as *network invariants*. Sample (slightly simplified from the ones checked for Azure) network invariants are:

Network Invariant 1: Traffic from a host leaf directed to a different cluster from the leaf is forwarded to a router in a layer above. In other words, suppose that *Router* belongs to a cluster given as a predicate *Cluster*, and that *RouterAbove* is the set of routers above *Router*, then

$$dst \notin Cluster \wedge Router \Rightarrow \bigvee_{n \in RouterAbove} n$$

On the other hand,

Network Invariant 2: Traffic from a host leaf directed to the same cluster is directed to the local VLAN or a router in the layer above that belongs to the same cluster as the host leaf router:

$$dst \in Cluster \wedge Router \quad \Rightarrow \\ VLAN \ \vee \ \bigvee_{n \in RouterAbove}(n \wedge n \in Cluster)$$

The routing behavior of routers at the same level from the same cluster should also act uniformly for addresses within the cluster (they can behave differently for addresses outside of a cluster range).

Network Invariant 3: Let $Router_1$, $Router_2$ be two routers at the same layer within the cluster *Cluster*, then

$$dst \in Cluster \Rightarrow Router_1 \equiv Router_2$$

IV. Differential Network Reachability

In the previous section we described how SecGuru performs local checks on routers. These local checks often imply global properties of the network. This approach works fine in the context of the Azure architecture, which is fixed and data-centers are deployed in cookie-cutter form. Finding local invariants, however, is an entirely manual process and the approach does not generalize to arbitrary networks (though there is a really good point to capturing and checking architecture based invariants for Azure). The behavior of a router is commonly a combination of ACLs, forwarding rules, and packet rewriting. It is therefore not generally possible to check global network invariants from a fixed set of local network invariants. To check global network properties we developed a specialized tool in Z3 that handles configurations for packet switching networks efficiently.

This time we represent forwarding logic and networks as a set of constrained *Datalog* rules. Suppose that n_r is a predicate representing the current router from our example, and $n_1, n_2, \ldots$ are the names of next-hop routers, represented as predicates, then the rules for representing the routing behavior can be written:

$$\forall dst \ . \ n_1(dst) \leftarrow \left(\begin{array}{ll} & n_r(dst) \\ \wedge & dst \neq 10.91.114.0/25 \\ \wedge & dst \neq 10.91.114.128/25 \wedge \ldots \end{array} \right)$$

$$\forall dst \ . \ n_2(dst) \leftarrow \left(\begin{array}{ll} & n_r(dst) \\ \wedge & dst \neq 10.91.114.0/25 \\ \wedge & dst \neq 10.91.114.128/25 \wedge \ldots \end{array} \right)$$

$$\forall dst \ . \ n_3(dst) \leftarrow \left(\begin{array}{ll} & n_r(dst) \\ \wedge & \left(\begin{array}{l} dst = 10.91.114.0/25 \vee \\ dst = 10.91.114.128/25 \end{array} \right) \\ \wedge & \ldots \end{array} \right)$$

$$\ldots$$

Constrained Datalog with stratified negation provides logical expressitivity that makes it easy to encode queries over pairs of paths. Thus, one can use Datalog to query for packets that are dropped along one route but not another.

Header Space Algebra (HSA) [9] was introduced to reason efficiently about reachability over sets of headers. The basic data-structure used by HSA is a difference of cubes (DOC) representation of three-valued bit-vectors. Three-valued bit-vectors encode address masks compactly using don't cares. An example DOC is the expression:

$$1 * 110 * * \setminus (*1 * * * 11 \cup *0 * * * 00)$$

It is shorthand for the set

$$\{1011011, 1011001, 1011010, 1111000, 1111001, 1111010\}.$$

In [11] we adapt DOC encodings as an underlying table representations for a Datalog engine in Z3. For a set of benchmarks extracted from Azure and Stanford networks we observed that the DOC representation scales well beyond competing representations, such as BDDs, or SAT based bounded model checking. Model checking techniques for (software defined) networks is actively investigated in several contexts, including the Anteater tool [12] and in [18].

