Designing VNT Candidates Robust Against Congestion Due to Node Failures

Onur Alparslan, Shin'ichi Arakawa, Masayuki Murata
Graduate School of Information Science and Technology
Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
{a-onur,arakawa,murata}@ist.osaka-u.ac.jp

***Abstract*—When wavelength division multiplexing with path (circuit) switching is used on an optical network, even the failure of a single link may tear down many lightpaths and cause abrupt and significant changes in the utilization of many links in the corresponding VNT (virtual network topology). In this paper, we propose an algorithm called MFLDA (Minimum Flow Logical topology Design Algorithm) for designing VNT candidates, which can accommodate a wide range of traffic patterns. Moreover, we show that the variant called MFLDA-FO (MFLDA with Failure Optimization) can design VNT candidates, which have lower probability of congestion right after the failure of multiple nodes compared to HLDA, which is one of the best performing VNT design algorithms in the literature. Furthermore, we show that when these VNT candidates are used as attractors in an attractor selection algorithm, the average time to recover from difficult failure scenarios is less than using the attractors designed by HLDA algorithm. Unlike HLDA, our VNT design algorithm and the attractor selection algorithm does not require the traffic matrix and the topology information after the failure.**

I. Introduction

In a wavelength division multiplexing optical network, a fiber link may carry hundreds of wavelengths. However, it is difficult to terminate each wavelength and moreover process and switch each packet carried on each wavelength at each node. Such networks are usually visualized by constructing a VNT (virtual network topology), where only the physical nodes, which are the transmitter and receiver edges of a lightpath, are shown as connected by a link. Modifying the VNT allows adapting the network for changing traffic conditions and new application layer services. There are many papers in the literature on VNT configuration, but they may be classified into two groups as online and off-line approaches [1], [2] in general. Off-line approaches create VNTs suitable for a set of possible traffic demand matrices. However, Internet traffic is difficult to predict as new applications and services, which can dramatically change the traffic, appear in time [3]. Moreover, it is difficult to predict the traffic changes due to node/link failures, cyber-attacks etc. Online approaches sample the traffic demand periodically and design a new VNT for the current environment [4]. However, they require up-to-date traffic demand matrix information, which can be challenging to retrieve. Moreover, some of them cannot handle traffic changes due to node or link failures and some of them need to know detailed information like exact place of failures in the topology in order to work.

Living organisms are well-known to adapt to the changes in the environment. It is shown that an attractor selection mechanism is adopted for using these genes to adapt to the environment and recover in order to increase the probability of survival. An attractor selection mechanism, which is similar to the system used by living organisms, was proposed in order to recover from topology failures and find a VNT, which minimizes the maximum lightpath load in the network [5]. Unlike most on-line methods in the literature, attractor selection does not require a priori knowledge like the place of failed nodes/links or detailed information about the current environment like the up-to-date traffic matrix information. Attractor selection requires only the maximum lightpath utilization level, which can be retrieved quickly by Simple Network Management Protocol (SNMP) [6]. Using the maximum load level in the network as a simple feedback, the attractor selection algorithm also recovers the network from high congestion after multiple node/link failures. Many papers on VNT failures in the literature concentrate only on preventing the disconnection of the remaining nodes in the VNT after a failure. Moreover, most off-line analytical approaches in the literature propose protection against failure of only one or two random nodes/links at a time or a regional failure, where all failures are inside a single region. On the other hand, attractor selection can solve complex problems with randomly distributed multiple node/link failures, which may occur due to a large scale distributed denial of service (DDoS) cyber-attack.

In attractor selection, the system tries to find an equilibrium point by evolving around the attractors where the conditions are known or expected to be preferable. The attractor selection algorithm uses a list of VNTs as attractors. Ref. [7] showed that even when random VNTs are used attractors, attractor selection has better performance than other algorithms in general. However, when the attractor VNTs were not suitable for the network, it took a long time to find a solution in some cases. The first work on designing VNT candidates as attractors was in [8], which showed that its attractors further decrease the convergence time compared to attractors selected in a random manner. While the proposed algorithm is good for designing VNTs with low utilization for a wide range of traffic matrices, it does not take the possible network failures into account.

