Personal Insights on Three Research Directions in Networked Systems

Kirill Kogan* Sergey Nikolenko† Vitalii Demianiuk*† Pavel Chuprikov*† Alex Davydow†
*IMDEA Networks Institute †Steklov Institute of Mathematics at St. Petersburg

***Abstract*—In this work, we draw from our research and industry experience in the design of networked systems and define research directions that can simplify management and better exploit network infrastructure. We introduce problems both on data and control planes related to the design of a single network element and network-wide behaviors, concentrating on three directions: processing a single packet with packet classifiers, processing streams of packets in network switches, and network-wide control plane optimization. In particular, we consider efficient representations of packet classifiers, expressive implementations of buffer management policies, the composition of heterogeneous control planes, network virtualization, and extension of the network stack to support interactive applications. For the considered research directions outlined here, we formulate problems that, we believe, are important in the design of network systems. The purpose of this work is to attract system researchers to specific problems introduced in this paper.**

I. Introduction

The Internet is built around a large variety of network elements, transferring information aggregated into packets that are forwarded towards their destinations. New economic models and services require networks to be more intelligent and expressive, which usually translates into more expensive network infrastructure. Reducing manageable network state and better reuse of underlying network infrastructure without compromising expressiveness allow to improve cost efficiency. In this work we explore several interesting research directions to address the raised limitations, surveying existing preliminary results and identifying important problems for future research. We identify three major directions that will, in our opinion, lead to exciting new advances in networking research in the near future.

We begin with Section II, where we discuss the efficiency of programs processing single packets. In particular, Section II-A discusses structural properties that can lead to drastic reductions in classifier table size, and Section II-B, we takes the next step forward and moves from optimizing a single classification table to joint optimization of multiple classifier instances. The main problem here is how to share the same processing programs that may have to be applied to multiple flows in the most efficient way.

In Section III, we discuss efficient representations of buffer management policies in various settings both from analytic and infrastructural perspectives. In Section III-A, we identify an important future direction for the analytic study of switching architectures: how to formalize latency requirements so that theoretical results will remain relevant for practical work. In Section III-B, we show possibilities for automation and virtualization in the design of buffer management policies.

Section IV outlines recent developments and future directions for network-wide optimization. In particular, in Section IV-A we consider the composition of heterogeneous control planes, Section IV-B presents novel ideas related to network virtualization that preserves routing decisions, and Section IV-C shows how extensions of the network stack can support and optimize interactive distributed applications, in particular for compute-aggregate problems. We conclude with Section V.

Throughout the paper, we pose a number of problems that, we believe, represent worthwhile goals for the efforts of networking researchers; we believe that addressing these problems will grow into important research directions in the near future.

II. Processing of single packets

The ability to characterize traffic and understand its properties is critical for efficient and effective network operations. In particular, one has to determine service classes for various applications and different traffic patterns to facilitate accurate capacity planning and ensure that resources are used appropriately in support of implemented management policies. This yields new opportunities to justify and optimize vast investments involved in building networks, from understanding network behavior to exploiting the underlying infrastructure.

Data plane services can be split into two major categories: (1) services that change properties of single packets (e.g, recoloring specific fields, changing encapsulations, access control lists) and (2) services that change inter-packet properties (e.g., rate-limiting, shaping). In this part we consider services of the first category, and will return to the second in Section III.

Usually, a service on the level of single packets is represented by a *packet classifier*. It is, in a nutshell, a program consisting of an ordered set of conditions. The first condition that matches an incoming packet defines which actions apply. Each condition can be expressed as a multi-field classification rule. Packet classifiers based on more than one field have become very common. If a classification rule looks for exact values for all fields, it can be represented by simply concatenating all fields; such classifiers can be implemented in content-addressable memory (CAM) or by a simple hash function in space linear in the number of rules?. The problem becomes harder if a field is represented by a prefix (looking for specific values of only the most significant bits) or a range

(confining values inside an interval) since a concatenation of prefixes or ranges is no longer a prefix or range.

Many sophisticated software-based approaches have been proposed [1]. Bounds derived from computational geometry imply that a software-based packet classifier with N rules and $k \geq 3$ fields have to use either $O(kN)$ space and $O(\log N)$ lookup time or $O(N)$ space and $O(\log^{k-1} N)$ time [2], [3]. Thus, software-based approaches are either too slow or too memory-intensive even with few prefix- or range-based fields.

More advanced services require additional expressiveness on existing classification fields or new ones. Given the above theoretical bounds, increasing expressiveness (adding new fields) and number of rules in a classifier can significantly impede efficiency of implementations. In this section, we look at potentially interesting directions to address this fundamental tradeoff between scalability and expressiveness and improve efficiency of classifier representations. We begin with structural properties that allow to represent classifiers efficiently.

A. Structural properties of classifiers

We begin with some definitions (for a more complete picture see [4], [5]). A packet *header* $H = (h_1, \cdots, h_w)$ is a sequence (string) of w bits, where each bit h_i of H can have values of zero or one. We denote by W the ordered set of w *indices* of the bits in headers, $W = (1, ..., w)$. A *classifier* $\mathcal{K} = (R_1, ..., R_N)$ is an ordered set of *rules*, where each rule R_i consists of an ordered set (string) of w ternary values 0, 1, and $*$ ("don't care") corresponding to bits in headers and a pointer to the corresponding *action* A_i. A header H *matches* a rule R if for every bit of H, the corresponding bit of R has either the same value or $*$. The set of rules has non-cyclic ordering $\prec$: an incoming packet goes down the list to be matched, so if a header matches both R_i and R_j for $R_i \prec R_j$ the action of rule R_i is applied. We say that two classifiers are *semantically equivalent* if they match the same action for every header.