V. Programmable Controllers

Network controller programs operate at their core by receiving packets from routers. The packets are rewritten, forwarded and used to update both local state and routing tables. In [1] we developed a language, VeriCon, capturing core features of network controllers relevant to verification of network controllers. State, local and external routing tables, are uniformly represented as predicates (Boolean arrays). Proving invariants of the controllers turns out to requite a limited expressive logical power close in style to the Bernays-Schönfinkel-Ramsey, otherwise known as Effectively Propositional Reasoning (EPR). EPR formulas are of the form: $\forall \vec{y} \ . \ \varphi[\vec{c}, \vec{y}]$, where $\vec{c}$ is a set of constant symbols, and the formula φ is quantifier-free over equalities and uninterpreted relations over the constants and bound variables. Thus, EPR formulas do not have nested functions.

The VeriCon verification conditions are discharged automatically by Z3, or in case of properties that are not invariants, Z3 provides counter-examples. Furthermore, invariants that were not already inductive are in some cases inductive after conjoining the invariants with their weakest pre-conditions. Weakest pre-condition strengthening is a folklore approach used in variations in deductive-algorithmic model checking. While it is simple to implement it does not scale very well and current efforts include replacing the strengthening by more sophisticated

approaches and also ensuring that the assertion language remains within a decidable extension of EPR.

VI. The Logical Power of Networks

A common experience so far has been that network verification is matched well by logics and solvers that exploit how ACLs, forwarding rules and controller programs handle sets of packets the same way: Transitions are guarded by predicates on bit-ranges and state updates copy or update bit-ranges to constant values. In other words, the tools exploit and support packet ranges and how the state of controllers is updated based on a few enumerable attributes. Yet, the underlying algorithms from our experiences are orthogonal. The bit-vector solver used in SecGuru reduces verification to propositional SAT; DNA pairing requires a Datalog engine; controller verification uses invariants expressed over quantified first-order logic so it requires efficient quantifier instantiation. The Z3 SMT solver exposes much richer functionality than the fragments we used here: Z3 supports reasoning about logical formulas using linear integer, linear and non-linear real arithmetic, algebraic data-types and arrays. It contains specialized engines for solving Horn clauses over arithmetic [3], [7], [13] that so far target applications from symbolic software model checking.

We believe the mutual exposure of formal methods to modern packet switched network engineering is a significant area of opportunity for both camps. An indication that this is broadly the case is that we are not the only ones who apply SMT, SAT, QBF, finite state model checking and other verification and synthesis technologies for programmable packet switched networks [14], [17], [19], [16]. More narrowly, the use of SMT solving and other theorem proving technologies for Network Verification offers mutual opportunities to improve scale and reliability of modern (large scale) data-center networks. On the other hand, the applications that emerge from Network Verification inspire new algorithms and data-structures for theorem proving and model checking.

Acknowledgment Our experiences with network verification is based on joint work with several collaborators, including: George Varghese, Mooly Sagiv, Charlie Kaufman, Geoff Outhred, Nuno Lopes, Mingchen Zhao, Jeff Jensen, Monika Machado, Garvit Juniwal, Ratul Mahajan, Ari Fogel, Jim Larus, Thomas Ball, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Michael Schapira and Asaf Valadarsky.

References

[1] Thomas Ball, Nikolaj Bjørner, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Mooly Sagiv, Michael Schapira, and Asaf Valadarsky. VeriCon: towards verifying controller programs in software-defined networks. In Michael F. P. O'Boyle and Keshav Pingali, editors, *PLDI*, page 31. ACM, 2014.

[2] Thomas Ball, Vladimir Levin, and Sriram K. Rajamani. A decade of software model checking with SLAM. *Commun. ACM*, 54(7):68–76, 2011.

[3] Nikolaj Bjørner, Kenneth L. McMillan, and Andrey Rybalchenko. Program Verification as Satisfiability Modulo Theories. In Pascal Fontaine and Amit Goel, editors, *SMT@IJCAR*, volume 20 of *EPiC Series*, pages 3–11. EasyChair, 2012.