In this paper, we propose an algorithm called MFLDA (Minimum Flow Logical topology Design Algorithm) for designing VNT candidates, which can accommodate a wider range of traffic patterns compared to a random VNT. Moreover, we present an extended version called MFLDA-FO (MFLDA with Failure Optimization), which is more robust against traffic changes after failure of multiple nodes. We show that right after a network failure a VNT designed by MFLDA-FO has a lower congestion probability compared to a VNT designed HLDA (Heuristic Logical topology Design Algorithm) [2], which is one of the best performing VNT design algorithms in the literature. Furthermore, we show that when the VNT candidates designed by MFLDA-FO are used as attractors in an attractor selection algorithm, the average time to recover from difficult failure scenarios is less than using the attractors designed by HLDA. Unlike HLDA, our VNT design algorithm and the attractor selection algorithm does not require the traffic matrix and the topology information after the failure

The paper is organized as follows. In Section II, we present the algorithm for designing VNT candidates. In Section III, we present the architecture of attractor selection. Section IV shows the simulation results and discusses the performance of the architecture. Section V concludes the paper.

II. VNT Candidate Design Algorithm

Ref. [8] showed that it is possible to design VNT candidates, which give low maximum link utilization for a wide range of traffic matrices. However, the VNTs designed by [8] were not robust against network failures as it did not take the physical topology into account. In this paper, our aim is to design VNTs robust against both wide range of traffic matrices and network failures, so it is more challenging.

Let's denote the traffic from a source to destination node as a flow. The flows are carried over the lightpaths established on the physical topology. The probability of a congestion on a lightpath increases with the increasing number of flows passing through. In order to minimizes the number of flows on the lightpaths, we propose an algorithm called MFLDA (Minimum Flow Logical topology Design Algorithm) for designing VNT candidates, which can accommodate a wide range of traffic patterns. Moreover, we extend it to MFLDA-FO (MFLDA with Failure Optimization) variant, which minimizes the number of flows on the lightpaths after node failures.

The pseudocode code of the main algorithm is shown in Fig. 3. First, the parameters are initialized and the initial VNT is established as shown in Fig. 1. n denotes the number of nodes in the network. The number of transmitters and receivers available on each node are stored in tra and rec arrays. When choosing the node pairs for new lightpaths, we give priority to the nodes, which have highest number of available transmitters/receivers, so we apply a token based priority scheme. The transmitter and receiver tokens on each node are stored in $token\_tra$ and $token\_rec$ arrays and first initialized to the number of transmitters and receivers available. In the FOR loop on line 8, initially the VNT is set to the physical topology by establishing lightpaths on the fibers

```
1:  function INITIALIZE
2:      $n \leftarrow$ number of nodes
3:      $stop\_all \leftarrow 0$
4:      for $k \leftarrow 0$ to $n$ do
5:          $tra[k], token\_tra[k] \leftarrow$ number of transmitters on $k$
6:          $rec[k], token\_rec[k] \leftarrow$ number of receivers on $k$
7:      end for
8:      for each fiber from $src$ to $dst$ on physical topology do
9:          if $tra[src] > 0$ AND $rec[dst] > 0$ then
10:             establish a lightpath from $src$ to $dst$
11:             decrease $tra[src]$, $rec[dst]$, $token\_tra[src]$, $token\_rec[dst]$ by one
12:         end if
13:     end for
14:     $token \leftarrow$ MIN(MAX($token\_tra$), MAX($token\_rec$))
15:     for $k \leftarrow 0$ to $n$ do
16:         $token\_tra[k] \leftarrow token\_tra[k] - token + 1$
17:         $token\_rec[k] \leftarrow token\_rec[k] - token + 1$
18:     end for
19:     return $n$, $tra$, $rec$, $token\_tra$, $token\_rec$, $stop\_all$ and initial VNT
20: end function
```

