Flow Allocation in Industrial Intent-based Networks

Conference Paper · December 2019

DOI: 10.1109/ANTS47819.2019.9117936

Barun Saha
Hitachi ABB Power Grids
31 PUBLICATIONS **151** CITATIONS

Flow Allocation in Industrial Intent-based Networks

Barun Kumar Saha
Corporate Research Center
ABB Global Industries and Services Pvt. Ltd.
Bangalore, India
barun.kumarsaha@in.abb.com

Luca Haab
Power Grids, PGGA-P
ABB Schweiz AG
Bern, Switzerland
luca.haab@ch.abb.com

Łukasz Podleski
Power Grids, PGGA-P
ABB Schweiz AG
Bern, Switzerland
lukasz.podleski@ch.abb.com

***Abstract*—Intent-based Networks (IBNs) are expected to add various degrees of autonomy and intelligence at different stages of a network, such as planning and operating. A particular use case of IBNs relevant to industry is that of end-to-end flow (service) allocation prior to commissioning such networks, for example, in power grids. In this context, we investigate the flow allocation problem for large networks by considering four schemes. In *Shortest Path-based Allocation* (SPA), all-pair shortest paths in a network are computed, which are then used to allocate the feasible flows. To prevent near-maximal link utilization, we consider a variation of SPA with usage *threshold* (SPA-T). The two other schemes, SPA with *probabilistic Hill Climbing* (SPA-HC) and SPA with *Simulated Annealing* (SPA-SA), also impose a similar utilization limit on each link. Moreover, SPA-HC and SPA-SA allocate the flows in a way to improve the relative *fairness*. Results of performance evaluation using data from real-life and synthetically generated networks show that the proposed schemes can allocate 86%–99% flows when link utilization threshold is varied from 0.75 to 0.95.**

***Index Terms*—Intent-based Networks, flow allocation, meta-heuristics, hill climbing, simulated annealing, shortest path**

I. Introduction

IBNs [1]–[4] present the next wave in the evolution of communication networks. IBNs promise many interesting features, such as specification of network operations in high-level languages [1] and continuous verification of a network's operating state. In other words, with IBNs, future networks will have various degrees of autonomy and intelligence inbuilt within them.

IBNs offer several industrial use cases [2] targeting different stages of a network's lifetime, for example, planning, commissioning, and maintaining operations. A particularly interesting one is that of planning and commissioning a new industrial network, for example, in oil and gas fields and power grids. Before deploying at a customer's site, extensive planning is done taking into account the overall network utilization. Moreover, mission-critical networks typically have networking devices, such as switches and routers, pre-configured and flows allocated. In particular, given a set of required network flows (or services), each (feasible) flow is mapped to an end-to-end path in the network. Accordingly, relevant rules are installed in the concerned devices. Moreover, Industry 4.0 is moving toward Time Sensitive Networks (TSNs) [5]. TSNs need to schedule flows based on input from field device controllers. The flow allocation service of IBNs can help TSNs in this regard. Moreover, operational IBNs can also use this solution to plan future allocation of additional network flows.

Integral flow allocation (i.e., when no flow is split across multiple paths) is a combinatorial optimization problem, which becomes difficult—and costly, in terms of time and skill needed by network engineers—to solve for large networks. This is especially true for power grid networks, which often contain a few hundred densely connected nodes. Motivated by these challenges, in this paper, we investigate the problem of flow allocation in large networks with finite link capacities.

We begin by considering a generic scheme, SPA, where all-pair shortest paths in the network are computed. Subsequently, feasible flows, which do not cause violation of link capacity, are allocated along those shortest paths. However, such a greedy strategy, can make certain links operate at their near maximum capacity. As a consequence, any increase in instantaneous bit rates of one or more flows will potentially lead to packet loss and delay. To mitigate these issues, we consider SPA-T, where utilization of each link is limited up to a threshold $\theta \in [0, 1]$.