[4] Achim D. Brucker, Lukas Brügger, and Burkhart Wolff. holTestGen/fw - An Environment for Specification-Based Firewall Conformance Testing. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *ICTAC*, volume 8049 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2013.

[5] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[6] P. Godefroid, J. de Halleux, A. V. Nori, S. K. Rajamani, W. Schulte, N. Tillmann, and M. Y. Levin. Automating Software Testing Using Program Analysis. *IEEE Software*, 25(5):30–37, 2008.

[7] Krystof Hoder and Nikolaj Bjørner. Generalized Property Directed Reachability. In Alessandro Cimatti and Roberto Sebastiani, editors, *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012.

[8] Karthick Jayaraman, Nikolaj Bjørner, Geoff Outhred, and Charlie Kaufman. Automated Analysis and Debugging of Network Connectivity Policies. Technical Report MSR-TR-2014-102, Microsoft Research, July 2014.

[9] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: static checking for networks. In *NSDI*, 2012.

[10] K. Rustan M. Leino. Developing verified programs with dafny. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *ICSE*, pages 1488–1490. IEEE / ACM, 2013.

[11] Nuno Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. DNA Pairing: Using Differential Network Analysis to find Reachability Bugs. Technical Report MSR-TR-2014-58, Microsoft Research, 2014.

[12] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the Data Plane with Anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, New York, NY, USA, 2011. ACM.

[13] Kenneth L. McMillan. Lazy Annotation Revisited. In Armin Biere and Roderick Bloem, editors, *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2014.

[14] Sanjai Narain, Gary Levin, Sharad Malik, and Vikram Kaul. Declarative Infrastructure Configuration Synthesis and Debugging. *J. Netw. Syst. Manage.*, 16(3):235–258, September 2008.

[15] Timothy Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The Margrave tool for firewall analysis. In *LISA*, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[16] Andrew Noyes, Todd Warszawski, Pavol Cerný, and Nate Foster. Toward synthesis of network updates. In Bernd Finkbeiner and Armando Solar-Lezama, editors, *SYNT*, volume 142 of *EPTCS*, pages 8–23, 2014.

[17] Shuyuan Zhang, Abdulrahman Mahmoud, Sharad Malik, and Sanjai Narain. Verification and synthesis of firewalls using SAT and QBF. In *ICNP*, pages 1–6. IEEE, 2012.

[18] Shuyuan Zhang and Sharad Malik. SAT Based Verification of Network Data Planes. In Dang Van Hung and Mizuhito Ogawa, editors, *ATVA*, volume 8172 of *Lecture Notes in Computer Science*, pages 496–505. Springer, 2013.

[19] Shuyuan Zhang, Sharad Malik, and Rick McGeer. Verification of computer switching networks: An overview. In *ATVA*, 2012.

On QoS Management in SDN by Multipath Routing

E. Chemeritskiy
Lomonosov Moscow State University
Moscow, Russia
tyz@lvk.cs.msu.su

R. Smelansky
Applied Research Center for Computer Networks
Moscow, Russia
smel@arccn.ru

***Abstract*—The Quality of Service (QoS) management is one of the urgent problems in networking which doesn't have an acceptable solution yet. In the paper the approach to this problem based on multipath routing protocol in SDN is considered. The proposed approach is compared with other QoS management methods. A structural and operation schemes for its practical implementation is proposed.**

Keywords—Quality of Service; Multipath Routing; Software-Defined Networks; Network Management

I. INTRODUCTION

QoS (Quality of Service) as a term is a general description of the performance of a network connection. This term is treated either as qualitative assessment of the connection performance by a user, or as a set of objective quantitative parameters characterizing the one. Qualitative evaluation of QoS is defined as the degree of satisfaction of a user by communication quality as for example in Skype – the sound quality, the presence of a distortion, the appearance of echo, jitter, quality of the picture etc. There are two basic methods for QoS qualitative evaluation: Mean Opinion Score and Quality of Experience [1]. These methods provide an integrated assessment of all subjective assessment of service.

