DyCroNO: Dynamic Cross-layer Network Orchestration and Real-time Deep Learning-based Network Load Prediction

Venkat Sai Suman Lamba Karanam and Byrav Ramamurthy
School of Computing
University of Nebraska-Lincoln
Lincoln, NE, USA
saisuman@huskers.unl.edu and ramamurthy@unl.edu

***Abstract*— In this paper, we present Dynamic Cross-layer Network Orchestration (DyCroNO), a dynamic service provisioning and load balancing mechanism for IP over optical networks. DyCroNO comprises of the following components: i) an end-end (E2E) service provisioning and virtual path allocation algorithm, ii) a lightweight dynamic bandwidth adjustment strategy that leverages the extended duration statistics to ensure optimal network utilization and guarantee the quality-of-service (QoS), and iii) a load distribution mechanism to optimize the network load distribution at runtime. As another contribution, we design a real-time deep learning technique to predict the network load distribution. We implemented a Long Short-Term Memory-based (LSTM) method with a sliding window technique to dynamically (at runtime) predict network load distributions at various lead times. Simulations were performed over three topologies: NSFNet, Cost266 and Eurolarge using real-world traffic traces to model the traffic patterns. Results show that our approach lowers the mean link load and total resources significantly while improving the resource utilization when compared to existing approaches. Additionally, our deep learning-based method showed promising results in load distribution prediction with low root mean squared error (RMSE) and $\sim$90% accuracy.**

***Index Terms*—IP over optical networks; Software-defined network; Load distribution; Real-time Deep Learning.**

I. Introduction and Background

In this paper, we present a Dynamic Cross-layer Network Orchestration (DyCroNO) for IP over optical networks such as the IP-based MPLS over Optical Transport Networks (OTN) using the software-defined networking (SDN). The main contributions of this paper are summarized as follows. We address three problems native to IP over optical networks: (1) dynamic service provisioning, (2) dynamic resource allocation and (3) dynamic load distribution mechanism. Additionally, we introduce a real-time deep learning-based network load distribution prediction mechanism and integrate it into DyCroNO. We used autoencoder-based variation of Long Short-Term Memory (LSTM) [1] method modified with a sliding window procedure to fit our problem statement. DyCroNO is designed using SDN paradigm with OpenFlow standard [2]. We assume Software-Defined IP-based MPLS over Optical Networks (SDION) and present results for the same. However, our proposed work can be extended to most multi-layer networks.

First, we discuss the limitations of current state-of-the-art and generalize them into categories. We later compare the results of DyCroNO against the generalized categories. The authors in [3] propose a software solution for tunable configuration and routing selection in optical networks called HeCSON. HeCSON selects a configuration for the lightpath using combination of data rate, modulation scheme and forward error correction based on the optical fiber's characteristics (number of channels, length, etc). We categorize works along this line as Bandwidth Separation Allocation methods (BSA). BSA separates the services and over-provisions based on their initial bandwidth request instead of the bandwidth they actually use.

The authors in [4] propose RADWAN, an optical link capacity reconfiguration approach that adapts the rate based on the measured SNR in the physical layer channel using a simple heuristic. Motivated by reducing transfer completion times, utilization and transfer costs, authors in [5] optimized bulk datacenter transfers connected via optical networks by formulating the resource provisioning as an optimization problem. The authors in [6] proposed an online linear programming approach to improve transfer completion times by optimizing the sum of flow times. The works proposed in [4]–[6] are limited to specific transport services (single layer) which makes them difficult to adopt jointly to IP over optical networks. The existing works on joint service provisioning in IP-based optical WANs use priority matching between the end-to-end (E2E) services and the virtual resources based on the QoS requirements [7], [8]. We generalize these works as Bandwidth Aggregation Allocation methods (BAA). BAA methods utilize the maximum or peak traffic statistics of the service to allocate bandwidth. Follow-up adjustments to the bandwidth is done based on the monitored resource utilization by the services.

