ST. PETERSBURG DEPARTMENT OF V.A.STEKLOV INSTITUTE OF MATHEMATICS OF THE RUSSIAN ACADEMY OF SCIENCES

as a manuscript

Pavel Sergeyevich Chuprikov

THEORETICAL AND EMPIRICAL ANALYSIS OF FUNDAMENTAL BOTTLENECKS IN NETWORKING AND DISTRIBUTED COMPUTING

PhD Dissertation Summary

for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science HSE

Saint Petersburg — 2019

The PhD dissertation was prepared at St. Petersburg Department of V.A.Steklov Institute of Mathematics of the Russian Academy of Sciences.

Academic Supervisors:

- Sergey Igorevich Nikolenko, PhD, PDMI RAS, Researcher at Laboratory of Mathematical Logic
- Kirill Kogan, PhD, IMDEA Networks Institute, Research Assistant Professor

1 Introduction

This section begins with a brief introduction into the current limitations of big data processing in relation to computer networks and distributed data processing, which are the primary topics of analysis in this dissertation, and then states the main goals and objectives of the research.

1.1 Dissertation topic and its relevance

Over the last decades, the volume and variety of data which is necessary to transmit and process has been increasing exponentially. Between 2013 and 2020, the global data volume is predicted to grow exponentially, from 4.4 to 44 zettabytes [1]. The expression "big data" usually refers to datasets whose size is beyond the ability of typical computing platforms to store and analyze. Due to these storage and computational limitations, data chunks should be *distributed* over multiple locations for big data processing. In this case, a network *interconnects* distributed instances of big data applications.

Big data both aggravates classical challenges of networking and presents new difficulties. For example, as one consequence of data growth, traditional cloud computing is increasingly replacing dedicated hardware by dynamically allocated virtual computing resources. They are usually paid based on allocation rather than actual use, which often leads to customers paying more than necessary. Recently, *serverless* architectures have exploded in popularity, and many major players in the cloud computing market introduced serverless solutions: AWS Lambda [2], Google Cloud Functions [3], Azure Functions [4], and others.

Although serverless computing is only a special case of cloud computing and is not able to cover every desired functionality, it remains an important representative case providing cost savings to the end user. In particular, the "function as a service" paradigm brings new challenges to resource allocation. Recent works have only begun to pose these questions for serverless computing, studying it in the context of queueing theory [5] and measuring relevant metrics for already existing solutions [6]. The present thesis takes the next step towards a theory of serverless cloud computing, proposing a comprehensive and novel formalization of the resource allocation model for serverless computing and presenting new resource allocation algorithms supported by both theoretical analysis and a comprehensive evaluation study.

In addition to requiring more flexible resource allocation methods, exascale data amounts and delay-sensitive objectives of big data applications impose significant feasibility constraints on the cloud computing paradigm. One particular limitation is in deploying a network solely as a simple interconnect. Although traditional networks mostly operate on partial information from data streams (packet headers), in some cases network elements are already able to process entire data streams at line-rate (e.g., in case of encryption). Therefore, exposing the structural layout of packet payloads can allow preprocessing of transmitted data before it is actually processed by cloud computing resources. According to [7, 8], the average final output size jobs is 40.3% of the initial data sizes in Google, 8.2% in Yahoo, and 5.4% in Facebook. The present thesis exploits these properties of data streams and aims to reduce processing load on traditional cloud computing resources. We use two major innovative approaches to achieve this goal: (1) intermediate in-network aggregations of data streams and (2) in-network data processing on the data plane of network elements. Implementations of both approaches extend existing network interconnects.

Finally, the challenges also propagate down to the level of an individual network element. The two basic functions of a network element are *packet classification*, i.e., deciding which action to take based on a packet's header, and *buffer management*, i.e., making admission, processing, and transmission decisions for incoming packets. The present thesis advances both.

1.2 Goals and objectives of the research

In this section, we define the specific objectives that we set for the research outlined in the thesis. We define four major areas of the results, proceeding from the level of an individual network element up to the level of large-scale distributed computing: packet classification, buffer management, cost-efficient serverless computing, and intermediate data aggregations.

Buffer management: multiple packet characteristics. When *multiple data streams* meet at a network element, the corresponding data packets are stored in a single network buffer waiting for processing and transmission. Buffers are controlled by *buffer management policies*, which play a major role in optimization of the desired objectives. Advanced economic models bring novel objectives, such as weighted throughput, which are largely not supported by existing policies. In addition, the heterogeneity in processing requirements exacerbated by in-network data processing is also not accounted for. The previous works consider only one of the two packet characteristics: in [9, 10] authors study behavior of a single queue with heterogeneous processing requirements, while [11] addresses a multi-valued case for FIFO queues; in addition, more involved buffering architectures [12] (combined input-output switches) and [13] (crossbar switches) have been considered for multi-valued packets. The interaction of the two characteristics and their impact on weighted throughput optimization has not been studied before. In order to provide theoretical guidance for the optimal choice of buffer management policy in such a setting for a single queue buffering architecture, this dissertation aims to: (1) build a model capturing both packet weights and processing; (2) devise new management policies with worst-case performance guarantees; (3) evaluate their performance on realistic data traces.

Packet classification: inter-device resource sharing and LPM infrastructure reuse. The basic function of *single packet processing* at a network element is *packet classification*. General-purpose data stream processing brings new flexibility requirements and adds more complexity to the policies implemented through the packet classification. Current single-switch approaches consist of software-based solutions and hardware-based solutions. The former include decision tree structures [14, 15], hashing [16], and encoding [17], in general they require modification to the classification algorithm. The latter rely on *ternary content-addressable memory (TCAM)* and optimize its space usage [18, 19] and, in particular, range encoding [20]. On the network-wide level the two existing works [21, 22] present algorithms that duplicate rules and are hard computationally. It is highly desirable to support the added complexity *without* expensive infrastructure upgrade, e.g., in the form of additional TCAMs or complex logic changes. This includes two very specific objectives: (1) first, it should be made possible to implement general ternary bit string classification using traditional *Longest-Prefix-Match (LPM)* representation in an implementation-agnostic way; (2) second, already rising to the network-wide level, an efficient inter-

device resource sharing scheme is necesssary to implement large policies (classifiers) when rule capacity is limited in each network element.

Serverless computing: cost-efficient resource allocation. The first essential step towards cost-efficient serverless computing is to construct a model that would realistically capture the costs and constraints of resource management. These include *allocation cost*, *maintenance cost*, and, most importantly, the *delay* that resource allocation incurs. For the resulting model, it is then necessary to construct efficient algorithmic solutions that would require no assumptions regarding the arrival patterns of requests and would have the performance guarantees suitable for an economic setting. In contrast, existing rule-based solutions [23] require deep knowledge of arrival behavior, while learning-based techniques [24] assume the behavior similar to that seen during the training. Last but not least, it is important to control the latency the proposed solutions add to the request processing, and understand how it affects the revenue.

Data aggregation: planning with both network infrastructure and applications. In cloud computing, the network infrastructure with its constraints and objectives and an application with its specific behavior and its own set of objectives seldom interact in a meaningful way, and there is little information exchange between them. For instance, an application aggregating too much data at a single node may cause performance degradation due to TCP-incast [25] or link overload. To exploit intermediate data aggregations efficiently, one has to find a way for the two layers to cooperate. The most relevant work [26] is tied to a specific network topology; [27] introduces implementation support for intermediate aggregations, and [28] is concerned exclusively with the processing latency and does not take network infrastructure into account. The specific challenge addressed in the present thesis is to find the minimal necessary information required to share from each layer that would allow for unified design principles of aggregation planning.

2 Key results and conclusions

This section summarizes the key contributions of this thesis, addressing specific objectives introduced above. Here we provide a brief list of key results, and then will expand upon each research direction in more detail in Section 4.

2.1 New buffer management settings and policies

Buffer management with two packet characteristics. We analyze the buffer management problem for a single queue that stores packets heterogeneous *both* in their relative values (rewards) and the amount of required processing [29]. Previous works [9] addressed either one or the other of the two characteristics. We prove lower bounds on the competitive ratio (see [30]) for several natural priority-based algorithms, including a constant general lower bound shown for an arbitrary online algorithm. The main result of this part is a $\left(1 + \frac{W+2}{V}\right)$ upper bound for the special case with only two possible values; here W is the maximal amount of required processing, and the two possible values are 1 and V. The proof is based

on an inductive argument with a per-packet alignment between optimal and priority-based algorithms (similar to that in [9]), augmented by special treatment of packets with value 1.

2.2 New packet classification schemes and algorithms

Ternary to LPM transformation algorithms. We have designed two algorithms, `MinGroupPartition` and `MaxCoveragePartition`, for the task of transforming a general ternary bit-string packet classifier into an *equivalent group* of more constrained LPM classifiers [31]. The key part of this result is a novel connection between the ability to construct complex lookup keys and order theory, specifically Dilworth's decomposition theorem [32]. We show that `MinGroupPartition` minimizes the total number of groups, while `MaxCoveragePartition` maximizes the number of rules covered by a given number of LPM groups. These algorithms prove it possible to represent expressive ternary bit-string classification rules using traditional longest-prefix match capabilities of existing network infrastructures.

Complex network-wide packet classification. To improve classification across multiple network elements, we design a simple and space-efficient scheme, called `OneBit`, for distributing data flow (stream) classification rules along a flow's path [33]. `OneBit` relies on just a single bit of packet metadata and, compared to previous work, does not employ any heuristics, is very computationally easy (linear in the number of rules), and allows for a polynomial time decision procedure for the feasibility of multi-flow distribution. Similarly to the ternary-to-LPM transformation algorithms above, it allows to implement very complex policies without any network upgrade.

2.3 New cost-efficient resource allocation for serverless computing

Elastic resource allocation. We have explored the trade-off between delaying user requests and using available resources efficiently for the elastic resource allocation problem [34]. The proposed model incorporates an allocation cost α, a maintenance cost β, $\beta < 1$, and the time allocation takes (normalized to 1). We have shown a simple greedy algorithm `NRAP` to be at most $2 \cdot \left(1 + \frac{\alpha}{1-\alpha-\beta}\right)$-competitive for $\alpha < 1 - \beta$. For $\alpha \geq 1 - \beta$, we present a parameterized algorithm ρ-`AAP` that actively delays requests; we prove that ρ-`AAP` is $\frac{\rho}{1-\rho}\left(\left\lceil \frac{\rho\alpha}{1-\beta} \right\rceil + 1\right)$-competitive (with a small amount of extra buffer space). Compared to previous works [35], which largely relied either on past history or on a deep understanding of request arrivals, the guarantees we provide in this part of the dissertation are worst-case and thus hold for any arrival sequence.