Fig. 1. The initialization of parameters and setting the initial VNT

between adjacent nodes. The reason is that as the number of physical hops a lightpath traverses increases, the probability of being hit by a failure increases, so priority is given to establish single hop lightpaths. In case the nodal degree is higher than the number of transmitters/receivers in a node, priority is given to the links that make the topology connected. This initial VNT serves as a substrate for adding new lightpaths. If network failures are not considered, it is also possible to use another substrate like a simple random ring topology passing through all nodes as the initial VNT. The initial VNT should be fully connected, so all nodes are reachable. On line 14 and in the next FOR loop, the number of tokens are normalized.

In order to select and add new lightpaths, the algorithm starts the main loop on line 2 in Fig. 3. In order to select the s-d pairs for establishing lightpaths, the algorithm collects data on the current logical topology as shown in Fig. 2. In order to analyse the effect of node failures, the algorithm simulates single node failures in the FOR loop on line 2 in Fig. 2 and stores the data about the lightpath stats after the failure. The f variable is set to ID of the nodes, which may fail. In the last loop, the stats are calculated for the VNT without any failure by setting f to n, which prevents node failures. As MFLDA creates a VNT without considering any failures, f is set to only n in MFLDA. The algorithm is called MFLDA-FO, when f includes the set of nodes that may fail, which allows the created VNT to be more robust against the failures at these nodes. If f loops over IDs of all nodes, the algorithm creates a VNT, which is robust against all possible node failures.

As a first stat, the routing algorithm is run to determine the paths on line 6 in Fig. 2. When there are multiple possible

```
1:  function CollectData
2:    for $f \leftarrow$ ID of nodes, which may fail, and finally $n$ do
3:      if $f < n$ then
4:        simulate failure of node $f$
5:      end if
6:      $routing[f] \leftarrow$ new routing table
7:      for each node pair $src$ and $dst$ do
8:        $hop[f][src][dst] \leftarrow$ number of hops from $src$ to $dst$
9:      end for
10:     for each lightpath from $src$ to $dst$ do
11:       $pass[f][src][dst] \leftarrow$ number of s-d pairs on lightpath from $src$ to $dst$
12:     end for
13:     for each $src, dst$ pair without a direct lightpath do
14:       $decrease[f][src][dst] \leftarrow$ number of node pairs whose hop count will decrease if a lightpath is established from $src$ to $dst$
15:     end for
16:     if $f < n$ then
17:       node $f$ recovers
18:     end if
19:   end for
20:   return $routing$, $hop$, $pass$, $decrease$
21: end function
```

Fig. 2. Collecting data on the current logical topology

paths, the selection varies with the implementation of the algorithm, so the exact behavior of the routing algorithm must be known. Using the routing information, the hop count distribution among s-d (source-destination) pairs is calculated on line 7. Then the number of total number of s-d pairs on each lightpath is calculated on line 10 and stored in the array $pass$. On line 13, we find the number of node pairs whose hop count will decrease if a lightpath is established between a s-d node pair and store it as a metric for this s-d pair in an array denoted by $decrease$.