The SPA-HC and SPA-SA schemes are based on metha-heuristics. In HC, one iteratively keeps choosing a relatively better solution. However, in probabilistic HC, one also chooses a worse solution with a small probability. SA [6], [7], on the other hand, is modeled after the physical process of heating a metal to a high temperature and then cooling it down very slowly. Such a process helps in eliminating or reducing defects. Since HC and SA both are well studied and characterized, we find them suitable for industrial usage.

Each step of SPA-HC and SPA-SA compares two or more solutions and selects a relatively better one. The (small) probability of selecting a bad solution is fixed in SPA-HC, whereas it varies with iterations in SPA-SA. A "solution" is constructed by considering the potential allocation of a very few candidate flows. The value of a solution is a quantitative measure of fairness in overall flow allocation. When a solution is selected at the end of an iteration, the corresponding candidate flows get allocated to the concerned shortest paths.

The scope of this work is limited to wireline networks. Moreover, the proposed flow allocation strategies are executed at a logical level; we do not discuss how the routing rules are actually installed in physical devices.

A. Contributions

The specific *contributions* of this work are as follows.

- Proposing greedy and metaheuristics-based algorithms for efficient flow allocation in large networks.
- Using a threshold, θ, to limit the maximum utilization of any link and thereby, help them decongest.
- Evaluating the efficiency of the proposed schemes via simulations using data from real-life and synthetically generated networks.

B. Organization

The remainder of this paper is organized as follows. Section II presents a summary of the related works. Section III presents the system model and discusses the flow allocation problem. The greedy algorithms SPA and SPA-T are discussed here. In Section IV, we discuss the proposed metaheuristics-based algorithms for flow allocation. Experimental set-up and performance metrics are discussed in Section V. The corresponding results are presented in Section VI. Finally, Section VII concludes this paper.

II. Related Work

Optimal flow allocation is a key requirement not only for communication networks [8]–[11], but also transportation networks [12]. The large volume of works on flow allocation and scheduling only underscore how fundamentally important it is for communication networks.

Hartert et al. [13] developed Declarative and Expressive Forwarding Optimizer (DEFO) to achieve fast and scalable routing with low congestion. DEFO partitions the network graph into smaller "partial graphs," and labels nodes shared by any two such graphs as middlepoints. Subsequently, traffic for a flow is forwarded from one middlepoint to another until it reaches the concerned partial graph. DEFO, therefore, reduces the scale of flow allocation problem to selection of a few middlepoints and routing among them. This, however, introduces a dependency in the network in order to achieve inter-middlepoint routing.

Layeghy et al. [14] developed Software-defined Constrained Optimal Routing (SCOR), a constraint programming (CP)-based framework for optimal routing in SDNs. CP allows to specify the desired objective function as well as different networking constraints in a high-level language. Moreover, multiple constraints can be combined together as required. This can potentially simplify operations of a network manager. In particular, SCOR constructs a binary matrix of links and flows, and determines which flow should be assigned to which link based on the given objective function. However, CP, in general, lacks scalability. Obtaining an optimal solution—or even a feasible sub-optimal solution—for a large network may not be practically feasible.

Consideration of fairness is a common aspect in most flow allocation works. Allybokus et al. [15], for example, used α-fairness for multi-path flow scheduling, i.e., where flows can be split across multiple paths. On the other hand, Ito et al. [10] observed that bandwidth requirements in a network can change dynamically and lead to bottleneck links and unfair load distribution. To alleviate this, the authors proposed to reallocate unused capacity of the links to high-demand traffic flows. On a similar note, Boley et al. [8] proposed traffic flow bandwidth reassignment to improve quality of service of Software-defined Networks (SDNs).

Huang et al. [16] noted that, in practice, a legacy network may upgrade into an SDN only in steps. Consequently, such hybrid SDNs would require multi-path traffic to support both legacy and SDN-enabled devices. The authors, therefore, propose a utility maximization scheme for bandwidth allocation in hybrid SDNs.