2.4 New data aggregation schemes for compute-aggregate tasks

Compute-aggregate task planning. To optimize the placement of intermediate aggregations during the execution of compute-aggregate tasks, we formally introduce the *compute-aggregate minimization (CAM)* problem [36]. The formulation succinctly captures the characteristics of both the network (topology and transmission costs) and the application (aggregation size function). We argue that these characteristics are sufficient for the construction of an *aggregation plan* that effectively exploits intermediate data aggregations and minimizes the overall transmission cost. In particular, as part of this dissertation,

a hardness result was established for the non-associative case, and the relation between CAM and the minimum Steiner tree was characterized. Moreover, the aggregation plan, as it is defined in the CAM problem, does not restrict when exactly transmissions and aggregations happen; it is conveniently decoupled both from the network and from the application. Compared to previous work exploiting in-network aggregation [26], the treatment we present in this thesis is more general (in particular, it does not restrict network topology) and, apart from specific new results, provides a novel general framework for reasoning about compute-aggregate task planning based on the notion of the aggregation size function.

3 Publications and approbation of the research

Each of the key results presented in the previous section has been published in one of the peer-reviewed research papers listed below.

First-tier publications:

- Chuprikov P., Nikolenko S. I., Davydow A., Kogan K. Priority Queueing for Packets with Two Characteristics* // IEEE/ACM Transactions on Networking. 2018. vol. 26 (1). pp. 342-355.

  Contribution of the disseration's author: Theorems 1–3 characterizing the behaviour of priority-queue (PQ) based policies; Theorem 7 establishing a general lower bound; Theorem 11 presenting lower bounds for PQ-based policies in a two-valued case; Theorem 12 showing an upper bound for $\mathtt{PQ}_{v,-w}$ in the two-valued case; and Theorem 17 with lower bounds for PQ-based policies with β-pushout.

- Chuprikov P., Kogan K., Nikolenko S. General Ternary Bit Strings on Commodity Longest-prefix-match Infrastructures* // Proceedings of IEEE ICNP 2017.

  Contribution of the dissertation's author: Theorem 1 establishing a necessary and sufficient condition for prefix-reorderability; the `PrefixToLPM` algorithm with a proof of correctness in Theorem 2; the `MinGroupPartition` algorithm with a proof of correctness in Theorem 3; Theorem 4 showing a hard example for MinGR; the `MaxCoveragePartition` algorithm with a proof of correctness in Theorem 5; bit-expanding heuristic described in Section V; Observations 3 and 4 linking prefix-reorderability with rule disjointness; implementation of the above algorithms.

- Chuprikov P., Nikolenko S., Kogan K. On Demand Elastic Capacity Planning for Service Auto-scaling* // Proceedings of IEEE INFOCOM 2016.

  Contributions of the dissertation's author: Theorems 1 and 2 stating non-competitiveness in a bufferless setting; A lower bound in a buffered setting with small allocation cost in Theorem 3; the `NRAP` algorithm and its competitiveness in Theorems 4 and 5; the ρ-`AAP` algorithm and its analysis in Theorems 6 and 7; Theorem 8–11 providing latency guarantees for `NRAP` and ρ-`AAP`; Theorems 12 and 13 presenting general lower bounds for a bounded-delay (BD) model; Theorem 14 that upper-bounds `NRAP` in the BD model; Theorem 15 with a general lower bound for limited resources BD model (LBD); Theorems 16 and 17 providing bounds for `NRAP` in LBD model;

*The author of the dissertation is the main author of the paper

Second-tier publications:

- Chuprikov P., Davydow A., Kogan K., Nikolenko S. I., Sirotkin A. Formalizing Compute-Aggregate Problems in Cloud Computing // Proceedings of SIROCCO 2018.

  Contributions of the disseration's author: formalization of a compute-aggregate task planning problem and nonassociative hardness (Section 3 of the paper); Theorems 2, 6 and 7 connecting CAM to the minimum Steiner tree problem (Sections 4.1 and 4.2 of the paper).

Other publications:

- Chuprikov P., Kogan K., Nikolenko S. I., How to implement complex policies on existing network infrastructure* // Proceedings of ACM SOSR 2018.

  Contributions of the dissertation's author: the `OneBit` algorithm and its performance guarantees in Theorem 5.2; a network-wide solution presented in Section 6; performance comparison with existing algorithms.

The obtained results are supported in two ways. First, in almost all settings considered in this work we present rigorous proofs of worst-case performance guarantees valid under a realistic model. Second, for practical purposes where worst-case behavior might be rare, we evaluate proposed solutions on synthetic traces. Buffer management policies (Section 4.1) were run under traffic based on CAIDA traces [37]. Classifier transformation algorithms (Section 4.2) have been evaluated on the *Classbench* test suite [38]. Resource allocation algorithms (Section 4.3) were tested using randomly generated network-traffic-like pattern of arrivals [39].

4 Contents

This Section provides an overview of the research projects that led to the results presented in Section 2 and describes how these results achieve the objectives stated in Section 1.2. Again, we begin with the level of an individual network element, presenting our results on buffer management and packet classification, and then proceed upwards to resource allocation for serverless computing and compute-aggregate task planning.

4.1 Processing of multiple data streams

Interconnecting infrastructure should support advanced economic models expressed through new types of objectives. Buffer management policies play a critical role in the optimization of those objectives. Network traffic is highly heterogeneous, and its characteristics have an impact on the process of optimization, but they are often not being taken into account by buffer management policies. Introduction of in-network data processing would bring even more heterogeneity into processing requirements, making the current policies even less efficient. In response to new challenges, the paper titled "Priority Queueing for Packets with Two Characteristics" (see [29]) studies the effect of two packet characteristics, namely, processing requirements and value, on the optimization of weighted throughput in a single queue. This study leads to the first set of results from Section 2, and their overview is presented in this section.

The considered model assumes a single queue that is able to hold B unit-sized packets and that all arriving packets are unit-sized. Each arriving packet p has two characteristics: processing requirements *(work)* $w(p) \in \{1, \ldots, W\}$, and value $v(p) \in \{1, \ldots, V\}$, we will sometimes denote such p as $(w(p) \mid v(p))$. Both $w(p)$ and $v(p)$ are known at arrival. The time is assumed to be slotted, and each time slot t is subdivided intro three phases: (1) *admission*, when a set of new packets arrives, and the management policy decides which packets to admit to the queue, possibly *pushing out* already admitted ones; (2) *processing*, when a single packet from the queue is scheduled for processing; and (3) *transmission*, when a single *fully processed* packet, i.e., a packet p that has been scheduled for processing at least $w(p)$ times, is chosen for transmission. Note, the results that follow impose no constraints on the transmission order, so the transmission decision is trivial: transmit the only fully processed packet if any. The goal is to find a buffer management algorithm that would maximize the total weighted throughput, i.e., the total value of all transmitted packets.

Unfortunately, due to the online nature of the problem it is impossible to find an algorithm producing an optimal solution for every sequence of requests. The performance guarantees are, thus, provided relative to an unknown optimal solution using competitive analysis [30]. An *online* algorithm ALG is called α-competitive for some $\alpha \geq 1$ iff for *any* finite arrival sequence σ the total weighted throughput is at least $1/\alpha$ times the total weighted throughput of an optimal *offline* algorithm OPT.

Algorithms. The previous works have demonstrated that if either $V = 1$ or $W = 1$, then simple priority-based algorithms that prefer either higher value or lower processing requirement are, in fact, optimal [9]. For that reason, priority-based algorithms are natural candidates for the model described above. The difference is that here the choice of the priority (the admission/processing order) is not obvious. A generalized notion of a priority-queue-based algorithm is defined next.

Definition. *Let f be a function of packets, $f(p) \in \mathbb{R}$, with the intuition that better packets have larger values of f. Then the $\mathtt{PQ}_f$ processing policy is defined as follows:*

- $\mathtt{PQ}_f$ *is greedy, i.e., it accepts incoming packets as long as there is space in a buffer;*
- $\mathtt{PQ}_f$ *is work-conserving, i.e., it processes a packet as long as its buffer is not empty;*
- $\mathtt{PQ}_f$ *orders and processes packets in its queue in the order of decreasing values of f;*
- $\mathtt{PQ}_f$ *pushes out a packet p and adds a new packet p' to the queue at time slot t if the buffer is full, p is currently the worst packet in the buffer and p' is better than p: $f(p) = \min_{q \in \mathrm{IB}^{\mathtt{PQ}_f}} f(q)$, and $f(p') > f(p)$. Here $\mathrm{IB}^{\mathtt{PQ}_f}$ denotes the set of packets in $\mathtt{PQ}_f$'s buffer on the current timeslot.*

In short, on admission $\mathtt{PQ}_f$ considers new packets and buffered packets together keeping those with higher value of f; for processing $\mathtt{PQ}_f$ chooses a packet p with the highest $f(p)$. There are three promising candidates for the priority function f; in what follows, v and w denote, respectively, value and *current* processing requirements: (1) $\mathtt{PQ}_{v,-w} = \mathtt{PQ}_{v-w/(W+1)}$ that prefers packets with higher value, ties given to lower processing; (2) $\mathtt{PQ}_{-w,v} = \mathtt{PQ}_{-w+v/(V+1)}$ that prefers packets with lower processing, ties given to higher value; and (3) $\mathtt{PQ}_{v/w}$ that prefers packets with higher value-to-work ratio. The example of their behavior is shown in Figure 1.

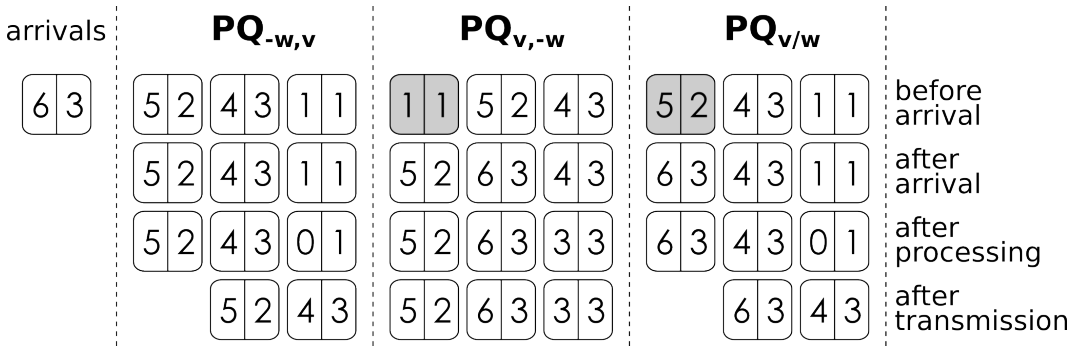

Figure 1: A sample time slot of $\mathsf{PQ}_{-w,v}$, $\mathsf{PQ}_{v,-w}$, and $\mathsf{PQ}_{v/w}$.

