Two Rerouting-Based Congestion Control Algorithms for Centrally Managed Flow-Oriented Networks

Andrzej Kamisiński, *Student Member, IEEE*, Jerzy Domżał, *Member, IEEE*, Robert Wójcik, *Member, IEEE*, and Andrzej Jajszczyk, *Fellow, IEEE*

***Abstract*— The evolution of the Internet and fiber-optic technologies has created new opportunities in terms of availability of different network services. The increasing demand for capacity, combined with strong reliability requirements, defines new challenges in the area of network management and design. Today, several users share the limited network resources, which may lead to frequent link congestions in communication networks. Therefore, we propose two effective solutions to respond to link congestions in centrally managed flow-oriented IP networks, such as software-defined networks. We verify their performance through simulation using a real backbone network topology and conclude that as long as enough resources are available in the network, the presented algorithms have the potential to reduce the number of fully loaded links in the network.**

***Index Terms*— Congestion control, routing, flow, software-defined networks.**

I. Introduction

With the rapid growth of telecommunication networks in the recent years, the daily volume of traffic forwarded through backbone networks has risen to an enormous level. Due to high costs of service disruption, especially in the case of services of strategic importance, strong reliability requirements are imposed on the core infrastructure. As limited network resources need to be shared across many users, it may happen that certain network links become overloaded, either blocking the subsequent traffic flows routed via these links or causing packet losses affecting the overall performance.

To reduce the probability of a link overload in the backbone network, Internet service providers have a few options to choose from. For example, the following general strategies may be considered:

- assign more resources to particular network segments than actually needed [1], taking as a reference the results of regularly performed load assessments;
- some new or existing traffic flows may be denied access to the network either for a predefined time or until the congestion is mitigated [2];
- accept new flows and send them via one or more alternative paths [3], [4], thus limiting or reducing the load on the primary path (there are also related proposals for data center networks, such as [5]–[7]);
- avoid congestions through the use of rate control mechanisms [8].

In the third case, the existing flows may stay on their original paths, but it is also possible to use a more flexible approach and relocate the selected existing flows if necessary.

In this paper, we explore the opportunities for effective congestion control mechanisms in the third category (i.e., accepting new flows and sending them via one or more alternative paths) with an assumption that the existing traffic flows may be relocated if needed. We discuss the related solutions in Section II and we show that while the algorithms proposed in this paper have limitations, they can significantly improve network performance in the presence of link congestions, compared to the strategies discussed in [5] and [7], and the case when routing is only managed by the OSPF protocol configured in such a way that it relies on a single path to each destination. In particular, we demonstrate that in the case of the considered evaluation scenario, the presented algorithms were able to reduce the number of fully loaded links down to 0 (or to a close value), provided that enough network resources were available.

II. Related Work

An example of a solution representing the third strategy (see Section I) in the case of centrally-managed networks was presented in [5]. It assumes that new traffic flows are transmitted along the shortest (in terms of the number of hops) paths (candidate paths) that also have the minimum cumulative utilization of links belonging to them. This approach has one major limitation: as it was designed for data center networks which usually offer multiple shortest paths to the selected destination node, it does not consider longer alternative paths.

A similar solution that reroutes traffic flows one-by-one in the case of link congestion is discussed in [7]. In this proposal, rerouting may take place at the point of congestion or one hop before and it is restricted to *elephant* flows (i.e., flows for which the observed size is at least 100 kB). The alternative path is selected based on the following criteria: minimum length (determines the set of candidate paths), the maximum load among links belonging to the path, and whether the new path will be overloaded once the selected flow is moved. While the idea to consider the maximum load among all links of a path is reasonable,[1] the proposed solution suffers from the

Manuscript received June 3, 2016; revised July 22, 2016; accepted July 23, 2016. Date of publication July 27, 2016; date of current version October 7, 2016. The research was carried out with the support of the project "High quality, reliable transmission in multilayer optical networks based on the Flow-Aware Networking concept" founded by the Polish National Science Centre under project no. DEC-2011/01/D/ST7/03131. The associate editor coordinating the review of this paper and approving it for publication was A. Vinel.

The authors are with the Department of Telecommunications, AGH University of Science and Technology, 30-059 Kraków, Poland (e-mail: kamisinski@kt.agh.edu.pl; jdomzal@kt.agh.edu.pl; robert.wojcik@kt.agh.edu.pl; jajszczyk@kt.agh.edu.pl).