The authors in [9] have highlighted the gap between the optical and IP networks for joint provisioning in optical WANs. The authors in [10] propose an integer linear programming approach for joint routing over non-segregated static and reconfigurable links. These works take considerable time to converge and do not operate on the sub-second time. We generalize these works as Virtual Dynamic Bandwidth Allocation (VDBA). VDBA methods adopt dynamic bandwidth adjustment in re-

978-3-903176-54-6

sponse to varying runtime resource demands, most often using SDN. VDBA methods treat the individual IP and optical layers as blackboxes. Existing research tells us that IP and optical layers each exhibit their own influence on the traffic flowing through the network [9]. Consequently, the runtime demands of the services in IP over optical networks exhibits high temporal and spatial locality [11], [12]. This can lead to resource over or under utilization and may impact the QoS for the services.For such an effort, individual transport layers cannot be treated as blackboxes to make accurate predictions. This requires not only the runtime monitoring of the virtual network resources but also the IP and optical elements comprising them. Additionally, estimating traffic demands currently and into the future in IP over optical networks is needed so that dynamic resource adjustments, most often in sub-second time frames. The results presented in Section III compare DyCroNO against BSA, BAA, and VDBA strategies. Rest of the paper is organized as follows. Section II presents the technical design, the algorithms and the proposed deep learning method. Section III presents the results. Section IV concludes the paper.

II. DyCroNO

In this section, we present the technical details of the proposed mechanism, DyCroNO. The symbols and definitions used in this section are described in Table I.

A. Problem Statement

Consider a software-defined IP over optical network. Due to the runtime dynamics, (1) the network resource usage (e.g. links) is imbalanced i.e. load imbalance exists, (2) some services are not being served up to their QoS, and (3) some resources can be re-provisioned from services with loose QoS requirements. While these can be addressed by following an algorithmic optimization similar to the existing works discussed in Section I, it does not ensure that (i) the overall network utilization is improved, (ii) the load distribution across the network is uniform i.e. there are no bottlenecks and, (iii) the re-provisioning can be made in advance by predicting the network usage and service QoS.

B. Methods

In this subsection, we explain the methodology of DyCroNO. First, we define the concept of virtual container and explain its usage in DyCroNO. Second, we explain the three main components in DyCroNO, namely, a) the service provisioning (par. II-B0a), b) an improved dynamic bandwidth adjustment (DBA) (par. II-B0b), and c) the load distribution (par. II-B0c). Three algorithms, namely, Algs. 1, 2, 3 are referred when the components are explained. The definitions of terminology used are presented in the Table I. Last, we present the proposed real-time deep learning approach for predicting network loads in Section II-C.

a) Service Provisioning: When a new service request arrives, the controller does a best match with the virtual container that can serve the request while considering its priority, required bandwidth, and maximum allowed turnaround time (1). The service rank r_{s_i} for service s_i is given by the 3-tuple $(u_{s_i}, \mu_{i,req}, b_{i,req})$. u_{s_i} is the user given priority to the service, $\mu_{i,req}$ is the required throughput calculated from the $(amount\_of\_data/max\_allowable\_time)$ and $b_{i,req}$ is the required minimum bandwidth. Each v_i has a set of circuits C_i and each circuit c_{i_j} in C_i has a length (i.e. number of hops), available bandwidth, and a guaranteed minimum turnaround time. To make sure that there are no large load imbalances, the distribution of services that on the circuits in C_i need to be as

Algorithm 1 Service Provisioning

procedure ASSIGN_VIRTUAL_CONTAINER(s_i)
  Create $list\_0$ of virtual containers matching rank r_{s_i}
  Create $list\_1$ from $list\_0$ with $\frac{b_{v_i} - b_{R_i,P_{v_i}}}{|C|} \leq th_{u,P_{V_i}}$
  Order $list\_1$ based on rank
  Pick the first virtual container v_j from $list\_1$
  Order all circuits c_k in C_j based on their ranks
  Split circuits in C_j into C_a, C_b and C_c for top, middle and bottom 33% in their ranks
  if r_{s_i} is high **then**
    Pick the circuit c_k with lowest $util_{c_k}$ from C_a
  else if r_{s_i} is medium **then**
    Pick the circuit c_k with lowest $util_{c_k}$ from C_b
  else
    Pick the circuit c_k with lowest $util_{c_k}$ from C_c
procedure SERVICEFINISH(v_j)
  while true **do**
    if Service s_{i_j} finishes **then**
      Call UpdateVirtualContainerRank(v_j)
      Call UpdateCircuitMetrics(c_k)

