SP-PIFO: Approximating Push-In First-Out Behaviors Using Strict-Priority Queues



Albert Gran Alcoz,

Alexander Dietmüller, Laurent Vanbever

sp-pifo.ethz.ch

Barefoot Networks February, 28 2020

EHzürich

a 144179.144199.pdf (page 1 of 13) ~

• • •

🗶 🗸 📩 🙆 Q Search

Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism

David D. Clark¹ Laboratory for Computer Science Massachusetts Institute of Technology ddc@lcs.mit.edu

Abstract

This paper considers the support of real-time applications in an Integrated Services Packet Network (ISPN). We first review the characteristics of real-time applications. We observe that, contrary to the popular view that real-time ap-plications necessarily require a fixed delay bound, some real-time applications are more flexible and can adapt to current network conditions. We then propose an ISPN architec-ture that supports two distinct kinds of real-time service: guaranteed service, which is the traditional form of real-time service discussed in most of the literature and involves -computed worst-case delay bounds, and predicted service which uses the measured performance of the network in com-puting delay bounds. We then propose a packet scheduling mechanism that can support both of these real-time services as well as accommodate datagram traffic. We also discuss two other aspects of an overall ISPN architecture: the ser-vice interface and the admission control criteria.

1 Introduction

The current generation of telephone networks and the current generation of computer networks were each designed to carry specific and very different kinds of traffic: analog voice and digital data. However, with the digitizing of telephony in ISDN and the increasing use of multi-media in computer applications, this distinction is rapidly disappearing. Merg-ing these sorts of services into a single network, which we re-fer to here as an Integrated Services Packet Network (ISPN), would yield a single telecommunications infrastructure offer would yield a single electromannications infrastructure buer-ing a multitude of advantages, including vast economies of scale, ubiquity of access, and improved statistical multiplex-ing. There is a broad consensus, at least in the computer networking community, that an ISPN is both a worthy and an achievable goal. However, there are many political, ad-ministrative, and technical hurdles to overcome before this vision can become a reality. vision can become a reality

¹Research at MIT was supported by DARPA through NASA Grant NAG 2-582, by NSF grant NCR-8814187, and by DARPA and NSF through Cooperative Agreement NCR-8919038 with the Corporation through Cooperative Agre-for National Research Init

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for ect commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific pern COMM'92-8/92/MD,USA

* 1992 ACM 0-89791-526-7/92/0008/0014...\$1.50

14

Scott Shenker Lixia Zhang Palo Alto Research Center Xerox Corporation shenker, lixia@parc.xerox.com One of the most vexing technical problems that blocks the path towards an ISPN is that of supporting real-time applications in a packet network. Real-time applications are quite different from standard data applications, and rervice that cannot be delivered within the typical data gene service that cannot be derivered within the typical data service architecture. In Section 2 we discuss the nature of real-time applications at length; here, however, it suffices to observe that one salient characteristic of the real-time applications we consider is that they require a bound on the delivery delay of each packet². While this bound may be statistical, in the sense that some small fraction of the packets may fail to arrive by this bound, the bound itsel must be known a priori. The traditional data service archi tecture underlying computer networks has no facilities for prescheduling resources or denying service upon overload, and thus is unable to meet this real-time requirement. Therefore, in order to handle real-time traffic, an enhanced architecture is needed for an ISPN. We identify fou

key components to this architecture. The first piece of the architecture is the nature of the commitments made by the etwork when it promises to deliver a certain quality of ser vice. We identify two sorts of commitments, guaranteed and predicted. Predicted service is a major aspect of our paper. While the idea of predicted service has been considered be fore, the issues that surround it have not, to our knowledge been carefully explored. The second piece of the architecture is the service inter-

face, i.e., the set of parameters passed between the source and the network. The service interface must include both and the network. The service interface must include both the characterization of the quality of service the network will deliver, fulfilling the need of applications to know when their packets will arrive, and the characterization of the source's traffic, thereby allowing the network to knowledgeably al-locate resources. In this paper we attempt to identify the critical aspects of the service interface, and offer a particular interface as an example. We address in passing the need for enforcement of these characterizations

enforcement of these characterizations. The third piece of the architecture is the packet scheduling behavior of network switches needed to meet these ser-vice commitments. We discuss both the actual scheduling algorithms to be used in the switches, as well as the schedul-ing information that must be carried in packet headers. This

²Since the term bound is tossed around with great abandon in the rest of the paper, we need to identify several different meanings to the term. An a priori bound on delay is a statement that none of the future delays will exceed that amount. A post facto bound is the maximal value of a set of observed delays. Statistical bounds allow for a certain percentage of violations of the bound; absolute bounds allow none.