Lower Bounds. However promising, all three algorithms turn out to perform badly in terms of competitiveness. The next theorem combines Theorems 1, 2, and 3 from [29] demonstrating that each of the three policies has an approximation ratio linear either in W or V, suggesting that the interplay between the two packet characteristics makes the buffer management problem substantially more difficult.

Theorem. *For a buffer of size B, maximal amount of required processing W, and maximal value V:*

- $\mathsf{PQ}_{-w,v}$ *is exactly V-competitive;*
- $\mathsf{PQ}_{v,-w}$ *is at least $\left(W \cdot \frac{V-1}{V} - o(1)\right)$-competitive; and*
- $\mathsf{PQ}_{v/w}$ *is at least $\min\{V, W\}$-competitive.*

The bounds presented above hold only for specific algorithms and do not imply that an optimal one does not exist. For the latter kind of result, a *general* lower bound is required, constructing an adversarial input sequence able to "fool" *any* online algorithm. Section V of [29] presents a number of general lower bounds[†] that are polynomial in either V or W. The bounds differ in constraints imposed on an algorithm's implementation: FIFO processing order, fixed buffer size B, or being based on priority (PQ_f). A bound free of constraints is much weaker, but is still able to show that no optimal online algorithm exists.

Theorem. *For a buffer of size B, maximal packet required processing W, and available packet values 1 and $V > 1$, every online deterministic algorithm* ALG *is at least $\left(1 + \frac{V-1}{V^2} - O\left(\frac{1}{W}\right)\right)$-competitive.*

Two-valued case (an upper bound). The lower bounds demonstrate that in general natural priority-based algorithms are far from optimal and are even far from general lower bounds. The logical step is to add additional restrictions to the problem. One such restriction is to assume that there are only *two possible values*, i.e. either $v(p) = 1$ or $v(p) = V$. It corresponds to a scenario where all packets are divided into "commodity" packets $(w \mid 1)$ and "golden" or high-priority packets $(w \mid V)$.

Theorem. *For a buffer of size B, maximal required processing W, and available packet values 1 and V:*

1. $\mathsf{PQ}_{-w,v}$ *is at least V-competitive;*
2. *if $W \geq V$ then $\mathsf{PQ}_{v/w}$ is at least V-competitive;*
3. $\mathsf{PQ}_{v,-w}$ *is at least $\left(\frac{W}{V} + o(1)\right)$-competitive.*

[†]Not a part of this dissertation.

First, note that if $W < V$ then $\mathtt{PQ}_{v/w}$ operates exactly as $\mathtt{PQ}_{v,-w}$ since any $(w \mid V)$ is better than $(w' \mid 1)$. Second, the bound for $\mathtt{PQ}_{v,-w}$ has become much less strict, which is intuitive: if a processing of a given higher-valued packet was a wrong decision, loss per time slot should not exceed $\frac{V}{W}$ vs $\frac{1}{1}$. In fact, the intuition is (almost) true.

Theorem. *For a buffer of size B, maximal required processing W, and available packet values* 1 *and* V $\mathtt{PQ}_{v,-w}$ *is at most* $\left(1 + \frac{W+2}{V}\right)$*-competitive.*

The proof of the above result is based on an inductive argument similar to that in [9] comparing sorted (according to priority) sequences of packets in $\mathtt{PQ}_{v,-w}$'s and OPT's buffers. The novelty is in the mapping between "commodity" $(w \mid 1)$ packets processed by OPT and processing cycles spent by $\mathtt{PQ}_{v,-w}$. Given the V-competitiveness of $\mathtt{PQ}_{-w,v}$, $\mathtt{PQ}_{v,-w}$ and $\mathtt{PQ}_{-w,v}$ come very close to a $\min\{V, W/V\}$ lower bound[†] on priority-based algorithms (Theorem 6 in [29]).

Beta push-out case. The previous discussion treated a decision to drop a newly arrived packet and a decision to push-out an already admitted one equally. However, there are reasons to prefer the former to the latter (e.g., the latter consumes more resources on a network device). The work [9] introduced a copying cost model to account for the difference; in that model, each admitted packet reduces the objective function by α. Similarly to [9] a modified version of the $\mathtt{PQ}_f$ family of algorithms, $\mathtt{PQ}_f^\beta$ is considered here. $\mathtt{PQ}_f^\beta$ only pushes out a packet if a new packet is β times better ($\beta > 1$).

Theorem. *For a buffer of size B, maximal required processing W, and maximal packet value V:*

(1) $\mathtt{PQ}_{-w,v}^\beta$ *is at least V-competitive both in the case of arbitrary values and in the two-valued case;*

(2) $\mathtt{PQ}_{v/w}^\beta$ *is at least* $\min\{V, W\}$*-competitive in the case of arbitrary packet values and is at least V-competitive in the two-valued case with $\beta W \geq V$;*

(3) $\mathtt{PQ}_{v,-w}^\beta$ *is at least* $\left(\frac{(V-1)}{V}W - o(1)\right)$*-competitive in the case of arbitrary packet values and at least* $\left(\frac{W}{V} + o(1)\right)$*-competitive in the two-valued case.*

Results summary. Table 1 summarizes all the theoretical results presented in [29]. In the simulation study based on CAIDA [37] traces, the throughput of $\mathtt{PQ}_{v/w}$ has mostly remained within 90% of optimal (we used an overapproximation) with $\mathtt{PQ}_{v,-w}$ showing similar performance when there are only two packet values (see Figure 5 in [29]).

4.2 Packet classification: a basic network function for single packet processing

As we have stated earlier, *packet classification* is one of the core functionalities behind single packet processing, and it directly affects the performance of every network element. The purpose of packet classification is to distinguish among different types *(classes)* of packets in order to perform class-specific handling. This is useful both as a form of data processing, and as a way to differentiate among multiple data streams.

[†]Not a part of this dissertation.

Processing policy	General case	Two-valued case	
Adversarial general lower bounds			
Any online algorithm	$\frac{\min\{\sqrt[2B]{W}, \sqrt[B]{V}\}-1}{2}$†	$1 + \frac{V-1}{V^2} - O\left(\frac{1}{W}\right)$	
Any priority queue	$\sqrt{W}$†	$\min\{V, W/V\}$†	
Any FIFO online algorithm	$\left(\frac{\sqrt{W}}{B} + 1 - \frac{1}{B}\right)$†	$\frac{V+B-1}{W+B-1}$†	
Lower and upper bounds for specific algorithms			
Processing policy	Lower bound	Lower bound	Upper bound
$\mathsf{PQ}_{-w,v}$, $\mathsf{PQ}^{\beta}_{-w,v}$	V	V	V
$\mathsf{PQ}_{v,-w}$, $\mathsf{PQ}^{\beta}_{v,-w}$	$\frac{W(V-1)}{V} - o(1)$	$\frac{W}{V} + o(1)$	$1 + \frac{W+2}{V}$
$\mathsf{PQ}_{v/w}$, $W \geq V$	V	V	
$\mathsf{PQ}^{\beta}_{v/w}$, $\beta W \geq V$	V	V	
$\mathsf{PQ}_{v/w}$, $W < V$	W	$\frac{W}{V} + o(1)$	$2 + \frac{2}{V}$

Table 1: Results summary for buffer management: lower and upper bounds.

$\mathcal{K}$	#1	#2	#3	#4	Action
R_1	0	1	0	0	A_1
R_2	0	$*$	$*$	$*$	A_2
R_3	1	0	1	$*$	A_3
R_4	1	$*$	0	$*$	A_4

(a) Original classifier $\mathcal{K}$

$\mathcal{K}^B$	#1	#2	#3	#4	Action
R_1^B	0	0	1	0	A_1
R_2^B	0	$*$	$*$	$*$	A_2
R_3^B	1	1	0	$*$	A_3
R_4^B	1	0	$*$	$*$	A_4

(b) A B-reordering of $\mathcal{K}$, where $B = (1, 3, 2, 4)$

Figure 2: An example of a packet classifier with four rules and a packet header's width $w = 4$.

An input to the classification process is a packet *header* H that is represented by a sequence of bits $(h_1, \ldots, h_w)$ over the $\{0, 1\}$ alphabet and the output is an opaque *action* to be applied to the packet. The w above is the *classification width*. The classification behavior is defined by a *packet classifier* $\mathcal{K} = (R_1, \ldots, R_N)_{\prec}$, which is an ordered (prioritized) by $\prec$ set of *rules*. Each rule R_i consists of a *filter* F_i and an *action* A_i. The filter defines constraints on a packet header and is represented by a sequence of ternary bits $(f_1, \ldots, f_w)$ over the $\{0, 1, *\}$ alphabet, $*$ representing "don't care". A header $(h_1, \ldots, h_w)$ *matches* a filter (or a rule containing it) $(f_1, \ldots, f_w)$ iff for all i either $h_i = f_i$ or $f_i = *$. To classify a packet, a *lookup* into a classifier is performed, and an action of a rule whose filter matches a packet's header is returned as a result. If there are two rules R and R' that *intersect*, i.e., there exists a header matching both, the action from the higher priority rule is returned. A toy example of a packet classifier $\mathcal{K}$ is presented in Figure 2(a), higher priority rules are at the top.

The discussion that follows is largely concerned with transforming one classifier $\mathcal{K}$ into another classifier $\mathcal{K}'$. It is absolutely required that the transformation does not change $\mathcal{K}$'s semantics, in other words $\mathcal{K}$ and $\mathcal{K}'$ must be *equivalent*. Formally, two classifiers $\mathcal{K}$ and $\mathcal{K}'$ are equivalent iff for every header the lookup results for $\mathcal{K}$ and $\mathcal{K}'$ are the same.

