Dynamic Routing and Wavelength Assignment in Networks with Wavelength Conversion using Fast Learning Automata and Deterministic Learning Automata

Ammar Muqaddas
ammar.mu@ku.edu.kw
Feb - 2019

This work is still incomplete.

Acknowledgment

Dr. Hamed Al-Azmi has collaborated with me on a DIFFERENT and improved version of this paper. But since the work was never complete due to various reasons. I release this partial paper WITHOUT including his contributions to the best of my knowledge. Still worm thanks go to him for his valued help.

Simulation Tool

I've built my own simulation tool for this work. Called Hegons, A HEterogeneous Grooming Optical Network Simulator. Supporting mixed routing & wavelength assignment algorithms and optional wavelength conversions capability on each node. Link in reference [31]

Fast Learning Automata

In the following two sections we will illustrate two main problems that stand in the face of learning automata's success as a routing and wavelength assignment algorithm in converter enabled optical networks. We will also propose two new algorithms trying to solve these problems. Combining the two algorithms is what we call "FLA" which is our proposed algorithm that we're going to show the results for.

Non-Exhaustive Learning Automata (NXLA)

The use of learning automata (LA) in routing and wavelength assignment has shown promising results [1]. However, in optical networks with nodes that have wavelength conversion capability, the LA technique's complexity and memory requirements grows exponentially once 3 or more converter nodes are added to the network. The reason behind this difficulty is as follows.

Learning automata (as described in [1]) considers all possible route/wavelength items and associates a probability of selection for each combination item. But for networks with wavelength converter nodes, the number of route/wavelength combinations multiplies exponentially for paths (routes) that contain converter nodes. Figure 1 illustrates.

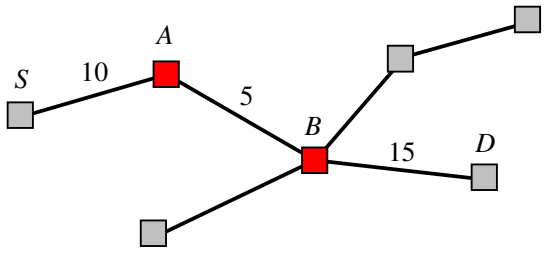


Figure 1, A and B are converter nodes. Numbers on links indicate wavelength capacity

Suppose that a call arrives that needs a connection from node *S* to *D*. The only shortest path available is *S,A,B,D*. Numbers on links indicate link capacities[*]. If nodes *A* and *B* had no conversions capability then the maximum number of wavelengths that can be used on path *S,A,B,D* is 5 (i.e wavelengths λ_1, λ_2, λ_3, λ_4, λ_5) since link *A,B* has a capacity of 5. Hence, for the LA algorithm, we have only 5 combinations (items) as shown in table 1.

Table 1: LA list

Path	Wavelength	Probability
S,A,B,D	λ_1	0.1
S,A,B,D	λ_2	0.4
S,A,B,D	λ_3	0.2
S,A,B,D	λ_4	0.1
S,A,B,D	λ_5	0.2

As shown, each of these combinations are associated with a probability of selection. These probabilities are the values for $P^{W}_{r_{(s,d)}}$ mentioned in [1] that are updated by the LA algorithm as described in the same paper. The probability values shown in table 1 (and all other tables) are just for example. For the rest of this paper, we call all such tables an **LA list**. We also refer to each row (RWA combination) as an **LA item**.

If node *A* was a converter node, the number of possible combinations will be 50 (5×10) as shown in table 2:

Table 2: LA list when A is a converter

Path	Wavelengths	Probability
S,A,B,D	λ_1 on link *S,A* and λ_1 on links *A,B* and *B,D*	0.01
S,A,B,D	λ_1 on link *S,A* and λ_2 on links *A,B* and *B,D*	0.02
S,A,B,D	λ_1 on link *S,A* and λ_3 on links *A,B* and *B,D*	0.001
S,A,B,D	λ_1 on link *S,A* and λ_4 on links *A,B* and *B,D*	0.09
….	….	….
….	….	….
S,A,B,D	λ_{10} on link *S,A* and λ_5 on links *A,B* and *B,D*	0.09

If nodes *A* and *B* both had conversion capability, then the number of combinations will be 750 ($5\times10\times15$). That's only for 2 converter nodes with a single path. Having multiple paths and more converter nodes will definitely cause the algorithm to consume huge amounts of memory and extended computation time in order to store and compute selection probabilities.

[*] Link capacity is the number of wavelengths that this link can carry. Note that wavelengths (in our simulation) are numbered from 0 upwards. For example, for link *S, A.* the wavelengths available are 0, 1, ,2…., 9

The exhaustive nature of the basic LA technique caused these difficulties. We intend here to experimentally explore the benefits of applying a non-exhaustive LA algorithm. Instead of considering all possible combinations in a single list of items, we propose creating multiple lists, one for each non-converting sub-path (segment). In other words, we will divide the full paths into segments that contain converting nodes only at their ending nodes. For our example above, when *A* is a converting node and *B* is not. We will have the **two** lists shown in table 3 and table 4.

Table 3: LA list for segment *S,A*

Path	Wavelength	Probability
S,A,B,D	λ_1 on link *S,A*	0.05
S,A,B,D	λ_2 on link *S,A*	0.2
S,A,B,D	λ_3 on link *S,A*	0.15
S,A,B,D	λ_4 on link *S,A*	0.2
S,A,B,D	λ_5 on link *S,A*	0.05
S,A,B,D	λ_6 on link *S,A*	0
S,A,B,D	λ_7 on link *S,A*	0.1
S,A,B,D	λ_8 on link *S,A*	0.1
S,A,B,D	λ_9 on link *S,A*	0.15
S,A,B,D	λ_{10} on link *S,A*	0

Table 4: LA list for segment A,B,D

Path	Wavelength	Probability
S,A,B,D	λ_1 on links *A,B* and *B,D*	0.5
S,A,B,D	λ_2 on links *A,B* and *B,D*	0.2
S,A,B,D	λ_3 on links *A,B* and *B,D*	0.2
S,A,B,D	λ_4 on links *A,B* and *B,D*	0.05
S,A,B,D	λ_5 on links *A,B* and *B,D*	0.05

When a new call arrives (from *S* to *D*), The LA algorithm is applied on the list in table 3 for the segment *S,A*. If LA succeeds in finding an unused wavelength, the algorithm updates the list in table 3 as described in [1]. Moving to segment *A*,*B*,*D*, the LA algorithm should be now applied on table 4 trying to find an unused wavelength. If it succeeds again then table 4 is updated for this success and the two wavelengths found are reserved for the call on their corresponding segments. Finally, the connection starts. Note here that if at any step LA fails in finding an unused wavelength, the call is blocked directly and the corresponding list (table 3 or table 4) should be updated for this failure.