To *synthesize*, we see that different methodologies have been used to allocate network flows, be it during planning or online operating states. Many of the allocation schemes, for example SCOR, take an optimization approach. However, such an approach may lack scalability with the growing size of networks. On the other hand, many schemes improve allocation efficiency by splitting flows. This may not be applicable for all industrial scenarios, for example, power grid networks, where communication path symmetry is required. Therefore, in this paper, we investigate the integral flow allocation problem for large networks using metaheuristics and greedy strategies.

III. The Flow Allocation Problem

We represent a wireline network as graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of (directed) edges (or links). Let F be the set of flows that needs to be allocated in the network G. Moreover, let $F_A \subseteq F$ be the set of allocated flows in the network at any instant. $F - F_A$ denotes the set of elements present in F, but not in F_A, i.e., the set of unallocated flows. For any edge $e \in E$, let c_e and r_e, respectively, be the capacity and residual bandwidth of e; then $x_e = c_e - r_e$ denotes the bandwidth already used by the edge. Moreover, for any flow $f \in F$, let d_f be the bandwidth demand of the flow. We assume that $c_e > 0$ and $d_f > 0$ for each $e \in E$ and $f \in F$, respectively.

The objective of optimal flow allocation problem is to find a set of paths (sets of subsets of E) so that traffic corresponding to the flows in F_A can pass from their respective source devices to target devices. The definition of "optimal" is highly contextual. For example, some may want to maximize the size of F_A, whereas others may attempt to improve allocation fairness. In the following, we discuss SPA and SPA-T, which attempt to maximize the number of flows allocated.

SPA adopts a greedy strategy to allocate all flows that it possibly can. In particular, SPA begins with computing all-pair shortest paths (i.e., from any node to any other node) for the given network. Subsequently, SPA iterates over all the flows (F) that need to be installed. At each iteration, based on the source and target of the concerned flow, SPA evaluates whether or not the corresponding shortest path has the minimum capacity to carry the flow. Let P_E be the set of edges used by any path. Then, the minimum available capacity of the path is $\min_{e \in P_E} r_e$, i.e., the minimum residual link bandwidth among all edges in P_E. If a path is found to have minimum capacity available to carry a flow, then SPA allocates the flow to the links of the concerned path. This ensures that no link carries traffic more than its designated capacity. The concerned flow is moved from set F to F_A. Moreover, residual bandwidths of the links along the path are reduced by an amount equal to the flow's bandwidth demand. Relevant data structures are updated to keep track of which link carries what flows. When SPA terminates, the set F_A contains all the flows that are allocated for a given network.

SPA-T is exactly similar to SPA, except that maximum utilization of any link (considering all flows passing through it) is kept below θ. In other words, the above mentioned minimum available capacity of the path is evaluated as $\min\{\min_{e \in P_E} r_e, \theta\}$.

IV. Metaheuristics-based Flow Allocation

The metaheuristics-based SPA-HC and SPA-SA schemes also rely on pre-computed all-pair shortest paths and restrict the utilization of each link up to θ. At each step, SPA-HC and SPA-SA try to improve the overall fairness in link utilization. We use Jain's fairness index [17] for this purpose. In particular, the fairness in bandwidth utilization of links arising out of a flow allocation scheme is defined as:

$$J(F_A, E) = \frac{(\sum_{e \in E} x_e)^2}{|E| \sum_{e \in E} x_e^2}, \qquad (1)$$

where x_e is the bandwidth usage of any edge $e \in E$. Equation (1) requires that $F_A \neq \varnothing$, i.e., at least one flow must be allocated to compute $J(F_A, E)$. In our implementation, SPA-HC and SPA-SA do an initial allocation of 10 randomly selected flows using SPA. In both the schemes, the value of a solution is given by (1), which ranges between $\frac{1}{|E|}$ and 1.

