<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *IEEE ICC 2019: IEEE International Conference on Communications 2019 Shanghai, China 20-24 May*.

Citation for the original published paper:

Bhamare, D., Kassler, A., Vestin, J., Khoshkholghi, M A., Taheri, J. (2019)
IntOpt: In-Band Network Telemetry Optimization for NFV Service Chain Monitoring
In: *2019 IEEE International Conference on Communications (ICC) Proceedings*
<https://doi.org/10.1109/ICC.2019.8761722>

N.B. When citing this work, cite the original published paper.

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-74631>

in the substrate network on which SFCs are routed. The proposed framework uses active telemetry probing [12], that is, inserting separate monitoring probes for the MFs periodically in the network to gather telemetry information from the data-plane using programmable data plane elements and P4. The sink nodes in the physical network send the probes along with the collected telemetry information back to the controller for further analysis. The IntOpt controller executes its commands by communicating them through the SDN controller, which in turn communicates with the underlying physical switches through the control plane [13]. For input to our meta-heuristic, we benchmark the P4 INT framework using P4FPGA [14] to approximate the delay induced by INT-operations.

The remaining article is organized as follows. In section II we discuss the related work in brief. In section III we discuss IntOpt architecture with a motivational example. In section IV we discuss the implemented heuristics. Section V presents the heuristic results while section VI concludes the paper addressing future directions.

II. RELATED WORK

Many solutions have been proposed in academia as well as industry for network monitoring. However, existing solutions mainly focus on trade-off between expressiveness, accuracy, speed and scalability [1], [5]. For example, systems such as NetQRE [7] and others can support a wide range of queries using stream processors running on general-purpose CPUs, but they incur substantial bandwidth and processing costs to do so. Telemetry systems such Chimera [6] and Gigascope [8] are expressive in nature by covering wide range of telemetry items, however, can only support lower packet rates. This is because these systems process all packets at the stream processor which can become a bottleneck.

On contrary, telemetry systems that rely on programmable switches alone can scale to high traffic rates, however, they can accommodate a limited set of telemetry items in order to achieve the scalability. For example, Sketchvisor [10], UnivMon [15] and OpenSketch [11] can perform telemetry tasks by executing queries solely in the data-plane at line rate, but the queries that they can support are limited by the computational capabilities and memory in the data-plane, sacrificing the expressiveness and accuracy.

Systems such as ElasticSketch [16], Marple [17] obtain a good balance between the expressiveness and scalability, however, they incur substantial processing overheads, delays and traffic overheads. To overcome this problem, in this work, we propose an approach to minimize the overhead associated with monitoring so as to make the underlying monitoring framework scalable as well as expressiveness.

Authors in [18] propose use of piggyback technique to monitor network statistics. However, we argue that the traffic in NFV and SFC architecture is unpredictable and hence might come in bursts. Due to this, piggybacking may fail to deliver accurate per-flow statistics. In this work, we advocate active telemetry probing [12], since it is an effective way to perform network monitoring. It is especially effective in the dynamic service chaining architecture, since each service chain may have been allocated different network slices with different QoS requirements undergoing different treatment in

the data-plane. Inserting active probes, however, can be expensive and may lead to queue buildups, buffer-bloat, packet drops and network congestion as well as delays, especially if it is performed in unplanned ad-hoc manner. To minimize the overhead associated with active probing, we develop a simulated annealing based random greedy meta-heuristic (SARG) that determines the optimal set of monitoring flows (MFs) in order to fulfill all the monitoring requirements of service flows (SFs). In the subsequent sections we discuss our proposed IntOpt architecture along with the proposed SARG meta-heuristic in more details.

III. ACTIVE PROBING OPTIMIZATION

A. INT - InBand Network Telemetry

An important goal of any network monitoring framework is to enable fine-grade monitoring at scale. Typically, such monitoring frameworks involve expensive interaction with the control plane and are either not scalable or lack expressiveness. With recent advances in programmable data-plane devices based on P4 [8] together with compiler and runtime support, adding monitoring support in the data plane becomes possible at line-rate. The main idea of the INT Framework [9] is that programmable data plane elements (e.g. a P4 capable router that runs the INT framework) can add telemetry instructions to individual monitoring probes, which the switch would parse and understand, in order to collect desired telemetry items. Such monitoring information may include packet queuing time, port utilization, etc. and would be added as custom headers for each probe using the INT-framework.