SIGCOMM'13, August 12-16, 2 Copyright 2013 ACM 978-1-450 Minimize flow

[SIGCOMM '13

Minimize tail packet delays [SIGCOMM '92]

functions, ma

to use in prac

with no prior hardware. To this en

ing mechanis

ing Shortest .

size is not k

ages multiple

modity swite

back Queue (

ly demoted fr

queues based

result, short flo

high-priority

flows in gene

without know

We have i

uated PIAS (

2 simulation

able with co

ible with le

sults show th

information-

FCT by up

L2DCT [27]

mance gap to

ric [13], for sl

1 Introdu

There has be

community (

center netwo

the flow com

¹PIAS, Practi introduced in an o

inary design and

load.



a nsdi15-paper



pFabric:

Moh

{alizade, shyang

ABSTRACT

In this paper we present pFa port design that provides ne tion times even at the 99th minimizing average flow co over, pFabric delivers this p that is based on a key concep decouple flow scheduling f packets carry a single prior flow; switches have very sn ple priority-based scheduling also correspondingly simple back only under high and pe oretical intuition and show bination of these two simple near-optimal performance

Categories and Subject Descr Networks]: Network Architectu General Terms: Design, Perfor Keywords: Datacenter network

1. INTRODUCTIO

Datacenter workloads imp on the transport fabric. Inte as the ones seen in search, s large number of small reque ter that are stitched togethe tation (e.g., delivering search low latency for each of the sh perceived performance is di-(or a large fraction of) the req to the user. However in c the latency for these short flo (FCT) can be as high as tens flows could complete in 10-these flows often get queued flows of co-existing worklo mining, etc) which significan

Permission to make digital or personal or classroom use is gran made or distributed for profit or of this notice and the full citation of of this work owned by others that credit is permitted. To copy others that redistribute to lists, requires prio permissions from permissions@

-bai.pdf (page 2 of 15)	λ Search	
		344
		AG
eduling for Commodity I	Jata Centers	to
sigcomm13-pfab	ic.pdf (page 1 of 12)	
	V C Search	
		Abstract—" users of an i context of rat
		a virtual circu the use of Ger with Leaky 1 make a wide
Minimal Near-Opt	imal Datacenter Transport	throughput a users may be and is efficien
		We present a (first propose name of Wei
ammad Alizadeh ^{†‡} , Shuang Nick McKeown [†] , Balaji Pra	Yang ¹ , Milad Sharif ¹ , Sachin Katti ¹ , bhakar ¹ , and Scott Shenker ¹	GPS. This all packet schem In this pape
[†] Stanford University [‡] Insieme	Networks [§] U.C. Berkeley / ICSI ii)@stanford.edu, shankar@icsi.berkeley.edu	analyzed exac and burstines The worst-cas to this paper.
, monam, oram, morin, baie	ingestaniora.eau sheriker@icsi.berkerey.eau	networks with
	Motivated by this observation, recent research has proposed new datacenter transport designs that, broadly speaking, use rate con-	This paper in the contro
abric, a minimalistic datacenter trans- ar theoretically optimal flow comple- percentile for short flows, while still	trol to reduce FCT for short flows. One line of work [3, 4] im- proves FCT by keeping queues near empty through a variety of	networks. Tr been traded
mpletion time for long flows. More- erformance with a very simple design	mechanisms (adaptive congestion control, ECN-based feedback, pacing, etc) so that latency-sensitive flows see small buffers and consequently small latencies. These <i>implicit</i> techniques generally	performance switched netv
om rate control. For flow scheduling, ity number set independently by each	improve FCT for short flows but they can never precisely determine the right flow rates to optimally schedule flows. A second line of	performance carry a wide
all buffers and implement a very sim- t/dropping mechanism. Rate control is	work [21, 14] explicitly computes and assigns rates from the net- work to each flow in order to schedule the flows based on their sizes or deadlines. This approach can potentially provide very good per-	and video. W apparently co
r; nows start at line rate and throttle ersistent packet loss. We provide the- ia extensive simulations that the com-	formance, but it is rather complex and challenging to implement in practice because accurately computing rates requires detailed flow	for link usag We propos based on Ge
e mechanisms is sufficient to provide	the bottleneck for each flow and avoid under-utilization (§2). Our goal in this paper is to design the simplest possible datacen-	Manuscript re by IEEE/ACM 1
iptors: C.2.1 [Computer-Communication re and Design mance	ter transport scheme that provides near-optimal flow completion times, even at the 99 th percentile for latency-sensitive short flows.	was presented i was partly funde Control Systems National Science
racket transport, Flow scheduling	whose entire design consists of the following:	Office under D/ A. K. Parekh Heights, NY 10 R. G. Collar
ose unique and stringent requirements aractive soft real-time workloads such	that encodes its priority (e.g., the flow's remaining size, dead- line). The priority is set independently by each flow and no	Systems, Massa IEEE Log Nu
ocial networking, and retail generate a sts and responses across the datacen-	coordination is required across flows or hosts to compute it. • Switches are simple; they have very small buffers (e.g., 36KB per port in our evaluation) and decide which packets to ac-	
h results). These applications demand nort request/response flows, since user-	cept into the buffer and which ones to schedule strictly ac- cording to the packet's priority number. When a new packet	· · · · ·
ctated by how quickly responses to all quests are collected and delivered back	arrives and the buffer is full, if the incoming packet has lower priority than all buffered packets, it is dropped. Else, the low- est priority packet in the buffer is dropped and replaced with	
ows is poor — flow completion times of milliseconds while in theory these	the incoming packet. When transmitting, the switch sends the packet with the highest priority. Thus each switch oper- otes independently in a grand and local fachion	
20 microseconds. The reason is that up behind bursts of packets from large ads (such as backup, replication, data	 Rate control is minimal; all flows start at line-rate and throttle their sending rate only if they see high and persistent loss. 	
atly increases their completion times.	pFabric thus requires no flow state or complex rate calculations at	
and copies of all or part of this work for ited without fee provided that copies are not commercial advantage and that copies bear	the switches, no large switch bullers, no explicit network feedback, and no sophisticated congestion control mechanisms at the end- host. pFabric is a clean-slate design; it requires modifications both	
n the first page. Copyrights for components in ACM must be honored. Abstracting with rwise, or republish, to post on servers or to	at the switches and the end-hosts. We also present a preliminary de- sign for deploying pFabric using existing switches, but a full design for incremental deployment is beinged the scene of this energy	
r specific permission and/or a fee. Request acm.org. 2013, Hong Kong, China.	¹ pFabric was first introduced in an earlier paper [5] which sketched	
J3-2030-0(15/08\$13.00.	a promining) acoder and mining summarion results.	
		- c
/ compl	etion times	Ente
•		
	151	
וטכאו , נ		