Algorithm 2 Dynamic Bandwidth Adjustment

procedure DBA(s_i)
  Call UpdateServiceProgress()
  All services with $progress_{s_i} \geq \alpha \rightarrow S\_heavy$
  All services with $progress_{s_i} < \alpha \geq \beta \rightarrow S\_medium$
  Put rest of the services into list S_light
  for each s_k in S_heavy, S_medium and S_light **do**
    Order on $dr_{s_i}/tr_{d_{s_i}}$
  Update $util_{v_i}$
  All v_j with $util_{v_j} \geq th_{high} \rightarrow V\_heavy$
  All v_j with $util_{v_j} < th_{high} \geq th_{medium} \rightarrow V\_medium$
  Put rest of the virtual containers into list V_light
  $Predictions = MovingWindow\_LSTM()$
  Call $LoadBalance()$
  for all virtual containers in each list V_heavy, V_medium and V_light **do**
    Assign first service s_i from s_heavy to first virtual container in V_heavy that matches the required source-destination pair
procedure UPDATEVIRTUALCONTAINERRANK(v_j)
  $rank_{v_j} = (b_{R_j,P_{v_j}}/|C|, b_{R_j}/|C|, b_{v_j}/|C|, th_{n_i} - curr_{n_i})$
procedure UPDATECIRCUITMETRICS(c_k)
  Update $\tau_{c_k} = [\tau_{c_{k,prev}} * (\frac{s}{1+d})] + \tau_{c_{k,prev}} * [1 - \frac{s}{1+d}]$
  Update $util_{c_k} = [util_{c_{k,prev}} * (\frac{s}{1+d})] + util_{c_{k,prev}} * [1 - \frac{s}{1+d}]$

Table I
TERMINOLOGY

Symbol	Definition	Symbol	Definition
S	set of all services in the network	s_{i_j}	some service s_j in virtual container v_i
C	set of all circuits in the network	C_i	Set of all circuits in v_i
R_j	set of service requests for v_j	c_{i_j}	some circuit c_j in C_i
r_{s_i}	rank of service s_i	$rank_{v_j}$	rank of v_j
b_{R_j}	currently used bandwidth in v_j	$b_{R_j,P_{v_j}}$	total bandwidth by all services in R_j with priority p_{v_j}
b_{v_j}	sum of bandwidths of the circuits in C_j belonging to v_j	$curr_{n_i}$	the current number of services being served by v_i
$util_{c_k}$	current utilization value of c_{i_k}	$util_{c_{i_{k,prev}}}$	previously computed util. value of the circuit c_{i_k}
$\tau_{c_{k,prev}}$	previous turnaround time on c_k	th_{n_i}	number of services that virtual container v_i can support
avg_util_c	measured average util. of all circuits	τ_{c_i}	the calculated average turnaround time on c_i
S_{light}	lightly served services. S_{medium} and S_{heavy} are moderately and heavily served.	V_{light}	lightly utilized virtual containers. V_{medium} and V_{heavy} are moderately and heavily utilized virtual containers.
$util_{v_i}$	utilization of the virtual container v_i	th_{high}	upper threshold for $util_i$. Similarly, th_{med}, th_{light} are middle and lower thresholds.
$th_{u,P_{v_i}}$	bandwidth upper threshold of v_i	$tr_{d_{s_i}}$	time remaining for s_i
α, β, γ	upper, middle and lower thresholds of service's progress (in %)	$progress_{s_{i_j}}$	unit-less metric that represents how much the service in virtual container v_j has been served till now