Section 4.2.1 gives an overview of an approach published in a paper titled "General ternary bit strings on commodity longest-prefix-match infrastructure" (see [31]) corresponding to the second result from Section 2. Section 4.2.2 describes a paper "How to implement complex policies on existing network infrastructure" (see [33]) where the third result from the Section 2 is presented.

4.2.1 Representing general ternary bit strings on LPM infrastructure

A general packet classifier without any further constraints is called a *ternary* classifier. Ternary classifiers are the most powerful, they are found in modern packet processing abstractions [40] and are required for certain applications (e.g., [41]). Unfortunately, software-based implementations of ternary classifiers are inefficient: they require either too much space or time [42]; hardware solutions (TCAMs) being very expensive and power hungry are not present in sufficient capacity on commodity network devices. In contrast, once a classifier satisfies LPM constraints (to be defined shortly), it becomes much easier to represent the classifier efficiently, so the support for LPM classification is widely available. Formally, an *LPM classifier* satisfies the following: (1) all $*$s come after all 0s and 1s (e.g., rules R_2 and R_3 in Figure 2(a)), in other words, the classifier must be *prefix*; (2) given any two intersecting rules the one with a longer prefix has a higher priority (e.g., rules R_2^B and R_3^B in Figure 2(b) violate the property). The mismatch between application requirements and infrastructure capabilities demands a solution able to represent an arbitrary *ternary* classifier $\mathcal{K}$ on an *LPM* infrastructure with classification width at most w_{LPM} (usually, w_{LPM} is either 32 or 128 bits).

An approach presented here is agnostic to any particular LPM implementation. Its high-level overview is the following: first the classification width is reduced to w_{LPM}, then using a novel prefix-reorderability property the result is made prefix, and, finally, the priorities are adjusted to satisfy *LPM* constraints. The main contribution of the paper is the ternary-to-prefix transformation, to be explained first.

Prefix-reorderability. Network elements are capable of constructing complex lookup keys, in particular, they are able to reorder bits of a header. Reordering of header's bits allows for reordering of classifier's bits, which is a kind of transformation potentially able to turn the classifier into a prefix one. Formally, let $B = (b_1, b_2, \ldots, b_k)$, $k \leq w$ and $1 \leq b_i \leq w$, be a sequence of distinct bit indices representing the new bit order. Then for a header $H = (h_1, h_2, \ldots, h_w)$ and a filter $F = (f_1, f_2, \ldots, f_w)$, we define $H^B = (h_{b_1}, h_{b_2}, \ldots, h_{b_k})$ and $F^B = (f_{b_1}, f_{b_2}, \ldots, f_{b_k})$. Finally, for a rule $R = (F, A)$, $R^B = (F^B, A)$. The *B-reordering* of a classifier $\mathcal{K} = (R_1, R_2 \ldots, R_N)$ is a classifier $\mathcal{K}^B = (R_1^B, R_2^B, \ldots, R_N^B)$, where before each lookup an input header H is replaced with H^B. For example, a prefix $(1, 3, 2, 4)$-reordering of a classifier from Figure 2(a) is presented in Figure 2(b). It is not hard to see that if B is a permutation of $(1, \ldots, w)$ then classifiers $\mathcal{K}^B$ and $\mathcal{K}$ are equivalent. The question is whether there exists a permutation B such that $\mathcal{K}^B$ is a prefix classifier, i.e., whether $\mathcal{K}$ is *prefix-reorderable*.

It turns out useful to consider for a rule $R = (F, A)$ a set of bit indices i such that $f_i \neq *$ denoted by $\text{exact}(F)$, e.g., in Figure 2(a) $\text{exact}(R_4) = \{1, 3\}$. A curious observation is that if $\mathcal{K}^B$ is prefix, then for any $R \in \mathcal{K}$ bits from $\text{exact}(R)$ must precede in B bits from $\{1, \ldots, w\} \setminus \text{exact}(R)$; otherwise, the rule R would not be prefix. The observation imposes constraints on B, which may contradict each other: consider a classifier with $F_1 = (0\ *)$ and $F_2 = (*\ 1)$. Generally, if $\mathcal{K}$ contains two rules R and R' such that neither $\text{exact}(R) \subseteq \text{exact}(R')$ nor $\text{exact}(R') \subseteq \text{exact}(R)$, then $\mathcal{K}$ is *not* prefix reorderable. The first theorem shows that the above condition is sufficient and is easy to check; in what follows $\text{exact}(\mathcal{K}) = \{\text{exact}(R) : R \in \mathcal{K}\}$:

Theorem (chain criterion)**.** *A classifier $\mathcal{K}$ is prefix-reorderable iff for every $E_1, E_2 \in \text{exact}(\mathcal{K})$ either $E_1 \subseteq E_2$ or $E_2 \subseteq E_1$ holds, i.e., $\text{exact}(\mathcal{K})$ can be reordered to form a "chain": $E_{i_1} \subseteq \ldots \subseteq E_{i_{|\text{exact}(\mathcal{K})|}}$.*

$\mathcal{K}$	#1	#2	#3	#4	Action
R_1	0	0	0	*	A_1
R_2	0	0	1	*	A_2
R_3	*	1	0	0	A_3
R_4	0	0	*	*	A_4
R_5	*	0	1	*	A_5
R_6	*	1	0	*	A_6
R_7	*	0	*	*	A_7

(a) An original classifier $\mathcal{K}$

$\mathcal{K}_1$	#1	#2	#3	#4	Action
R_1	0	0	0	*	A_1
R_2	0	0	1	*	A_2
R_4	0	0	*	*	A_4
R_7	*	0	*	*	A_7

$\mathcal{K}_2$	#1	#2	#3	#4	Action
R_3	*	1	0	0	A_3
R_5	*	0	1	*	A_5
R_6	*	1	0	*	A_6

(b) A prefix-reorderable partition of $\mathcal{K}$

Figure 3: An example of a non-prefix-reorderable classifier

The permutation of bit indices can be found in $O(|\mathcal{K}| \cdot w)$ time (if one exists).

Once a permutation B witnessing $\mathcal{K}$'s prefix-reorderability is found, it only remains to transform the prefix classifier $\mathcal{K}^B$ into an LPM classifier, which can be done with a trie-based algorithm `PrefixToLPM` in $O(|\mathcal{K}| \cdot w)$ time (Theorem 2 in [31]). Note, neither transformation increases the number of rules.

Next, there will be considered two approaches dealing with non prefix-reorderable classifiers. The first approach relaxes the prefix-reorderability property and takes advantage of the parallelism available at network devices, and the second uses additional rules.

Minimal multi-group representation. Assume that classifier's rules are partitioned into prefix-reorderable groups. Then each group can be transformed into an equivalent LPM classifier using the algorithms discussed above. Since modern network devices are able to perform several lookups at line rate, it is feasible to perform a lookup in each of the constructed classifiers and return the highest priority result as a final answer. For example, the classifier $\mathcal{K}$ in Figure 3(a) is not prefix reorderable, nevertheless, it can be partitioned into $\mathcal{K}_1$ and $\mathcal{K}_2$ as in Figure 3(b). Both $\mathcal{K}_1$ and $\mathcal{K}_2$ are prefix-reorderable; hence, once they are transformed into LPM form, two LPM lookups would be enough to implement $\mathcal{K}$. The number of lookups supported at line rate is limited, thus it is desirable to minimize the number of groups.

Problem (MinGR)**.** *Find a partition of rules of a given classifier $\mathcal{K}$ into a minimal number of disjoint prefix-reorderable groups.*

The crucial step towards a solution for MinGR is to move from partitioning of $\mathcal{K}$ to partitioning of $\mathrm{exact}(\mathcal{K})$. From the chain criterion it is known that $\mathcal{K}' \subseteq \mathcal{K}$ is prefix reorderable iff $\mathrm{exact}(\mathcal{K}')$ is a chain. It follows that given a solution to MinGR it is possible to produce a *chain cover* of $\mathrm{exact}(\mathcal{K})$ of the same size. In the opposite direction, from any chain cover of $\mathrm{exact}(\mathcal{K})$ it is possible to build group-wise prefix-reorderable partition of $\mathcal{K}$ of the same size by grouping the rules according to exact. The MinGR problem is, thus, equivalent to the problem of finding the smallest chain partition of $\mathrm{exact}(\mathcal{K})$.

Note, the last problem is expressed solely in terms of a *partial order* $\mathrm{exact}(\mathcal{K})$ with relation $\subseteq$, and we can employ existing algorithmic solutions to the chain cover problem running in $O(|\mathrm{exact}(\mathcal{K})|^{5/2})$ time [32]. Adding the time to construct a partial order and the time to recover rule partitions leads to the `MinGroupPartition` algorithm. It is expected in practice (and was confirmed in evaluation) that $|\mathrm{exact}(\mathcal{K})|$ is much less than $|\mathcal{K}|$.

Theorem. *The* `MinGroupPartition` *algorithm finds an optimal solution for the* MinGR *problem in time* $O(|\operatorname{exact}(\mathcal{K})|^{5/2} + |\mathcal{K}|^2 w)$.

The number of groups produced by `MinGroupPartition` and, hence, the number of lookups required, can still be too large to implement at line rate on the LPM infrastructure.

Theorem. *There exists a classifier* $\mathcal{K}$ *with* $|\mathcal{K}| = O(\frac{1}{\sqrt{w}} 2^{w/2})$ *such that an optimal solution for the* MinGR *problem requires exactly* $|\mathcal{K}|$ *groups.*

Even though such a degenerate case as in the previous theorem is unlikely to occur, there may exist some *small* subset of rules that are "bad" for prefix-reorderablity rendering a soultion produced by `MinGroupPartition` infeasible to implement. The remedy to the issue is a "mixed" representation, where some part of the classifier is represented using traditional non-LPM techniques (e.g., in a small TCAM). The optimization objective is to minimize the size of that part given a limit on the number of LPM groups.

Problem (MaxCov)**.** *Given a classifier* $\mathcal{K}$ *and a constant* $\beta > 0$*, partition the largest possible subset of* $\mathcal{K}$*'s rules into at most* β *prefix-reorderable groups.*

To solve the MaxCov problem a `MaxCoveragePartition` algorithm slightly alters the underlying graph from [32] and runs minimum weight matching to account for "non-covered" rules.

Theorem. *The* `MaxCoveragePartition` *algorithm produces an optimal solution for the* MaxCov *in time* $O(|\operatorname{exact}(\mathcal{K})|^2 |\mathcal{K}| + |\mathcal{K}|^2 w)$.