Programmable data plane devices complying with the INT standard are divided into three categories: INT sources, INT forwarders and sinks. INT sources add telemetry instruction headers thus instructing forwarding nodes, which telemetry items to collect. Using P4 framework, INT forwarders parse instruction headers, collect relevant telemetry information and add it to the packet headers. Finally, INT sinks remove telemetry headers and forward collected information to a data sink (e.g. stream processor) where further processing can be done (e.g. detecting per switch micro bursts, applying machine learning on collected state information, etc.). Using the INT framework, packets collect observed network state from INT enabled devices and aggregate such state in flexible headers while traversing the network [9], [12].

B. IntOpt Architecture

The proposed IntOpt architecture is shown in Fig. 1. IntOpt controller communicates with SDN controller using East-West interfaces. It retrieves information such as underlying physical topology, physical links as well as the SFCs, their actual deployment over the physical nodes and their monitoring demands from the SDN controller (black double-dashed line from SDN controller). The figure shows a network topology with six (SW1 to SW6) physical switches. Also, two service flows, flow A-Z and flow B-Y are deployed through these switches as shown in Fig. 1.

The IntOpt controller maps service flow (SF) telemetry items and frequency demands to the respective physical links. It then finds out the optimal probing frequency as well as total telemetry items which need to be monitored for each

Figure 1. IntOpt Architecture

physical link in order to cover all SFs with minimal overhead at the data plane as well as the controller. The controller then prepares an optimal set of monitoring flows (MFs) so that all the given SFs are monitored along with their monitoring demands in terms of telemetry frequency and telemetry items. The IntOpt controller performs these tasks by executing the SARG meta-heuristic proposed in this work. The details of the SARG meta-heuristic are explained in section IV.

The IntOpt controller then identifies telemetry sources (SW1 in Fig. 1), forwarders (SW2-SW5) as well as sinks (SW6), and commands the SDN controller to populate flow tables accordingly (black dashed line). Since we are using active telemetry probing to monitor the network, IntOpt also commands SDN controller to send the periodic monitoring probes for each monitoring flow as per the telemetry frequency determined in the previous step (black double-dashed line to SDN controller). P4 programmable switches are responsible for parsing the monitoring probes, inserting the telemetry items and forwarding of the probes to correct output port (red dotted line). The Controller acts as data-sink by instructing the sink switch (SW6) of each monitoring flow to forward the collected information to itself through SDN controller (red double dashed line). It then maps the collected telemetry information back to individual SF requirements to check any SLA violation, such as exceeding the total delays or buffer queue size at the intermediate switch etc..

C. Optimized Active Telemetry Probes Generation

In this sub-section we illustrate our concept of preparing optimal monitoring flows (MFs) with a toy example. We have considered six different service function chains (SFCs) as service flows (SFs) as shown in Fig. 2, with 15 virtual functions (VFs), numbered from 1 to 15. The numbers on SFC blocks (inside circles and rectangles) indicate the VFs that particular SFC is comprised of. Please note that SFCs may share the VFs. SFCs may have different sizes and shapes as shown in Fig. 2. This may be due to back and forth traffic flows among the VFs and their execution order. We have considered different complex SFC shapes to make the solution generic [2], [19]

While deploying the monitoring flows, however, we only consider linear flows. That is, we don't allow forking, or

Figure 2. Service Flows and deployment over Atlanta network.

loop formation in MFs. The probes can simply be forwarded on to the port which is inserted as a next hop in telemetry header. Since we propose to perform mapping and extraction of telemetry data at the controller, similar to the approach proposed in [5], implementing linear MFs reduces the controller overhead while preparing the MFs as well as gathering the collected information from the probes and mapping it back to the SFs. Also, this is typically handy allowing the probe forwarding logic at the switches to be simple and fast.

Let us consider a 15-node Atlanta topology from SNDLib [20] for deployment of the six SFs. A possible deployment of SFs over the given substrate network is given in Fig. 2. To illustrate our approach, we focus on a specific SF, SF1, with blue rectangular VFs. Deployment of SF1 over the substrate network is shown in Fig.2 with double dashed blue lines. Let's assume for simplicity we implement separate monitoring flow for each SF, following its exact shape on the substrate network. As a result, monitoring flow MF1 for SF1 will also follow the same path as SF1. As we notice, at node 15, the MF1 has to split and the probes gets forwarded to two nodes, node 8 and 9. If we aim to implement simple "next-hop look-up and forward" functionality at the intermediate switches, then it becomes complex to keep track of the next forwarding port due to split-up of the MF. One way to achieve this is to let the controller keep track of switch unique ID (such as MAC) and its forwarding port on that switch for that particular MF and embed the whole information in the probe. Each switch then performs the match and forwards the monitoring probe accordingly.