Previous research on the efficiency of packet classifiers has been mostly centered around semantically equivalent representations. Some of them introduce proprietary techniques, and most approaches are variations of Boolean minimization methods [6]–[9].

As a running example, consider the following classifier $\mathcal{K}$:

$\mathcal{K}$	#1	#2	#3	#4	#5	*Action*
R_1	0	1	0	0	0	A_1
R_2	0	0	1	1	1	A_2
R_3	0	1	0	0	1	A_1
R_4	1	1	1	0	0	A_3
R_5	1	1	0	0	0	A_3
R_6	*	*	*	*	*	drop

Here, an incoming filter $(0\ 1\ 0\ 0\ 1)$ will match R_3 but not R_1 or R_2, so A_1 will be applied; we will omit the "catch-all" rule R_6 in what follows. Note that R_4 and R_5 have the same action A_3 and differ in one bit only, so they can be compressed into $R_4' = (1\ 1\ *\ 0\ 0)$, which is semantically equivalent to the original classifier. Boolean minimization methods can reduce both classification width (number of bit identities involved in classification) and number of rules. But what does one do if optimized classifiers are irreducible but still do not reach desired efficiency metrics such as required memory and lookup time? Increasing classification width and number of rules significantly affects the efficiency of classifier representation, and hence feasibility; theoretical bounds show that there are no efficient representations without exponential memory and with feasible lookup time. The answer may come from structural properties of classifiers that lead to efficient implementations.

Since multiple rules can match the same header, a classifier has global priorities (order of rules) to resolve this ambiguity. But how much complexity can intersecting rules add? The first structural property that we consider here is *rule-order independence* [4], [10]. Two rules are *rule-order-independent* (ROI) if no header matches both; the name comes from the fact that such rules can come in any order inside a classifier with no change in semantics. If any two rules of a classifier are ROI, the entire classifier is ROI. Now, if a given ROI classifier remains ROI on a subset of bit identities, classification can be done on this subset, reducing classification width. In particular, $R_1, \ldots, R_5 \in \mathcal{K}$ are ROI, and we can remove two out of five bit identities from $\mathcal{K}$ while preserving ROI:

$\mathcal{K}^{\mathrm{ROI}}$	#1	#2	#3	#4	#5	*Action*
R_1^{ROI}	0	.	0	.	0	A_1
R_2^{ROI}	0	.	1	.	1	A_2
R_3^{ROI}	0	.	0	.	1	A_1
R_4^{ROI}	1	.	1	.	0	A_3
R_5^{ROI}	1	.	0	.	0	A_3

This classifier takes up only $\frac{3}{5}$ of the space needed for $\mathcal{K}$, and every packet that previously matched R_i now matches R_i^{ROI}.

The cost here is that this equivalence is not bijective: e.g., $(0\ 0\ 0\ 0\ 0)$ would be dropped by $\mathcal{K}$ but matches R_1^{ROI}. To get a semantically equivalent construction, we need a false-positive check that can be applied separately after a match has been found; e.g., in this case R_1^{ROI} matches $(0\ 0\ 0\ 0\ 0)$, ROI guarantees that no other rule matches this header, and the false-positive check only needs to test $(0\ 0\ 0\ 0\ 0)$ against one rule, the original R_1.

ROI reduces classification width and does not reduce the number of rules. A more relaxed structural property is *action-order independence* [11]. Two rules are *action-order independent* (AOI) if they either are ROI or have the same action. Equivalence is now preserved in terms of the same action applied to incoming packets rather than literally applying the same rules. Hence, AOI can often reduce classifiers even further, and fewer actions yield better results. In our running example, AOI allows to remove the fifth bit; R_1 and R_3 will thus become the same, $(0\ .\ 0\ .\ .)$, so R_3 can be removed entirely, and R_4 and R_5 can be resolved as above:

$\mathcal{K}^{\mathrm{AOI}}$	#1	#2	#3	#4	#5	*Action*
R_1^{AOI}	0	.	0	.	.	A_1
R_2^{AOI}	0	.	1	.	.	A_2
$R_4'^{\mathrm{AOI}}$	1	.	*	.	.	A_3

Now $\mathcal{K}^{\mathrm{AOI}}$ takes up only six bits instead of the original $|\mathcal{K}| = 25$. Though the efficiency of the main classifier (even versus the ROI variant) is improved, the false-positive check becomes

a full-fledged lookup: a header $(0\ 0\ 0\ 0\ 0)$ matched by R_1^{AOI} would now have be tested against both R_1 and R_3, unlike ROI where it was always sufficient to test against a single matched rule.

Can we delay or possibly completely avoid a false-positive check? When we reduce a classifier's width, we are often left with a classifier that does not preserve the semantics of the original one. Removing a bit identity is equivalent to changing its value to $*$ in all filters, so reducing classification width cannot decrease the set of headers covered by filters. Thus, during this relaxation we only need to guarantee correct matching for headers matched by the original classifier. Note that headers matched by both ROI and AOI classifiers have the same resulting action except the headers matched by the last "catch-all" entry that now may be matched by one of the actions. When a classifier represents a forwarding table, these headers represent "uncovered traffic", left outside a given traffic matrix for which forwarding decisions have been defined. Representations of classifiers based on structural properties without the false-positive check lead to the notion of *relaxed semantic equivalence*.