A nice property that both `MaxCoveragePartition` and `MinGroupPartition` share is that non-prefix-reorderable classifiers are handled without increasing the number of rules. Still, sometimes it might be worthwhile to trade some memory for a smaller number of groups or a smaller size of the traditionally-represented part.

Problem (MaxCov–m)**.** *Given a classifier* $\mathcal{K}$*, a constant* $\beta > 0$*, and a maximal number of rules* M*, find the largest subset* $\mathcal{K}' \subseteq \mathcal{K}$ *and a multigroup classifier* $\mathcal{K}^*$ *equivalent to* $\mathcal{K}'$ *with* $|\mathcal{K}^*| \leq M$ *such that* $\mathcal{K}^*$ *can be split into at most* β *prefix-reorderable groups.*

As the MaxCov–m problem does not restrict the ways in which $\mathcal{K}'$ can be modified, it is unlikely that an optimal solution can be found efficiently. A proposed heuristic repeatedly takes a rule R that cannot be fit into any of $\mathcal{K}^*$'s groups, and adds more indices to $\operatorname{exact}(R)$ by expanding non-exact bits in the R's filter, essentially duplicating the rule.

Shrinking the classification width. It still may be the case that the classification width of the input classifier $\mathcal{K}$ may be larger than w_{LPM} supported by the infrastructure. To reduce the width in [18] authors suggest exploiting a *rule-disjointness* property. A classifier $\mathcal{K}$ is rule-disjoint on a set of bit indices B iff in $\mathcal{K}^B$ no two rules intersect. If $\mathcal{K}$ is rule disjoint on B, then $\mathcal{K}$ is equivalent to $\mathcal{K}^B$ with each action augmented by a false-positive check against the corresponding rule from $\mathcal{K}$. The width is reduced if $|B| < w$. A notable observation is that rule-disjointness and prefix-reorderability do not conflict with each other. Nevertheless, it is unclear which property should be satisfied first for the best performance.

Implementation and Evaluation. The optimizations described above were implemented in a platform-agnostic way using the P4-language infrastructure [40]. They also were evaluated based on synthetic

classifiers from the Classbench [38] suite. A heuristic that first splits the rules into rule-disjoint groups and then into LPM groups while doing bounded "don't care" bit expansions performed the best. For instance, we were able to represent on 32-bit-wide LPM infrastructure 80% to 99% of rules for ACL-based classifiers, 38% to 100%—for firewall-based, and almost 100%—for IP-chain-based (see Tables I and II in [31]).

4.2.2 Representing complex policies

The previous section approached the representation of complex policies on existing infrastructures from the point of view of a single network element. To satisfy network-wide rule capacity constraints, [21] and [22] suggested a "virtual pipeline" approach to resource sharing. In particular, [22] introduced the splitting of a policy among the switches on the *path of a network flow* as a building block in the network-wide policy representation. Formally, given a classifier $\mathcal{K}$, the *splitting* of $\mathcal{K}$ is defined as a *sequence* of classifiers that is *equivalent* to $\mathcal{K}$. A lookup into a sequence $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_l$ is performed through a sequence of lookups into each of $\mathcal{K}_i$ in order, each time applying an action returned from the ith lookup before proceeding to the $(i+1)$th. The intuition is that if $\mathcal{K}$ represents a policy for a given flow, then $\mathcal{K}_i$ should be stored on the ith switch along the flow's path. An *l-splitting* of a classifier $\mathcal{K}$ is a splitting of $\mathcal{K}$ having l elements. The formal problem is stated below.

Problem (FlowSplit)**.** *Given a classifier $\mathcal{K}$ and a sequence of switch capacities $c_1, c_2, \ldots, c_l$, find an l-splitting $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_l$ of $\mathcal{K}$ such that $\mathcal{K}_i$ has at most c_i rules.*

The complexity of the FlowSplit problem comes from intersecting rules, i.e., rules that match the same header. For instance, if a classifier $\mathcal{K}$ in Figure 4(a) is split arbitrarily as in Figure 4(b), then the equivalency may be lost: a header $(1\ 0\ 1\ 0)$ is matched twice: first in R_1 and then in R_2. A possible fix is to add **nop**-rules to $\mathcal{K}_2$ as shown in Figure 4(c).

There are two existing solutions to the issue of intersecting rules. Palette [21] expands $*$ bits, so that rules could be split into non-intersecting groups, i.e. no two $\mathcal{K}_i$ and $\mathcal{K}_j$, $i \neq j$, would match the same header. The optimization problem that arises from such approach is computationally hard, forcing Palette to resort to heuristics. The other approach, *One Big Switch* (OBS) [22], allows rules at different switches to intersect. To construct $\mathcal{K}_i$ OBS chooses a multidimensional rectangle r_i and sets $\mathcal{K}_i$ to be a "projection" of all remaining (not covered by $\mathcal{K}_j$, $j < i$) rules on r_i. To preserve the equivalency, in every later switch $\mathcal{K}_j$, $j > i$, a special high priority rule is added, mapping r_i to the **nop** action (similar to Figure 4).

A boolean minimization technique. The first technique, introduced in this dissertation, takes the iterative approach of OBS as a basis, but allows $\mathcal{K}_i$ to represent an arbitrary subset of rules, not necessarily a rectangle. The price of that flexibility is a larger number of **nop**-rules introduced to subsequent switches.

The technique relies on *Boolean Minimization* [43] and is called BM. It operates in l steps, the ith step, $1 \leq i \leq l$, takes as input a classifier $\mathcal{K}^{(i-1)}$ ($\mathcal{K}^{(0)} = \mathcal{K}$) and using a heuristics (an algorithm's parameter) selects a subset $\mathcal{K}'_i \subseteq \mathcal{K}^{(i-1)}$. Then, $\mathcal{K}_i$ is set to $\mathcal{K}^{(i-1)}$ with all rules *not* in $\mathcal{K}'_i$ having the **nop** action, and boolean minimization is then run over $\mathcal{K}_i$. If the result fits in c_i, $\mathcal{K}^{(i)}$ is constructed similarly to $\mathcal{K}_i$, except that all rules *in* $\mathcal{K}'_i$ are being mapped to **nop**. The $(i+1)$th step begins next. Figure 4 shows one

$\mathcal{K}$	#1	#2	#3	#4	Action
R_1	*	*	1	0	A_1
R_2	1	0	*	*	A_2
R_3	0	0	*	*	A_3
R_4	*	*	1	1	A_4

(a) An original classifier $\mathcal{K}$

$\mathcal{K}_1$	#1	#2	#3	#4	Action
R_1	*	*	1	0	A_1
R_3	0	0	*	*	A_3

$\mathcal{K}_2$	#1	#2	#3	#4	Action
R_2	1	0	*	*	A_2
R_4	*	*	1	1	A_4

(b) Not a splitting of $\mathcal{K}$

$\mathcal{K}'_2$	#1	#2	#3	#4	Action
R_1	*	*	1	0	**nop**
R_2	1	0	*	*	A_2
R_3	0	0	*	*	**nop**
R_4	*	*	1	1	A_4

(c) A version of $\mathcal{K}_2$ augmented with **nop**-rules to preserve equivalency

Figure 4: An example of a classifier splitting

possible result of BM. The subset-selection heuristic can supply more than one candidate for $\mathcal{K}'_i$.

One-bit of metadata. It can be shown (see Section 4 in [33]) that all three approaches BM, Palette, and OBS are incomparable in general. Still, they all share three common limitations: (1) *memory expansion* due to either **nop** rules (OBS, BM) or rule duplication (Palette); (2) *slow running times* due to hard underlying optimization problems; and (3) *lack of support for dynamic fields*, i.e., the fields that are changed by actions and are also used for classification. A solution avoiding all three limitations and requiring just one bit of metadata is presented next.

Consider a naive approach of putting first (according to priority) c_1 rules of $\mathcal{K}$ into $\mathcal{K}_1$, then next c_2 rules into $\mathcal{K}_2$, and so on until all rules have been assigned. The *only* thing that may go wrong is that a header first matched at some $\mathcal{K}_i$ would be matched again at $\mathcal{K}_j$, $j > i$ since the first match in such a scheme is *always* the right one. A simple fix for repeated matching implemented in a OneBit algorithm introduces a special **matched** bit to a header indicating whether the header has already been matched, the classification is then performed only if **matched** is not set. Every action in each $\mathcal{K}_i$ is modified to set the **matched** bit, and the default action (applied in case of no-match) of $\mathcal{K}_1$ resets **matched**. The following theorem states that OneBit has indeed avoided the limitations of the earlier approaches.

Theorem. *Given a* FlowSplit*'s instance* $(\mathcal{K}, \{c_i\}_i)$ *with* $\sum_i c_i \geq |\mathcal{K}|$*, the* OneBit *algorithm constructs in* $O(|\mathcal{K}|)$ *time an* l*-splitting of* $\mathcal{K}$ *that remains correct in the presence of dynamic fields and requires* $|\mathcal{K}|$ *rules in total.*

Network-wide solution. The OneBit algorithm solves the policy-representation problem only for a single flow. Network-wide, there are multiple flows, each is a subject to its own policy and has its own forwarding path. In [22] the following joint policy placement problem was stated:

Problem (MultiFlowSplit)**.** *Given a set of nodes* V*, node capacities* $c : V \to \mathbb{N}$*, and a set of* k *path/classifier pairs* $(P_i, \mathcal{K}_i)$*, where* P_i *is a sequence of vertices* $(v^i_1, \ldots, v^i_{l_i})$*, find a capacity allocation* $a : V \times [k] \to \mathbb{N}$ *such that* $\sum_{i=1}^{k} a(v,i) \leq c(v)$*, and a solution for each* FlowSplit *problem with* $\mathcal{K} = \mathcal{K}_i$ *and* $c_j = a(v^i_j, i)$ *for* $j = 1, \ldots, l_i$.

To solve MultiFlowSplit OBS's authors suggested an iterative heuristic that repeatedly refined per-flow capacity allocation, attempting to solve individual FlowSplit problems on every iteration using OBS technique. There was no bound on the number of iterations since there was no criterion for OBS to succeed in policy splitting. In contrast, the simplicity of a similar criterion for OneBit leads to a one-shot

solution to the MultiFlowSplit problem via a reduction from the per-flow capacity allocation problem to the maximum flow problem.