The major drawback with such scheme is the delays incurred due to the processing overhead at intermediate switches. Also, the probe size increases as more data needs to be embedded (here MAC of the switch) at every hop, adding to the overhead. Alternatively, we can implement two linear MFs, that is, MF1 with path as (15-9-10-7-14) and MF2 with path (15-8-1-7-14). In this case, forwarding is linear and simple. A simple next-hop port number can be inserted in the probe to guide the intermediate switch to forward the probe to the next hop. However the solution is still non-

Figure 3. 5 monitoring flows to cover all the service flows.

optimal as the link $E_{7,14}$ is covered twice unnecessarily by the probes. Such overhead due to non-optimal deployment of monitoring flows may increase significantly with increase in the number of service flows. In this simple case with a single SF, the optimal solution would be to deploy two linear MFs with paths as (15-9-10-7-14) and (15-8-1-7). With this motivational example, we demonstrate that it is sufficient to deploy 5 monitoring flows as shown in Fig. 3, so that all the given SFs can be monitored. As we note that, all the used physical links in the topology of Fig. 2 are covered by at least one MF in Fig. 3. We cast this as an optimization problem and develop a simulated annealing based random greedy meta-heuristic (SARG) approach to prepare optimal set of MFs as explained in the next section.

IV. ALGORITHMS FOR ACTIVE PROBING OPTIMIZATION

In this section we develop simulated annealing based random greedy heuristic that determines the optimal set of monitoring flows (MFs) in order to fulfill all the monitoring requirements of service flows (SFs) while minimizing the overhead. We also benchmark the INT framework using P4FPGA, as explained in Section V. We now explain our SARG approach to prepare optimal set of MFs. As a first step towards preparing the optimal set of MFs, we prepare set E of links covered by the SFs and map the SF telemetry items and frequency demands to the respective physical links. Then we calculate the minimum number of MFs to monitor all the links in the set E . To achieve this, the heuristic prepares two sets, μ_{ij} and δ_{ij} . μ_{ij} denotes a strict bound for the telemetry frequency demands and δ_{ij} denotes a strict bound for the telemetry item demands for all SFs passing through the link E_{ij} . We implement a pre-processing step at the controller and maintain separate data structures to keep track of μ_{ij} and δ_{ij} .

We now demonstrate the above concept with an example that shows a simple substrate network and three service flows (SFs) in Fig. 4. Let us denote the sets of telemetry items demanded by SF1, SF2 and SF3 as set S_1 , S_2 and S_3 respectively. The frequency and telemetry item demands for each SF are given in table I, with values selected randomly. Relationship between telemetry item demands for each SFs is shown by Venn diagram in Fig. 5. That is, S_2 , a set of

Figure 4. Service Flow demands to link demands mapping.

Figure 5. Telemetry item demands for service flows.

10 telemetry items demanded by SF2 is a super-set of S_1 , 5 items demanded by SF1. However, S_3 for SF3 is intersecting S_1 and S_2 as shown in Fig. 5, with 2 items in common with SF1 and 3 more items in common with SF2 (in total 5 items common with SF2).

Table I
SERVICE FLOW FREQUENCY AND TELEMETRY DEMANDS

Service Flow	Frequency (ms)	Telemetry Items
SF1	5	5
SF2	1	10
SF3	10	10

The pre-processing step maps SF monitoring demands to link demands and fills sets μ_{ij} and δ_{ij} as shown in table II. As we observe, if only one monitoring flow is passing through the link, then telemetry frequency demand mappings are straightforward (such as links E_{12} , E_{35} , E_{56} and E_{46}). However, for links accommodating more than one SF, we need to determine the appropriate mappings. For example, monitoring frequency demand for link E_{23} should be 1 ms since, from table 1, it is the most strict telemetry frequency demand for all the SFs passing through link E_{23} . Also, it will cover the telemetry frequency demands for other SFs, which are greater than 1 ms. Similarly, for the link E_{34} monitoring frequency demand should be 5 ms as it is more strict (5 ms) compared to the other (10 ms), and so on.