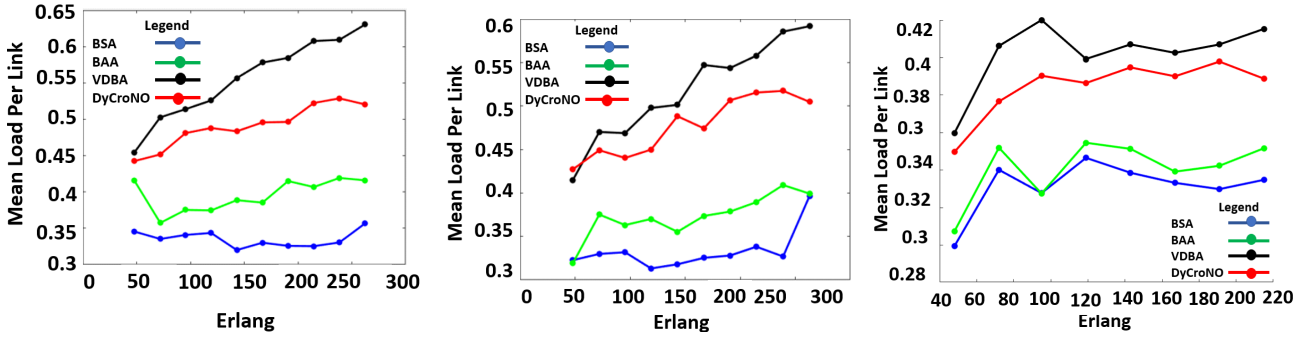


Figure 1. Mean load per link as the ratio of occupied versus free bandwidth in: (a) NSFNet, (b) Cost266 and, (c) EuroLarge networks.

uniform as possible. Each c_{i_j} in C_i has a rank associated with it, which can be given as $c_{i_{j_{rank}}} = (\tau_{C_{i_j}}, c_{i_{j_{length}}}, c_{i_{j_{bw}}})$. $c_{i_{j_{bw}}}$ is the available bandwidth of circuit c_{i_j}. $\tau_{c_{i_j}}$ is the calculated average turnaround time given in the form of exponential smoothing average shown in Alg. 2 line 19. Here, s is the smoothing factor with value 2 and d is the time in days.

To encapsulate the spatio-temporal behavior, we go beyond the E2E utilization statistics. We form statistical relations between the physical network elements (the circuits) and the services with their virtual containers. For each circuit c_{i_i} in v_i, we calculate its historical utilization $util_{c_{i_i}}$ periodically. $util_{c_{i_i}}$ is calculated as an exponential smoothing average and is a network wide statistic (see Alg 2 line 20). This lets us prioritize the physical layer circuits in v_i that are most likely to be underutilized if not being chosen. This strategy is different from existing works, which consider the virtual containers as blackboxes. Finally, we calculate the rank of v_i as $rank_{v_i} = (b_{R_i,P_{v_i}}/|C|, b_{R_i}/|C|, b_{v_i}/|C|, th_{n_i} - curr_{n_i})$ (Alg. 2 line 17).

b) Improved Dynamic Bandwidth Adjustment (DBA): To efficiently monitor a service s_i's progress and make DBA decisions, we periodically keep track of the historical average

Algorithm 3 Periodic Update

1: **procedure** LOADBALANCE
2: **for** each v_j in V_heavy **do**
3: **for** each c_k in C_j **do**
4: **if** $util_{c_k} > avg\_util_c$ **then**
5: Add c_k to C_l belonging to the first virtual container v_l from list V_light with same source and destination
6: **if** Projected Finish Times for services in v_l <Previously Measured Finish Times **then**
7: Exclude c_k from list C_j
8: Repeat until $util_{v_j}$ equals th_{med}
9: **procedure** UPDATESERVICEPROGRESS(s_i)
10: Compute $progress_{s_i} = [progress_{s_{i,current}} * (\frac{s}{1+d})] + progress_{s_{i,prev}} * [1 - \frac{s}{1+d}]$
11: **procedure** UPDATEVIRTUALCONTAINERUTIL
12: **for** each virtual container v_i **do**
13: Compute $util_{v_i} = [util_{v_{i,current}} * (\frac{s}{1+d})] + util_{v_{i,prev}} * [1 - \frac{s}{1+d}]$

progress of the service since its inception. This also relieves some nuances associated with the service complexity. The historical average progress of s_i *i.e.* $progress_{s_i}$ is shown in Alg. 3 line 10. Here, $progress_{s_{i,current}}$ is given by $dt_{s_i}/td_{d_{s_i}}$ where dt_{s_i} is the amount of data transferred, $td_{d_{s_i}}$ is the time since data transfer began. We break any ties with $dr_{s_i}/tr_{d_{s_i}}$, where dr_{s_i} is the amount of data remaining and $tr_{d_{s_i}}$ is the time remaining expressed as $(maximum\_allowable\_time\_limit - time\_since\_data\_transfer\_began)$.