One possible alternative to this approach is *compact routing* that can reduce forwarding table size by *stretching* the quality of the objective requirements [12], in difference from relaxed semantic equivalence that affects only decisions for "uncovered" traffic.

Another interesting line of structural properties allow representations of general classifiers on existing longest-prefix-match infrastructures. The recently introduced *prefix reorderability* property [13] defines how to order bit identities of a given classifier with general priorities to get a prefix classifier with the longest-prefix match priorities.

Thus, different flavors of structural properties of classifiers lead to different efficient representations. We believe that many exciting developments are still possible in this direction and we expect new results in this direction. We highlight two potentially interesting problems, each leading to a potentially new direction of study.

Problem 1. *Find new structural properties of classifiers that allow for more efficient representations.*

Problem 2. *Find and consider cases where* approximate *classification results are useful; approximate classification can improve efficiency of representations even further while still guaranteeing a desired quality of the classification process.*

B. Optimizing multiple classifier instances

Growing scalability needs impose additional constraints on accountability and resolution of applied services at the data plane in order to support increasing numbers of flows. Even in traditional broadband environments, gateways often support tens of thousands of coexisting flows. Real life scenarios usually have only a few different policies (e.g., "gold", "silver", and "bronze") applied to tens of thousands of flows. Most existing works optimize a single policy instance and attempt to optimize required space and/or lookup time.

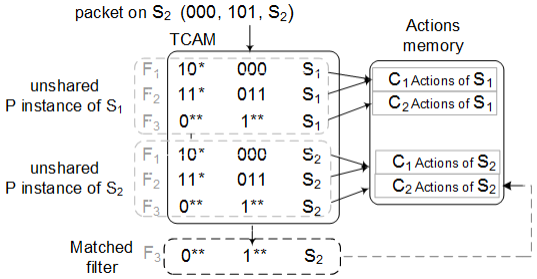

Fig. 1. Traditional attachment model: to implement per flow per action state, a separate instance of a policy classifier $\{F_1, F_2, F_3\}$ is allocated in TCAM per flow. The policy consists of two classes $C_1 = \{F_1, F_2\}$ and $C_2 = \{F_3\}$. To distinguish different instances of classifiers, flow identifier S_i occurs in the lookup key (in this example $(000, 101, S_2)$) and in classifier representation.

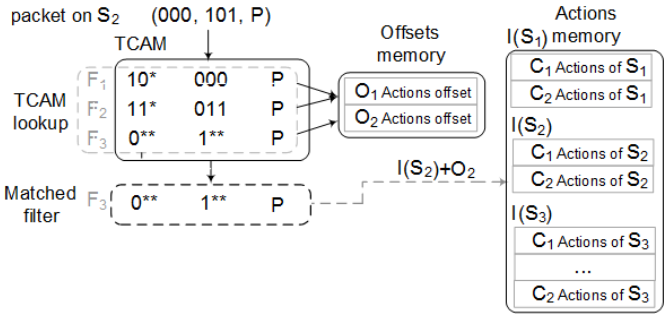

Fig. 2. Attachment model with equivalent actions layout: a single classifier instance is configured in TCAM for all flows associated with the same policy. The TCAM lookup result is an offset O_i from a flow-specific block of allocated actions $I(S_j)$. S_3 has different action structure than S_1 and S_2, so actions of the class C_2 cannot be found for flow S_3. P is a policy identifier.

But one can also look at the same problem from a different angle. A policy consists of a predefined set of traffic patterns, or *classes*, represented as classifiers like the ones considered in Section II-A. The notion of classes can be explicit, as in Cisco Modular Quality-of-service Command-line (MQC) [14], or implicit, as in various access-control list implementations (classification rules or subsets of them with the same actions) [15]. Typically, policies share classes extensively but can use them in different orders.

Traditionally, to implement state with per flow per action (or per class) resolution a separate policy instance is allocated per flow (see Fig. 1). The alternative approach studied in [16] hinges on the fact that the number of applied classification policies (e.g., QoS, firewall, etc.) and the number of flows (where these policies are configured) differ by orders of magnitude. For instance, in real-life scenarios "gold", "silver", and "bronze" policies can be applied to a huge number of flows. Since we need to maintain state with per flow per action resolution, the number of action instances for a policy P equals the number of flows with P configured independently. To maintain a single instance of a policy classifier, the work [16] applies an additional constraint on allocated actions, assuming that each flow S has a start address $I(S)$ of its instance in allocated policy actions. All allocated instances for actions of a policy P have an *equivalent layout* if all action

sets $A_i \in P$ have the same offset O_i from $I(S_j)$ for every flow S_j with P attached. For example, on Fig. 2 action layouts of S_1 and S_2 are equivalent but action layout of S_3 is not equivalent to S_1 or S_2 since actions of the class C_2 have a different offset in S_3's instance of the actions. The idea is to keep the equivalent layout for all instances of actions in the same policy (there are as many such instances as flows with the same policy) during allocation. In this case, the result of a matched class C_i is an offset O_i of A_i from $I(S)$ (rather than an address of an action as in the traditional attachment model). Thus, the absolute address of A_i corresponding to flow S_j is $I(S_j) + O_i$ (see Fig. 2).