Returning back to complexity measures, the total number of combinations now is only 15 (10+5). Similarly, if both nodes *A* and *B* were converter nodes, we would have 30 (10+5+15) combinations. The number of combinations will simply grow linearly in this method. If more than one paths existed from *S to D* (and we intend to use LA for routing), then we should have created separate LA lists for each one of the other paths. Then create a parent LA list that is used to choose the route for the incoming call using also the usual LA algorithm.

It should be pointed out here that we do not claim that a non-exhaustive method can replace the exhaustive one with complete efficiency. What we are trying to do here is to study the effect of neglecting the exhaustive combinations.

Call Setup Time in Learning Automata and Failure Negligence

The real-time (online) nature of the dynamic RWA problem in WDM networks requires immediate responsiveness from the network when a call is requested. Stochastic learning automata algorithms iterate indefinitely (choosing an item from a LA list probabilistically) until a successful item is found, i.e an available RWA combination. This means that an unsuccessful LA item might be selected multiple times before the algorithm finds a successful item, which in turn can result in long call setup times and unnecessary added computation. Although LA probabilities ($P^{w}_{r_{(s,d)}}$) are being decreased for LA items that are found unavailable which can lessen the chance that those items will be selected again, simulation results show that relying only on this basic fact can still lead to lengthened call setup time[†].

An efficient LA algorithm should be designed to totally avoid selecting unsuccessful list items more than once while serving an incoming call. Simply marking the unsuccessful items will not help allot since it will not stop the LA stochastic selector from choosing the same unsuccessful item again and again. Figure 2 illustrates the difficulty.

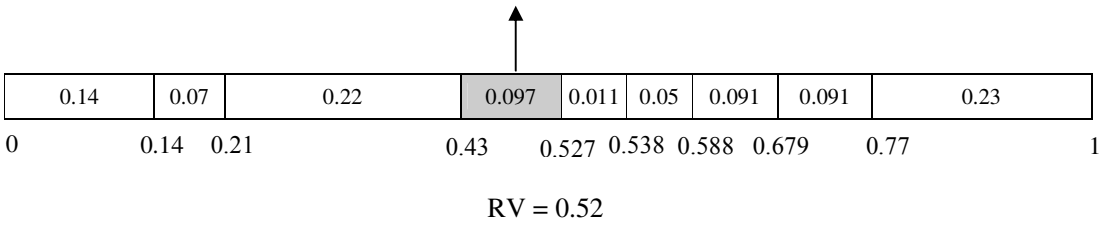


Figure 2, Example of a random variable that chooses a value between 0 and 1. The LA selection is the interval (rectangle) where the random variable hits, which is the gray rectangle in this case. Numbers inside the rectangle indicate item probabilities. (RV: Random Variable).

A typical method for implementing LA works as follows. All the items of the LA list are laid down in the interval [0,1] subdividing it into subintervals as shown in the example in Figure 2. Where the probability of each item determines the percentage of width that this item's subinterval occupies of the total width of the interval [0,1]. The order of the list items is not important. A programmed random variable is implemented (via a random number generator). The random variable should give values between 0 to 1 uniformly. At each LA iteration, a new random value is given to the random variable. The list item that will be selected for the iteration is the item that the random variable pointes inside it's region (The gray region in Figure 2). It should be clear by now that the random variable might choose the same item more than once.

[†] Other stochastic RWA algorithms (such random RWA) also suffer from the same problem. For random RWA, we need a similar but simpler approach as the one described here to avoid this problem.

We propose the following modification. Assume the following definitions:

L: The maximum limit for the random variable were $0 \le L \le 1$
RV: A real valued random variable were $0 \le RV \le L$
S: Size of the LA list. i.e number of items in the LA list.
f : The index for the first excluded subinterval in the full [0,1] interval.
a, ε *:* The convergence parameters mentioned in [1]. Where $0 < a < 1$, $0 < \varepsilon < 1$
FC: Failure constant. Which is equal to $\frac{\varepsilon}{S-1}$
Note: We're assuming that the first subinterval has an index of zero.

Initially (When a new call arrives), we set L=1 and set $f = S$. Once an LA item i with probability P_i is selected and found unsuccessful. The usual updating of the LA list probabilities is done. Then, the selected item's subinterval will be excluded from further LA selection by swapping this subinterval with the last unexcluded subinterval in the full [0,1] interval. Then L will be decreased by a certain amount in such a way that RV (were $RV \le L$) is forced to give random values in the interval [0,L] were this interval is guaranteed to exclude the failed selected item. In other words, the following LA iterations will produce RV values that will never pick our excluded item again. Every time an item is excluded, f is decreased by 1. Once $f = 0$, we know that all items have been excluded and we can stop the LA search and block the pending call. Other wise if a successful item is found before f becomes 0 we accept the call allocating the route and wavelengths associated with the item. The following c-like pseudo code provides the algorithms.

Structure La_item

```
struct la_item
{
	void *pitem;
	double probability;
};
```

Algorithm ChooseLAItem
Input: The LA list. Assumed here to be a global array of **la_item** structures called item[]
Output: The index of the item that is chosen by the LA algorithm

```
integer ChooseLAItem()
{
	double ran,d1;
	integer choice;

	ran = L*GenRand();
	choice = 0; d1 = item[0].probability;
	while (ran>=d1)
	{	choice++;
		d1 = d1 + item[choice].probability;
	}
	return choice;
}
```

Algorithm NeglectFailedChoice
Input: The **choice** index that was returned by ChooseLAItem( )
Output: Cast the failed choice at the end of the LA list and exclude it from further selection by lowering L properly.

```
void NeglectFailedChoice(integer choice)
{
	la_item latemp;

	L = (f-1)*FC+(1-ε)*L - item[choice].probability;

	// Swap newly excluded with last non-excluded
	f--;
	latemp = item[choice];
	item[choice] = item[f];
	item[f] = latemp;
}
```

The function GenRand( ) generates random numbers in the interval [0,1]. The ChooseLAItem( ) should be called on every LA iteration to choose the item. Then the item should be checked wither it's a success or failure, and accordingly, probabilities of all LA items should be updated as in [1]. After this, In case the selected item was a failure, NeglectFailedChoice( ) should be called.

