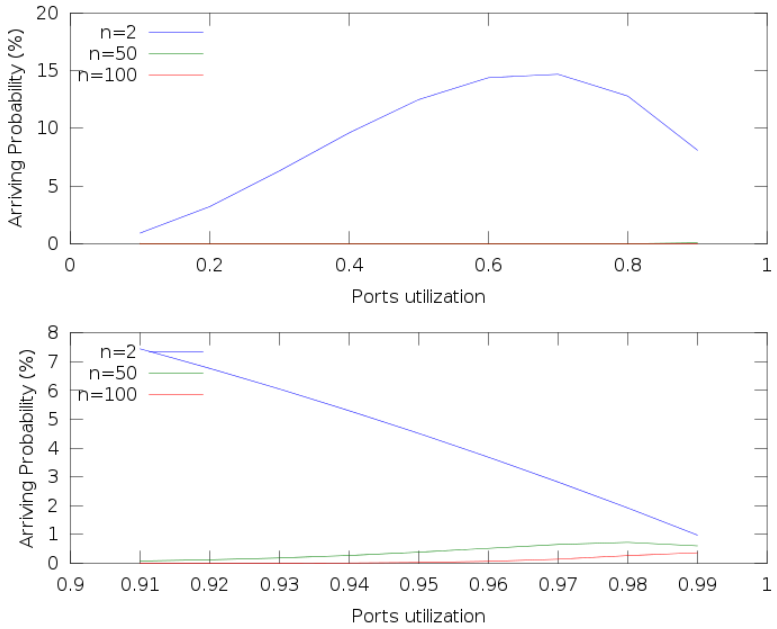

Fig. 10 Probability to have n flows in the considered system.

a comprehensive theoretical and empirical analysis of the effectiveness of the resource management in SDNs under different scheduling policies and conditions.

Acknowledgements The research presented in this paper is supported by the following projects: *ERRIC - Empowering Romanian Research on Intelligent Information Technologies*", FP7-REGPOT-2010-1, ID: 264207; "*SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", (PN-II-CT-RO-FR-2012-1-0084);"; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012.

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insights.

References

1. Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.
2. Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM Computer Communication Review*, 40(4):63–74, 2010.
3. Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal near-optimal datacenter transport, 2013.

4. Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.
5. Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association, 2005.
6. Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4):1–12, 2007.
7. Martin Casado, Tal Garfinkel, Aditya Akella, Michael J Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. SANE: A protection architecture for enterprise networks. In *USENIX Security Symposium*, 2006.
8. Ng Chee-Hock and S Boon-He. *Queueing modelling fundamentals*. Wiley & Son, Chichester, 2002.
9. Jon Crowcroft, Markus Fidler, Klara Nahrstedt, and Ralf Steinmetz. Is sdn the deconstraining constraint of the future internet? *ACM SIGCOMM Computer Communication Review*, 43(5):13–18, 2013.
10. Andrew Curtis, Jeffrey Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4):254–265, 2011.
11. Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review*, 19(4):1–12, 1989.
12. Wu-chang Feng, Kang G Shin, Dilip D Kandlur, and Debanjan Saha. The BLUE active queue management algorithms. *IEEE/ACM Transactions on Networking (TON)*, 10(4):513–528, 2002.
13. Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking: An api for application control of sdns. *SIGCOMM Comput. Commun. Rev.*, 43(4):327–338, August 2013.
14. Sally Floyd. TCP and explicit congestion notification. *ACM SIGCOMM Computer Communication Review*, 24(5):8–23, 1994.
15. Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.
16. Crispin W Gardiner et al. *Handbook of stochastic methods*, volume 3. Springer Berlin, 1985.
17. Albert Greenberg, Gisli Hjalmtysson, David A Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, 2005.
18. Albert Greenberg, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 57–62. ACM, 2008.
19. Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.
20. Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.
21. Brad Hedlund. Starting a new journey with dell force10. `http://bradhedlund.com/2011/10/05/starting-a-new-journey-with-dell-force10/`, 2011. [Accessed November 14th, 2013].
22. Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: Saving energy in data center networks. In *NSDI*, volume 3, pages 19–21, 2010.
23. Chi-Yao Hong, Matthew Caesar, and P Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.
24. Pan Hui, Teemu Koponen, Pan Hui, and Teemu Koponen. Software defined networking (dagstuhl seminar 12363). *Dagstuhl Reports*, 2(9):95–108, 2012.

25. FK Hwang. Rearrangeability of multi-connection three-stage clos networks. *Networks*, 2(4):301–306, 1972.
26. Hesam Izakian, Ajith Abraham, and Václav Snášel. Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments. *Neural Network World*, 19(6):695–710, 2009.
27. Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM Computer Communication Review*, volume 18, pages 314–329. ACM, 1988.
28. Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 3–14. ACM, 2013.
29. Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth delay product networks. *ACM SIGCOMM Computer Communication Review*, 32(4):89–102, 2002.
30. Bo-Yu Ke, Po-Lung Tien, and Yu-Lin Hsiao. Parallel prioritized flow scheduling for software defined data center network. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, pages 217–218, 2013.
31. Paul E McKenney. Stochastic fairness queueing. In *INFOCOM'90. Ninth Annual Joint Conference of the IEEE Computer and Communication Societies.'The Multiple Facets of Integration'. Proceedings., IEEE*, pages 733–740. IEEE, 1990.
32. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
33. Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.
34. Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 13–18. ACM, 2012.
35. Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J. Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 151–162, New York, NY, USA, 2013. ACM.
36. Cristina Serbanescu. Noncommutative markov processes as stochastic equations' solutions. *Bull. Math. Soc. Sc. Math. Roumanie Tome 41*, 89(3):219–228, 1998.
37. Cristina Serbanescu. Stochastic differential equations and unitary processes. *Bull. Math. Soc. Sc. Math. Roumanie Tome 41*, 89(3):311–322, 1998.
38. Anirudh Sivaraman, Keith Winstein, Suvinay Subramanian, and Hari Balakrishnan. No Silver Bullet: Extending SDN to the Data Plane. In *Twelfth ACM Workshop on Hot Topics in Networks (HotNets-XII)*, College Park, MD, November 2013.
39. Kun Tan Jingmin Song, Q Zhang, and M Sridharan. Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. *Proceedings of PFLDnet 2006*, 2006.
40. Cisco Systems. Cisco's Massively Scalable Data Center. `http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/MSDC/1.0/MSDC_AAG_1.pdf`, 2013. [Accessed November 14th, 2013].
41. Chia-Hui Tai, Jiang Zhu, and Nandita Dukkipati. Making large scale deployment of RCP practical for real networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 2180–2188. IEEE, 2008.