In this paper we are primarily interested in the second interpretation of the term QoS as a set of the parameters a network connection. Under term QoS requirements we will mean a set of the QoS parameters a network connection has to meet. The term QoS management we will treat as ability of network to maintain a set of connection parameters compliant with the QoS requirements of the application it is due to. Saying "connection" we mean end-to-end (e2e) connection. A set of QoS parameters includes:

- Throughput – a part of the channel bandwidth available to the particular connection;
- End-to-end delay – time is needed to deliver a packet from one source host to a destination host;
- Jitter – a deviation of the end-to-end delay from its mean value;
- Error Rates - the share of packets lost or damaged during a transmission through connection.

Different parameters of QoS play a different role for different applications. For example, multimedia application requires high throughput, videoconferencing and real time simulation – small jitter and end-to-end delay, telemedicine (distance surgery) – high throughput and low error rate.

Providing a connection with an appropriate QoS require a certain network resources. However, the network has only a limited amount of the resources to handle data flows. Thus we get a problem how to allocate network resources to meet QoS requirements of different applications operate at the same time? In practice usually there is problem connected to the previous one - what level of utilization (efficiency) of the network resources under allocation have been made? Thus, a network has to be selective while spreading bandwidths of its channels and capacities of its switching devices over the applications. Thereby, the solution for the quality of service problem we are looking for should meet the following criteria: (1) ensure compliance of granted e2e connections with the QoS requirements of applications, (2) provide a small resource fragmentation, and (3) to be a practical method delivering a suboptimal resource allocation.

Although QoS issue has been addressed since the first attempts to transmit voice over a packet switched network [2], and the community has developed a set of diverse approaches to conquer it, none of them is successful enough to be implemented by default. They are either too expensive to deploy or provide insufficient increase to the admissible utilization of a network. Thereby, the existing practices of the network management advice to obtain the missing resources by a straightforward resource extension, rather than to invest into an intricate piece of hardware, gain better control over the resource distribution and attune the performance in an intelligent way.

In this paper we propose a new approach to QoS management in SDN networks [3] based on Multi Path Routing (MPR) called MPRSDN with the following features:

- MPRSDN refuses resource reservation in favor of their efficient utilization. Thereby, it provides no strict guarantees and implements a best effort approach.
- Although we propose to construct a QoS-compliant resource allocation with a heuristic search, our approach uses a considerably large search space to allocate the resources for each of the requested connections. Thus, if it fails to meet the requirements of a given application, most likely, there are no more suitable resources left.
- It does not require specialized hardware and may be deployed in any SDN network with an appropriate

This research is supported by the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60914X0003 and Russian Foundation for Basic Research, project 14-07-00625

control over the switches. The hosts have to be preinstalled with the software agent for multipath routing enabling to involve some idling resources.

In section II we provide the comparative analysis of existing approaches to QoS management. Section III introduces the structural and operational schemes of the proposed QoS control toolset.

II. Related Work

A. Conventional QoS management

There are multiple well-known approaches to the quality of service management. Introduced by the model of Integrated Services (IntServ) [4], signaling protocol RSVP (and later NSLP [5]) provides applications with guarantees over throughput and delay of the granted connection by resource reservation at each router along the flow path calculated by a routing protocol. The reservation restricts schedule of packet handling at each affected router because the allocated resources are assigned to the flow exclusively and cannot be used even if the flow does not fully utilize them at that time. An application has to announce its QoS requirements before the connection setup and cannot modify them until the connection close. Thus, the application is forced to over pledge and reserve resources with a margin for the maximum traffic burst.

IntServ relies on static resource reservation and brakes work-conserving operation of switching devices. This results into an unnecessary resource fragmentation, similar to the one in a computer with paged allocation of RAM. As a result, in some cases network fails to supply the connection with the requested QoS even if accumulative amount of the network resources is enough to make it. The similar problem may be also caused by the independence of the signaling and routing protocols. There might be a bypass route to avoid the overloaded network component, however reservation is separated from routing and cannot take this advantage.