A question remains, "*How to decrease L ?"*. At first, one might be tempted to assume that L should be decreased an amount equal to the probability of the excluded item. Which is true. But before doing so, we should realize that we have updated the probabilities of the whole LA list[‡]. Which means that our previous L is not excluding the previously excluded items correctly. The very definition of L imposes that it is a value that covers precisely the unexcluded interval. Hence L is the total probability of all unexcluded items. In other words:

$$L = \sum_{i=0}^{f-1} P[i] \tag{1}$$

Where $P[i]$ is the probability of item i. However, we will try here to calculate L in an efficient manner. Referring to [1], probabilities should be updated as follows:

(2)

$$P'[j] = (1-\varepsilon)P[j] \tag{2}$$

$$P'[i] = \frac{\varepsilon}{S-1} + (1-\varepsilon)P[i] \quad \forall i \neq j \tag{3}$$

Where j is the index of the item that has failed. $P'[i]$ and $P'[j]$ are the new updated probabilities of items i and j respectively. So we can rewrite (1) as:

$$L' = \sum_{i=0}^{f-1} P'[i] \tag{4}$$

Where L' is the updated value of L. Using (2) and (3)

[‡] Decreasing the failed item and increasing all other items

$$L' = \left[\sum_{\substack{i=0 \\ i \neq j}}^{f-1} \frac{\varepsilon}{S-1} + (1-\varepsilon)P[i] \right] + (1-\varepsilon)P[j] \tag{5}$$

$$\Rightarrow \quad L' = \frac{(f-1)\varepsilon}{S-1} + (1-\varepsilon) \sum_{\substack{i=0 \\ i \neq j}}^{f-1} P[i] + (1-\varepsilon)P[j] \tag{6}$$

$$\Rightarrow \quad L' = \frac{(f-1)\varepsilon}{S-1} + (1-\varepsilon) \sum_{i=0}^{f-1} P[i] \tag{7}$$

Using (1), the term $\sum_{i=0}^{f-1} P[i]$ is equal to L (The previous value of L). Hence:

$$L' = \frac{(f-1)\varepsilon}{S-1} + (1-\varepsilon) \cdot L \tag{8}$$

Equation (8) represents a more optimized method of calculation than (4). L' is the corrected value after updating the item probabilities. Now we can exclude the newly excluded item by subtracting it's updated probability $P'[j]$ as follows:

$$L' = \frac{(f-1)\varepsilon}{S-1} + (1-\varepsilon) \cdot L - P'[j] \tag{9}$$

Which is the method we are using in NeglectFailedChoice( ) algorithm. An alternative method is to update L only when we call ChooseLAItem( ) as shown bellow:

Algorithm ChooseLAItem2
Input: The LA list. Which is assumed to be a global array of **la_item** structures called item[]
Output: The index of the item that is chosen by the LA algorithm

```
integer ChooseLAItem()
{
       double ran,d1;
       integer choice;

       ran = L*GenRand();
       choice = 0; d1 = item[0].probability;
       while (ran>=d1)
       {      choice++;
              if (choice>=f) { L=ran; ran = L*GenRand(); choice=0; d1=0; }
              d1 = d1 + item[choice].probability;
       }
       return choice;
}
```

The bold face line added above will do all updating for L. No need to update L in NeglectFailedChoice( ). This method exercises a trial and error technique. L is left unchanged until it produces random values in the excluded interval, only then L will be changed. This method can consume less computation time in some cases, like in high traffic demands were the number of blocked calls (item failures) is large which results in NeglectFailedChoice( ) being called so many times updating L repeatedly.

Least Used & Most Used Wavelength Assignment Revisited

Before we proceed with the results, we would like to clarify an inconsistency that we have faced while observing the literature. In almost all papers that we have seen [2][3][4][5][6][7], the definitions of the Least Used (LU) and Most Used (MU) wavelength assignment has been similar to the following:

Least Used: Selects the wavelength that is currently the least used in the network.
Most Used: Selects the wavelength that is currently the most used in the network.

Both definitions (As has been explained by many papers) require global knowledge about wavelength usage allover the network. In other words the network has to know in how many links each wavelength is used anywhere in the network and provide such information to the nodes in a distributed manner. However, there exists another uncommon definition for LU and MU:

Least Used: Selects the wavelength that is currently used on the least number of links in the path.
Most Used: Selects the wavelength that is currently used on the largest number of links in the path.

Clearly, the two latter definitions differ from the former ones by the scope of the wavelength counting. The former common definitions are **global** in the sense that they count the wavelength usage through the whole network while the latter definitions are **local** since they count only through the selected path. These two latter definitions appeared in a version of paper [8]. However the newest version of [8] surprisingly reverts back to the two common definitions we stated above.

For convenience, we have produced simulation results for both global and local versions of LU and MU. For differentiation, we will refer to the localized versions as Localized Least Used (LLU) and Localized Most Used (LMU). While the globalized versions will still have the names LU and MU. A comparison between the two versions have never been presented in the literature before. It turns out that LLU and LMU produce much better results than LU and MU as shown in the following sections.

Simulation Results

We conduct our simulation using two different topologies. The 14 nodes NSFNET topology and a 20 nodes randomly generated network topology. For the NSFNET we use the same heterogeneous link capacities and call arrival rates mentioned in [1] (case 1). However, in some scenarios, we ignored the heterogeneous link capacities and enforced homogenous capacities with the same wavelength (W=50) all over the network. For the other 20 nodes random network, links are distributed over 30% of the 20x20 possible source to destination pairs. Where each link has a randomly generated capacity in the range 20 to 100 wavelengths uniformly distributed over the links. Call arrival rates are also uniformly distributed random integers in the range 0 to 10 calls/unit time. Each of the simulation runs were carried out for 1000 time units. Each simulation run is repeated for 10 batches and the average of the 10 results is our final result. As in [1] we use $\alpha = 0.1$ and $\varepsilon = 0.001$ for LA algorithm.

Exhaustive LA vs. Non-Exhaustive LA

Using the NSFNET topology with two converter nodes (namely 4 and 7). Simulation memory usage for both exhaustive LA (XLA) and the non-exhaustive LA (NXLA) described in section #.# are shown in Figure 3. Figure 4 also shows the total computation time on a 2GHz PC. It is clear from both figures that XLA is far more resource consuming and CPU intensive than NXLA. Remember that this is when only 2 converter nodes are present. If more converter nodes are deployed, XLA takes much more memory and CPU time and becomes totally unfeasible for more than 4 converter nodes. While NXLA described in #.# scales linearly.