Table II
FREQUENCY DEMANDS MAPPINGS

Link	Frequency (μ_{ij})	Telemetry Items (δ_{ij})
E_{12}	5	S_1
E_{23}	1	$S_4 = S_1 \cup S_2 \cup S_3$
E_{34}	1	S_2
E_{35}	5	$S_5 = S_1 \cup S_3$
E_{46}	1	S_2
E_{56}	10	S_3

Similarly, we map the telemetry item demands of the SFs to the link demands (column 2 in table II). For example,

we observe that on the link E_{23} , the set S_2 with 10 items for SF2 also covers S_1 with 5 items for SF1 (as $S_1 \subset S_2$ as mentioned earlier). However, every 10^{th} ms, we need to insert a new set S_4 of telemetry items in the monitoring flow over E_{23} such that $S_4 = S_1 \cup S_2 \cup S_3$. This is because SF3 has some telemetry items in S_3 which are not covered by S_2 . As we can see, the size of S_4 is 15 telemetry items.

Once the mappings are done, *Random Greedy* procedure maps link monitoring demands to MFs so that all SFs with the specified SLAs are covered while the number of total MFs are minimized. Algorithm 1 illustrates the steps for our Random Greedy policy. We begin by initializing a monitoring flow and adding any random link to it. We keep adding more links to the MF by selecting them sequentially thereafter, given the link being added has similar or less strict monitoring demands than the existing set of links in the given MF. This is repeated until the size of the MF grows beyond the threshold. At that instance, the heuristic terminates the monitoring flow and start a new one. The process is repeated until all links are covered.

Algorithm 1 Random Greedy Approach integrated with Simulated Annealing

---

```

procedure RANDOM_GREEDY
   $E \Rightarrow$  set of the edges covered by all service flows
   $\lambda_f \Rightarrow$  monitoring flow size threshold
   $E_{ij} \leftarrow \text{Random\_Select}(E)$ 
  Initialize a monitoring flow  $m_f$ 
  Monitoring Frequency( $m_f$ )  $\leftarrow \mu_{ij}$ 
  Telemetry Items( $m_f$ )  $\leftarrow \delta_{ij}$ 
  Set  $E_{ij}$  as a start link of  $m_f$ 
  while  $E \neq \Phi$  do
     $E_{jk} \leftarrow \text{Sequential\_Select}(E)$ 
    if  $\mu_{ij} = \mu_{jk}$  and  $\delta_{ij} = \delta_{jk}$  then
       $m_f \leftarrow m_f + E_{jk}$ 
       $E \leftarrow E - E_{jk}$ 
    else
      terminate  $m_f$  and initiate new flow  $m_{f+1}$ 
      break the for loop
    end if
    if  $\text{Size}(m_f) > \lambda_f$  then
      terminate  $m_f$  and initiate new flow
      break the for loop
    end if
  end while
end procedure

```

We develop a simulated annealing based meta-heuristic shown in Algorithm 2, which prevents the random greedy approach from getting stuck in local minima. The *quality* attribute of the solution returns the total number of en-cap/decap plus forwarding instances at the data plane due to the proposed MF deployment scheme in the given solution. It has been used as the fitness function (explained in more depth in Section V) for comparison of the solutions.

We also implement an ad-hoc approach, which we denote as naïve algorithm, which is unaware of the optimization policies and just tries to avoid forking or looping of the MFs, which is the basic requirement for MF to be a valid flow. In the naïve implementation, we just start the MF for each SF and follow it linearly. If there is any forking or loop formation in the SF, we just break the existing MF and form a new one. This is the typical approach which is generally

---

Algorithm 2 Simulated Annealing meta-heuristic steps

```

procedure SIMULATED_ANNEALING
   $temperature \leftarrow \lambda$ 
   $cooling\_rate \leftarrow \alpha$ 
   $previous\_sol \leftarrow new\_sol \leftarrow best\_sol \leftarrow NULL$ 
  while  $temperature > 1$  do
     $new\_sol \leftarrow \text{Random\_Greedy\_Procedure}()$ 
    if  $e^{(new\_sol.quality - prev\_sol.quality)/\lambda} > \text{Random}(0, 1)$  then
       $prev\_sol \leftarrow new\_sol$ 
      if  $new\_sol.quality > best\_sol.quality$  then
         $best\_sol \leftarrow new\_sol$ 
      end if
    end if
     $temperature \leftarrow temperature \times (1 - cooling\_rate)$ 
  end while
end procedure

```

---

followed in the absence of any sophisticated algorithm for the MFs formation. Steps for the naïve approach are given in Algorithm 3. In the next section, we analyze the benchmarking results as well as numerical results obtained through simulation of the two algorithms discussed.