It would be interesting to move from optimizing a single policy instance to efficient representations of multiple policies; the ultimate goal would be to maintain a single class instance for all policies that contain it, but due to different order constraints maximizing reuse and minimizing the total size turns into an interesting research problem. We expect that here, again, the structural properties considered in Section II-A may help since they can relax constraints on the ordering of classes. Overall, we arrive at two important problems for multiple classifier optimization.

Problem 3. *Find new invariants that allow efficient representation of classifiers with multiple coexisting flows.*

Problem 4. *Construct algorithms for joint optimization of policies consisting of classes that minimize total size.*

III. From single packets to packet streams

Buffering architectures define how input and output ports of a network element are connected [17], [18]. Their design and management directly impact performance and cost of each network element. If a burst of packets arrives, there are cases when it is impossible to transmit all packets immediately, and some of them must be buffered inside the network element (or dropped). A *buffer management policy* consists of three major parts: *admission control* (that decides which traffic should be admitted to the buffer), *processing policy* (defining a processing order of packets) and *packet scheduling* (that choose which packet is transmitted next).

A. Analytic results for various switch architectures

To analyze performance of buffer management policies, the networking community has historically relied on stochastic models to evaluate performance. These problems were first introduced in the theoretical community in early 2000s with the works [19]–[21] that applied competitive analysis [22] to this domain. Competitive analysis not only helps better understand buffer management policies but also provides worst-case guarantees on performance, which is important for real life switches with unpredictable and highly variable traffic patterns. We identify two major directions in recent developments in this domain: incorporating new traffic characteristics into buffer management policies and studying the impact of new characteristics on final objectives.

Packet characteristics that have been studied in previous works include *value*, i.e., how much the packet contributes to the objective function when it is transmitted [23]–[25], *required processing*, i.e., how many processing cycles the packet has to go through before it can be transmitted (this is motivated by many different tasks of varying complexity that a modern network edge can apply to a packet) [26]–[30], *size*, i.e., how much memory a packet takes up in the buffer, and others. Some works have considered packets that have multiple characteristics, e.g., value and required processing at the same time [31].

One major open question is how to formalize *latency* in the form of a final objective function. The first option is not to optimize latency but rather satisfy it by setting a "slack" (maximal delay) on per-packet basis. On the switch level, this model is called *bounded delay*, studied analytically in [32]. This lets one optimize other objectives such as throughput by incorporating multiple packet characteristics (value, required processing etc.). A major drawback of this approach that it is not conducive to end-to-end management decisions. In particular, it is unclear how to divide the slack of a given flow among network elements and packets in the flow.

Optimizing greedily for latency at every switch addresses this issue. Such recently proposed transports as *pFabric* and *pHost* optimize for the so-called *flow completion time* (FCT) [33], [34]. Under this approach, there is still some variation on how to represent FCT in the final objective and which additional information is required on the flow level. FCT is defined in [33] as the difference between the time e_f when a flow is fully processed at the last packet's destination and the time b_f when the first packet arrives at the source. If there is no multiplexing of multiple flows (i.e., the ith packet of a flow f arrives at the source at time $b_f + i$), and the flow sizes (in packets) are known to the scheduling algorithm *a priori*, there is an algorithm with SRPT (shortest remaining processing time) priorities that optimizes average FCT represented by $e_f - b_f$ for every flow f, and *pFabric* is based on this algorithm [33].

However, analytic results in realistic settings with respect to FCT remain elusive; the *pFabric* work [33] is inspired by [35] that makes two key assumptions: that flows are *atomic*, i.e., a flow is an uninterrupted stream of packets available all at once, and that all flow sizes are available in advance. Both these assumptions can be problematic in practice: in a network of switches, delays on a previous network element can result in interruptions in flows arriving on the next one, and flow sizes are often unknown when their first packets arrive. It is an important and challenging open problem to find suitable theoretical models of latency in buffer management.

Problem 5. *Devise latency representations that are amenable for analytic results in terms of competitive analysis while still incorporating sufficiently general settings relevant to networking practice.*

B. Towards software-defined buffer management

Traditional network management only allows to deploy a predefined set of buffer management policies whose parameters can be adapted to specific network conditions. Incorporation of new traffic characteristics introduces significant complexities in understanding the behaviors of various buffer management policies and requires complex control/data plane code changes and sometimes even hardware redesign. Current developments in software-defined networking mostly eschew these challenges and concentrate on flexible and efficient representations of packet classifiers, which do not really capture buffer management aspects. But can we program flexible buffer management policies by packet classifiers? In particular, can we use such languages as P4 [36] (possibly with some extensions) for this purpose?

While packet classifiers also define a packet stream whose inter-packet properties can be changed, buffer management policies can affect not necessarily the packet currently being processed. For instance, in case of congestion one can drop the head-of-line (HOL) packet to allow faster reaction of sources on congestion. This may require as many rules in a classifier as the number of buffered packets. Moreover, high-end network elements implement queueing modules in specialized hardware. Given both these constraints, it is important to find the right abstractions to express buffer management policies.

To specify an adequate language for software-defined buffer management, we need to identify primitive entities and their properties and a logic for manipulating these primitives. The choice of primitives dictates the simplicity and expressivity of the language. For example, the main primitives in OpenFlow are *flows*, *actions*, etc., flow properties are *fields*, and the logic to manipulate flows is an ordered set of *conditions*.