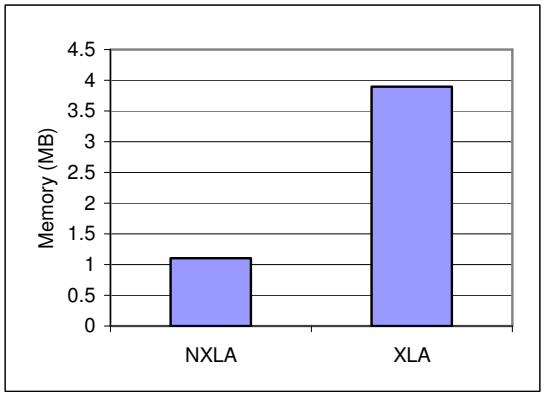


Figure 3: Memory usage for NSFNET with 2 converter nodes. (XLA: Exhaustive LA).

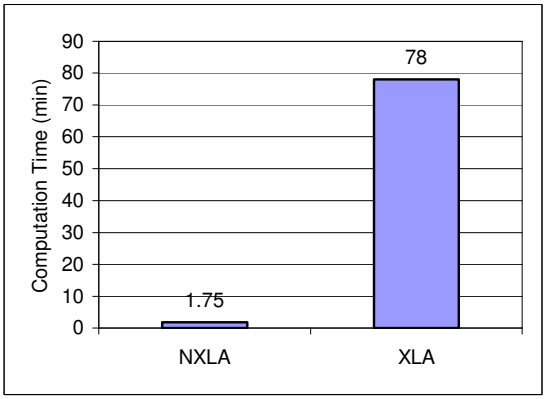


Figure 4: Computation Time in minutes for NSFNET with 2 converter nodes. (XLA: Exhaustive LA).

These results were all expected. But the interesting results were the blocking probabilities. As shown in Figure 5. The new non-exhaustive LA gave lower blocking probabilities than XLA in all cases of different traffic loads. This is probably due to long convergence time of XLA because of the existence of a large number of LA items in it's LA lists. In other words, it takes a long time for XLA to figure out the optimum probabilities of it's LA lists simply because there are so many items in the lists. However, this observation needs further investigation and is out of the scope of this paper.

For the rest of this paper, XLA is not used any more. All LA algorithms shown are non-exhaustive.

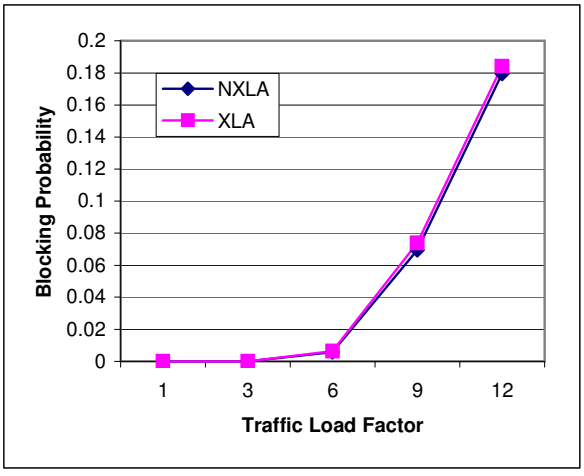


Figure 5: Blocking probability for NSFNET with 2 converter nodes.

NXLA vs. FLA

As it has been mentioned above at the beginning of section #.#. Fast Learning Automata (or FLA for short) is our proposed algorithm which combines the two methods mentioned in #.# and #.#. Roughly speaking:

$$\text{FLA} \equiv \text{NXLA} + \text{Failure Negligence algorithm}$$

Figure 6 and 7 compare NXLA and FLA in both networks in terms of the **Average Number of Attempts** a call takes before it finds an available RWA combination or fails the search and gets blocked. The average number of attempts represents one of the measures for the calls setup time. It can be seen from both figures that FLA outperforms NXLA in this measure.

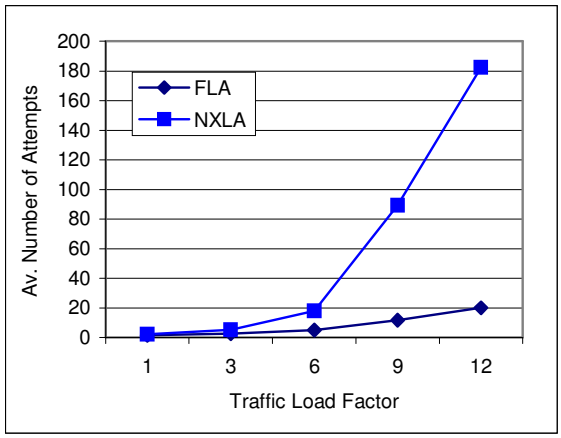


Figure 6: Average Number of Attempts for NSFNET with 2 converter nodes.

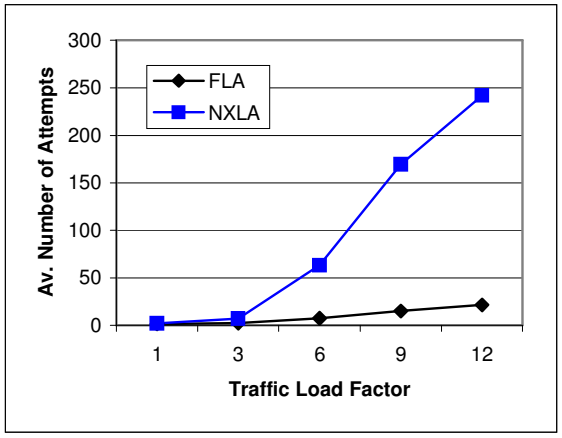


Figure 7: Average Number of Attempts for the 20 nodes network with 2 converter nodes.

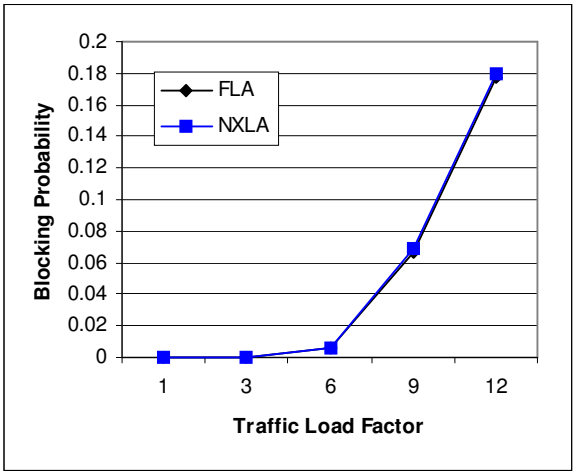


Figure 8: Blocking Probability for NSFNET with 2 converter nodes.