A. SPA-HC: SPA with Probabilistic Hill Climbing

Algorithm 1 presents the detailed steps of SPA-HC. The Algorithm runs for a finite number of steps. In each iteration, SPA-HC creates η feasible solutions.[1] The detailed steps for creating a solution are depicted in Algorithm 2. The Algorithm essentially selects up to ϕ flows so that the cumulative bandwidth usage (line number 10)[2] of the individual links do not exceed the given threshold θ (line number 12). Note that κ and ϕ ensure that Algorithm 2 terminates after a finite number of steps.

Once a feasible solution is available, Algorithm 1 computes the updated fairness index value (line number 8)[3] by considering link usages based on F_A as well as the potential flows returned by the solution. If the projected fairness value is better than the previous one, the solution is accepted (line number 10). Moreover, the consideration of the probability p_{bad} potentially helps to avoid getting stuck in local optima. When a feasible solution is accepted, the sets F and F_A are updated accordingly; link bandwidth measures are also updated appropriately.

[1] In classical hill climbing, η is set to 1.

[2] To keep the presentation of this Algorithm simple, we skip listing the steps where link usage from the other flows in the $flows$ list are also considered.

[3] In our implementation, we maintain a running sum of x_e and x_e^2 so that we do not have to repeatedly iterate over E to compute J.

Algorithm 1: Flow allocation using SPA-HC

```
Input:
  • $F$: Set of flows to allocate
  • $F_A$: (Very small) set of already allocated flows
  • $P$: Set of all-pair shortest paths
  • $\theta$: Link utilization threshold
  • $\eta$: Number of solutions to consider in an iteration
  • $\phi$: Maximum number of flows in a solution
  • $p_{bad}$: Probability of selecting a bad solution
Output: $F_A$: Set of allocated flows
1  $max\_steps = |F - F_A|$
2  $istep = 0$
3  $best\_value =$ Compute $J(F_A, E)$ using (1)
4  while $istep < max\_steps$ and $|F - F_A| > \eta$ do
       // Solutions generated in this iteration
5      $S = [\,]$
6      for $j \in 1 \ldots \eta$ do
7          $s = create\_solution(F - F_A, P, \theta, \phi, S)$
           // $s.flows$ indicate the flows selected by
              solution $s$
8          $v =$ Compute $J(F_A \cup \{s.flows\}, E)$ using (1)
9          $r =$ Uniform random number between 0 and 1
10         if $v > best\_value$ or $r < p_{bad}$ then
11             $best\_value = v$
12             $best\_solution = s$
13         end
14     end
15     $flows = best\_solution.flows$
16     $paths = best\_solution.paths$
17     foreach $(f, p) \in (flows, paths)$ do
18         $F = F - \{f\}$
19         $F_A = F_A \cup \{f\}$
20         foreach $e \in p$ do
               // Initially, $r_e = x_e, \forall e \in E$
21             $r_e = r_e - d_f$
22             $x_e = x_e + d_f$
23         end
24     end
25     $istep = istep + 1$
26 end
```

B. SPA-SA: SPA with Simulated Annealing

The SPA-SA scheme bears close resemblance to SPA-HC, except the fact that, in each iteration, only one feasible solution is evaluated. Algorithm 3 presents the detailed steps of SPA-SA. The initial *temperature* T is set to a high value, which is scaled down by a factor $0 < \gamma < 1$ (usually close to 0.99) in each iteration until T goes below ε (a very small positive fraction) or the upper limit on number of iterations is exceeded, whichever is earlier. Line numbers 8–10 represent the heart of SA. Given a solution, we define *cost* as the