However, abstractions of ordered sets of conditions and actions that govern the handling of individual incoming packets as in OpenFlow or P4 appear insufficient to specify expressive and efficient buffer management behaviors. But how to adequately express policies? Buffer management policies are generally concerned with *boundary conditions* (e.g., upon admission a packet with *smallest* value can be dropped). Hence, *priority queues* arise as a natural choice for implementing actions related to user-defined priorities (e.g., FIFO processing order chooses a packet with *smallest* arrival time). The priority criteria does not change at runtime (e.g., a queue's ordering cannot change from FIFO to LIFO). Thus, each admission, processing, and scheduling policy maintains its priority queue data structure whose behavior is defined by a simple *comparator* – a Boolean function comparing two objects of same type via arithmetic/Boolean operators and accesses to packet and object attributes.

In addition, to specify when a queue or buffer should be considered congested, we introduce simple Boolean *conditions*. These simple constructs reconcile expressivity and simplicity, guaranteeing a constant number of insert/remove and lookup operations during admission or scheduling of a packet. Based on these principles [37], [38] propose a concise yet expressive language to define buffer management policies at runtime without control/data-plane code changes.

We believe that this abstraction can enable and accelerate innovation in the domain of buffering architectures and management, similar to programming abstractions that exploit OpenFlow for services with sophisticated classification modules. An interesting future research direction is to automate the design of management policies, given desired objectives and involved traffic characteristics that will allow to adapt automatically to desired network behaviors by using machine learning techniques. Moreover, it would be interesting to define a virtualization layer on top of heterogeneous computing resources that will distribute processing given service characteristics and a desired objective.

Problem 6. *Construct algorithms for automated design of management policies based on previous traffic history and desired objectives.*

Problem 7. *Design a virtualization layer able to distribute processing and buffer management among a network with heterogeneous structure for given objectives and network structure.*

IV. Expressiveness and simplicity in control planes

Network infrastructure is an expensive resource that requires complex management. Network providers are often unable to fully leverage this huge investment. In this section, we discuss control plane abstractions and mechanisms that allow to better exploit network infrastructure.

A. Composition of heterogeneous control planes

The software-defined networking (SDN) paradigm allows network operators to deploy network services through a centralized controller. Recent interest in SDN has fueled the implementation of a variety of network services on controllers written in different languages and supported by different organizations. Given the large number of network services and their increasing complexity, no single controller can provide all network services. Even if a controller provides all desired services, it is unlikely to have the best-in-class implementation of all those services. To address this problem, the works [39], [40] propose a framework with control plane composition that can use controllers from different vendors. The framework applies services implemented on heterogeneous controllers to the same network traffic by requiring only a standardized south-bound API. This prevents vendor lock-in at the control plane by allowing network operators to deploy services implemented on heterogeneous controllers. The framework is designed to operate in a way that is transparent to the controllers and does not require additional standardization as north-bound API. Finding new abstractions can significantly simplify network management and improve flexibility of control planes.

Originally, it was assumed that standardization of south-bound API can serve SDN requirements. Later it became clear that a single virtual controller for network management

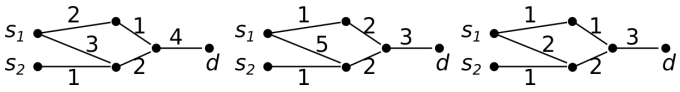

Fig. 3. Example of three bandwidth equivalent networks.

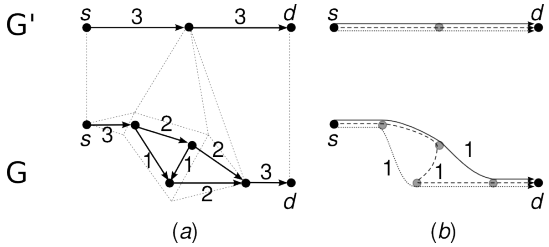

Fig. 4. Bandwidth equivalence: (a) two equivalent graphs, (b) a bandwidth allocation on G' with three routes of bandwidth 1 maps to G.

is not sufficient. The decoupling of control and data planes and standardization of south-bound API only improve the resolution of management but do not address the problem in general since if interoperability of more than one virtual controller is required the east-west API should be standardized as well. Before SDN, the standardization of east-west API was done with per protocol resolution, which is a complex task. We believe that standardization of east-west API should be protocol oblivious, and potentially the infrastructure of routing protocols can be exploited to achieve this goal.

Problem 8. *Build protocol oblivious abstraction for east-west API for inter-operation of heterogeneous control planes.*

B. Network virtualization preserving bandwidth and routing capabilities

To simplify network management, one can propose to represent the original network with a simpler/cheaper network that still implements specific properties of the original, which results in a tradeoff between the simplicity of representations and efficient reuse of the underlying infrastructure. Preserving network capabilities allows to operate services on the simpler representation transparently from the physical infrastructure, and understanding the constraints of a given network shows which resources need additional investments. There have been attempts to virtualize specific network architectures [33], [41], but there is no well-understood process to get a simplified representation of a network while preserving its "bandwidth" and "routing" capabilities.

The work [42] provides a first step in this direction, introducing new structural properties for networks and providing transformations that preserve these properties.