a 00234856.pdf (page 1 of 14) 🗶 👻 🛅 🛞 🔍 Q Search IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 1, NO. 3, JUNE 1993 Generalized Processor Sharing Approach Flow Control in Integrated Services Networks: The Single-Node Case nsdi18-sharma.pdf (page 2 of 17) 🗶 🗸 📩 🙆 Q Search Approximating Fair Queueing on Reconfigurable Switches Naveen Kr. Sharma^{*} Ming Liu^{*} Kishore Atreya[†] Arvind Krishnamurthy^{*} of the bandwidth under all circumstances, regardless of Abstract other network traffic. Having the network enforce fair Congestion control today is predominantly achieved via bandwidth allocation offers several benefits. It simplifies end-to-end mechanisms with little support from the netcongestion control at the end-hosts, removing the need work. As a result, end-hosts must cooperate to achieve to perform slow-start or complex congestion avoidance optimal throughput and fairness, leading to inefficiencies strategies. Further, flows can ramp up quickly without and poor performance isolation. While router mechaaffecting other network traffic. It also provides strong nisms such as Fair Queuing guarantee fair bandwidth alisolation among competing flows, protects well-behaved location to all participants and have proven to be optimal flows from ill-behaving traffic, and enables bounded dein some respects, they require complex flow classificalay guarantees [34]. tion, buffer allocation, and scheduling on a per-packet A fair bandwidth allocation scheme is potentially well basis. These factors make them expensive to implement suited to today's datacenter environment, where mulin high-speed switches. tiple applications with diverse network demands often In this paper, we use emerging reconfigurable switches co-exist. Some applications require low latency, while to develop an approximate form of Fair Oueueing that others need sustained throughput. Datacenter networks operates at line-rate. We leverage configurable permust also contend with challenging traffic patterns - such packet processing and the ability to maintain mutable as large incasts or fan-in, micro-bursts, and synchronized state inside switches to achieve fair bandwidth allocaflows, - which can all be managed effectively using a fair tion across all traversing flows. Further, present our queueing mechanism. Fair queueing mechanisms can design for a new dequeuing scheduler, called Rotating also provide bandwidth guarantees for multiple tenants Strict Priority scheduler that lets us transmit packets from of a shared cloud infrastructure [35]. multiple queues in approximate sorted order. Our hard-Over the years, several algorithms for enforcing fair ware emulation and software simulations on a large leafbandwidth allocation have been proposed [25, 27, 28 spine topology show that our scheme closely approxi-33], but rarely deployed in practice, primarily due to their mates ideal Fair Queueing, improving the average flow inherent complexities. These algorithms maintain state completion times for short flows by 2-4x and 99th tail and perform operations on a per-flow basis, making them latency by 4-8x relative to TCP and DCTCP. challenging to implement at data rates of 3-6 Tbps in 1 Introduction hardware. However, recent advances in switching hardware allow flexible per-packet processing and the abil-Most current congestion control schemes rely on endity to maintain limited mutable state at switches without to-end mechanisms with little support from the netsacrificing performance [12, 6]. In this paper, we explore work (e.g., ECN, RED). While this approach simplifies whether an efficient fair queueing implementation can be switches and lets them operate at very high speeds, it realized using these emerging reconfigurable switches. requires end-hosts to cooperate to achieve fair network We present Approximate Fair Queueing (AFQ), a fair sharing, thereby leading to inefficiencies and poor perbandwidth allocation mechanism that approximates the formance isolation. On the other hand, if the switches various components of an ideal fair queueing scheme were capable of maintaining per-flow state, extracting using features available in emerging programmable rich telemetry from the network, and performing conswitches, such as the ability to maintain and mutate figurable per-packet processing, one can realize intelliswitch state on a per-packet basis, perform limited comgent congestion control mechanisms that take advantage putation for each packet, and dynamically determine of dynamic network state directly inside the network and which egress queue to use for a given packet. We deimprove network performance scribe a variant of the packet-pair flow control proto-One such mechanism is Fair Queueing, which has col [24], designed to work with AFQ, that achieves close been studied extensively and shown to be optimal in sevto optimal performance while maintaining short queues. eral aspects. It provides the illusion that every flow (or We further prototype an AFQ implementation on a Cavparticipant) has its own queue and receives a fair share ium networking processor and study its feasibility on upcoming reconfigurable switches. Using a real hard-*University of Washington ware testbed and large-scale simulations, we demon-[†]Cavium Inc.