Algorithm 2: Feasible solution creation

```
Input:
  • $F_U$: Set of unallocated flows
  • $P$: Set of all-pair shortest paths
  • $\theta$: Link utilization threshold
  • $\phi$: Maximum number of flows in a solution
  • $\kappa$: Maximum number of attempts
Output: A set of up to $\phi$ feasible flows and their
        corresponding paths
1  $istep = 0$
2  $flows = [\,]$
3  $paths = [\,]$
4  while $istep < \kappa$ do
5      for $j \in 1 \ldots \kappa$ do
6          $f$ = Randomly selected flow from $F - F_A$
7          $p$ = Shortest path from $P$ indexed by $f$'s
             source and target
8          $flag =$ True
9          foreach $e \in p$ do
               // Potential new usage of edge $e$
                  along the path $p$
10             $x'_e = x_e + d_f$
11             $\theta_e = x'_e / c_e$
12             if $\theta_e > \theta$ then
13                 $flag =$ False
14                 break
15             end
16         end
17         if $flag$ is True then
18             Append $f$ to $flows$ and $p$ to $paths$
19             foreach $e \in p$ do
20                 $r_e = r_e - d_f$
21                 $x_e = x_e + d_f$
22             end
23         end
24         if $|flows| = \phi$ then
25             break
26             $istep = \kappa$          // Exit the outer loop
27         end
28     end
29     $istep = istep + 1$
30 end
31 return $(flows, paths)$
```

Algorithm 3: Flow allocation using SPA-SA

```
Input:
  • $F$: Set of flows to allocate
  • $F_A$: (Very small) set of already allocated flows
  • $P$: Set of all-pair shortest paths
  • $T$: Initial temperature
  • $\gamma$: Cooling coefficient
  • $\varepsilon$: Zero temperature
  • $\theta$: Link utilization threshold
  • $\phi$: Maximum number of flows in a solution
Output: $F_A$: Set of allocated flows
1  $max\_steps = |F - F_A|$
2  $istep = 0$
3  $best\_value =$ Jain's index based on $F_A$ and $E$
4  while $T > \varepsilon$ and $istep < max\_steps$ do
5      $s = create\_solution(F - F_A, P, \theta, \phi)$
6      $v =$ Compute $J(F_A \cup \{s.flows\}, E)$ using (1)
7      $\Delta = \frac{1}{v} - \frac{1}{best\_value}$
8      $A = \exp(\frac{-\Delta}{T})$
9      $r =$ Uniform random number between 0 and 1
10     if $\Delta \leq 0$ or $r < A$ then
11         $best\_value = v$
12         $best\_solution = s$
13         $flows = best\_solution.flows$
14         $paths = best\_solution.paths$
15         foreach $(f, p) \in (flows, paths)$ do
16             $F = F - \{f\}$
17             $F_A = F_A \cup \{f\}$
18             foreach $e \in p$ do
19                 $r_e = r_e - d_f$
20                 $x_e = x_e + d_f$
21             end
22         end
23     end
24     $T = \gamma T$
25     $istep = istep + 1$
26 end
```

reciprocal of its value. The difference in cost between the current solution and a previous one is stored in Δ. Based on this, the acceptance probability A is defined, which depends both on the contemporary temperature and the cost difference. New solution that provides a lower cost is definitely accepted; otherwise it is chosen with probability A.

V. Performance Evaluation

In this Section, we discuss the experimental setup and the metrics used to evaluate the proposed flow allocation schemes.

A. Experimental Setup

We implemented the proposed algorithms using Python 2.7 in a Linux system with Intel(R) Core(TM) i5-4570S 2.90 GHz CPU and 8 GB RAM. The "networkx" library of Python was used to compute the all-pair shortest paths.

We used two real-life network topologies (labeled as "NR1" and "NR2," respectively) and a synthetic topology (labeled as "NS1") from the data set made available by Hartert et al. [13]. These network topologies and the corresponding flow demands are summarized in Table I.