c) Load Distribution: We calculate the ranks of virtual containers and the circuits when provisioning the services to help balance the load within a virtual container (and the network, by extension) while still meeting the QoS. We use the historical average utilization of each v_i, given by $util_{v_i}$ in Alg. 3 line 13 in making load balancing decisions. Here, $util_{v_{i,current}}$ is the variance of the utilization of all circuits in C *i.e.* $\sigma^2(util(C))$. Larger value of $util_{v_{i,current}}$ implies uneven historical distribution of utilization among the circuits in v_i. Line 6 in Alg. 3 is the "make before break" condition which checks if the new allocation for the container– including the constant overhead for reallocating the circuits– improves the finish times of services carried by it. If the reallocation does not improve the finish times then no reallocation is made.

C. Real-time Deep Learning-based LSTM Method for Network Load Distribution Prediction

We use a well-known version of the deep learning method called LSTM) [13] after modifying it to support a sliding window procedure. We encode the multivariate time-series data collected from the network at run-time into an input sequence using a sliding window methodology. For example, a given day may be divided into x time slices, which means that the input sequence for each day consists of link utilization metrics collected at x interleaved steps *i.e.* time-steps of length x. Each time slice consists of k input features or variables used in a multivariate fashion to make predictions in time slice $x + 1$. This encoded input sequence is fed to the next component (or submodel), a hidden encoder layer, chosen as an LSTM model. The encoder layer interprets and encodes the data and feeds it to a dense decoder layer. The decoder layer interprets the encoded data and feeds the final layer, the dense output layer. The final single value output is the prediction for the link load distribution for one time slice $x + 1$. The sliding window now moves over by one time slice and the process is repeated. We forego the in-depth implementation specifics of the conventional LSTM for the sake of brevity [1]. We modified the conventional LSTM model with a sliding window variation specifically to learn and predict in run time. Figure 2 shows one iteration of our model. We strategically run the model at Alg. 2 line 12 *i.e.* at each time slice so that learning is done in an iterative fashion until the current time slice and day (*i.e.* to-date). The runtime implementation of our model in the SDN controller opens the possibility to dynamically balance the loads based on predicted values.

III. Results and Discussion

We chose three topologies, namely, 1) NSFNet with 14 nodes and 19 links, 2) Cost266 with 16 nodes and 24 links and 3) EuroLarge with 43 nodes and 180 links. We used real-world traffic traces from a local Internet Service Provider (ISP) from the US midwest region. We compare the performance of DyCroNO with other methods discussed in I, namely, Static and Separated Bandwidth Allocation method (BSA), Static and Aggregated Bandwidth Allocation method (BAA) and Virtual DBA (VDBA). ODU0 and ODU2 are the supported OTN transport channels with a bandwidth of 1 Gbps and 10 Gbps respectively. The simulation runs services of 3 priorities high, mid and low.

a) Load Distribution: Figures 1a, 1b and 1c present the mean distribution of loads per link across the three topologies. The mean load per link of DyCroNO is just below VDBA while BSA and BAA present much lower loads per link. Our approach exhibits a relatively uniform distribution as well as lower values of link loads compared to the other methods method while still maintaining the desired QoS (as shown by blocking probability in Fig. 5). In BSA and BAA, bottlenecks exist because certain links exhibit very high link loads while rest of the network is lightly loaded. This can be attributed to the reallocation of circuits (paths) and services transported among them on top of the service re-provisioning. We discuss more on this when we present our results on resource utilization and blocking probability.

b) Resource Utilization: Figures. 3b and 3d present the instantaneous resource utilization for NSFNet and Cost266 topologies respectively. DyCroNO exhibits highest overall resource utilization in addition to the overall 55-60% reduced total resources (shown in Fig. 3). This is because: (i) our approach reduces the internal fragmentation of resource utilization within the virtual containers because the under utilized circuits are reallocated among existing virtual containers.)ii) Our approach reduces the external fragmentation of the virtual containers by prioritizing the under utilized circuits for service re-provisioning instead of a simple priority matching.