The first property considered in [42] is *bandwidth equivalence*. For a weighted directed network $G = (V, E, w, S, D)$ without self-loops with weights representing capacities and predefined sets S and D of source and destination vertices, a *bandwidth allocation* (BA) $A = (P, f)$ is a set of edge-simple paths $P = \{p_1, \ldots, p_k\}$ that start at sources, end at destinations, and satisfy capacity constraints (for a rigorous definition and proofs see [42]). Two networks with the same graph structure but possibly different capacities are *bandwidth equivalent*, $G \simeq G'$, if every BA on one is also a BA

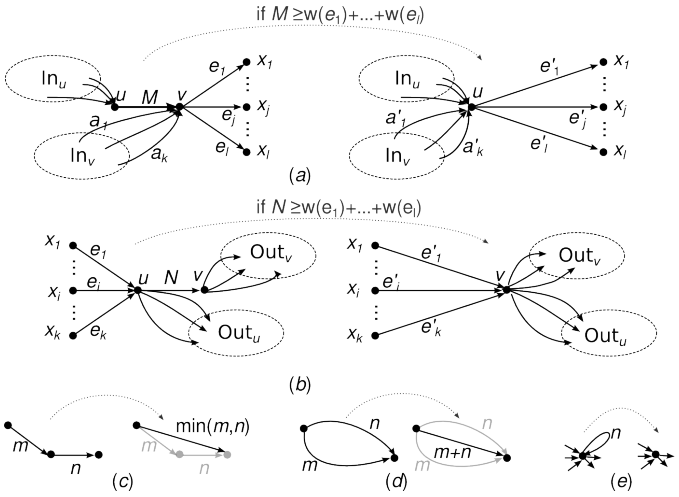

Fig. 5. Topology transformations: (a) shrinking non-bottleneck incoming edge; (b) shrinking non-bottleneck outgoing edge; (c) merging parallel edges; (d) removing self-loops; (e) path of length 2.

on another. For a simple example, the three networks on Fig. 3 are all bandwidth equivalent: any set of routes from s_1 and s_2 to d that satisfies capacity constraints shown on the edges of one of the graphs will also satisfy them in the two others. However, the graphs have very different total capacities, and the result of capacity planning for all three networks on Fig. 3 should be the network on the right, which has minimal aggregate capacity and cannot be reduced further. The work [42] introduces the notion of a *minimal bandwidth network*, where capacities cannot be reduced without violating bandwidth equivalence, and presents a polynomial algorithm on DAGs and efficient heuristic algorithms on general graphs for finding minimal bandwidth networks, all without changing the underlying network graph.

But the graph itself can also be simplified. The notion of *routing equivalence* extends bandwidth equivalence to transformations that change graph structure: two networks G and G' with the same sources S and destinations D are *routing equivalent* if there is a one-to-one mapping $g : \mathrm{Path}_G(S, D) \to \mathrm{Path}_{G'}(S, D)$ between paths from sources to destinations on G and G' that preserves BAs (we again refer to [42] for a formal definition and proofs). Basically, if we can find a way to reduce a network while preserving routing equivalence, it means that we can solve routing problems on the simpler reduced network and then "lift" the solutions back to the original network.

Experiments in [42] indicate that by using relatively simple heuristic algorithms one can reduce complex graphs with real life network topologies by a factor of 2-3, often reducing even further with non-equivalent routing transformations. Thus, preliminary results reported in [42] are promising and lead to the following problems for further work.

Problem 9. *Find new transformations that preserve original bandwidth and routing capabilities of a network while simplifying its structure/topology.*

Problem 10. *Construct efficient algorithms for recovering routing decisions from a network reduced by bandwidth preserving but not routing equivalent transformations.*

Note that the work [42] only the first step towards a much more ambitious goal. The final objective is to represent the original network as a small virtual switch, where a richer set of decisions (e.g., scheduling) can be made and mapped to operations in the physical network.

Problem 11. *Define transformations that allow to represent a given network as a virtual switch and show how to implement scheduling decisions in the virtual switch on the represented physical network.*

C. Formalizing compute-aggregate problems

To address access bandwidth constraints to storage, data centers maintain data at different interconnected locations. Modern big data applications are highly distributed, and requests need to satisfy various objectives such as latency and cost efficiency [43]–[45]. *Compute-aggregate* problems, where several data chunks must be aggregated in a network sink, encompass an important class of big data applications implemented in modern data centers. To compute the final result, multiple data chunks must be aggregated in one place while satisfying combined latency constraints.

The works [46], [47] study optimal compute-aggregate structures to minimize latency. Unfortunately, the final result can heavily depend on the properties of underlying transports and utilization of network infrastructure. To address these limitations, the work [48] proposes a model that constructs a schedule of aggregations under budget constraints that can be specified for each compute-aggregate task. Later, an underlying transport can redistribute the computed schedule in time to optimize desired objectives such as latency or throughput. We believe that this approach will allow to decouple optimization problems from underlying transports and provide better fine-grained control to exploit network infrastructure.

Several network infrastructure characteristics can potentially interact with an aggregation schedule. These characteristics include link latencies, current link loads (for example, due to concurrently running aggregations), and energy efficiency parameters. In order to keep things as simple as possible and decoupled from the network, in [48] all of these characteristics are modeled as a single link transmission cost per unit of data.