4.3 Elastic processing

Elasticity offered by the cloud allows for greater flexibility and operational savings. To be efficient, the resource capacity should match the offered load as close as possible. Since it is infeasible to adjust the capacity instantaneously, some degree of capacity planning is required. Traditionally, it has been a user's responsibility, but recently, a new "pay-per-use" approach has emerged in the form of *serverless computing* [2, 3, 4], where users only pay for actually processed requests. The work "On demand elastic capacity planning for service auto-scaling" (see [34]) addresses the challenges of building an efficient capacity planning scheme in a serverless setting, it constitutes the fourth set of results from Section 2.

Model description. The time is assumed to be slotted. While a given request may need different types of resources (e.g., memory, processing, or network bandwidth), all are jointly modelled as a single *(virtual) resource unit.* Every request r has an associated value $v(r)$ and is considered processed if it has been allocated exactly one resource unit for exactly one time slot. Once processed, r adds $v(r)$ to the revenue; the scaling is such that $\min_r\{v(r)\} = 1$. Reflecting operational expenses, the allocation of a resource unit costs α, and keeping a resource unit allocated has a maintenance cost β per time slot. Finally each time slot t is divided into three phases: (1) *arrival:* a set A_t of new requests arrives; (2) *processing:* a set P_t of requests is chosen for processing, $|P_t|$ must be less than N_t, the number of allocated resources; (3) *prediction* the resource capacity N_{t+1} is planned for the next time slot. Note, phases (2) and (3) can be handled in parallel, the essential constraint is that the prediction happens *before* the next arrival to account for the allocation delay. The set of requests to choose P_t from depends on the ability to delay request processing, the study of a fundamental trade-off between delaying requests and dropping them altogether is the main topic of [34].

Objective. The objective is to minimize $\sum_t(\sum_{r\in P_t} v(r) - \beta N_t - c_t)$, where c_t is the cost of changing the resource capacity from N_t to N_{t+1}. Hereafter a linear cost model is assumed, i.e., $c_t = \max\{0, N_{t+1} - N_t\}$. In the economic setting, average case guarantees are insufficient, so the competitive worst-case analysis [30] is used instead. A given *online* algorithm ALG is called α-competitive, $\alpha \geq 1$, if for any finite sequence of requests σ ALG's objective value ALG(σ) is at least OPT(σ)/α, where OPT(σ) is an objective value of the *optimal offline* algorithm OPT; ALG is not competitive if no such α exists.

Bufferless setting. If a request cannot be delayed, i.e., $P_t \subseteq A_t$ must hold, then such a setting is called *BufferLess with Heterogeneous values* (BLH). Unfortunately, even if it is profitable to process one request alone ($\alpha < 1 - \beta$), there is no competitive algorithm:

Theorem. *If $\alpha < 1 - \beta$, then in* BLH *any deterministic online allocation policy is non-competitive.*

The above theorem relies on the inability to predict the right amount of resources for the first arrival: there may always arrive more requests. To make the setting more realistic, an upper bound B on the

number of allocated resource units is imposed in the *Bounded BufferLess* setting with *Heterogeneous values* (BBLH). Nonetheless, B can be arbitrarily large, so non-competitiveness still holds:

Theorem. *If $\alpha < 1 - \beta$, then in* BBLH *any deterministic online allocation policy is non-competitive.*

Buffered setting. Motivated by the negative results of the bufferless setting, a *buffer* of size B is introduced into the model. Requests from A_t that have not been immediately serviced during t are hold in the buffer (at most B in total) for later processing. There are two variations of the model: *Buffered with Heterogeneous values* (BH) useful for upper bounds (as a more general one) and *Buffered with Uniform values* (BU) for lower bounds. Still, no optimal online algorithm exists.

Theorem. *If $\alpha < 1 - \beta$, then in* BU *every online algorithm is at least $\left(2 - \frac{\alpha}{1-\beta}\right)$-competitive.*

The first strategy that takes advantage of the buffer is called NRAP.

Definition. *Next Round Allocation policy (*NRAP*) operates as follows:*

- on arrival: *requests are accepted as long as there is room in the buffer; if the buffer is full then higher-valued requests always push out lower-valued ones from the buffer;*
- on processing: *either requests left from the previous time slot or those that have pushed them out are chosen for processing;*
- on prediction: *k resource units are predicted iff there are going to be k requests in the buffer by the end of the timeslot (i.e., not counting the requests under processing).*

It follows that NRAP never allocates more resources than necessary, but being to eager it may spend a lot on (de-)allocation. Nevertheless, NRAP has constant competitiveness once $\alpha < 1 - \beta$:

Theorem. *If $\alpha < 1 - \beta$, then in* BH *the* NRAP *algorithm is at most $\left(2 \cdot \frac{1-\beta}{1-\alpha-\beta}\right)$-competitive.*

The $\alpha < 1 - \beta$ condition is essential for NRAP to stay competitive since in general it performs one allocation per processed request. If allocation cost is higher and approaches 1 then NRAP's competitiveness tends to infinity.

Theorem. *If $\alpha < 1 - \beta$, then in* BU *the* NRAP *algorithm is at least $\left(1 + \frac{1-\beta}{1-\alpha-\beta}\right)$-competitive.*

In order to support allocation costs greater than $1 - \beta$, the next capacity planning algorithm, called ρ-AAP, does not allocate a resource unit until there is a sufficient number of pending requests to cover the allocation cost. Each resource unit has its own batch of *pre-assigned* requests.

Definition. *The* Amortizing Allocation Policy *ρ-AAP with parameter $\rho > 1$ operates as follows.*

- on arrival: *incoming requests are accepted as long as the remaining buffer capacity plus the number of allocated resource units is more than $(B - \lceil \frac{\rho\alpha}{1-\beta} \rceil)/(\lceil \frac{\rho\alpha}{1-\beta} \rceil + 1) - \lceil \frac{\rho\alpha}{1-\beta} \rceil$ (positive integer for simplicity); only requests accepted in the current time slot are allowed to be pushed out.*
- on processing: *for each allocated resource unit choose one of* preassigned *requests for processing.*
- on prediction: *while the total value of* non-preassigned *requests in the buffer is at least $\lceil \frac{\rho\alpha}{1-\beta} \rceil$:*
  - *predict that an additional resource unit v' will be needed for the next time slot;*

– *preassign to v' as few as possible non-preassigned requests with total value at least $\lceil \frac{\rho\alpha}{1-\beta} \rceil$;*

for every allocated resource unit with preassigned requests, predict that this unit will be needed for the next timeslot; deallocate those units that do not.

The following theorem summarizes the competitiveness guarantees of the ρ-AAP policy. For a simpler analysis ρ-AAP is given small extra buffer space compared to OPT.

Theorem. *If $\alpha \geq (1-\beta)$, the ρ-AAP algorithm with a buffer of size $B + \lceil \frac{\rho\alpha}{1-\beta} \rceil$ has competitive ratio at most $\frac{\rho}{\rho-1}\left(\left\lceil \frac{\rho\alpha}{1-\beta} \right\rceil + 1\right)$ in BH and at least $(\lceil \frac{\rho\alpha}{1-\beta} \rceil + 1)$ in BU against OPT with a buffer of size B.*

Latency analysis. To maintain quality of service it is important to control latency experienced by requests. If a request r arrives at time slot $t_a(r)$ and leaves the system at $t_l(r)$, then $\mathrm{lat}(r) = t_l(r) - t_a(r)$. For an algorithm ALG there are two possible definitions of latency: *worst case latency* $\mathrm{lat}(\mathtt{ALG}) = \sup_\sigma \max_{r\in\sigma} \mathrm{lat}(r)$; and *average case latency* $\mathrm{lat}_{\mathrm{avg}}(\mathtt{ALG}) = \sup_\sigma \left(\sum_{r\in\sigma} \mathrm{lat}(r)/|\sigma|\right)$.

While latency of the NRAP algorithm is optimal, ρ-AAP may hold requests indefinitely until enough of them are accumulated. In order to perform latency analysis for ρ-AAP, we assume that at least one request arrives every time slot. We introduce a modified version of ρ-AAP, called ρ-AAPm, that drops all non-preassigned requests if no requests arrive.

Theorem. *In the BH model: (1) $\mathrm{lat}(\mathtt{NRAP}) = 1$; (2) $\mathrm{lat}_{\mathrm{avg}}(\mathtt{NRAP}) = 1$; (3) $\mathrm{lat}(\rho\text{-}\mathtt{AAPm}) = \left\lceil \frac{\rho\alpha}{1-\beta} \right\rceil$; and (4) $\mathrm{lat}_{\mathrm{avg}}(\rho\text{-}\mathtt{AAPm}) \geq \frac{1}{2}\lceil \frac{\rho\alpha}{(1-\beta)} \rceil + \frac{1}{2}$.*

Bounded Delay Model. Instead of bounding latency *a posteriori*, latency can be incorporated into the model. In the *Bounded Delay* (BD) model each request r comes with a deadline $d(r) \geq 1$, s.t., if r arrives on $t_a(r)$ it is *automatically* dropped after $t_a(r) + d(r)$. The number of resources and the buffer size are unbounded. There is still an implicit bound on the buffer size: *maximal arrival rate* $\times$ *maximal deadline*. The BD model does not allow for competitiveness even under a weaker assumption than BU.

Theorem. *If $\alpha > 1 - \beta$, then in BD any deterministic online policy is not competitive.*

The NRAP policy given an unbounded buffer never pushes out requests and, hence, always processes the request from the previous time slot, allowing to achieve constant competitiveness once $\alpha < 1 - \beta$:

Theorem. *If $\alpha < 1-\beta$, then in BD any deterministic online policy is at least $\left(1 + \frac{\alpha}{2(1-\alpha-\beta)}\right)$-competitive, and NRAP is at most $\left(1 + \frac{\alpha}{1-\beta-\alpha}\right)$-competitive*

The last setting called *Limited buffer Bounded Delay* (LBD) imposes a fixed bound B on the number of allocated resources. Compared to BD the LBD model only slightly affects competitiveness.

Theorem. *If $\alpha < 1 - \beta$, then in LBD any deterministic online policy is at least $(2 + \frac{\alpha}{1-\beta-\alpha})$-competitive, while NRAP is at most $3(1 + \frac{\alpha+\beta}{1-\beta-\alpha})$ and at least 3 competitive.*

The experimental evaluation (see Section VIII in [34]) using synthetically generated traces has shown that if the allocation cost α remains small and the buffer size B is large, the proposed algorithms outperform learning-based straw-man approaches. As the allocation cost grows ρ-AAP looses noticably due to being over-cautious and underutilizing available resources.