We conducted several experiments to investigate the flow allocation schemes. First, we executed SPA, SPA-T, SPA-HC, and SPA-SA with the aforementioned three networks of different sizes as input. This helped us to collect various data, such as the number of flows allocated, average path length, number of hot links, and execution time. Next, keeping all other

TABLE I: Network topologies and flow demands

	NR1 [13]	NR2 [13]	NS1 [13]
Nodes	79	87	199
Links	294	322	522
Flows	6162	7527	37881
Min. link bandwidth (bps)	2400000		
Max. link bandwidth (bps)	10000000		
Min. flow demand (bps)	1	11	11
Max. flow demand (bps)	499374	396931	118476

parameters constant, we varied the value of θ to investigate how utilization threshold affected the flow allocation. We also performed sensitivity analysis of the algorithms' parameters. We varied the value of p_{bad} from 0.002 to 0.012 to understand how selecting a bad solutions affected SPA-HC.

At the beginning of each experiment, the network and corresponding flow details were read from an external file. Based on these, a networkx directed graph data structure was created, which was then used to compute the all-pair shortest paths. Each experiment was repeated for five times, based on which the mean and 95% confidence interval were computed. Unless otherwise specified, we assumed $\theta = 0.85$, $T = 100000.0$, $\gamma = 0.999$, $\varepsilon = 10^{-5}$, $p_{bad} = 0.01$, $\eta = 10$, and $\phi = 5$.

We considered the following metrics to measure the performance of SPA, SPA-HC, and SPA-SA.

- Percentage of flows allocated ($\frac{|F_A|}{|F|}$) by each scheme
- Fairness of flow allocation across the network measured by J in (1)
- Utilizations of the links breaching threshold θ (*only* in case of SPA)
- Processing time of the proposed schemes

The final metric, processing time of a scheme, indicates the time spent in reading the network data file, constructing a graph data structure from it, computing the all-pairs shortest path, performing algorithm-specific operations, and finally, verifying the allocations.

VI. Results

Figure 1a shows the percentage of flows allocated by SPA, SPA-T, SPA-HC, and SPA-SA in the three networks. Note that as we move from NR1 to NR2 and NS3, both the size of the network and number of flows to allocate increase. In general, SPA allocated more than 90% flows in all the three networks. SPA-HC, on the other hand, was found to allocate more flows than SPA in the networks NR1 and NR2. This can be accounted to the comparison of multiple solutions at each step of SPA-HC. However, for larger networks, the maximum limit on the number of steps executed by SPA-HC potentially disallowed consideration of enough candidate solutions. This largely explains why SPA-HC allocated marginally less number of flows than SPA in case of NS1. On the other hand, since SPA-SA evaluates only one potential solution in each iteration, enough alternative flow allocations may not have been considered by SPA-SA. This possibly resulted in relatively less number of flow allocations by SPA-SA in the three networks. Finally, SPA-T allocated fewer flows than SPA because of the additional link utilization threshold constraint.

Figure 1b plots the Jain's fairness index obtained with respect to flow allocation. In general, SPA-HC resulted in a relatively more fair flow allocation than its peers. This is because SPA-HC, by considering η solutions, had a better view of the global solution space. On the other hand, even SPA-T was found to be fair than SPA, because the former did not indiscriminately allocate flows to links.

Figure 1c shows the total time taken by the algorithms to complete their execution. SPA and SPA-T were found to be the fastest schemes, which took less than or up to a second. SPA-HC and SPA-SA, in contrast, took several minutes to complete execution. In particular, SPA-HC takes about 40 minutes, on average, to allocate the flows in NS1. This is largely due to the several solutions (combinations of multiple flows) that SPA-HC generate and evaluate in each iteration. SPA only evaluates whether or not a given path can meet requirements of a flow. Consequently, SPA takes much lower execution time compared to its counterparts.

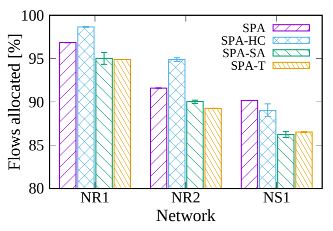
We also look at the links whose utilization crossed θ under SPA. In particular, SPA created six, ten, and seven such links in NR1, NR2, and NS1, respectively. Figure 2 shows that most of those links were very close to 100% utilization. As noted earlier, this can lead to potential network disruptions in the near future.