The model of Differentiated Services (DiffServ) [6] proposes to replace an awkward resource scheduling for end-to-end connections with predefined qualities by a local flows grading at the network devices. Each device defines a set of service classes and attributes each class with a certain QoS. Although each flow has a right to request a class with an appropriate service, the model does not provide any guarantees over the provided packet processing quality. Instead, each switch undertakes to share its resources among the flows of different classes in accordance with their relative shares. If there are no flows for a certain class of service then the resources of this class are allocated among the other classes. Thereby, switches are work-conserving and never idle when there are some packets to process. Although the application may specify required class of service for its packets explicitly, it is optional. In practice switching devices often calculate the class of service for a packet automatically by a certain set of its attributes and a mapping preinstalled by the administrator.

Differentiated Services introduce a way to deal with switch-level resource fragmentation and increase the overall network performance. However, it manages only the network resources along the primary route of an application. Thus, some idling and suitable resources away from this route are unavailable. Moreover, the class of service of the flow is set statically for the whole path. Although it is possible to improve granularity by dynamic changing of class of service at some points in the network this interference into the switching logic is beyond the capabilities of the networks of ordinary switching devices without a centralized control.

QoS-routing [7] was intended to improve allocation of network resources by constructing individual data transmission paths for each connection. Such a fine-grained routing is used to balance data flows among several paths, bypass congestion involve idling resources aside from heavy loaded channels, and take into account the QoS requirements of the application. For example, the delay sensitive traffic is usually routed along the shortest path, whereas the other flows may be forced to use the longer paths. However, a practical implementation of this method requires a low-level and centralized control over the switching devices unavailable back in time of its emergence. Moreover, QoS-routing algorithms tried to treat the problem of resource allocation as a global optimization problem with multiple constraints and their implementations were too slow to run on the fly.

B. QoS management with SDN

SDN supplies a complete control over the packet handling rules of each switch in the network, and an SDN controller may easily implement each of the mentioned approaches to QoS management without a regard to a complex distributed exchange algorithms for service data. Controller can mimic resource reservation by dynamic adjustment of traffic shaping parameters at its border switches of the network. It is also capable to collect a comprehensive set of the QoS metrics and implement a relevant QoS-aware routing on a per-flow basis, or improve capabilities of DiffServ with dynamic reassigning the class of service mark for any flow at any point of the network. Unfortunately, neither flexibility, nor convenience of SDN removes the inherent disadvantages of these methods.

SDN provides a technical capability to gather the relevant information about the network, but it is a hard task to construct a comprehensive algorithm to dispose the collected data properly. This algorithm is expected to analyze a set of heterogeneous parameters and synthesize such a set of appropriate forwarding instructions for the switches to achieve a better network performance. It is hardly believable there are real opportunities to construct routing algorithm able to work on the fly [8].

SDN does not give us any advantage to cope the problem of how to transmit QoS requirements from the user application to the Control Plane. However, this problem has been realized. FLARE [9] proposes to enable such an interaction by appending of arbitrary data to the tail of a packet and introducing corresponding handlers for the piggy-backed data at both end-host and switches. PANE [10] considers direct communication of the end-host application and the controller. On the other hand, loosening of the separation between the Data Plane and the Control Plane leads to potential security breach, and there is a lot of skepticism about its overall advantage.

Another reason for controller to avoid interference in applications communication is Internet Architecture Principles [11, 12]. As an evolutionary development of the network architecture SDN should not violate these principles. End to End principle states "The network's job is to transmit datagrams as efficiently and flexibly as possible. Everything else should be done at the fringes…" [11]. Clark explained this principle with the following words "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)" [13].

C. *Multi-Path Routing*

An SDN controller has a number of options to provide an application with a connection of an appropriate QoS: controller can route the flow through the underused links, reallocate the resources along the existing routes and/or impose stronger restrictions to the other flows. However, it requires too complicated algorithm to manage all the listed possibilities simultaneously. MPRSDN proposes to decompose this global resource management problem into a set of smaller problems with help of Multi Path Routing.