The input to aggregation schedule construction contains the initial distribution of data and, most importantly, a distributed application is required to provide a way to approximate the size of two data chunks after they are aggregated. In this model, this approximation is called an *aggregation size function* μ. It estimates the size of aggregated data chunks, $\mathrm{size}(\boxed{xy}) = \mu\big(\mathrm{size}(\boxed{x}), \mathrm{size}(\boxed{y})\big)$, and its properties significantly affect aggregation schedule end cost. We do not expect μ to be exactly correct, but it should provide the correct order of magnitude in order for the optimal solution to be actually good in practice. Some examples of μ for practical problems include:

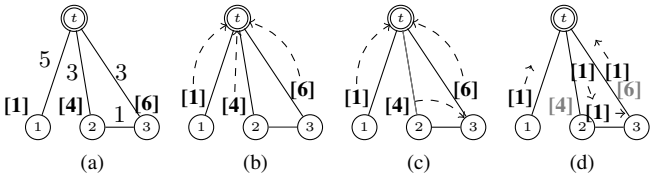

Fig. 6. Different μ lead to different plans: (a) a sample task; (b) optimal plan for $\mu(a,b) = a + b$; (c) optimal plan for $\mu(a,b) = \max(a,b)$; (d) optimal plan for $\mu(a,b) = \min(a,b)$.

- $\mu(a,b) = \mathrm{const}$ for finding the top k elements in data with respect to some criterion;
- $\mu(a,b) = \min(a,b)$ or $\mu(a,b) = \max(a,b)$ for choosing the best data chunk;
- $\mu(a,b) = a + b$ for concatenation or sorting;
- $\max(a,b) \le \mu(a,b) \le a + b$ for set union (word count).

Fig. 6 shows how the choice of μ can affect the optimal aggregation plan. Fig. 6a shows chunks of size 1 at vertex 1, of size 4 at vertex 2, and of size 6 at vertex 3, and the goal is to aggregate them at vertex 0. For $\mu(a,b) = a + b$, the optimal plan is to move each chunk to the root separately (Fig. 6b). For $\mu(a,b) = \max(a,b)$, it is cheaper to first move the chunk of size 4 along edge $2 \to 3$ and merge it, then move the resulting chunk of size 6 to the root (Fig. 6c). Finally, for $\mu(a,b) = \min(a,b)$ the optimal plan is to leave large chunks in place and traverse the whole graph with the smallest chunk, merging larger ones along the way (Fig. 6d). Thus, even in a simple example the aggregation plan can change drastically depending on μ.

The works [46], [47] introduce optimization problems for minimizing the resources needed to fulfill a compute-aggregate task based on μ, [48] proposes new algorithms (based on spanning and Steiner trees) for the extended problem that takes into account network infrastructure, and show approximation bounds. However, this direction of study is far from closed. We formulate the general goal of this field as an important research problem.

Problem 12. *Construct a classification of aggregation functions, complete with theoretical and practical analysis, which will unify design principles of "perfect" aggregations and help design aggregation plans for a variety of compute-aggregate problems.*

V. Conclusion

In this work, we have discussed three research directions in the design of networked systems. Based on our experience in both research and industry, we have defined a number of potentially interesting problems for future studies in these domains. The primary goal of this work is to attract more attention of systems researchers to the topics we have discussed and motivate to explore them from alternative angles.

Acknowledgements

This work was supported by the Russian Science Foundation grant 17-11-01276 "Networking and distributed systems and algorithms and related fundamental problems".

REFERENCES

[1] G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*, ser. The Morgan Kaufmann Series in Networking. Morgan Kaufmann, 2005.

[2] M. H. Overmars and A. F. van der Stappen, "Range searching and point location among fat objects," *J. Algorithms*, vol. 21, no. 3, pp. 629–656, 1996.

[3] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *Micro*, vol. 20, no. 1, pp. 34–41, 2000.

[4] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "Sax-pac (scalable and expressive packet classification)," in *SIGCOMM*. ACM Press, 2014, pp. 15–26.

[5] P. Chuprikov, K. Kogan, and S. I. Nikolenko, "General ternary bit strings on commodity longest-prefix-match infrastructures," in *ICNP*, 2017, pp. 1–10.

[6] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. E. Saks, "Minimizing disjunctive normal form formulas and ac^0 circuits given a truth table," *SIAM J. Comput.*, vol. 38, no. 1, pp. 63–84, 2008.

[7] S. Khot and R. Saket, "Hardness of minimizing and learning DNF expressions," in *FOCS*, 2008, pp. 231–240.

[8] C. Umans, "The minimum equivalent DNF problem and shortest implicants," *J. Comput. Syst. Sci.*, vol. 63, no. 4, pp. 597–611, 2001.

[9] K. Kogan, S. I. Nikolenko, W. Culhane, P. Eugster, and E. Ruan, "Towards efficient implementation of packet classifiers in SDN/OpenFlow," in *HotSDN*, 2013.

[10] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. T. Eugster, "Exploiting order independence for scalable and expressive packet classification," *IEEE Transactions on Networking*, vol. 24, no. 2, pp. 1251–1264, 2016.

[11] K. Kogan, S. I. Nikolenko, P. Eugster, A. Shalimov, and O. Rottenstreich, "FIB efficiency in distributed platforms," in *ICNP*, 2016, pp. 1–10.

[12] D. V. Krioukov, K. C. Claffy, K. R. Fall, and A. Brady, "On compact routing for the internet," *Comput. Commun. Rev.*, vol. 37, no. 3, pp. 41–52, 2007.