After collecting the stats, the main algorithm starts selecting the s-d pairs for establishing lightpaths in Fig. 3. Among the possible candidates, the algorithm gives priority to the ones, which will decrease the number of s-d pairs on the lightpaths, which are carrying the highest number of s-d pairs. Therefore, the values in the $pass$ array is sorted in descending order with corresponding (f, src, dst) on line 5. By the FOR loop on line 7, the algorithm loops over $pass$ list with corresponding (f, src, dst) in order to decrease the number of s-d pairs on the lightpath from src to dst with the failure scenario f, until a new lightpath is established or $pass$ is empty. In order to decrease the s-d pair count on the selected lightpath, the algorithm tries to establish a direct lightpath between one of the s-d pairs on this lightpath. Therefore, the algorithm creates a list of s-d pairs passing through this lightpath and stores them in the array $impact$ with an impact factor metric, which is the amount of decrease in total hop count between all node pairs after a lightpath is established between this

```
1:  Initialize()
2:  repeat
3:    CollectData()
4:    $stop\_pass \leftarrow 0$
5:    sort $pass[f][src][dst]$ in descending order
6:    repeat
7:      repeat
8:        get next $(f, src, dst)$ in the sorted $pass$ list
9:        set the routing table to $routing[f]$
10:       for each node pair $src2$ and $dst2$ whose route includes the lightpath from $src$ to $dst$ do
11:         $impact[src2][dst2] \leftarrow decrease[f][src2][dst2] * (hop[f][src2][dst2] - 1)$
12:       end for
13:       sort $impact[src2][dst2]$ in descending order
14:       repeat
15:         get next $(src2, dst2)$ in the sorted $impact$ list
16:         if $token\_tra[src2] > 0$ AND $token\_rec[dst2] > 0$ AND $tra[src2] > 0$ AND $rec[dst2] > 0$ then
17:           establish a lightpath from $src2$ to $dst2$
18:           decrease $tra[src2]$, $rec[dst2]$, $token\_tra[src2]$, $token\_rec[dst2]$ by one
19:           $stop\_pass \leftarrow 2$
20:         end if
21:       until $stop\_pass$ is 2 OR $impact$ list is empty
22:       if $stop\_pass$ is 0 AND $pass$ list is empty then
23:         if MIN($token\_tra$)$\leq$0 OR MIN($token\_rec$)$\leq$0 then
24:           increase $token\_tra, token\_rec$ of nodes by one
25:           $stop\_pass \leftarrow 1$
26:         else
27:           $stop\_all \leftarrow 1$
28:         end if
29:       end if
30:     until $stop\_pass > 0$
31:     re-initialize $pass$ to last sorted list
32:   until $stop\_all$ is 1 OR $stop\_pass$ is 2
33: until $stop\_all$ is 1
```

Fig. 3. The main algorithm

s-d pair. After sorting the impact factor list in descending order with corresponding $(src2, dst2)$, the algorithm tries to establish a lightpath among s-d pairs $(src2, dst2)$ in the list in a loop starting on line 14. Before establishing the lightpath, the algorithm checks whether the lightpath satisfies the conditions like the node pair has enough number of tokens, transmitter and receivers. It may also need to satisfy other conditions like the maximum number of wavelengths.

While not mandatory, checking whether the congestion probability decreases after establishing the lightpath can further decrease the congestion probability. If the distribution of the traffic matrix is known, the non-congestion probability for a given of s-d pairs on a lightpath can be estimated by a simulation. As a comparison metric, the overall non-congestion probability of a VNT can be roughly approximated by multiplying the non-congestion probability of all lightpaths and the lightpath is established only if it decreases.

If a new lightpath is established, the algorithm stops trying establishing new lightpaths and goes back to the beginning of the main loop on the line 2. When $pass$ is empty and no new lightpaths are established, the algorithm checks the tokens of all nodes. If there is a node with token less than one, the algorithm increases the tokens of all nodes by one so that more nodes can establish a lightpath and re-runs the loop after re-initializing the $pass$ array to the last sorted list. Otherwise, the algorithm stops and outputs the designed VNT. As many (f, src, dst) give the same value in $pass$ and $impact$ arrays, it is possible create different VNTs by random shuffling the order of (f, src, dst) sets before sorting the $pass$ and $impact$ arrays. Due to the token based architecture, it is difficult to state an order of complexity to the algorithm. As a reference, it takes around 40 minutes to design a VNT on a Waxman topology with 100 nodes and 400 optical fibers with 16 transmitters and receivers per node, using a not-so-optimized single-threaded C++ simulator on a single core of Intel 3960x CPU. Speed improvement of several orders of magnitude seems possible using an optimized and multi-threaded program.