Algorithm 3 Naïve ad-hoc Approach

```

procedure NAIVE_APPROACH
   $s \leftarrow \text{Sequential\_Select}(R)$ 
   $E \Rightarrow$  set of the edges covered by service flow  $s$ 
   $\lambda_f \Rightarrow$  monitoring flow size threshold
  Initialize a monitoring flow  $m_f$ 
   $E_{ij} \leftarrow \text{Sequential\_Select}(E)$ 
  Set  $E_{ij}$  as a start link of  $m_f$ 
  while  $E \neq \Phi$  do
     $E_{jk} \leftarrow \text{Sequential\_Select}(E)$ 
    if (no loop or fork) and  $\mu_{ij} = \mu_{jk}$  and  $\delta_{ij} = \delta_{jk}$  then
       $m_f \leftarrow m_f + E_{jk}$ 
       $E \leftarrow E - E_{jk}$ 
    else
      terminate  $m_f$  and initiate new flow  $m_{f+1}$ 
      break the for loop
    end if
    if  $\text{Size}(m_f) > \lambda_f$  then
      terminate  $m_f$  and initiate new flow
      break the for loop
    end if
  end while
end procedure

```

V. NUMERICAL RESULTS

In this section we first evaluate the monitoring overhead of INT-operations by benchmarking the P4 implementation with P4FPGA. We use the benchmarking results to compare our proposed SARG scheme with ad-hoc naïve approach in terms of monitoring overheads and present our numerical evaluations.

In order to approximate the delay induced by INT-operations, which is mainly due to packet parsing and pushing telemetry headers, we performed a series of experiments using the NetFPGA-SUME hardware platform. We implemented a P4 program using the P4FPGA toolkit [14], based on the INT specification, which parses incoming packets, and pushes INT headers (encapsulation), accordingly [9]. The headers are divided into two types, Telemetry Instruction and Telemetry Data headers. Instruction headers contain a set of instructions that determine, which telemetry data items

should be pushed by each switch, along with various meta-data. Telemetry data headers contain actual INT data such as queue occupation, switch traversal latency, etc. The P4 INT program checks incoming packets for instruction headers, and if found one, it pushes the telemetry data specified in the instruction header. Should a packet arrive without an instruction header, the switch can be configured through the control plane, to either forward it as normal packet or insert a telemetry instruction header. This is configured through a match key, such as source port or destination address.

We ran experiments with both edge and core switch P4 programs, pushing from 0 to 8 INT data headers. Edge switches were configured to push both the instruction header and the configured number of data headers, while the core switches only pushed data headers. Fig. 6 shows the experienced delays against the number of header fields inserted for core as well as edge switches. As we observe, there is a linear relationship between the telemetry data pushed in the packet and the delays observed. Also, the edge switch has higher latency due to additional push operation for instruction headers.

The tasks performed at the switches while monitoring probes are passed through them and which cause the significant delays are (1) encap/decap of the probes with telemetry headers at the source and the sink and (2) parse the telemetry header and insert the telemetry items accordingly (which we call Forwarding or FW for simplicity). We now evaluate the performance of our proposed SARG scheme and the naïve approach in terms of total encap/decap plus FW instances as well as total delays incurred due to such operations while all the probes pass through the network. We apply a linear fit with the delay results obtained using P4-NetFPGA switches (Fig. 6) to extrapolate delay values for larger topologies.

We have used Europe topology from SNDLib which has 37 nodes in total and 178 links [20]. We generate the service chains randomly. We also select specific parameters for SFs such as average length in terms of hops, telemetry items demands and telemetry frequency demands randomly from specific ranges given as an input to the heuristic. We also vary the total number of actual SFs to be deployed in the network to observe the effect on the final results.

We have considered two different cases for average hop-length and telemetry item demands for the SFs. That is, the first case with average hop-length as 5, telemetry items needed are 10 and telemetry frequency demand as 5 ms. For the second case, the hop-length as 10, telemetry items needed are 10 and the telemetry frequency demand is 10 ms. Fig 7 shows the total encap/decap plus forwarding instances in the given network along the Y-axis against different number of service flows along the X-axis. In Fig. 8 we plot the graphs for the total delays incurred due to the monitoring flows. For this purpose, we use the overhead values obtained from the experimental benchmarking of P4 INT framework on the NetFPGA-SUME hardware as discussed above.

As we observe, in Fig. 7, the total encap/decap plus FW overhead is minimum for the proposed SARG scheme. For example, for 25 SFs with case 1 (10H/5L), the number is 100 (green squared solid line), however for the naïve approach the number is 163 (red circled dotted line). For case 2 (10H/10L) the numbers are 122 and 210 for SARG and naïve approach

Figure 6. NetFPGA results for Core and edge switches.