[13] P. Chuprikov, K. Kogan, and S. Nikolenko, "General ternary bit strings on commodity longest-prefix-match infrastructures," in *ICNP*, 2017, pp. 1–10.

[14] "Quality of Service (QoS)," http://www.cisco.com/c/en/us/products/ios-nx-os-software/quality-of-service-qos/index.html.

[15] "Cisco IOS Security Configuration Guide," http://www.cisco.com/c/en/us/td/docs/ios/12_2/security/configuration/guide/fsecur_c.pdf.

[16] K. Kogan, S. I. Nikolenko, P. T. Eugster, and E. Ruan, "Strategies for mitigating TCAM space bottlenecks," in *HOTI*, 2014.

[17] M. Goldwasser, "A survey of buffer management policies for packet switches," *SIGACT News*, vol. 41, no. 1, pp. 100–128, 2010.

[18] S. I. Nikolenko and K. Kogan, "Single and multiple buffer processing," in *Encyclopedia of Algorithms*. Springer, 2016, pp. 1988–1994.

[19] W. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosén, "Competitive queue policies for differentiated services," in *INFOCOM*, 2000, pp. 431–440.

[20] Y. Mansour, B. Patt-Shamir, and O. Lapid, "Optimal smoothing schedules for real-time streams (extended abstract)," in *PODC*, 2000, pp. 21–29.

[21] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, "Buffer overflow management in qos switches," in *STOC*, 2001, pp. 520–529.

[22] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[23] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, "Buffer overflow management in QoS switches," *SIAM J. Comput.*, vol. 33, no. 3, pp. 563–583, 2004.

[24] J. Kawahara, K. Kobayashi, and T. Maeda, "Tight analysis of priority queuing policy for egress traffic," *CoRR*, vol. abs/1207.5959, 2012.

[25] P. T. Eugster, K. Kogan, S. I. Nikolenko, and A. Sirotkin, "Shared memory buffer management for heterogeneous packet processing," in *ICDCS*, 2014, pp. 471–480.

[26] I. Keslassy, K. Kogan, G. Scalosub, and M. Segal, "Providing performance guarantees in multipass network processors," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1895–1909, 2012.

[27] K. Kogan, A. López-Ortiz, S. I. Nikolenko, and A. V. Sirotkin, "A taxonomy of semi-FIFO policies," in *IPCCC*, 2012, pp. 295–304.

[28] K. Kogan, A. López-Ortiz, S. I. Nikolenko, and A. V. Sirotkin, "Online scheduling fifo policies with admission and push-out," *Theory of Computing Systems*, vol. 58, no. 2, pp. 322–344, 2016.

[29] P. Eugster, K. Kogan, S. I. Nikolenko, and A. V. Sirotkin, "Heterogeneous packet processing in shared memory buffers," *Journal of Parallel and Distributed Computing*, 2017.

[30] P. T. Eugster, A. Kesselman, K. Kogan, S. I. Nikolenko, and A. Sirotkin, "Essential traffic parameters for shared memory switch performance," in *SIROCCO*, 2015, pp. 61–75.

[31] P. Chuprikov, S. I. Nikolenko, and K. Kogan, "Priority queueing with multiple packet characteristics," in *INFOCOM*, 2015, pp. 1418–1426.

[32] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, "Buffer overflow management in QoS switches," *SIAM Journal on Computing*, vol. 33, no. 3, pp. 563–583, 2004.

[33] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: minimal near-optimal datacenter transport," in *SIGCOMM*, 2013.

[34] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in *CONEXT*, 2015, pp. 1–12.

[35] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai, "Sum multicoloring of graphs," *J. Algorithms*, vol. 37, no. 2, pp. 422–450, 2000.

[36] "P416 language specification," https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.pdf, 2017.

[37] K. Kogan, D. Menikkumbura, G. Petri, Y. Noh, S. I. Nikolenko, and P. T. Eugster, "BASEL (buffer management specification language)," in *ANCS*, 2016, pp. 69–74.

[38] K. Kogan, D. Menikkumbura, G. Petri, Y. Noh, S. I. Nikolenko, A. Sirotkin, and P. T. Eugster, "A programmable buffer management platform," in *ICNP*, 2017, pp. 1–10.

[39] K. Kogan, A. Dixit, and P. Eugster, "Serial composition of heterogeneous control planes," in *ONS*, 2014.

[40] A. Dixit, K. Kogan, and P. Eugster, "Composing heterogeneous SDN controllers with flowbricks," in *ICNP*, 2014, pp. 287–292.

[41] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *CoNEXT*, 2013.

[42] S. I. Nikolenko, K. Kogan, and A. Fernández Anta, "Network simplification preserving bandwidth and routing capabilities," in *INFOCOM*, 2017.

[43] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *USENIX*, 2010, pp. 281–296.

[44] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," in *OSDI*, 2006, pp. 205–218.

[45] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[46] W. Culhane, K. Kogan, C. Jayalath, and P. Eugster, "LOOM: Optimal aggregation overlays for in-memory big data processing," in *HotCloud*. Philadelphia, PA: USENIX Association, 2014. [Online]. Available: https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/culhane

[47] ——, "Optimal communication structures for big data aggregation," in *INFOCOM*, April 2015, pp. 1643–1651.

[48] P. Chuprikov, A. Davydow, K. Kogan, S. Nikolenko, and A. Sirotkin, "Planning in compute-aggregate problems as optimization problems on graphs," in *ICNP*, 2017, pp. 1–2.