III. Attractor Selection Control

In biological systems, the interaction between metabolic reaction network and gene regulatory network controls the growth of cells. The gene regulatory network produces the proteins necessary for the cell growth. In the metabolic reaction network, proteins use the nutrition in the environment and produce the substances necessary for the growth. When the system conditions are preferable, the deterministic behavior drives the system to converge to an attractor. If the conditions are not preferable, the stochastic behavior dominates the system. The system searches for a new attractor by randomly changing the state by adding noise.

A. VNT Control

As the number of receivers and transmitters in a node is limited, it is not possible establish lightpaths on each wavelength between adjacent nodes. Therefore a virtual network topology can be drawn by considering the lightpaths as direct links between their receiver and transmitter nodes. As the maximum number of lightpaths on a fiber or node is limited, a VNT control is necessary to choose the optimum set of lightpaths in a network. In a biological system, the gene regulatory network adapts to the changing environment by taking the growth rate as a feedback as a result of the metabolic reaction network. In our work, we tried to adapt to the changing IP network conditions by reconfiguring the VNT, so we interpret the VNT as the gene regulatory network and the IP network as the metabolic reaction network. Our aim is to minimize the congestion on the highest utilized lightpath in order to decrease the packet drop rates and buffering delays. When node/link failures occur, the capacity of available lightpaths may no longer enough to carry the traffic and cause high congestion in the IP network. The VNT control method takes the maximum congestion level as a feedback and reconfigures the VNT until the maximum utilization in the IP network decreases below a threshold value.

B. Analytical Model of VNT Control

Each lightpath is controlled by a gene, so i-th lightpath has an expression level of x_i, which is calculated by

$$\frac{dx_i}{dt} = \alpha \cdot \left(f\left(\sum_j W_{ij} \cdot x_j - \theta \right) - x_i \right) + \eta. \qquad (1)$$

The α is the growth rate, which is calculated according the maximum utilization level in the IP network as a feedback. The η is the Gaussian noise term showing the strength of stochastic behavior. If the maximum utilization level is high, α decreases, so η dominates the equation, which causes the VNT to change randomly to find a new attractor. The rate of change in the expression level by the deterministic behavior is given by the sigmoidal regulation function $f(z) = \tanh(\mu z)$, where μ is the gain parameter. W is the regulatory matrix, which shapes the system to convergence to an attractor state when the growth rate is high. θ is the threshold value for expression level. The maximum utilization level u_{max} is converted to α by

$$\alpha = \frac{1}{1 + exp(\delta \cdot (u_{max} - \zeta))}, \qquad (2)$$

where δ is the gradient and the ζ is the threshold utilization parameter. When the u_{max} surpasses ζ, the value of α converges to zero, which means low growth rate. In this case, the noise term η dominates the control and the VNT changes randomly, until the VNT control finds a VNT with low u_{max}. After each iteration, the x_i values are sorted from highest to lowest and the corresponding lightpaths are established in this order.

The regulatory matrix W is a Hopfield neural network containing a set of possible attractors [9], [10]. We use it as an associative memory to store the attractors by using orthogonal projection [11], [12]. Let's assume that we have m attractors, where attractor k has the VNT expression vector $x(k) = (x_1^{(k)}, \dots, x_i^{(k)})$. The VNT lightpaths are coded according bipolar coding by setting x_i to 1 if the lightpath is established and -1 otherwise. Let X be a matrix whose rows are the attractors. Using the pseudo-inverse matrix X^+, the regulatory matrix is calculated by $W = X^+X$.