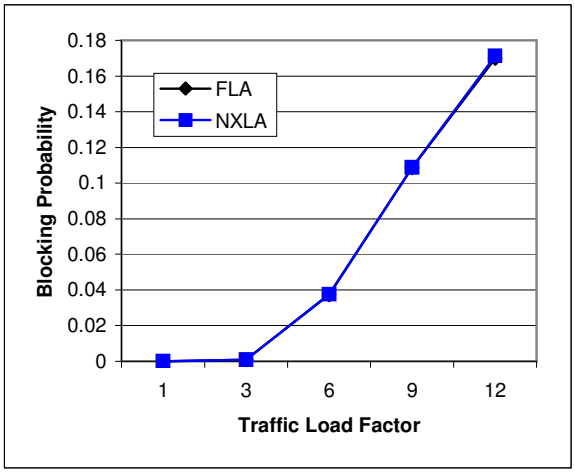


Figure 9: Blocking Probability for the 20 nodes network with 2 converter nodes.

Further more, Figures 8 and 9 show that FLA and NXLA are almost identical in terms of blocking probability. FLA has slightly improved (lower) values in high load cases. Similar

results have been obtained for networks with 4 and 8 converter nodes. Fully converting networks gave agreeing results as well.

Deterministic Learning Automata

Another problem in LA (even in FLA) that is not very apparent at first glance is the computation time of the probabilities updating process. Remember that when LA chooses an item from the LA list, and the item is tested wither it's a success or failure. In both cases, the probabilities of all the LA items in the list has to be updated properly. For a few tens of items this should be no problem. But as the number of wavelengths per fiber increases in the near future, The number of items must increase and the updating process will consume more CPU time. Typical technologies can multiplex 40 or 80 wavelengths per fiber [9]. However, Recent advances have reached 128,176,256 and 273 wavelengths per fiber [10] [11]. Furthermore, research at the Bell Labs Innovations has demonstrated that the number of wavelengths in a single fiber could be increased to 1000 [9].

Since LA is a probabilistic algorithm, the sum of all the probabilities of all the items has to be equal to 1. And that is exactly the reason why all the probabilities of all the items have to be updated (not only the checked item's). We propose here a new simple algorithm (DLA: Deterministic Learning Automata) that gives similar performance as LA in many cases without the need to update any probabilities.

Instead of having a probability value for each item, DLA will have a counter for each item. The counter simply counts how many times it's corresponding item has succeeded *through out the whole history of the network*. When a call arrives, DLA will choose the item with highest counter first. If it fails, the next highest is chosen, and the next until it finds a successful one, inform the network to use it and increase it's counter by 1. Now, this algorithm might sound more like the Most Used (MU) algorithm. But, in MU the counters are *decreased* once the router and/or wavelength is released from the network. DLA never decreases the counters. In other words MU chooses the most *currently* used item, while DLA chooses the most successful item *throughout the entire network history*.

Programmatically speaking, a counter can not keep increasing infinitely. To solve this, in DLA we occasionally subtract a constant value from all the counters to keep them from overflowing. It's obvious that this will not change the order of selecting the items. This subtraction need NOT to be done very often since counters (in current programming languages) can usually count upto very large number (2^{32}-1 for a 32 bit counter). We also define DLA to be a non-exhaustive algorithm in a network with converter nodes. Just as FLA is, each non-converting segment will have a separate DLA list with it's own items with their counters.

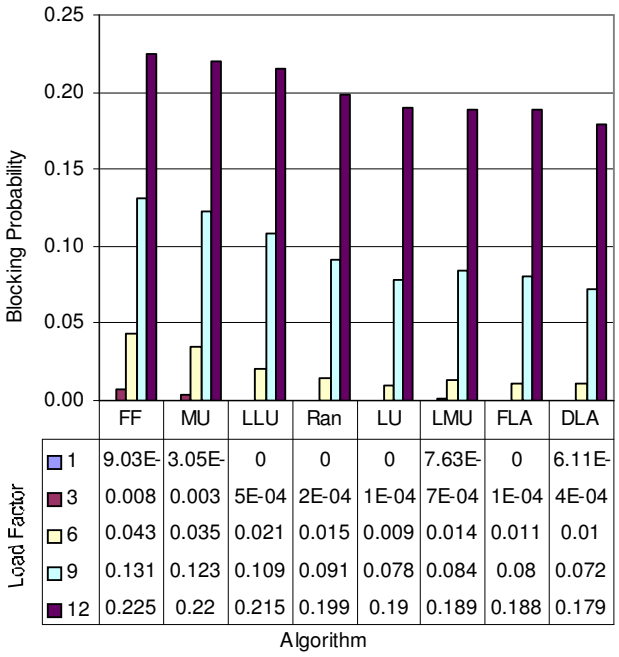
Results and comparisons

We have done simulations for both networks using First Fit (FF), Random (Ran), Most Used (MU), Least Used (LU), Localized Most Used (LMU), Localized Least Used (LLU) wavelength assignment algorithms, To compare with LA and DLA. For routing, we used Fixed Alternate Routing (FAR) which considers all shortest paths from each source to each destination. However, The literature does not specify the order of FAR path selection if there is more than one shortest path from a source to a destination. We use First Fit route selection

in the following subsection. We follow that with a subsection to show the results for the Least Loaded Routing (LLR).

In addition to the blocking probability and average number of attempts, we also show results of the *variance* in blocking probabilities. Remember that each source to destination pair have their own local blocking probability which might differ significantly from the overall average blocking probability. The variance is an indicator for the fairness of an RWA method. Furthermore, results of the *maximum* blocking probability are shown which can help in realizing the worst cases.

Figures 10,12 show average blocking probability results for the NSFNET without converter nodes. For the highest traffic load factor 12, DLA achieves the lowest result of 0.1792. Next comes FLA with 0.1885, then LMU with 0.1893 followed by LU with 0.1901. Lowering the load factor to 9 we see that DLA is still holding first position while LU comes in second. FLA comes next followed by LMU. However, as the load is decreased more, DLA falls back giving marginally higher blocking probabilities. On the other hand FLA still gains lower values staying among the three top algorithms.