Digital Object Identifier 10.1109/LCOMM.2016.2594774

[1]It is worth noting that the mechanism proposed in [7] compares absolute load values, which may lead to wrong decisions if network links have significantly different capacities.

```
Require: network topology modeled as a weighted graph $G = (V, E)$; relative link
    overload threshold $l_{th} \in [0, 1]$; overloaded link $e_{ov} \in E$; weight parameter $\alpha$
1:  $F \leftarrow$ GetFlows $(e_{ov})$
2:  $f \leftarrow$ GetTheMostSignificantFlow $(F)$
3:  repeat
4:      $f_{previous} \leftarrow f$
5:      $P_{current} \leftarrow$ GetCurrentPath $(f)$
6:      $R_f \leftarrow$ GetDemand $(f)$
7:      $Q \leftarrow \emptyset$
8:      for all $v \in V$ do
9:          $D[v] \leftarrow$ MAX_COST {distance/cost vector relative to the source node of flow $f$}
10:         $B[v] \leftarrow$ NULL {predecessor nodes according to the computed routing scheme}
11:         $Q \leftarrow Q \cup \{v\}$ {set of nodes to be considered in the while loop (line 14)}
12:     end for
13:     $D$ [GetSource $(f)$] $\leftarrow 0$
14:     while $Q \neq \emptyset$ do
15:         $u \leftarrow$ GetVertexWithMinDistance $(Q, D)$
16:         $Q \leftarrow Q \setminus \{u\}$
17:         for all $v \in$ GetNeighbors $(u) \cap Q$ do
18:             if $(u, v) = e_{ov}$ then
19:                 Proceed to the next iteration
20:             end if
21:             $P_t \leftarrow$ GetShortestPath $(D, B,$ GetSource $(f), u)$
22:             $P_t \leftarrow P_t \cup \{(u, v)\}$
23:             $l \leftarrow 0$ {the initial value of the maximum expected relative load among all links of path $P_t$}
24:             $p \leftarrow 1$ {the initial value of the estimated probability that path $P_t$ will not be overloaded}
25:             for all $e \in E \cap P_t$ do
26:                 $p \leftarrow p \cdot (1 -$ GetEstimatedOverloadProb $(e, l_{th}))$
27:                 if GetExpectedRelativeLoad $(e, R_f) > l$ then
28:                     $l \leftarrow$ GetExpectedRelativeLoad $(e, R_f)$
29:                 end if
30:             end for
31:             $d = \alpha l + (1 - \alpha)(1 - p)$
32:             if $d < D[v]$ then
33:                 $D[v] = d$
34:                 $B[v] = u$
35:             end if
36:         end for
37:     end while
38:     $P_{new} \leftarrow$ GetShortestPath $(D, B, f)$
39:     if GetCost $(P_{new}) <$ GetCost $(P_{current})$ then
40:         RelocateFlow $(f, P_{new})$
41:         $F \leftarrow F \setminus \{f\}$
42:     end if
43:     $f \leftarrow$ GetTheMostSignificantFlow $(F)$
44: until IsCongested $(e_{ov})$ = FALSE or $f = f_{previous}$ or $f$ = NULL
```

Fig. 1. Congestion control algorithm relying on information about the estimated overload probability of a path and the maximum relative load among all links belonging to the path.

same limitation as the previous approach with respect to the set of candidate paths.

Another related solution relies on counting flows of the same type (e.g., http or ftp flows) for each link individually [4]. Each flow type has the corresponding predefined weight, which may also be determined based on the estimated bit rate. The main idea is that the more flows of significant types are observed on a link, the more likely the link is to enter the congestion state, which should be avoided. At the same time, the weighted scheme may not reflect the actual situation in the network if the size of flows varies considerably within each category, not to mention the unknown, yet potentially strong influence of unclassified flows.

In this paper, we present algorithms that do not suffer from the limitations identified and discussed above.

III. Congestion Control Algorithms

A. Algorithm I: Max Path Load and Path Overload Probability

The first algorithm relies on information about the estimated overload probability of a path and the maximum relative load among all links belonging to the path. As presented in Fig. 1, the algorithm requires access to the network graph G and must be invoked in the context of an overloaded link e_{ov}. l_{th} represents the global value of the link overload threshold.[2] Before entering the main loop, the algorithm prepares the list of flows forwarded via the overloaded link and selects the one with the highest demand as *the most significant flow*. Then, it enters the loop (lines 3-44) which is executed for as long as link e_{ov} remains congested, or alternatively, until one of the following conditions is satisfied:

- the most significant flow could not be relocated,
- all flows on link e_{ov} have already been considered.