Figure 7. Total encap-decap and FW instances.

respectively. Corresponding delays for case 1 as shown in Fig. 8 are, 300 and 700 micro-seconds for SARG and naïve approach respectively. For case 2 the delays are observed to be 400 and 800 micro-seconds. We also observe that such delays increase with the increase in average hop length of the SFs (green squared solid line vs. blue triangle solid line). In Fig 9 we keep the number of total number of SFs constant to 50 and vary the telemetry item requirements along the X-axis. We observe the linear growth, which is due to the linearly growing overhead for the P4 header pushing operations. The results show that the proposed SARG approach can reduce the monitoring overhead by 39% and the total delays by 57%. Numerical evaluation demonstrates that, with systematic approach such as SARG, the monitoring overheads can be reduced significantly.

VI. CONCLUSIONS

In this work, we propose IntOpt, a scalable and expressive telemetry framework and develop simulated annealing based random greedy (SARG) meta-heuristic for optimally deploying the MFs for flexible service function chain monitoring. The IntOpt controller prepares optimal set of MFs by executing the proposed SARG approach, calculates the optimal probing frequency as well as total telemetry items to be monitored for each link in order to cover all service flows with minimal overhead at the data plane as well as at the controller. The controller then identifies proper telemetry sources, forwarders as well as sinks, and populates flow tables

Figure 8. Total delays against total SFs.

Figure 9. Total delays against total telemetry items.

accordingly through the SDN controller. In addition to our proposed SARG meta-heuristic, we also implement an ad-hoc naïve approach, which is generally followed in the absence of a systematic flow generation strategy. We benchmark the actual incurred overheads and latency due to telemetry operations using the P4 INT framework, P4FPGA and SUME hardware platform for a variety of telemetry items. Our evaluation shows that using our heuristic significantly reduces the total monitoring overhead and the delays introduced due to the telemetry operations. We argue that such systematic approach can be incorporated with the existing monitoring frameworks to obtain scalability without losing the generality and expressiveness of the systems.

As a future work, we aim to develop an optimization model to obtain the optimal number of monitoring flows and compare the heuristics against the optimal solution as well as consider different substrate network topology. We also aim to provide an architecture to integrate the proposed scheme with the existing networking architectures such as ONAP.

VII. ACKNOWLEDGMENT

The authors received partial funding from the Knowledge Foundation of Sweden through the Profile HITS (Grant Number 20140037).

REFERENCES

[1] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide

measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 561–575, ACM, 2018.

[2] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, vol. 102, pp. 1–16, 2017.

[3] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[4] D. Bhamare, A. Erbad, R. Jain, M. Zolnari, and M. Samaka, "Efficient virtual network function placement strategies for cloud radio access networks," *Computer Communications*, 2018.

[5] A. Gupta, R. Harrison, A. Pawar, R. Birkner, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven network telemetry," *arXiv preprint arXiv:1705.01049*, 2017.

[6] K. Borders, J. Springer, and M. Burnside, "Chimera: A declarative language for streaming network traffic analysis," in *USENIX Security Symposium*, pp. 365–379, 2012.

[7] Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, and B. T. Loo, "Quantitative network monitoring with netqre," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 99–112, ACM, 2017.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[9] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[10] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 113–126, ACM, 2017.

[11] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *NSDI*, vol. 13, pp. 29–42, 2013.

[12] P. Lapukhov and R. Chang, "Data-plane probe for in-band telemetry collection," Internet-Draft draft-lapukhov-dataplane-probe-01, Internet Engineering Task Force, June 2016. Work in Progress.

[13] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[14] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4fpga: A rapid prototyping framework for p4," in *Proceedings of the Symposium on SDN Research, SOSR '17*, (New York, NY, USA), pp. 122–135, ACM, 2017.

[15] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with unimon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 101–114, ACM, 2016.

[16] A. Kumar, M. Sung, J. J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, pp. 177–188, ACM, 2004.

[17] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 85–98, ACM, 2017.

[18] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *International Conference on Passive and Active Network Measurement*, pp. 31–41, Springer, 2013.

[19] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Multi-objective scheduling of micro-services for optimal service function chains," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2017.

[20] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0-Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007)*, Spa, Belgium, April 2007. <http://sndlib.zib.de>, extended version accepted in *Networks*, 2009.