orce max-min fairness N '93, NSDI '18]



. a ups.pdf (page 1 of 21) 🗶 💌 📩 🛞 Q Search Universal Packet Scheduling Sylvia Ratnasamy[†] Scott Shenker^{†‡} Radhika Mittal † Rachit Agarwal[†] [†]UC Berkeley [‡]ICSI Abstract We can define a universal packet scheduling algorithm (hereafter UPS) in two ways, depending on our viewpoint In this paper we address a seemingly simple question: on the problem. From a theoretical perspective, we call a Is there a universal packet scheduling algorithm? More packet scheduling algorithm universal if it can replay any precisely, we analyze (both theoretically and empirically) schedule (the set of times at which packets arrive to and whether there is a single packet scheduling algorithm that, exit from the network) produced by any other scheduling at a network-wide level, can perfectly match the results of algorithm. This is not of practical interest, since such any given scheduling algorithm. We find that in general schedules are not typically known in advance, but it offers the answer is "no". However, we show theoretically that a theoretically rigorous definition of universality that (as the classical Least Slack Time First (LSTF) scheduling alwe shall see) helps illuminate its fundamental limits (i.e., gorithm comes closest to being universal and demonstrate which scheduling algorithms have the flexibility to serve empirically that LSTF can closely replay a wide range of as a UPS, and why). scheduling algorithms in realistic network settings. We From a more practical perspective, we say a packet then evaluate whether LSTF can be used in practice to meet various network-wide objectives by looking at pop-

ular performance metrics (such as mean FCT, tail packet

delays, and fairness); we find that LSTF performs com-

parable to the state-of-the-art for each of them. We also

discuss how LSTF can be used in conjunction with ac-

tive queue management schemes (such as CoDel) without

There is a large and active research literature on novel packet scheduling algorithms, from simple schemes such

as priority scheduling [31], to more complicated mechanisms to achieve fairness [16, 29, 32], to schemes that

help reduce tail latency [15] or flow completion time [7],

and this short list barely scratches the surface of past and

current work. In this paper we do not add to this impres-

sive collection of algorithms, but instead ask if there is a single *universal* packet scheduling algorithm that could

obviate the need for new ones. In this context, we consider

a packet scheduling algorithm to be both how packets are

served inside the network (based on their time of arrival and their packet header) and how packet header fields are initialized at the edge of the network; this definition

includes all the classical scheduling algorithms (FIFO,

LIFO, priority, round-robin) as well as algorithms that

incorporate dynamic packet state [15, 35, 36].

changing the core of the network.

1 Introduction

scheduling algorithm is universal if it can achieve different desired performance objectives (such as fairness, reducing tail latency, minimizing flow completion times). In particular, we require that the UPS should match the performance of the best known scheduling algorithm for a given performance objective. ¹

The notion of universality for packet scheduling might seem esoteric, but we think it helps clarify some basic questions. If there exists no UPS then we should *expect* to design new scheduling algorithms as performance objectives evolve. Moreover, this would make a strong argument for switches being equipped with programmable packet schedulers so that such algorithms could be more easily deployed (as argued in [33]; in fact, it was the eloquent argument in this paper that caused us to initially ask the question about universality).