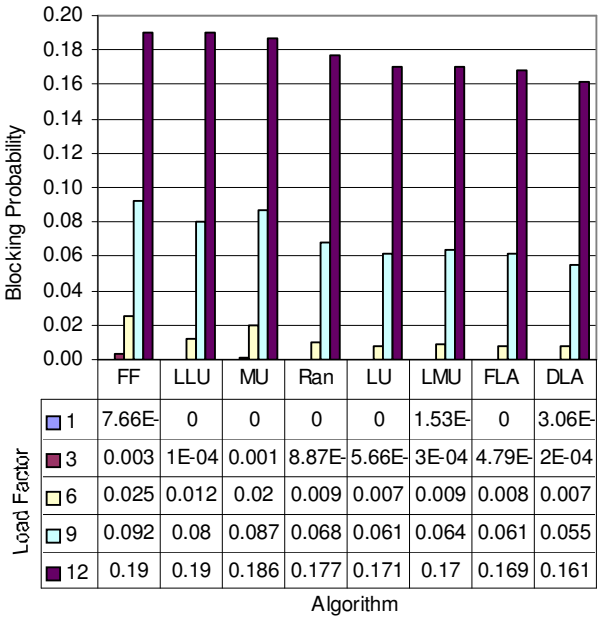


Load Factor	FF	MU	LLU	Ran	LU	LMU	FLA	DLA
1	9.03E-	3.05E-	0	0	0	7.63E-	0	6.11E-
3	0.008	0.003	5E-04	2E-04	1E-04	7E-04	1E-04	4E-04
6	0.043	0.035	0.021	0.015	0.009	0.014	0.011	0.01
9	0.131	0.123	0.109	0.091	0.078	0.084	0.08	0.072
12	0.225	0.22	0.215	0.199	0.19	0.189	0.188	0.179

Figure 10: Blocking Probability for the NSFNET without converter nodes.

In the next step, four randomly chosen nodes in the NSFNET are made to be converter nodes. Figure 11 illustrates the results. DLA ranks first and FLA second in both high load factors 12, 9. For loads 1,3,6 again DLA draws back performing poorly while FLA swaps in first and second place with LU.

It is worth clarifying one important point here regarding the performance of LU and MU. LU has inherently performed poorly in most references while MU has been noticed as one of the best (if not the best) WA algorithm. Results in figures 10,11 show the complete opposite. This is due to the fact that in the above results we are considering *heterogeneous* link capacities (**W** is not fixed) while most references consider homogenous link capacities [12, 2, 14, 15, 16, 17, 21, 22, 23, 24, 25, 26, 27]. Few



Load Factor	FF	LLU	MU	Ran	LU	LMU	FLA	DLA
1	7.66E-	0	0	0	0	1.53E-	0	3.06E-
3	0.003	1E-04	0.001	8.87E-	5.66E-	3E-04	4.79E-	2E-04
6	0.025	0.012	0.02	0.009	0.007	0.009	0.008	0.007
9	0.092	0.08	0.087	0.068	0.061	0.064	0.061	0.055
12	0.19	0.19	0.186	0.177	0.171	0.17	0.169	0.161

Figure 11: Blocking Probability for the NSFNET with 4 converter nodes.

authors considered heterogeneous link capacities [1, 18, 19, 20]. Later results in section #.# (Figure 24) show networks with homogenous link capacities, were indeed MU performs efficiently and LU's performance degrades as stated in most papers. This is an indicator of the great sensitivity of WA algorithms to the link capacities in the network.

We suspect that MU works better for homogenous link capacities because it *packs* the usage of the wavelengths more efficiently, trying the same wavelengths over and over so leaving all free wavelengths untouched. But when link capacities heavily differ in the network, this packing will not work since many wavelengths that are being used in high capacity links are simply not available in lower capacity links (i.e higher indexed wavelengths). The most used wavelengths will fail to be available so often. On the other hand LU will tend to use those wavelengths since they are actually unusable in lower capacity links. This creates a tendency to choose those higher indexed wavelengths for establishing short light paths which will free more lower indexed wavelengths for longer light paths.

Despite MU's unsuitability in this case, it is rather surprising to see the interesting performance of LMU in Figures 10 and 11. Which can raise questions about the behavior of the two versions, which is a subject out of the scope of this work.

As for call setup time, Figure 13 shows the average number of attempts a call takes before it finds an available RWA combination or fails the search and gets blocked. DLA and FLA gave the best results for the high load factors 6, 9, 12 were LU ranked either in 4^{th} or 5^{th} place. For load factor 3, DLA drops to 5^{th} place while FLA comes in 3^{rd}. FLA drops to 4^{th} place for load factor 1. Figures 13, 14 show further worst case and fairness indicators for convenience were both FLA and DLA gave either best or comparable results.

Figures 16, 17, 18, 19 show the results for the NSFNET with 2 of the nodes working as converter nodes. Again DLA performs best at high rates and FLA performs well but behind LU which ranks 2^{nd}. However, LU still suffers from the high Avg. number of attempts while FLA competes neck to neck with Random WA on second place.

We can draw an initial conclusion here that DLA is best at high loads but it's performance drops significantly at lower loads while FLA maintains a good to very good performance on all loads.

FAR with different WA On NSFNET

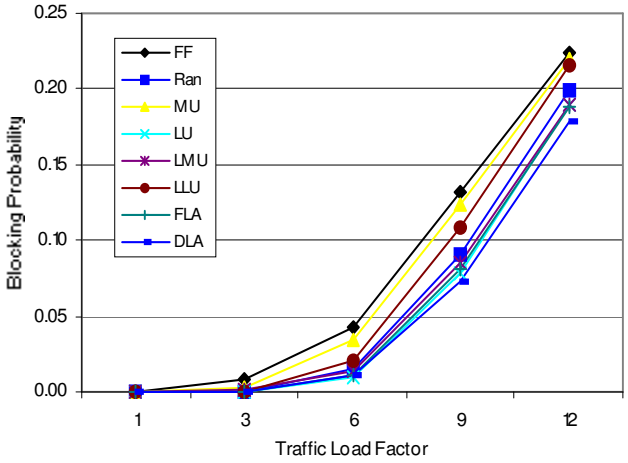


Figure 12: Blocking Probability for the NSFNET without converter nodes.

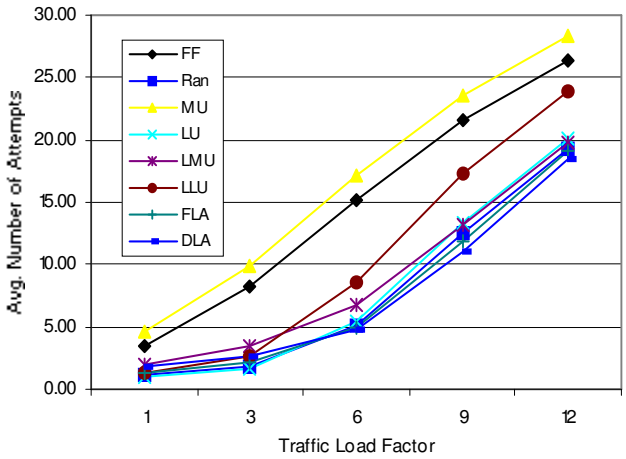


Figure 13: Avg. number of attempts for the NSFNET without converter nodes.