Figure 4a shows the total network resource utilization and the load distribution of the network links measured as a standard deviation (std. dev.) of utilization. The total resource utilization steadily decreases for the first few time slices then stabilizes. The link utilization distribution shows significantly lower std. dev. values over one day. Lower std. dev. values imply higher uniformity in network load distribution. Figure 4b shows the results of total network resource utilization and the distribution of load on the network link over a span of 10 days. The total network resource utilization stabilizes over the first couple of days while the link utilization distribution continues to show lower std dev. values. The improvements can be attributed to the $LoadBalance()$ procedure, which is called before every time slice in a day and hence the traffic is redistributed among the network resources (keeping in mind the relations between virtual containers, the physical circuits within them and the services running in the network).

c) Blocking Probability: Figure 5 presents the results of the blocking probability. Across all three topologies, DyCroNo maintains the lowest blocking probability (more evident at higher traffic loads). Higher blocking probability implies that newer services are blocked when the network cannot provision

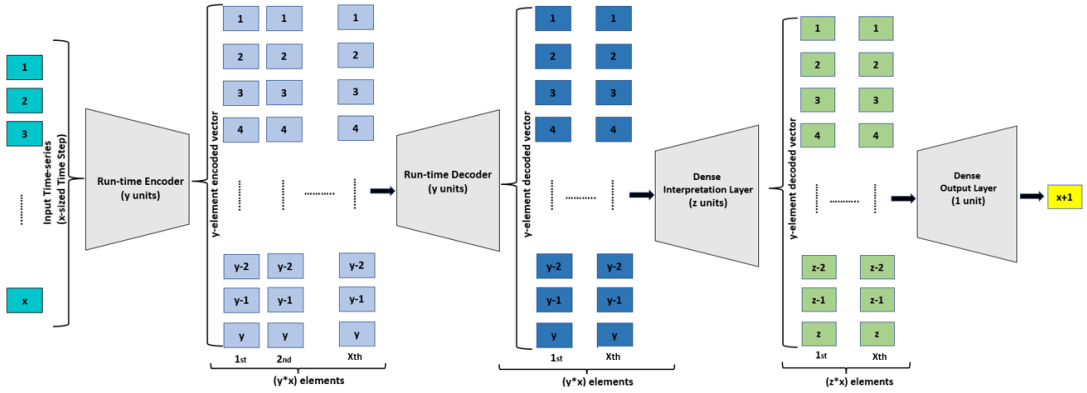
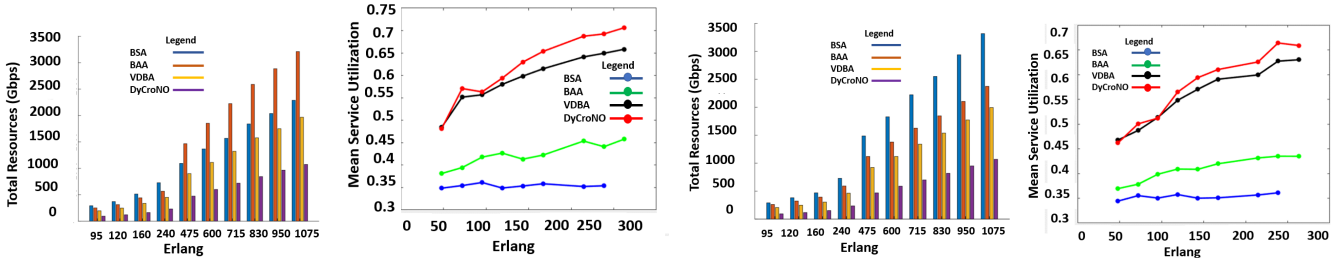
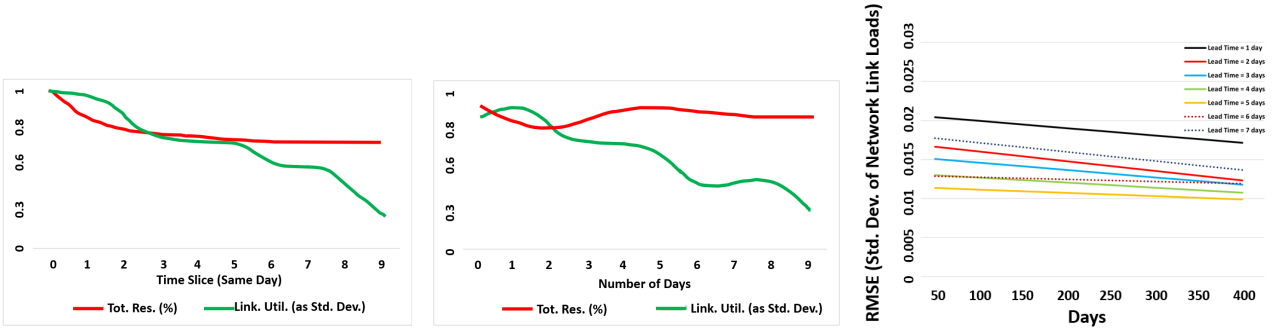