4.4 Optimizing Compute-Aggregate Task Planning

An important class of big data processing applications are compute-aggregate systems where several data chunks must be aggregated at a given location. These systems are being optimized for latency and cost-efficiency, but rarely does the optimization take network constraints into account leading to TCP-incast [25] issues and link overload. The full joint optimization problem contains many moving parts, for which reason we introduce two *independent* phases; (1) finding the cheapest possible aggregation plan; and (2) performing actual aggregations and data transmissions while optimizing desired objectives. In a paper titled "Formalizing Compute-Aggregate Problems in Cloud Computing" the focus is on the first phase and, in particular, on the unified properties of compute-aggregate tasks that lead to efficient aggregation plans, which is the last result from Section 2

The network is modelled as an undirected graph $G = (V, E)$, where V is a set of computing nodes, and E is a set of connecting links. The compute aggregate task is represented by a target vertex $t \in V$ where the final result aggregated, and a set of initial data chunks $C = \{\boxed{x_1}, \ldots, \boxed{x_n}\}$, where each chunk x has a location denoted by $\mathrm{v}(\boxed{x}) \in V$ and a size denoted by $\mathrm{size}(\boxed{x}) \in \mathbb{R}_{\geq 0}$. To combine a multitude of possible optimization objectives (e.g., latency, throughput, avoidance of congestion) a single per-link cost function $c : E \leftarrow \mathbb{R}_{\geq 0}$ is used: to transmit $\boxed{x}$ through $e \in E$ one must pay $c(e) \cdot \mathrm{size}(\boxed{x})$.

Move to root is suboptimal. The processing scheme that currently prevails is to send every chunk to the root t and perform all aggregations there. It may be suboptimal or infeasible for three reasons: (1) insufficient memory capacity for low-latency in RAM applications; (2) data is sent to the root simultaneously causing TCP-incast [25]; (3) suboptimal transmission cost (as defined earlier). It is possible to improve on the first two issues by sending data chunks to t one by one and immediately aggregating as they arrive. To gain more flexibility in the aggregation order operations should be *associative*: $\mathbf{aggr}(\boxed{x}, \mathbf{aggr}(\boxed{y}, \boxed{z})) = \mathbf{aggr}(\mathbf{aggr}(\boxed{x}, \boxed{y}), \boxed{z})$, and *commutative*: $\mathbf{aggr}(\boxed{x}, \boxed{y}) = \mathbf{aggr}(\boxed{y}, \boxed{x})$.

Intermediate Aggregations. The principle of *moving computation to data* is widely used in big data processing to reduce network traffic. In this work we further extend that principle to *moving aggregation to data*, which allows data aggregation to happen at *intermediate nodes*. The extension is formalized as follows: an *aggregation plan* P is a sequence of operations $(o_1, o_2, \ldots, o_m)$, where each o_i is either $\mathbf{move}(\boxed{x}, v)$ that moves a chunk $\boxed{x}$ to a vertex v; or $\mathbf{aggr}(\boxed{x}, \boxed{y})$ that merges two chunks $\boxed{x}$ and $\boxed{y}$ located at the same vertex, which results in a new chunk $\boxed{xy}$. After P has been entirely processed, there must remain a single chunk $\boxed{z}$, s.t., $\mathrm{v}(\boxed{z}) = t$.

Every aggregation plan P has an associated transmission cost $\mathrm{cost}(P)$ that is a sum of costs of all P's operations: $\mathrm{cost}(\mathbf{move}(\boxed{x}, v)) = \mathrm{size}(\boxed{x}) \cdot d(\mathrm{v}(\boxed{x}), v)$, where $d(u, v)$ is the cheapest $(u \rightsquigarrow v)$-path according to c, and $\mathrm{cost}(\mathbf{aggr}(\boxed{x}, \boxed{y})) = 0$ since no data transmission is taking place.

Compared with the move to root strategy, intermediate aggregations: (1) reduce incoming traffic of individual nodes, improving TCP-incast behavior; (2) reduce aggregated data at individual nodes, improving memory efficiency; and (3) reduce overall amount of transferred data and, as a result, the transmission cost. Also, note that an aggregation plan is fully decoupled both from the network transport responsible for data transmission and from the application responsible for aggregation.

Aggregation size function. To state the problem formally, it remains to define $\mathrm{size}(\boxed{xy})$ for any **aggr**$(\boxed{x}, \boxed{y})$ operation. Knowing the exact value is infeasible during the planning phase: the value may depend on $\boxed{x}$'s and $\boxed{y}$'s content, and it may require to actually perform the aggregation. As a trade-off between accuracy and feasibility each application is asked to provide a size-based approximation in the form of an *aggregation size function* $\mu : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, so that the following equality holds (on execution it holds only approximately) $\mathrm{size}(\boxed{xy}) = \mu(\mathrm{size}(\boxed{x}), \mathrm{size}(\boxed{y}))$. The μ function must satisfy the same commutativity and associativity constraints as **aggr**. These are some examples of μ: $\mu(a,b) = \mathrm{const}$ for finding the top k elements; $\mu(a,b) = \min(a,b)$ or $\mu(a,b) = \max(a,b)$ for choosing the best data chunk; $\mu(a,b) = a + b$ for concatenation or sorting; $\max(a,b) \leq \mu(a,b) \leq a + b$ for set union.

Problem (CAM — compute-aggregate minimization)**.** *Given a connected undirected graph $G = (V,E)$, a cost function c, a target vertex t, a set of initial data chunks C, and an aggregation size function μ, the* $\mathrm{CAM}[\mu]$ *problem is to find an aggregation plan P such that* $\mathrm{cost}(P)$ *is minimized.*

There are two important special cases of CAM: CCAM and TCAM. In the first, for security reasons it is prohibited to aggregate chunks at intermediate nodes. As a result, the set of vertices where initial chunks are located becomes equal to V. In the second case we restrict the topology G to a tree.

Hardness and connection to the minimum Steiner tree problem. Interestingly, without the associativity constraint on μ, there does not exist a polynomial time algorithm with a constant approximation factor unless $\mathrm{P} = \mathrm{NP}$. The proof of the following theorem is based on a reduction from the knapsack problem.

Theorem. *Unless* $\mathrm{P} = \mathrm{NP}$*, there is no polynomial time approximation algorithm for* CAM *without associativity constraint on μ even if G is restricted to two vertices.*

To understand the connection between CAM and the *minimum Steiner tree problem (*MStT*)* it is instructive to consider a special case where all initial chunks have equal sizes and $\mu(x,x) = x$. With such a choice of μ it is always beneficial to merge chunks; hence, the aggregation would happen strictly along the tree. Note, the cost of the aggregation plan would be equal to the total cost of the tree multiplied by x, and we see that the CAM problem becomes equivalent to MStT. In a general case we would need to pay a multiplicative factor, equal to the ratio between maximum and minimum chunk size among all possible chunks. Using commutativity and associativity we can express both extrema as follows: let $W_C[\mu] = \max_{C' \subseteq C}\{\mu(C')\}$ and $w_c[\mu] = \min_{C' \subseteq C}\{\mu(C')\}$.

Theorem. *If there exists a polynomial α-approximate algorithm for* MStT*, then there exists a polynomial algorithm that solves* $\mathrm{CAM}[\mu]$ *with approximation factor* $\alpha \frac{W_C[\mu]}{w_C[\mu]}$.

There is a simple 2-approximation to the MStT based on the minimum spanning tree of the G's distance closure and a much more involved algorithm leading to $\alpha = \ln 4 + \epsilon \leq 1.39$ [44]. The two special cases, namely CCAM and TCAM, when viewed from MStT perspective become the minimum spanning tree and the unique Steiner tree, respectively. Both can be found in polynomial time, and hence in the above theorem has $\alpha = 1$ for CCAM and TCAM.

Corollary. *There exist polynomial algorithms that solve* $\mathrm{CCAM}[\mu]$ *and* $\mathrm{TCAM}[\mu]$ *on a set of chunks C*

with approximation factor $\frac{W_C[\mu]}{w_C[\mu]}$.

Furthermore, if we know that $\min\{a,b\} \leq \mu(a,b) \leq \max\{a,b\}$, then $W_C[\mu] = W_C = \max_{c \in C}\{\text{size}(c)\}$ and $w_C[\mu] = w_C = \min_{c \in C}\{\text{size}(c)\}$. As a result, it is possible to obtain a multiplicative factor which is independent of the exact choice of μ:

Theorem. *If* $\min\{a,b\} \leq \mu(a,b) \leq \max\{a,b\}$ *for all* a,b *and there exists a polynomial* α*-approximate algorithm for* MStT*, then there exists a polynomial algorithm that solves* $\text{CAM}[\mu]$ *with approximation factor* $\alpha\frac{W_C}{w_C}$.

Corollary. *If* $\min\{a,b\} \leq \mu(a,b) \leq \max\{a,b\}$ *then there exist polynomial algorithms that solves* $\text{CCAM}[\mu]$ *and* $\text{TCAM}[\mu]$ *with approximation factor* $\frac{W_C}{w_C}$.

Finally, when $\mu(a,b) \geq a + b$, then it never makes sense to merge chunks and we are able to treat all the chunks independently reducing the CAM to the problem of finding shortest paths from t.

Theorem. *If* $\mu(a,b) \geq a + b$ *for all* a,b *then there exists a polynomial optimal algorithm for* $\text{CAM}[\mu]$, $\text{CCAM}[\mu]$, *and* $\text{TCAM}[\mu]$*; for* $\text{TCAM}[\mu]$ *the running time is* $O(|C| + |G|)$.

5 Conclusion

This dissertation addresses fundamental scalability constraints of modern computing infrastructures, following the specific goals and objectives introduced in Section 1.2.

First, we have made important progress in the field of in-network data processing. To support new objectives and data stream characteristics during processing of multiple data streams, we performed worst-case performance analysis for several natural priority-based buffer management policies (Section 4.1). This analysis has for the first time jointly considered two characteristics, value and processing requirements. This has allowed us to make an optimal choice of the management policy in such a setting. In particular, we have shown that prioritizing according to value-to-work ratio is not always the best option, contrary to natural intuition.