Distances assigned to particular destination nodes during the execution of the modified Dijkstra's shortest path algorithm on graph G (lines 7-37) depend on the maximum expected relative load among all links belonging to a path and the estimated overload probability of the entire path from the source node to the currently considered node (line 31),[3] based on the previously computed values.[4] The α parameter defines the importance of each of the two mentioned factors. In particular, it is possible to consider only one factor by setting α to either 0 or 1 (see line 31). Once the distances are computed, it is possible to determine the best alternative path P_{new} for flow f, such that does not include link e_{ov}. If its cost is smaller than the cost of the currently used path $P_{current}$, the flow is relocated. In all other cases, the flow remains on the original path.[5]

B. Algorithm II: Max Path Load and Path Length

The second algorithm is very similar to the first one, but instead of the estimated overload probability of a path it considers its length in terms of the number of links. For this reason, only three lines in Fig. 1 need to be changed as follows:

- line 24: $m \leftarrow 0$,
- line 26: $m \leftarrow m + 1$,
- line 31: $d = \alpha l + (1 - \alpha)\frac{m}{|E|}$,

where m is the number of links belonging to the constructed path.

The motivation for the second algorithm is that less congested alternative paths in a network do not necessarily have to be close to the corresponding primary paths in terms of their lengths. Thus, it is now possible to explicitly assign the importance to the length factor through setting the α parameter to a desired value.

The proposed solutions have some limitations. First, they require a complete view of a network in terms of the load of particular links, including the ability to reallocate flows and make decisions based on the previously collected

[2]The lowest value of the relative link load (link capacity utilization) for which the link is still perceived as congested.

[3]Note that the overload probability of a path is computed as 1 minus the probability that all links belonging to this path are not overloaded. In other words, the path is overloaded if at least one of its links is. The expected relative load of a link is computed as the capacity utilization (value in the range of [0, 1]) involving the cumulative demand of all current flows forwarded via this link and the demand of flow f, as if it was assigned to this link.

[4]Different implementation strategies are possible here, for example, a fixed number of samples may be kept in a circular buffer.

[5]One potentially interesting modification of the algorithm is to try to relocate the subsequent significant flows if the most significant flow could not be moved. We expect that this may improve the effectiveness of the algorithm at the cost of an increased runtime.

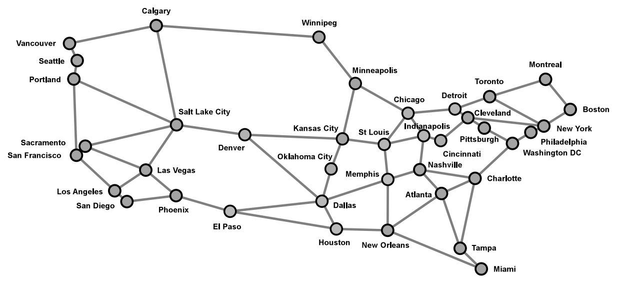

Fig. 2. Evaluation network topology based on the data provided by the *SNDlib* project [12] (name of the model: *janos-us-ca*).

data (Algorithm I, Section III-A). Second, none of them is perfect in all scenarios — while Algorithm I performed well in heavily congested networks, it was not the case for Algorithm II. At the same time, both algorithms are compatible with the concept of Software-Defined Networking [9] which is becoming increasingly popular among researchers and network engineers. In particular, they might be deployed in a logically-centralized network controller which makes routing decisions, configures traffic flows, and monitors the state of flows and network switches.

IV. Evaluation

In this section, we present the evaluation results of the proposed algorithms. The evaluation was based on a simulation study carried out with the aid of a discrete-event, flow-level network simulator written in the C++ programming language,[6] using the US backbone network topology shown in Fig. 2. The network contains 39 nodes and 122 unidirectional links.[7] For an interested reader, we have also prepared a short online Appendix [11] in which we present an extended study involving two different scenarios.