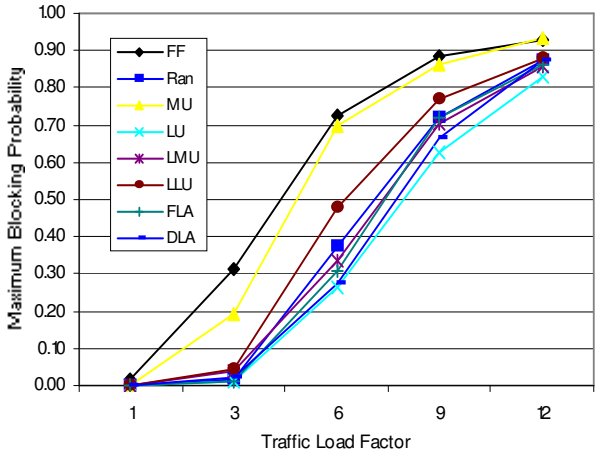


Figure 14: Maximum Blocking Probability for the NSFNET without converter nodes.

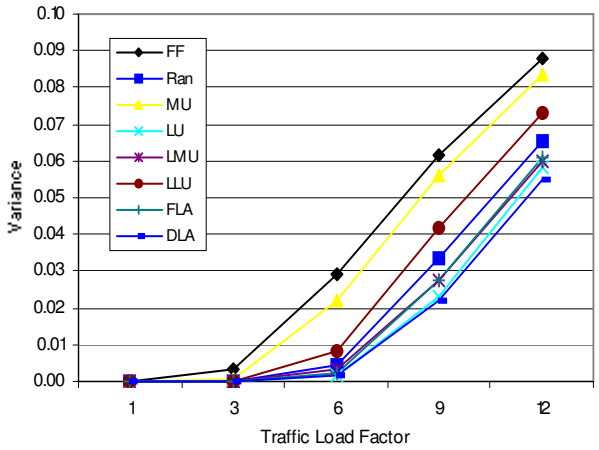


Figure 15: Variance of node blocking probabilities for the NSFNET without converter nodes.

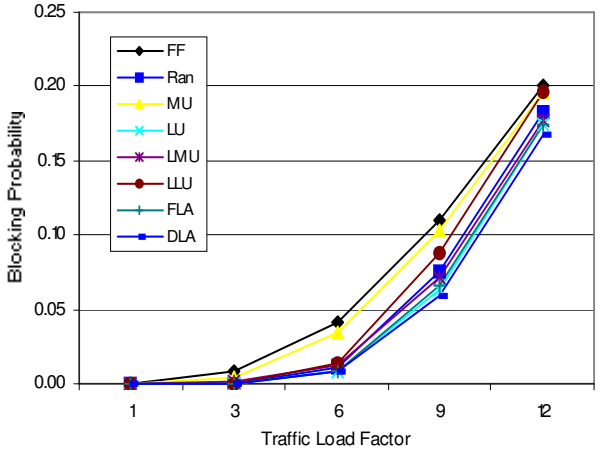


Figure 16: Blocking Probability for the NSFNET with 2 converter nodes.

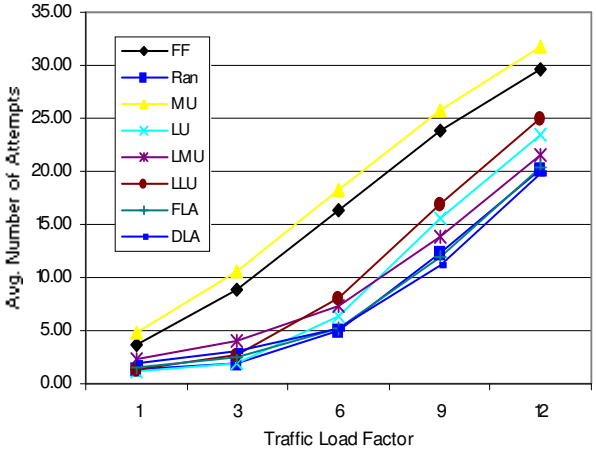


Figure 17: Avg. Number of Attempts for the NSFNET with 2 converter nodes.

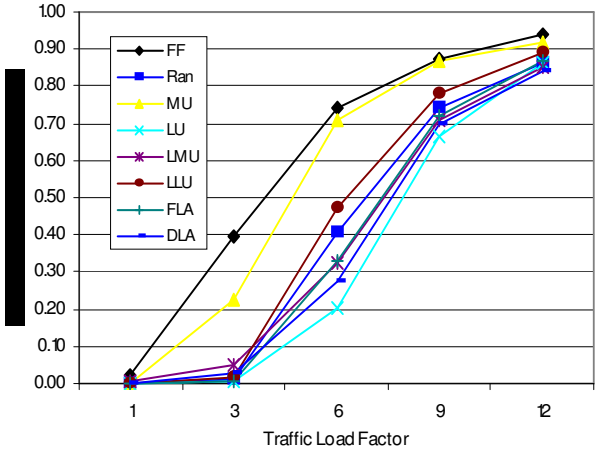


Figure 18: Maximum Blocking Probability for the NSFNET with 2 converter nodes.

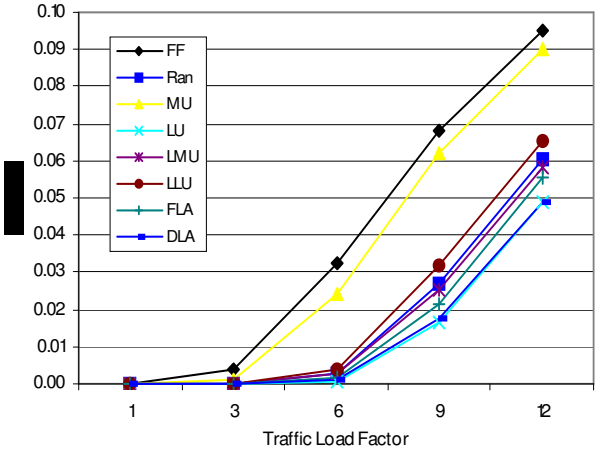


Figure 19: Variance of node blocking probabilities for the NSFNET with 2 converter nodes.