IV. Performance Evaluation

We evaluated the performance of the proposed algorithm against network failures by a simulation study. We used Waxman model with default parameters in BRITE tool [13] to create a physical topology with 100 nodes and 400 optical fibers, one optical fiber for each direction. We also tried other topologies like Erdős-Rényi model available in BRITE tool and got similar simulation results. The number of transmitters

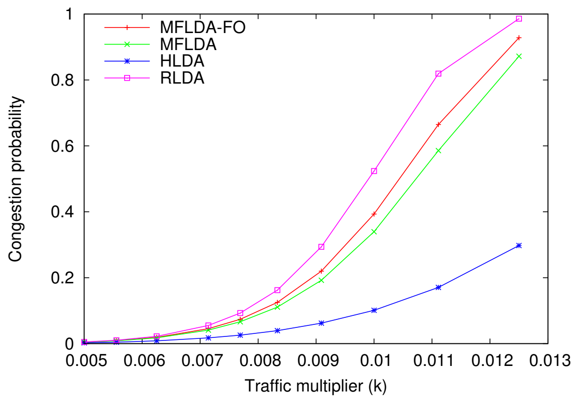

Fig. 4. The congestion probability when there is no failure

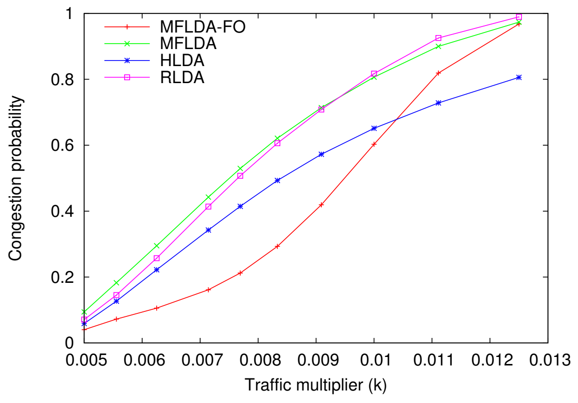

Fig. 5. The congestion probability after 5 nodes fail

and receivers in a node was limited to 16. As the transmitter/receiver count was the main limit, there were enough number of wavelengths on a fiber to carry the lightpaths. The amount of traffic per s-d node pair had a LogNormal(-0.5,1) distribution. In order to show the effect of traffic intensity, the traffic matrix was multiplied by k. A VNT was marked as congested if the utilization of one of its lightpaths was more than 50%. The attractor selection algorithm parameters were $\mu = 10$, $\delta = 50$, and $\zeta = 0.5$. The variance N of the noise

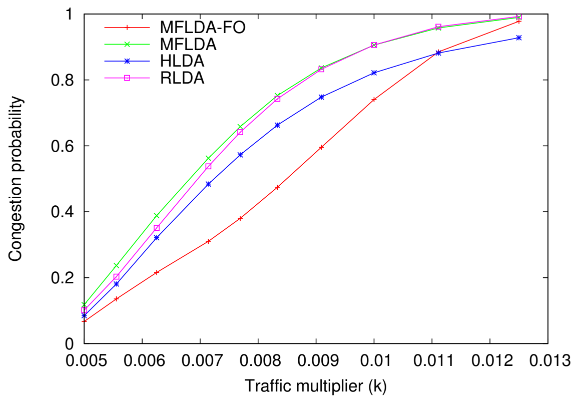

Fig. 6. The congestion probability after 10 nodes fail

η was 0.15. Shortest path routing is applied on both physical and logical topologies.

We compared four different VNT candidate sets. The VNT candidates denoted by MFLDA were designed by the proposed algorithm without optimization for node failure, by setting f to only n on line 2 of the pseudocode, which prevents an optimization for failures. The VNT candidates denoted by MFLDA-FO were designed by the proposed algorithm with optimization for possible node failures by looping f from 0 to n. A lightpath is established only if it decreases the overall non-congestion probability. For comparison, the VNT candidates denoted by RLDA (Random Logical Topology Design Algorithm) and HLDA (Heuristic Logical topology Design Algorithm) [2] were also simulated. In RLDA, the VNTs were created by establishing lightpaths among randomly chosen node pairs. In HLDA, the VNTs were created by establishing lightpaths among the s-d pairs with the highest traffic according to the traffic matrix. In order to maximize the performance of HLDA in failure scenarios, HLDA is applied to the topology after failure with the knowledge of the failed nodes. Therefore, the VNTs created by HLDA were specially designed for the topology after failure. On the other hand, we simulate MFLDA and MFLDA-FO under harsher conditions without providing the traffic matrix information and the place of failed nodes. As the exact place of failed nodes are not known to the network, when the transmitters/receivers of failed lightpaths become idle, these idle transmitters/receivers were used for establishing new lightpaths among randomly chosen node pairs.