In the main simulation scenario, each link had the capacity of 1 Gbit/s, while the corresponding relative overload threshold was set to 0.7 (it means that a link is perceived as congested whenever its capacity utilization equals 70% or more).[8] The congestion state of links was monitored every 1 s and the overload probability of each link was estimated based on 100 most recent samples stored in a circular buffer. 10000 traffic flows were generated between different nodes selected at random according to a uniform distribution. The flow inter-arrival time was selected according to the exponential distribution with the mean value of 0.1 s and the maximum value (bound) of 0.2 s, while the duration of each flow followed the Pareto distribution with the mean value of 120 s, the maximum value (bound) of 240 s, and the shape parameter equal to 1.5. The average flow demand was set to 10 Mbit/s, whereas the actual values were selected at random from range [7.5; 12.5] Mbit/s according to a uniform distribution. Each simulation was repeated 10 times to get credible results. The average values were computed over the period from the 150th to the 650th second of each simulation to avoid the warm-up and termination phases. The numerical values were selected in such a way that the generated traffic triggered network congestion events for multiple links simultaneously (also ensuring that a fraction of links were fully loaded — about $4-5\%$ of all links on average in the case of OSPF-like routing with no additional congestion avoidance measures), so that the effectiveness of the considered flow-level congestion mitigation strategies could be evaluated.[9]

[6]For the purpose of random data generation (e.g., flow inter-arrival time, flow duration and demand, source and destination nodes), we relied on the random data generation engine of the widely-used ns-3 network simulator [10].

[7]Note that Fig. 2 presents an undirected graph.

[8]We refer to a network link as *fully loaded* whenever its capacity utilization reaches 100%.

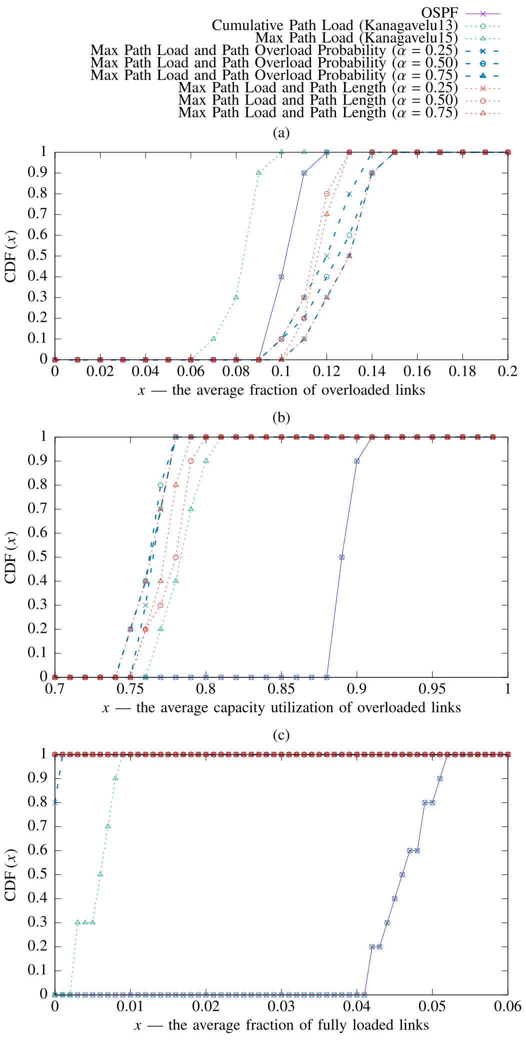

Fig. 3. Estimated Cumulative Distribution Function (CDF) of (a) the average fraction of overloaded links, (b) the average capacity utilization of overloaded links, and (c) the average fraction of fully loaded links between the 150th and 650th second of simulations.

The proposed algorithms were compared with two existing solutions introduced in [5] and [7], as well as with the third reference case (OSPF) in which traffic flows were forwarded along the shortest paths and there were no active congestion mitigation mechanisms in the network. The evaluation results are presented in Figs. 3(a)-(c). In terms of the average fraction of overloaded links, the main point was to observe how the

[9]Note that our study is focused on flow allocation strategies and does not extend to congestion control mechanisms introduced on other levels (e.g., in the TCP protocol).

proposed solutions influence the number of overloaded links in the network, compared to the other considered strategies. According to Fig. 3(a), only the algorithm presented in [7] rearranged traffic flows in such a way that the total number of overloaded links decreased, while the other methods (except for the algorithm introduced in [5] and the OSPF protocol which was not responding to congestions) often transferred the excessive load to previously uncongested links. However, by increasing the load of such links, it was possible to reduce the average capacity utilization of the overloaded links in the entire network, thus avoiding traffic losses on the most congested links. This observation corresponds to Fig. 3(b) in which the proposed algorithms performed better than all three reference solutions in terms of the average capacity utilization of all overloaded links in the network (the largest difference was observed for OSPF and the algorithm introduced in [5], both of which have demonstrated comparable performance). The results shown in Fig. 3(c) also confirm the previous statement. While the observed number of fully loaded links in the case of the algorithm presented in [7] was considerably smaller (on average) than the respective value for OSPF and the third reference solution [5], it needs to be emphasized that the two proposed strategies reduced the number of fully loaded links down to 0 almost for all of the considered values of α, effectively ensuring that the congested links were not causing traffic losses in the network. It means that except for the case of Algorithm I when α was equal to 0.25, the following relation was true: $\forall_{x \in [0;1]} \text{CDF}(x) = 1$, as presented in Fig. 3(c).