Figure 2. The input time-series data is collected during run-time and fed to the LSTM-based model during run-time. The input time-series consists of the run time statistics collected at x interleaved time steps. These statistics form the input features to the first layer *i.e.* the run-time encoder. Next, the encoder-decoder step outputs the decoded resultant units to the interpretation layer. Finally, the output layer outputs *1* unit of output, which is the predicted value for the *(x+1)* time step.



(a) (b) (c) (d)

Figure 3. (a), (c) Total occupied resources, (b), (d) Mean service Utilization i.e. the mean resource utilization by the services. which have high or mid-priority, (a), (b) using NSFNet topology. (c), (d) using COST266 topology



(a) (b) (c)

Figure 4. The total network resource utilization and the load distribution among network links (as std. dev.) in NSFNet topology using DyCroNO over (a) one day consisting of 10 time slices, (b) over 10 days. The traffic load was set to 120 Erlang. (c) RMSE values of the std. dev. of link load distribution in the network over several prediction lead times using our proposed deep learning-based method.

them. This happens when the circuits that are supposed to carry these services are overburdened. DyCroNO dynamically and periodically reroutes the services internally among the circuits in the virtual containers. DyCroNO's load balancing technique is different from existing works because the resource utilization of circuits inside each virtual containers are balanced, not just the virtual containers themselves as black boxes. As shown in Fig. 1, the network link loads are maintained relatively high but at a level while still being able to provision additional resources. Instead of a simple priority matching between services and the virtual containers, we (re)provision the specific services that have the highest likelihood of being blocked if not being served by selecting the best circuits inside the virtual containers (in addition to selecting the best virtual container).

d) Total Occupied Resources: Figures 3a and 3c show the total occupied network resources measured in Gbps for

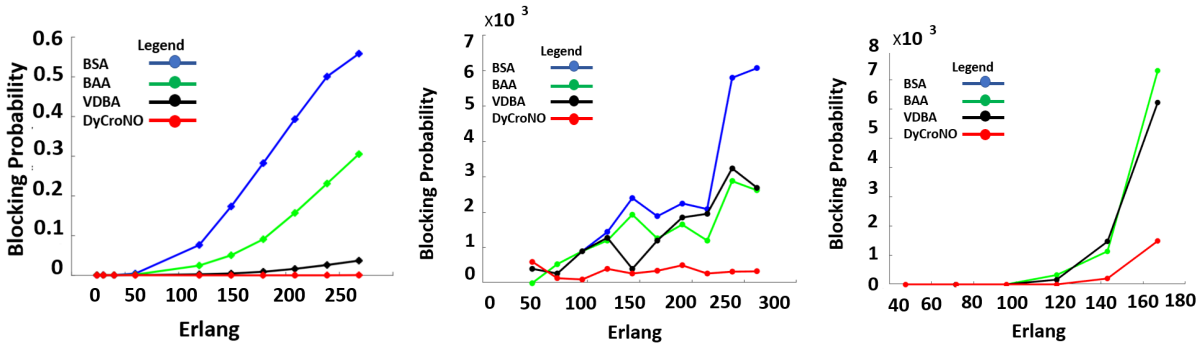


Figure 5. Blocking Probability over: (a) NSFNet, (b) Cost266 and (c) EuroLarge networks.