Each VNT candidate set was simulated with 500.000 traffic matrices and failure patterns to estimate their congestion probability. A set of 10 VNTs were designed by both MFLDA-FO and MFLDA. In each simulation one of the 10 designed VNT candidates was randomly selected and simulated. In order to increase the randomness, a different VNT was used by RLDA in each simulation. As HLDA optimizes the VNT for a given traffic matrix and failed node set, again a different VNT was designed and used by HLDA in each simulation. When a node fails, the lightpaths passing through its fibers fail at the same time. Instead of rerouting the failed lightpaths, the traffic on the failed lightpaths is rerouted to other available lightpaths like in [7].

Fig. 4 shows that when there was no node failure, HLDA gave the lowest probability of congestion. As HLDA has the traffic matrix information and it designs a specific VNT for each traffic matrix, this is an expected result. However, our aim is to design VNT candidates without traffic matrix information. In Fig. 4, MFLDA gave lower congestion probability than MFLDA-FO. The reason is that the failure optimization causes the VNT to include backup paths against possible failure scenarios. These lightpaths may not be so useful when there is no failure, so the congestion probability of MFLDA-FO was a bit higher than MFLDA. In all simulations RLDA gave the highest congestion probability.

Fig. 5 and 6 show that when 5 and 10 nodes failed, MFLDA-FO gave lower congestion probability than the others unless

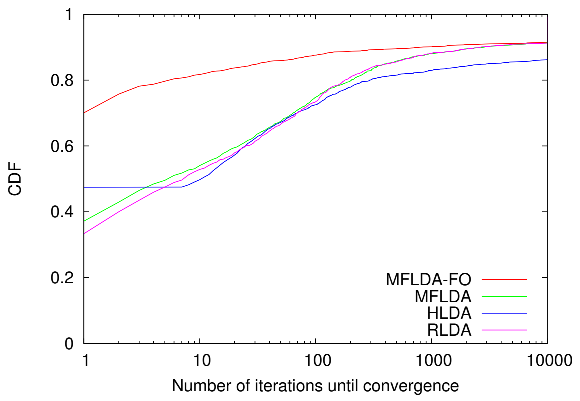

Fig. 7. The CDF of convergence time to a solution after 5 nodes fail

the traffic was too high. While HLDA had both the traffic matrix and failed node list information, it could not create direct lightpaths among some of the s-d pairs with high traffic, whose lightpath route pass through failed nodes. As HLDA does not provide any optimization for these s-d pairs, they may end up using multiple lightpaths and concentrate on some lightpaths and cause congestion. On the other hand, MFLDA-FO optimizes the VNT by taking failures into account to prevent hot-spots, so it gave lower congestion probability unless the traffic was too high. As many lightpaths become unavailable and the characteristics of the topology greatly changes after multiple node failures, the optimizations by MFLDA no longer work, so the VNTs designed MFLDA gave similar congestion probability to RLDA.