Until now, we have discussed the performance of the proposed solutions without considering the value of the α parameter. In the case of the second algorithm, the smallest value of α resulted in a considerably larger average fraction of overloaded links in the network, compared to the two other considered values for which the corresponding difference was not significant. This is in line with our expectations, as the small weight assigned to the maximum relative path load increased the priority of the load-independent factor (relative path length). In terms of the average capacity utilization of overloaded links, the smallest gain (relative to the OSPF case) was observed for $\alpha = 0.50$, while the highest gain was associated with $\alpha = 0.25$. One possible explanation of this observation is that as the algorithm generally preferred shorter alternative paths, less links were carrying the offloaded traffic flows.

In the first algorithm, both factors are load-dependent and their mutual importance is also determined by the value of the α parameter. Based on the results shown in Figs. 3(a)-(c), we noticed that setting α to different values had impact mainly on the average fraction of overloaded links (the smallest values were observed for $\alpha = 0.25$) and the average fraction of fully loaded links. At the same time, the influence of α is almost not visible in Fig. 3(c), as the proposed algorithms were effectively preventing all links from entering the fully-loaded state (except for a very small fraction that was observed only for $\alpha = 0.25$ when using Algorithm I).

V. Conclusion

In this paper, we presented two new algorithms for rerouting-based congestion control in centrally-managed flow-oriented networks. Their performance was assessed on the grounds of a simulation study in which we used a real backbone network topology. The simulation results have confirmed that the proposed solutions can be used as effective congestion prevention measures in centrally-managed networks, such as networks based on the Software-Defined Networking concept.

References

[1] F. Dikbiyik, M. Tornatore, and B. Mukherjee, "Exploiting excess capacity for survivable traffic grooming in optical backbone networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 6, no. 2, pp. 127–137, Feb. 2014.

[2] J. Domzal and A. Jajszczyk, "New congestion control mechanisms for flow-aware networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2008, pp. 12–16.

[3] R. Wójcik, J. Domżał, and Z. Duliński, "Flow-aware multi-topology adaptive routing," *IEEE Commun. Lett.*, vol. 18, no. 9, pp. 1539–1542, Sep. 2014.

[4] T.-T. Nguyen and D. S. Kim, "Accumulative-load aware routing in software-defined networks," in *Proc. IEEE 13th Int. Conf. Ind. Inform. (INDIN)*, Jul. 2015, pp. 516–520.

[5] R. Kanagavelu, B. S. Lee, R. F. Miguel, L. N. T. Dat, and L. N. Mingjie, "Software defined network based adaptive routing for data replication in data centers," in *Proc. 19th IEEE Int. Conf. Netw. (ICON)*, Dec. 2013, pp. 1–6.

[6] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "FlowBender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experim. Technol. (CoNEXT)*, 2014, pp. 149–160. [Online]. Available: http://doi.acm.org/10.1145/2674005.2674985

[7] R. Kanagevlu and K. M. M. Aung, "SDN controlled local re-routing to reduce congestion in cloud data center," in *Proc. Int. Conf. Cloud Comput. Res. Innov. (ICCCRI)*, Oct. 2015, pp. 80–88.

[8] J. Gruen, M. Karl, and T. Herfet, "Network supported congestion avoidance in software-defined networks," in *Proc. 19th IEEE Int. Conf. Netw. (ICON)*, Dec. 2013, pp. 1–6.

[9] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[10] The ns-3 Project. (Jul. 2016). *The ns-3 Discrete-Event Network Simulator*. [Online]. Available: https://www.nsnam.org

[11] A. Kamisiński, J. Domżał, R. Wójcik, and A. Jajszczyk. (Jul. 2016). *Online Appendix*. [Online]. Available: http://kt.agh.edu.pl/~kamisinski/pub/2016_commltr_congestion_control/

[12] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Netw. Optim.*, vol. 55, no. 3, pp. 276–286, May 2010.