However, if there is indeed a UPS, then it changes the lens through which we view the design and evaluation of packet scheduling algorithms: e.g., rather than asking whether a new scheduling algorithm meets a performance objective, we should ask whether it is easier/cheaper to

Is there a universal packet scheduler? [NSDI '16]

3

¹For this definition of universality, we allow the header initialization to depend on the objective being optimized. That is, while the basic scheduling operations must remain constant, the header initialization can depend on whether you are seeking fairness or minimal flow completion time.

ups.pdf (page 1 of 21) 🗶 🗸 📩 🛞 Q Search Universal Packet Scheduling Radhika Mittal† Scott Shenker^{†‡} Rachit Agarwal[†] Sylvia Ratnasamy[†] [‡]ICSI [†]UC Berkeley

Abstract

• • •

In this paper we address a seemingly simple question: Is there a universal packet scheduling algorithm? More precisely, we analyze (both theoretically and empirically) whether there is a single packet scheduling algorithm that, at a network-wide level, can perfectly match the results of any given scheduling algorithm. We find that in general the answer is "no". However, we show theoretically that the classical Least Slack Time First (LSTF) scheduling algorithm comes closest to being universal and demonstrate empirically that LSTF can closely replay a wide range of scheduling algorithms in realistic network settings. We then evaluate whether LSTF can be used in practice to meet various network-wide objectives by looking at popular performance metrics (such as mean FCT, tail packet delays, and fairness); we find that LSTF performs comparable to the state-of-the-art for each of them. We also discuss how LSTF can be used in conjunction with active queue management schemes (such as CoDel) without changing the core of the network.

1 Introduction

There is a large and active research literature on novel packet scheduling algorithms, from simple schemes such as priority scheduling [31], to more complicated mechanisms to achieve fairness [16, 29, 32], to schemes that help reduce tail latency [15] or flow completion time [7], and this short list barely scratches the surface of past and current work. In this paper we do not add to this impressive collection of algorithms, but instead ask if there is a single universal packet scheduling algorithm that could obviate the need for new ones. In this context, we consider a packet scheduling algorithm to be both how packets are served inside the network (based on their time of arrival and their packet header) and how packet header fields are initialized at the edge of the network; this definition includes all the classical scheduling algorithms (FIFO, LIFO, priority, round-robin) as well as algorithms that incorporate dynamic packet state [15, 35, 36].

We can define a universal packet scheduling algorithm (hereafter UPS) in two ways, depending on our viewpoint on the problem. From a theoretical perspective, we call a packet scheduling algorithm universal if it can replay any schedule (the set of times at which packets arrive to and exit from the network) produced by any other scheduling algorithm. This is not of practical interest, since such schedules are not typically known in advance, but it offers a theoretically rigorous definition of universality that (as we shall see) helps illuminate its fundamental limits (i.e., which scheduling algorithms have the flexibility to serve as a UPS, and why).

From a more practical perspective, we say a packet scheduling algorithm is universal if it can achieve different desired performance objectives (such as fairness, reducing tail latency, minimizing flow completion times). In particular, we require that the UPS should match the performance of the best known scheduling algorithm for a given performance objective. 1

The notion of universality for packet scheduling might seem esoteric, but we think it helps clarify some basic questions. If there exists no UPS then we should expect to design new scheduling algorithms as performance objectives evolve. Moreover, this would make a strong argument for switches being equipped with programmable packet schedulers so that such algorithms could be more easily deployed (as argued in [33]; in fact, it was the eloquent argument in this paper that caused us to initially ask the question about universality).

However, if there is indeed a UPS, then it changes the lens through which we view the design and evaluation of packet scheduling algorithms: e.g., rather than asking whether a new scheduling algorithm meets a performance objective, we should ask whether it is easier/cheaper to

"You can't have *everything* you want, but you can have *anything* you want"

Is there a universal packet scheduler? [NSDI '16]

Generality

Universal packet scheduler

Flexibility

Customized algorithms



¹ For this definition of universality, we allow the header initialization to depend on the objective being optimized. That is, while the basic scheduling operations must remain constant, the header initialization can depend on whether you are seeking fairness or minimal flow completion

• • • ups.pdf (page 1 of 21) 🗶 🗸 📩 🛞 Q Search Universal Packet Scheduling Radhika Mittal† Scott Shenker^{†‡} Rachit Agarwal[†] Sylvia Ratnasamy[†] [‡]ICSI [†]UC Berkeley

Abstract

In this paper we address a seemingly simple question: Is there a universal packet scheduling algorithm? More precisely, we analyze (both theoretically and empirically) whether there is a single packet scheduling algorithm that, at a network-wide level, can perfectly match the results of any given scheduling algorithm. We find that in general the answer is "no". However, we show theoretically that the classical Least Slack Time First (LSTF) scheduling algorithm comes closest to being universal and demonstrate empirically that LSTF can closely replay a wide range of scheduling algorithms in realistic network settings. We then evaluate whether LSTF can be used in practice to meet various network-wide objectives by looking at popular performance metrics (such as mean FCT, tail packet delays, and fairness); we find that LSTF performs comparable to the state-of-the-art for each of them. We also discuss how LSTF can be used in conjunction with active queue management schemes (such as CoDel) without changing the core of the network.