Figure 3 shows how p_{bad}, the relative frequency of choosing a bad solution, affected the performance of SPA-HC. The plots with solid lines show that, as p_{bad} increased, more flows were allocated in the networks NR1 and NR2. However, at the same time, the overall fairness of flow allocation measured by (1) consistently decreased (dotted lines, y-axis on the right side). In particular, beyond $p_{bad} = 0.01$, the increase in flow allocation was almost negligible. The plots indicate that the value of p_{bad} should typically range between 0.005 and 0.01.

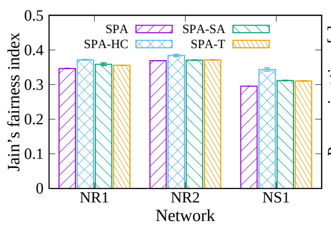
Figure 4 shows how the performance of SPA-HC, SPA-SA, and SPA-T varied with the link utilization threshold. As expected, when θ increased, all the schemes allocated more flows (plotted with solid lines), because capacity of links to accommodate new flows increased. At the same time, fairness in link utilization (plotted with dotted lines) decreased with increase in θ. Interestingly, the percentage of flows allocated did not seem to have a steep increase with θ. In other words, setting link utilization threshold to say, more than 90%, may not be of much practical use.

VII. Conclusion

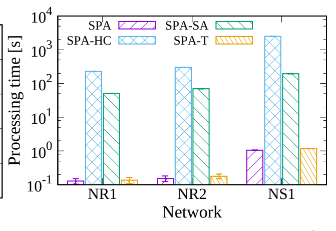
The rise of industrial IBNs would witness various forms of automation both in-network and to other/external networks. One such service is the flow allocation for large networks. In this paper, we investigated four schemes in this regard. All these algorithms rely upon pre-computed all-pair shortest paths for a network. SPA-T, SPA-HC, and SPA-SA tend to maximize the number of flows allocated while also limiting per-link bandwidth usage and thereby, improving the allocation fairness. SPA-HC fares better than the others because it



(a) Total flows allocated.



(b) Flow allocation fairness.



(c) Processing times (y-axis in log-scale).

Fig. 1: Comparative performance of the SPA, SPA-T, SPA-HC, and SPA-SA.

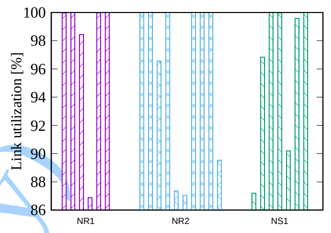


Fig. 2: Highly utilized links under SPA ($\theta = 85\%$).

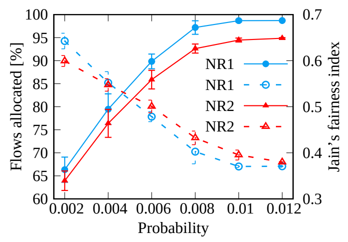
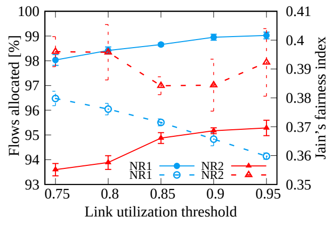
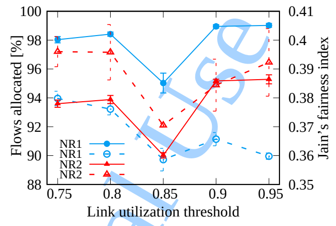


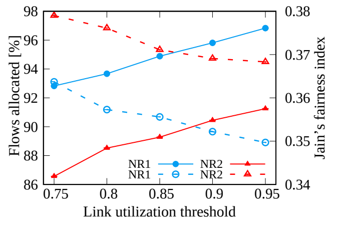
Fig. 3: Effects of p_{bad} on SPA-HC.