MPRSDN associate each connection with a simple module to detect violations of its QoS requirements and request the controller to supply additional resources on their occurrences. The controller module handles the requests by constructing of additional data transmission paths through the network. The set of paths granted to a connection is used to balance its packets and gain a larger amount of the resources. If controller provides connection with a path, it has not used before, there is a good chance, this path improves accumulated QoS of the connection.

There are multiple well-known approaches to implement the described splitting and balancing of a packet flow among a set of alternative paths. Routers often use Equal Cost Multi Path (ECMP) [14] to route the traffic addressed to the same destination along the different paths with equal cost. ECMP is simple to implement by distributing of the incoming packets with round-robin. However, such a naive approach to balancing results into packet reordering, the most of TCP congestion-avoidance algorithms treat as a packet loss. As a result, the size of congestion window decreases, and the original non-split connection may even outperform the balanced one. Thereby, practical balancer implementations send all the packets of a single connection along the same route. So, they are often unable to split the "elephant" flows and overcome the problem of fragmentation at the data channels.

In contrast to ECMP, Multi Path (MP) TCP [15] follows the End to End principle and proposes to split a single TCP session into smaller virtual sessions at the end hosts. MP TCP operates transparently for an application. Upon the setting up of the connection, it creates a static set of internal sockets. Each of these sockets is used to establish an individual connection trough the network. MP TCP balances the packets among this set of connections and uses an original congestion-avoidance algorithm to cope inter-connection packet reordering without a significant performance drop.

Although MP TCP implements an automatic adjustment for the packet ordering, it does not provide any means to ensure the allocated internal connection use different paths. Existing implementations of MP TCP send the information about the original connection the packet within an optional L4 field the most of network devices unable to distinguish. Thereby, flows of the same application are most likely to take the same path. This fact cancels all the advantages of a multipath routing, until the sender or/and receiver has multiple interfaces connected to different networks.

Fortunately, flexibility of SDN networks can surmount the disadvantages of MP TCP. Controller may easily detect a new connection is setting up by intercepting its first packet; get any of its attributes including the data stored inside of the payload; find out the original application connection it belongs to, and minimize intersection of its route with the other flows of the same connection.

III. QUALITY OF SERVICE IN MULTI PATH SDN

The paper refers a middleware designed to split a single Application Flow (AF) into a set of Sub Flows (SF) and multiplex these SFs into a single AF as a Multi Flow Agent (MPA). For a given AF, we will call the AF degree a number of SFs, carrying its data.

Each SF establishes a connection between a pair of unique L4 addresses: one at the source and one at the destination host. Network switches are supposed to distinguish different SFs by their headers and treat each of them as an ordinary and independent flow. In particular, each SF may attribute its packets with a higher TOS/DSCP mark and get a better service as compared to the other SFs of the same AF.

Although MP TCP agent may be considered as an example of MPA, we imply the latter to be a more general term. Different MPA implementation may go over TCP and provide the similar multi path transmission to other protocols, modify the number and intensity of SFs dynamically without the need to reestablish the parent AF, rate-limit or shape individual SFs with some arbitrary algorithms, and interact with an SDN controller explicitly or implicitly.

To design an efficient implementation of the MPRSDN one should answer on the following questions:

- How to retrieve the QoS requirements for an application?
- How to monitor and properly estimate the quality of the granted connections?
- How to keep connection properties compliant with the QoS requirements of applications by MPA?
- How should MPA and SDN controller interact?

A. *Deriving QoS requirements*

MPRSDN does not use the greedy approach. It requests extra resources dynamically and only when it founds that there

is a risk to violate the QoS requirements. Thus, it allows application to release the sparse part of the previously acquired resources and request the missing resources without reestablishing of the connection. For example, a network video-streaming application may loosen its requirements to the connection, while playing static scenes, and increase them at the moments of active motions.

Thereby, there is an issue, how to retrieve the initial QoS requirements of the application and how to modify them during the MPA operation? There are two options to resolve this problem: (1) make application to specify its QoS requirements through a socket-level API, or (2) derive these requirements from some application profile.

Using of the socket-level API results into a considerable complication of network programming for the application developer. Although this kind of effort may result into a reasonable benefit for applications with severe dependency on the connection QoS, in many cases this functionality will be considered as unnecessary and obscuring.