1 Introduction

There is a large and active research literature on novel packet scheduling algorithms, from simple schemes such as priority scheduling [31], to more complicated mechanisms to achieve fairness [16, 29, 32], to schemes that help reduce tail latency [15] or flow completion time [7], and this short list barely scratches the surface of past and current work. In this paper we do not add to this impressive collection of algorithms, but instead ask if there is a single universal packet scheduling algorithm that could obviate the need for new ones. In this context, we consider a packet scheduling algorithm to be both how packets are served inside the network (based on their time of arrival and their packet header) and how packet header fields are initialized at the edge of the network; this definition includes all the classical scheduling algorithms (FIFO, LIFO, priority, round-robin) as well as algorithms that incorporate dynamic packet state [15, 35, 36].

We can define a universal packet scheduling algorithm (hereafter UPS) in two ways, depending on our viewpoint on the problem. From a theoretical perspective, we call a packet scheduling algorithm universal if it can replay any schedule (the set of times at which packets arrive to and exit from the network) produced by any other scheduling algorithm. This is not of practical interest, since such schedules are not typically known in advance, but it offers a theoretically rigorous definition of universality that (as we shall see) helps illuminate its fundamental limits (i.e., which scheduling algorithms have the flexibility to serve as a UPS, and why).

From a more practical perspective, we say a packet scheduling algorithm is universal if it can achieve different desired performance objectives (such as fairness, reducing tail latency, minimizing flow completion times). In particular, we require that the UPS should match the performance of the best known scheduling algorithm for a given performance objective. 1

The notion of universality for packet scheduling might seem esoteric, but we think it helps clarify some basic questions. If there exists no UPS then we should expect to design new scheduling algorithms as performance objectives evolve. Moreover, this would make a strong argument for switches being equipped with programmable packet schedulers so that such algorithms could be more easily deployed (as argued in [33]; in fact, it was the eloquent argument in this paper that caused us to initially ask the question about universality).

However, if there is indeed a UPS, then it changes the lens through which we view the design and evaluation of packet scheduling algorithms: e.g., rather than asking whether a new scheduling algorithm meets a performance objective, we should ask whether it is easier/cheaper to

"You can't have *everything* you want, but you can have *anything* you want"

Is there a universal packet scheduler? [NSDI '16]

Generality

Universal packet scheduler

Programmable scheduling



¹ For this definition of universality, we allow the header initialization to depend on the objective being optimized. That is, while the basic scheduling operations must remain constant, the header initialization can depend on whether you are seeking fairness or minimal flow completion

1602.06045.pdf (page 1 of 14) 🗶 🗸 📩 🛞 Q Search

Programmable Packet Scheduling

Anirudh Sivaraman^{*}, Suvinay Subramanian^{*}, Anurag Agrawal[†], Sharad Chole[‡], Shang-Tse Chuang[‡], Tom Edsall[‡], Mohammad Alizadeh*, Sachin Katti+, Nick McKeown+, Hari Balakrishnan* *MIT CSAIL, [†]Barefoot Networks, [‡]Cisco Systems, ⁺Stanford University

ABSTRACT

• • •

Z

S

ý.

0604

2

60

 $\overline{}$

Xiv

9 Switches today provide a small set of scheduling algorithms. 201 While we can tweak scheduling parameters, we cannot modify algorithmic logic, or add a completely new algorithm, after the switch has been designed. This paper presents a þ design for a programmable packet scheduler, which allows scheduling algorithms-potentially algorithms that are unknown today-to be programmed into a switch without re-6 quiring hardware redesign.

Our design builds on the observation that scheduling algorithms make two decisions: in what order to schedule packets and when to schedule them. Further, in many scheduling algorithms these decisions can be made when packets are enqueued. We leverage this observation to build a programmable scheduler using a single abstraction: the push-in first-out queue (PIFO), a priority queue that maintains the scheduling order and time for such algorithms.

We show that a programmable scheduler using PIFOs lets us program a wide variety of scheduling algorithms. We present a detailed hardware design for this scheduler for a 64-port 10 Gbit/s shared-memory switch with <4% chip area overhead on a 16-nm standard-cell library. Our design lets us program many sophisticated algorithms, such as a 5-level hierarchical scheduler with programmable scheduling algorithms at each level.

1. INTRODUCTION

Today's line-rate switches provide a menu of scheduling algorithms: typically, a combination of Deficit Round Robin [34], strict priority scheduling, and traffic shaping. A network operator can configure parameters in these algorithms. However, an operator cannot change the core algorithmic logic in an existing scheduling algorithm, or program a new one, without building new switch hardware.