While MFLDA-FO had lower probability of congestion right after failure, not all possible failure scenarios could be solved by VNT optimization only. In such cases, attractor selection mechanism allows solving complex failure scenarios after some iterations. However, the convergence time to a solution depends on the performance of the attractors used. Fig. 7 shows the cumulative distribution function (CDF) of the number of iterations by attractor selection algorithm until it finds a VNT which has maximum lightpath utilization less than 50%. The VNT candidates designed by MFLDA-FO, MFLDA, HLDA and RLDA were set as the attractors of the attractor selection algorithm and the initial VNT. Each attractor set was simulated with 2000 different failure patterns and traffic matrices. The traffic intensity k was set to 0.0083. Five randomly chosen nodes fail before the first iteration. The x-axis is the number of iterations until convergence. As seen in the figure, the attractors designed our MFLDA-FO gave much faster convergence than both RLDA and HLDA algorithms, even though HLDA had both the traffic matrix and failed node list information that were not available to MFLDA-FO. Around 10% of the simulations could not converge as there was no solution or the solution domain was too small.

V. Conclusion

In this paper, we proposed an algorithm called MFLDA for designing VNTs, which can accommodate various traffic patterns. We also presented an extended version called MFLDA-FO to design VNTs robust against congestion due the traffic changes after network failures. The simulation results showed that the VNT candidates designed by MFLDA-FO can accommodate a wide range of traffic both before and right after a failure of multiple nodes. Moreover, the simulations showed that the converge time is faster when these VNTs are used as attractors in an attractor selection algorithm, compared to the attractors designed by HLDA algorithm. Unlike HLDA, our VNT design algorithm and the attractor selection algorithms do not require traffic matrix information and failed node list. While it is easy to estimate the routing stats used in our algorithm when shortest path routing is applied, it may be difficult with some routing algorithms. As a future work, we will investigate the possible implementation issues with other routing algorithms and evaluate their performance.

Acknowledgment

This research was supported in part by Grant-in-Aid for Scientific Research (A) 15H01682 of the Japan Society for the Promotion of Science (JSPS) in Japan.

References

[1] B. Mukherjee, D. Banerjee, S. Ramamurthy, and B. Mukherjee, "Some principles for designing a wide-area WDM optical network," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 5, pp. 684–696, Oct 1996.

[2] R. Ramaswami and K. Sivarajan, "Design of logical topologies for wavelength-routed optical networks," *Selected Areas in Communications, IEEE Journal on*, vol. 14, no. 5, pp. 840–851, Jun 1996.

[3] Y. Liu, H. Zhang, W. Gong, and D. Towsley, "On the interaction between overlay routing and underlay routing," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4, March 2005, pp. 2543–2553 vol. 4.

[4] A. Gencata and B. Mukherjee, "Virtual-topology adaptation for WDM mesh networks under dynamic traffic," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 2, pp. 236–247, Apr 2003.

[5] Y. Koizumi, T. Miyamura, S. Arakawa, E. Oki, K. Shiomoto, and M. Murata, "Adaptive virtual network topology control based on attractor selection," *Lightwave Technology, Journal of*, vol. 28, no. 11, pp. 1720–1731, June 2010.

[6] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on Internet traffic identification," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 3, pp. 37–52, rd 2009.

[7] Y. Koizumi, S. Arakawa, S. Kamamura, D. Shimazaki, T. Miyamura, A. Hiramatsu, and M. Murata, "Adaptability of virtual network topology control based on attractor selection against multiple node failures," in *18th OptoElectronics and Communications Conference / Photonics in Switching (OECC/PS)*, June 2013.

[8] T. Ohba, S. Arakawa, Y. Koizumi, and M. Murata, "Scalable design method of attractors in noise-induced virtual network topology control," *J. Opt. Commun. Netw.*, vol. 7, no. 9, pp. 851–863, Sep 2015.

[9] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[10] Y. Baram, "Orthogonal patterns in binary neural networks," Technical Memorandum 100060, NASA, 1988.

[11] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[12] Y. S. Hanay, Y. Koizumi, S. Arakawa, and M. Murata, "Virtual network topology control with oja and apex learning," in *Proceedings of the 24th International Teletraffic Congress*. International Teletraffic Congress, 2012, p. 47.

[13] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proceedings of MASCOTS '01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 346–353.