Transparent deriving of the application requirements does not imply any extra effort by the developers, and has more perspectives to be generally accepted. However, the only connection characteristic that can be estimated transparently is its intensity. This kind of data may be sufficient to derive the required bandwidth, but it does not allow estimate the other QoS characteristics such as a transmission delay.

B. Monitoring of a connection QoS

SDN controller has comprehensive possibilities to monitor QoS of an e2e connection. There are some researches devoted to constructing and maintenance of a traffic matrix formed by an enumeration of bandwidths consumed by each of the end-host applications [16] and measurement of one-way delay for an arbitrary flow while it moves through the network infrastructure [17]. However, a comprehensive fine-grained measurement imposes a frequent polling of the devices and results into excessive loading of both network devices and the controller. There are some attempts to reduce intensity of the controller requests to the devices by using the dead reckoning estimation [18]. The idea is to use a simple network model to approximate parameters of interest between the measurements and reduce their total number. However, the simulation of a network with an appropriate accuracy often results into even higher requirements to computation power of the controller.

As a result, controller has to delegate part of its monitoring functions to MPAs. However, monitoring at hosts becomes rather challenging, especially in case of a UDP-like half-duplex connections. UDP sender does not know the amount of packets dropped and both the connected hosts are unaware of an actual network delay value. In practice, this problem is usually moderated by wrapping the raw application data into RTP protocol [19]. It establishes an additional RTCP connection to send periodic statistics backwards from destination to source, and reduces the case of half-duplex connections to the simpler full-duplex one. TCP-like connection allows the hosts to detect bandwidth shortage by the amount of the lost packets and infer a one way delay of the connection from the RTT provided by the underlying congestion avoidance algorithm.

C. QoS management with MPRSDN

MPRSDN provides two ways to meet QoS requirements: adjustment of the number of SFs in the AF and individual regulation of their service classes. Upon QoS violation MPA scales AF partitioning and/or steps up the service for some of its SFs. Upon detecting excessive overprovisioning MPA rollbacks the parameters to avoid unnecessary overhead and simplify the AF maintenance.

The listed QoS management means are independent of each other, and may be applied in any order. However, one sequence may be superior in the first set of cases, while the other is more efficient in another set. Thus, it makes sense to develop a set of strategies to regulate the properties of some SFs and adjust their number for different types of requirement violations in a most efficient way. A set of appropriate MPA heuristics may include the following examples:

- When accumulated bandwidth of the SFs subsides, some network channel is likely to become congested. In this case rise in classes of service for the SFs with the lower throughput is usually less efficient than increase in the number of the SFs.
- If the estimated AF delay exceeds the allowed upper limit, MPA should accelerate the slowest of its SFs. One way to accomplish this task is to give up using this SF and reallocate its data among the others.
- If the violation is due to a change in the requirements of an application, there are no reasons to increase the degree of AF partitioning. Thereby, MPA should cover the lack of resources by rising of QoS requirements for some of the existing SFs in the first place, and consider increasing of SF number to be an auxiliary leverage.

D. Communication between an MPA and and SDN controller

SDN provides two different ways to install forwarding rules into the network devices: the proactive and the reactive one. The former one implies an SDN controller foresees the need in some paths through the network and sets up appropriate rules in advance. Any packets that match these rules are transmitted by the devices autonomously without further involvement of the controller. Thus, it is unable to track the establishment of new connections directly. The reactive approach implies the border network devices request packet processing instructions from the SDN controller upon receiving a packet without a match among the existing rules.

In order to support multipath routing an SDN controller should identify individual SFs of a single AF and provide them with different paths. This requires the controller to react MPA in dynamic. Thus, the controller either has to provide MPAs with ability to connect it directly through a dedicated channel, or operate in the reactive mode. Since the former one implies mixing of Data and Control planes and requires a fundamental change of the interaction between the host and the network, we give preference to a more practical second option.

While requesting controller for instructions to process a packet of an unknown flow, switching device either provide