(a) SPA-HC.



(b) SPA-SA.



(c) SPA-T.

Fig. 4: Effects of utilization threshold θ on the different schemes.

takes a more detailed look at the feasible solutions space. On the downside, it takes a longer time to run.

The results presented here offer various insights and directions that can be investigated in the future. In particular, speeding up SPA-HC while retaining its performance would be a very interesting task. Moreover, other local and global search techniques can be explored in quest of better solutions. Additional constraints for flow allocation, for example, latency and jitter, can also be considered, which, in turn, would make the problem still more difficult.

References

[1] M. Kiran, E. Pouyoul, A. Mercian, B. Tierney, C. Guok, and I. Monga, “Enabling intent to configure scientific networks for high performance demands,” *Future Generation Computer Systems*, vol. 79, pp. 205 – 214, 2018.

[2] B. K. Saha, D. Tandur, L. Haab, and L. Podleski, “Intent-based Networks: An Industrial Perspective,” in *Proceedings of the 1st International Workshop on Future Industrial Communication Networks (FICN ’18)*. New York, NY, USA: ACM, 2018, pp. 35–40.

[3] J. Augé and M. Enguehard, “A network protocol for distributed orchestration using intent-based forwarding,” in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, April 2019, pp. 718–719.

[4] A. Abhashkumar, J.-M. Kang, S. Banerjee, A. Akella, Y. Zhang, and W. Wu, “Supporting diverse dynamic intent-based policies using Janus,” in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. New York, NY, USA: ACM, 2017, pp. 296–309.

[5] L. Lo Bello and W. Steiner, “A perspective on IEEE time-sensitive networking for industrial communication and automation systems,” *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, June 2019.

[6] F. Dababneh and L. Li, “Integrated electricity and natural gas demand response for manufacturers in the smart grid,” *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 4164–4174, July 2019.

[7] B. K. Saha, S. Misra, and S. Pal, “SeeR: Simulated annealing-based routing in opportunistic mobile networks,” *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, pp. 2876–2888, Oct 2017.

[8] J. M. Boley, E. Jung, and R. Kettimuthu, “Adaptive QoS for data transfers using software-defined networking,” in *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Nov 2016, pp. 1–6.

[9] K. Sharma and V. Badarla, “FlowFurl: A flow-level routing for faulty data center networks,” in *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Nov 2016, pp. 1–6.

[10] Y. Ito, H. Koga, and K. Iida, “A bandwidth reallocation scheme to improve fairness and link utilization in data center networks,” in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, March 2016, pp. 1–4.

[11] D. Nadig, B. Ramamurthy, B. Bockelman, and D. Swanson, “Large data transfer predictability and forecasting using application-aware SDN,” in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec 2018, pp. 1–6.

[12] A. Chopra, D. S. Kalhan, A. S. Bedi, A. K. Gupta, and K. Rajawat, “On socially optimal traffic flow in the presence of random users,” in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec 2018, pp. 1–6.

[13] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, “A declarative and expressive approach to control forwarding paths in carrier-grade networks,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. New York, NY, USA: ACM, 2015, pp. 15–28.

[14] S. Layeghy, F. Pakzad, and M. Portmann, “SCOR: software-defined constrained optimal routing platform for SDN,” *CoRR*, vol. abs/1607.03243, 2016.

[15] Z. Allybokus, K. Avrachenkov, J. Leguay, and L. Maggi, “Multi-path alpha-fair resource allocation at scale in distributed software-defined networks,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2655–2666, Dec 2018.

[16] X. Huang, T. Yuan, and M. Ma, “Utility-optimized flow-level bandwidth allocation in hybrid SDNs,” *IEEE Access*, vol. 6, pp. 20 279–20 290, 2018.

[17] D. C. R. Jain and W. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer system,” DEC Technical Report 301, Sep. 1984.