Second, it has been proven possible to represent complex classification policies on existing network infrastructure through: (1) novel ternary-to-LPM classifier transformation algorithms (Section 4.2.1), and (2) an efficient but simple scheme for network capacity sharing (Section 4.2.2). As a result, we have introduced and justified techniques that achieve the expressiveness required for in-network data processing without excessive costs.

Third, for *efficient serverless computing* we have studied the trade-off between delaying service requests and revenue maximization in the serverless setting, using a novel formalization of serverless economics (Section 4.3). As part of this study, we have developed and studied two novel capacity planning algorithms, providing worst-case (assumption-free) performance guarantees and latency analysis for each. One major benefit of these contributions lies in cost efficiency improvements for the most elastic (i.e., flexible) computing paradigm.

Fourth, to exploit *intermediate data aggregations* we have developed a two-phase approach (Section 4.4). The first phase optimizes for budget constraints based only on the information necessary to construct an efficient aggregation plan. Desired objectives (e.g., latency or throughput) are captured indirectly and optimized by the network during the second phase. By separating the two phases, we

make network transport less reactive, reducing its task to scheduling of aggregations according to the constructed aggregation plan. Intermediate aggregations used in that plan bring an additional advantage that many aggregation issues (e.g., TCP-incast or memory capacity constraints) should become much less pronounced.

In total, this thesis represents significant progress towards the direction of exascale big data processing in a networked environment. This progress exposes a great potential of the underlying interconnecting infrastructure that has not seen much attention before. Taken together, our results allow to significantly enhance big data processing in computer networks without a radical redesign of underlying infrastructures.

References

[1] Hajirahimova Makrufa Sh., Aliyeva Aybeniz S. About Big Data Measurement Methodologies and Indicators // Intl. J. Modern Education and Computer Science. — 2017. — Vol. 9, no. 10. — P. 1–9.

[2] Amazon. AWS Lambda. — 2017. — https://aws.amazon.com/lambda/.

[3] Google. Cloud Functions. — 2017. — https://cloud.google.com/functions/.

[4] Microsoft. Azure Functions. — 2017. — https://azure.microsoft.com/en-us/services/functions/.

[5] Reserved, on demand or serverless: Model-based simulations for cloud budget planning / E. F. Boza [et al.] // 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM). — 2017. — Oct. — P. 1–6.

[6] Lloyd Wes [et al.]. Serverless Computing: An Investigation of Factors Influencing Microservice Performance // IC2E. — 2018. — P. 159–169.

[7] Dean Jeffrey, Ghemawat Sanjay. MapReduce: simplified data processing on large clusters // Commun. ACM. — 2008. — Vol. 51, no. 1. — P. 107–113.

[8] The Case for Evaluating MapReduce Performance Using Workload Suites / Yanpei Chen [et al.] // MASCOTS. — 2011. — P. 390–399.

[9] Providing Performance Guarantees in Multipass Network Processors / Isaac Keslassy [et al.] // IEEE/ACM Trans. Netw. — 2012. — Vol. 20, no. 6. — P. 1895–1909.

[10] Online Scheduling FIFO Policies with Admission and Push-Out / Kirill Kogan [et al.] // Theory of Computing Systems. — 2016. — Feb. — Vol. 58, no. 2. — P. 322–344. — URL: https://doi.org/10.1007/s00224-015-9626-4.

[11] Andelman Nir, Mansour Yishay, Zhu An. Competitive Queueing Policies for QoS Switches // Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. — SODA '03. — Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2003. — P. 761–770. — URL: http://dl.acm.org/citation.cfm?id=644108.644235.

[12] Kesselman Alex, Kogan Kirill, Segal Michael. Improved Competitive Performance Bounds for CIOQ Switches // Algorithmica. — 2012. — Jun. — Vol. 63, no. 1. — P. 411–424. — URL: https://doi.org/10.1007/s00453-011-9539-9.

[13] Kesselman Alex, Kogan Kirill, Segal Michael. Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing // Distributed Computing. — 2010. — Nov. — Vol. 23, no. 3. — P. 163–175. — URL: https://doi.org/10.1007/s00446-010-0114-4.

[14] Gupta P., McKeown N. Classifying packets with hierarchical intelligent cuttings // IEEE Micro. — 2000. — Jan. — Vol. 20, no. 1. — P. 34–41.

[15] Vamanan Balajee, Voskuilen Gwendolyn, Vijaykumar T. N. EffiCuts: Optimizing Packet Classification for Memory and Throughput // SIGCOMM Comput. Commun. Rev. — 2010. — Aug.. — Vol. 40, no. 4. — P. 207–218. — URL: http://doi.acm.org/10.1145/1851275.1851208.

[16] Fast packet classification using bloom filters / S. Dharmapurikar [et al.] // 2006 Symposium on Architecture For Networking And Communications Systems. — 2006. — Dec. — P. 61–70.

[17] Compressing Forwarding Tables for Datacenter Scalability / O. Rottenstreich [et al.] // IEEE Journal on Selected Areas in Communications. — 2014. — January. — Vol. 32, no. 1. — P. 138–151.

[18] SAX-PAC (Scalable And eXpressive PAcket Classification) / Kirill Kogan [et al.] // Proceedings of the 2014 ACM Conference on SIGCOMM. — SIGCOMM '14. — New York, NY, USA : ACM, 2014. — P. 15–26.

[19] Space and speed tradeoffs in TCAM hierarchical packet classification / Alexander Kesselman [et al.] // Journal of Computer and System Sciences. — 2013. — Vol. 79, no. 1. — P. 111 – 121. — URL: http://www.sciencedirect.com/science/article/pii/S0022000012001237.

[20] Bremler-Barr A., Hendler D. Space-Efficient TCAM-Based Classification Using Gray Coding // IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications. — 2007. — May. — P. 1388–1396.

[21] Kanizo Y., Hay D., Keslassy I. Palette: Distributing tables in software-defined networks // 2013 Proceedings IEEE INFOCOM. — 2013. — April. — P. 545–549.

[22] Optimizing the "One Big Switch" Abstraction in Software-defined Networks / Nanxi Kang [et al.] // Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies. — CoNEXT '13. — New York, NY, USA : ACM, 2013. — P. 13–24.

[23] Amazon AutoScaling. — http://aws.amazon.com/autoscaling/.

[24] Han R., et al. Lightweight Resource Scaling for Cloud Applications // CCGrid. — 2012. — P. 644–651.

[25] Understanding TCP Incast Throughput Collapse in Datacenter Networks / Yanpei Chen [et al.] // Proceedings of the 1st ACM Workshop on Research on Enterprise Networking. — WREN '09. — New York, NY, USA : ACM, 2009. — P. 73–82.

[26] Camdoop: Exploiting In-network Aggregation for Big Data Applications / Paolo Costa [et al.] // Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). — San Jose, CA : USENIX, 2012. — P. 29–42.

[27] Map-reduce-merge: Simplified Relational Data Processing on Large Clusters / Hung-chih Yang [et al.] // Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. — SIGMOD '07. — New York, NY, USA : ACM, 2007. — P. 1029–1040. — URL: http://doi.acm.org/10.1145/1247480.1247602.

[28] Optimal communication structures for big data aggregation / W. Culhane [et al.] // 2015 IEEE Conference on Computer Communications (INFOCOM). — 2015. — April. — P. 1643–1651.

[29] Priority Queueing for Packets With Two Characteristics / P. Chuprikov [et al.] // IEEE/ACM Transactions on Networking. — 2018. — Feb. — Vol. 26, no. 1. — P. 342–355.

[30] Borodin Allan, El-Yaniv Ran. Online Computation and Competitive Analysis. — New York, NY, USA : Cambridge University Press, 1998. — ISBN: 0-521-56392-5.

[31] Chuprikov P., Kogan K., Nikolenko S. General ternary bit strings on commodity longest-prefix-match infrastructures // 2017 IEEE 25th International Conference on Network Protocols (ICNP). — 2017. — Oct. — P. 1–10.

[32] Fulkerson D. R. Note on Dilworths decomposition theorem for partially ordered sets // Proc. Amer. Math. Soc. — 1956.

[33] Chuprikov Pavel, Kogan Kirill, Nikolenko Sergey. How to Implement Complex Policies on Existing Network Infrastructure // Proceedings of the Symposium on SDN Research. — SOSR '18. — New York, NY, USA : ACM, 2018. — P. 9:1–9:7. — URL: http://doi.acm.org/10.1145/3185467.3185477.

[34] Chuprikov P., Nikolenko S., Kogan K. On demand elastic capacity planning for service auto-scaling // IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications. — 2016. — April. — P. 1–9.

[35] Lorido-Botran Tania, Miguel-Alonso Jose, Lozano Jose A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments // Journal of Grid Computing. — 2014. — Dec. — Vol. 12, no. 4. — P. 559–592.

[36] Formalizing Compute-Aggregate Problems in Cloud Computing / Pavel Chuprikov [et al.] // Structural Information and Communication Complexity / Ed. by Zvi Lotker, Boaz Patt-Shamir. — Cham : Springer International Publishing, 2018. — P. 377–391.

[37] for Internet Data Analysis CAIDA The Cooperative Association. — [Online] http://www.caida.org/.

[38] Taylor David E., Turner Jonathan S. ClassBench: A Packet Classification Benchmark // IEEE/ACM Trans. Netw. — 2007. — Jun.. — Vol. 15, no. 3. — P. 499–511.

[39] Zukerman M., Neame T., Addie R. Internet traffic modeling and future technology implications // INFOCOM. — Vol. 1. — 2003. — March. — P. 587–596 vol.1.

[40] P4: Programming Protocol-independent Packet Processors / Pat Bosshart [et al.] // SIGCOMM Comput. Commun. Rev. — 2014. — Jul.. — Vol. 44, no. 3. — P. 87–95.

[41] Fast Regular Expression Matching Using Small TCAM / Chad R. Meiners [et al.] // IEEE/ACM Trans. Netw. — 2014. — Vol. 22, no. 1. — P. 94–109.

[42] Gupta P., McKeown N. Packet classification on multiple fields // SIGCOMM. — 1999. — P. 147–160.

[43] Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table / Eric Allender [et al.] // SIAM J. Comput. — 2008. — Vol. 38, no. 1. — P. 63–84.

[44] An Improved LP-based Approximation for Steiner Tree / Jaroslaw Byrka [et al.] // Proceedings of the Forty-second ACM Symposium on Theory of Computing. — STOC '10. — New York, NY, USA : ACM, 2010. — P. 583–592. — URL: http://doi.acm.org/10.1145/1806689.1806769.