By contrast, with a programmable packet scheduler, network operators would be able to deploy custom scheduling algorithms to better meet application requirements, e.g., minimizing flow completion times [9] using Shortest Remaining Processing Time [33], flexible bandwidth allocation across flows or tenants [31, 26] using Weighted Fair Queueing [17], or minimizing tail packet delays [16] using Least Slack Time First [28]. With a programmable packet sched-

uler, switch designers would implement scheduling algorithms as programs atop a programmable substrate. Moving scheduling algorithms into software makes it much easier to build and verify algorithms in comparison to implementing the same algorithms as rigid hardware IP.

This paper presents a design for programmable packet scheduling in line-rate switches. Our design is motivated by the observation that all scheduling algorithms make two key decisions: first, in what order should packets be scheduled, and second, at what time should each packet be scheduled. Furthermore, in many scheduling algorithms, these two decisions can be made when a packet is enqueued. This observation was first made in a recent position paper [36]. The same paper also proposed the push-in first-out queue (PIFO) [15] abstraction for maintaining the scheduling order or scheduling time for packets, when these can be determined on enqueue. A PIFO is a priority queue data structure that allows elements to be pushed into an arbitrary location based on an element's rank, but always dequeues elements from the

Building on the PIFO abstraction, this paper presents the detailed design, implementation, and analysis of feasibility of a programmable packet scheduler. To program a PIFO, we develop the notion of a scheduling transactiona small program to compute an element's rank in a PIFO. We present a rich programming model built using PIFOs and scheduling transactions (§2) and show how to program a diverse set of scheduling algorithms in the model (§3): Weighted Fair Queueing [17], Token Bucket Filtering [7], Hierarchical Packet Fair Queueing [10], Class-Based Queueing [19, 20], Least-Slack Time-First [28], Stopand-Go Queueing [22], the Rate-Controlled Service Disciplines [40], and fine-grained priority scheduling (e.g., Shortest Job First, Shortest Remaining Processing Time, Least Attained Service, and Earliest Deadline First).

Until now, all line-rate implementations of these scheduling algorithms-if they exist at all-have been hard-wired into switch hardware. We also describe the limits of the PIFO abstraction (§3.5) by presenting examples of scheduling algorithms that can't be programmed using a PIFO.

We present a detailed hardware design for a programmable scheduler using PIFOs (§4). We have imple-

De

Sca

PIFO abstraction for programmable scheduling [SIGCOMM '16]

Implementing PIFO queues in hardware is difficult

ployability	Requires new ASIC implementation, which might take years
alability	Supports ~1k flows and ~10 Gbps

Flexibility Assumes monotonic increase of ranks within flows

Can we approximate PIFO queues...

at line rate,

at scale, and

on existing devices?

SP-PIFO

PIFO approximation

Introducing...

A deployable, scalable and flexible

The PIFO queue

Allows packets to be pushed into arbitrary locations

Only drains packets from the head

Packet ranks

The PIFO queue

Allows packets to be pushed into arbitrary locations

Only drains packets from the head

Input sequence



PIFO Queue

Packet ranks





The PIFO queue

Allows packets to be pushed into arbitrary locations

Only drains packets from the head



Packet ranks

The PIFO queue

Allows packets to be pushed into arbitrary locations

Only drains packets from the head

Input sequence



Packet ranks



The PIFO queue

Allows packets to be pushed into arbitrary locations

Only drains packets from the head

Input sequence

PIFO Queue

Packet ranks





Programmable scheduler

Rank computation (programmable) PIFO queue (fixed logic)

Input sequence



Rank Computation

Programmable scheduler

Rank computation (programmable) PIFO queue (fixed logic)

Input sequence



Programmable scheduler Rank computation (programmable) PIFO queue (fixed logic)

Input sequence



Programmable scheduler Rank computation (programmable) Adaptation strategy + strict-priority queues (fixed logic)



Ideal case Perfect PIFO if number of queues >= number of ranks



Strict-Priority Queues

Ideal case Perfect PIFO if number of queues >= number of ranks



Strict-Priority Queues

In practice Number of queues < number of ranks

In practiceNumber of queues < number of ranks</th>ProblemOutput sequence can have scheduling errors

In practiceNumber of queues < number of ranks</th>ProblemOutput sequence can have scheduling errors



In practice Number of queues < number of ranks Problem Output sequence can have scheduling errors

Low-ranked packets drained after high-ranked packets

- In practice Number of queues < number of ranks
- Problem Output sequence can have scheduling errors
- Design mapping strategies that **Opportunity** minimize scheduling errors

MappingQueue bounds scanned bottom-up

Packet enqueued if rank >= queue bound

Queue bounds scanned bottom-up Mapping Packet enqueued if rank >= queue bound

Mapping Queue bounds scanned bottom-up Packet enqueued if rank >= queue bound

Queue bounds scanned bottom-up Mapping Packet enqueued if rank >= queue bound

Is rank >= queue bound ?