NSFNet and Cost266 topologies respectively. The total occupied resources are reduced by more than 100% for lower loads and by even greater levels for higher loads. BSA and BAA provision the services based on their peak demands. VDBA dynamically allocates the extra (unused) bandwidth to a lower priority service using a simple random weighted algorithm. Although this improves the network resource utilization, it fails in addressing the internal and external fragmentation among the circuits inside virtual containers and the virtual containers respectively. DyCroNo overcomes this limitation because it explicitly addresses the internal and external fragmentation (see III-0b).

e) Load Distribution Prediction: We use the std. dev. of the measured link utilization in the network as a metric for load distribution due to simplicity. Higher std. dev. values imply higher load imbalances in the network. Our deep learning model is applicable to any numeric metric other than std. dev., if need be. Figure 4c shows the root mean squared error (RMSE) results of the collected std. dev. of network link load distribution. The number of time slices x was set to 10, after experimenting with several values. The hyperparameters were set as follows: the number of epochs to 100, batch size to 50, LSTM hidden layer width to 200, dense layer width to 100, and output layer width to 1. We chose several prediction lead times ranging from 1 day to 7 days. We averaged the collected results shown in Fig. 4c over several runs of our algorithm. Our method predicts the link load distribution values with high accuracy (evidenced by the low RMSE values) across all prediction lead times.

IV. Conclusion and Future Work

In this paper, we presented DyCroNO for dynamic (1) bandwidth adjustment, (2) service provisioning, and (3) a load balancing mechanism for IP over optical networks. DyCroNo uses a real-time deep-learning based load prediction method using LSTM to make decisions. We presented the simulation results in an IP-based Packet over OTN network for blocking probability, resource utilization, resources utilization of the resources, total resources used, load distribution, and accuracy of load prediction. In the future, we will investigate the scalability of our methods. We would like to extend our approach to more than two types of networks (eg: optical transport network (OTN), wavelength division multiplexing (WDM), elastic optical network (EON) etc.). Additionally, we will investigate the scalability of our deep learning method at (i) much larger network sizes (eg: number of links $>$ 10,000) and (ii) various sliding window sizes (x).

V. Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant Number CNS-1817105.

References

[1] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[2] "120022, OpenFlow Specifications v.1.3.0, May, 2012." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[3] S. K. Patri, A. Autenrieth, D. Rafique, J.-P. Elbers, and C. M. Machuca, "HeCSON: Heuristic for configuration selection in optical network planning," in *Optical Fiber Communication Conference*. Optical Society of America, 2020, pp. Th2A–32.

[4] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "RADWAN: Rate adaptive wide area network," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 547–560.

[5] L. Luo, H. Yu, K.-T. Foerster, M. Noormohammadpour, and S. Schmid, "Inter-datacenter bulk transfers: Trends and challenges," *IEEE Network*, vol. 34, no. 5, pp. 240–246, 2020.

[6] M. Dinitz and B. Moseley, "Scheduling for weighted flow and completion times in reconfigurable networks," *IEEE INFOCOM*, pp. 1043–1052, 2020.

[7] S. Rahman, T. Ahmed, S. Ferdousi, P. Bhaumik, P. Chowdhury, M. Tornatore, G. Das, and B. Mukherjee, "Virtualized controller placement for multi-domain optical transport networks using machine learning," *Photonic Network Communications*, vol. 40, no. 3, pp. 126–136, 2020.

[8] R. Casellas, R. Martínez, R. Vilalta, and R. Muñoz, "Abstraction and control of multi-domain disaggregated optical networks with OpenROADM device models," *Journal of Lightwave Technology*, vol. 38, no. 9, pp. 2606–2615, 2020.

[9] R. Singh, M. Ghobadi *et al.*, "Run, walk, crawl: Towards dynamic link capacities," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 143–149.

[10] T. Fenz, K.-T. Foerster, S. Schmid, and A. Villedieu, "Efficient non-segregated routing for reconfigurable demand-aware networks," *Computer Communications*, vol. 164, pp. 138–147, 2020.

[11] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 1, pp. 1–29, 2020.

[12] C. Avin and S. Schmid, "Toward demand-aware networking: a theory for self-adjusting networks," *ACM SIGCOMM Computer Communication Review*, vol. 48, no. 5, pp. 31–40, 2019.

[13] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.