Mapping

Queue bounds scanned bottom-up Packet enqueued if rank >= queue bound

Mapping

Queue bounds scanned bottom-up Packet enqueued if rank >= queue bound

Input sequence

Mapping

Queue bounds scanned bottom-up Packet enqueued if rank >= queue bound

Mapping

Queue bounds scanned bottom-up Packet enqueued if rank >= queue bound

How can we design a mapping strategy that minimizes scheduling errors?

SP-PIFO: Approximating Push-In First-Out Behaviors Using Strict-Priority Queues

- Adaptation design How does it work
- Implementation 2

1

3 Evaluation

How can it be deployed

How well does it perform

SP-PIFO: Approximating Push-In First-Out Behaviors Using Strict-Priority Queues

Adaptation design How does it work

1

- Implementation 2
- 3 Evaluation

How can it be deployed

How well does it perform

Problem formulation

ObjectiveFind optimal queue bounds *q**That minimize the expected loss U for all ranks

$$\boldsymbol{q^*} = \underset{q \in Q}{\operatorname{argmin}} \underset{r \sim R}{E} \left[\begin{array}{c} U(\boldsymbol{q}, r) \end{array} \right]$$

Unpifoness (U) quantifies the scheduling errors

Initialization

Zero traffic knowledge Queue bounds set to zero

Initialization

Zero traffic knowledge Queue bounds set to zero

Initialization

Zero traffic knowledge Queue bounds set to zero

Initialization

Zero traffic knowledge Queue bounds set to zero

Initialization

Zero traffic knowledge Queue bounds set to zero

Output sequence

scheduling error

Initialization

Zero traffic knowledge Queue bounds set to zero

Dutput sequence

45

46

Push-up

(future low-rank packets to higher-priority queues)

After enqueue, queue bound set to the rank of the packet enqueued

Push-up

(future low-rank packets to higher-priority queues)

After enqueue, queue bound set to the rank of the packet enqueued

Strict-Priority Queues

Push-up

(future low-rank packets to higher-priority queues)

Input sequence

Strict-Priority Queues

After enqueue, queue bound set to the rank of the packet enqueued

Push-down

(future high-rank packets to lower-priority queues)

After potential error detected, all queue bounds decreased the error cost

- **Objective** Find optimal queue bounds q* That minimize the expected loss U for all ranks
- Result Packet-level adaptation of *q*

SP-PIFO: Approximating Push-In First-Out Behaviors Using Strict-Priority Queues

- Adaptation design How does it work
- 2 Implementation

1

- 3 Evaluation

How can it be deployed

How well does it perform

SP-PIFO has been fully implemented on Barefoot Tofino

SP-PIFO: Approximating Push-In First-Out Behaviors Using Strict-Priority Queues

- Adaptation design How does it work
- Implementation 2

1

3 Evaluation

How can it be deployed

How well does it perform

Evaluation

Question

How well does SP-PIFO approximate well-known scheduling objectives under realistic traffic workloads?

Scheduling objectives

Minimizing Flow Completion Time

pFabric* (8 queues)

Ranks are set to the remaining flow size

Max-min fairness

Start-Time Fair Queuing (32 queues)

Ranks based on a fluid model

* without starvation prevention

Methodology

Packet-level simulator

We integrated SP-PIFO in Netbench [SIGCOMM 2017]

Topology

We use a leaf-spine topology with 144 servers, links of 1Gbps and 4Gbps

Realistic workloads We generate traffic following pFabric web-search workload

SP-PIFO closely approximates pFabric, minimizing FCTs for both small and big flows

99th percentile FCT (ms)

Small flows <100KB

Big flows ≥1MB

SP-PIFO closely approximates state-of-the-art fair-queuing algorithms

Average FCT (ms)

Small flows <100KB

All flows @ Load 0.7

Check our website! sp-pifo.ethz.ch

SP-PIFO characterization, comparison with gradient

Hardware evaluation on Barefoot Tofino

Limitations and future improvements

All the code is available All our experiments are reproducible

SP-PIFO: Making scheduling programmable, today!

SP-PIFO approximates the behavior of PIFO queues at line rate, at scale and on existing devices

It adapts the mapping between packet ranks and strict-priority queues to minimize the scheduling errors

It reacts per-packet to traffic variations, without traffic knowledge required

SP-PIFO: Approximating Push-In First-Out Behaviors Using Strict-Priority Queues

Albert Gran Alcoz

Alexander Dietmüller

sp-pifo.ethz.ch

Laurent Vanbever

Networked Systems ETH Zürich — seit 2015