FAR with different WA on the random 20 nodes network

In all plots in this section, FLA-FLA represents combined routing and wavelength assignment using LA. We have not shown FLA routing in the above simulations in order to emphasize on the performance of FLA *wavelength assignment alone* and to give other algorithms a fair chance to compete by having the same routing method applied to all algorithms. However, using FLA routing combined with FLA WA will improve the performance significantly as shown in figures 18, 19, 20, 21. FLA-FLA gave the best results in almost all cases. FLA, DLA and LU achieved comparable results with some superiority for LU. However, LU took more setup time in high loads.

It can be noticed here that the differences between RWA algorithms are minor.

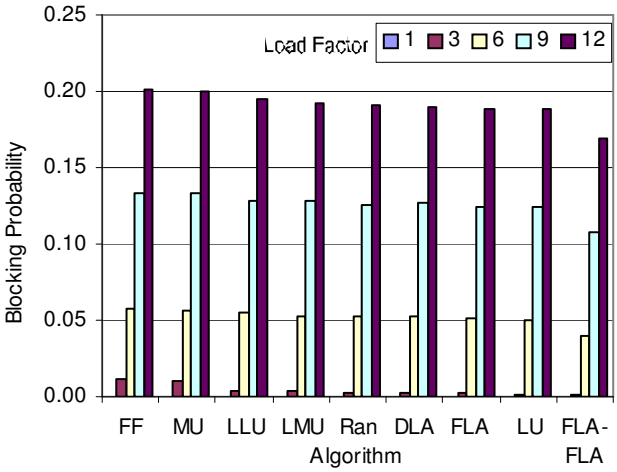


Figure 18: Blocking Probability for the 20 nodes network without converter nodes.

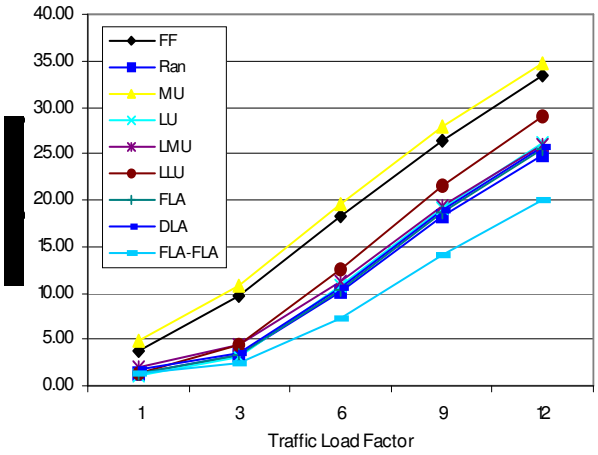


Figure 19: Avg. Number of Attempts for the 20 nodes network without converter nodes.

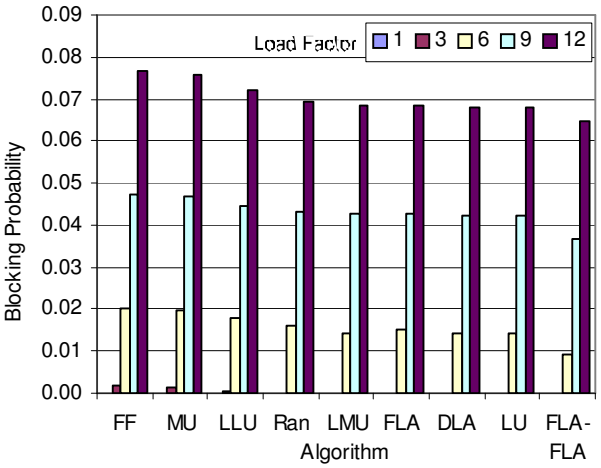


Figure 20: Variance of node blocking probabilities for the 20 nodes network without converter nodes.

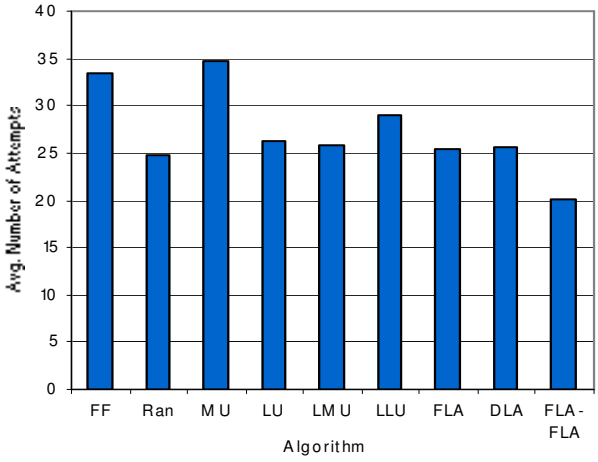


Figure 21: (At load factor 12). Avg. Number of Attempts for the 20 nodes network without converter nodes.

FAR with different WA On NSFNET with homogeneous capacities (W=50)

When links have the same capacities, behavior of some of the RWA algorithms change rapidly. In Figure 22, (aside from FLA-FLA's interesting performance) we can see that both FLA and DLA performed only moderately in terms of the blocking probability. LU and LLU gave the worst results while MU was the best after FLA-FLA. This confirms our notice earlier in section #.# regarding homogenous and heterogeneous capacities and agrees totally with the literature regarding the efficiency of MU. Performance of First Fit WA also improved. Figure 23 shows that FLA-FLA took the least average setup times in high load factors 6,9 and 12 while LU gave the least times for loads 1 and 3. Despite MU's interesting performance in the blocking probability, MU took the longest average setup times in all loads.

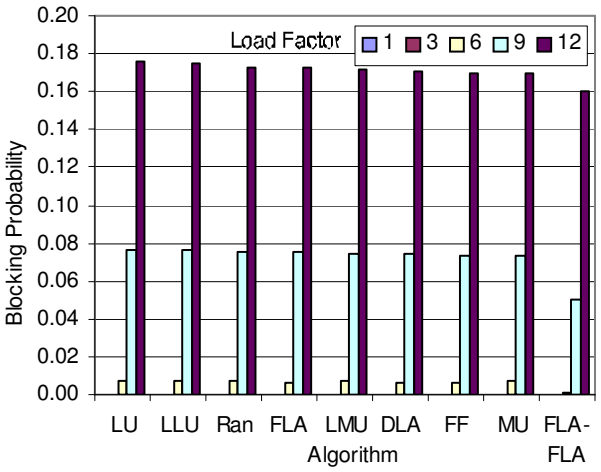


Figure 22: (W=50). Blocking Probability for the NSFNET with 4 converter nodes.

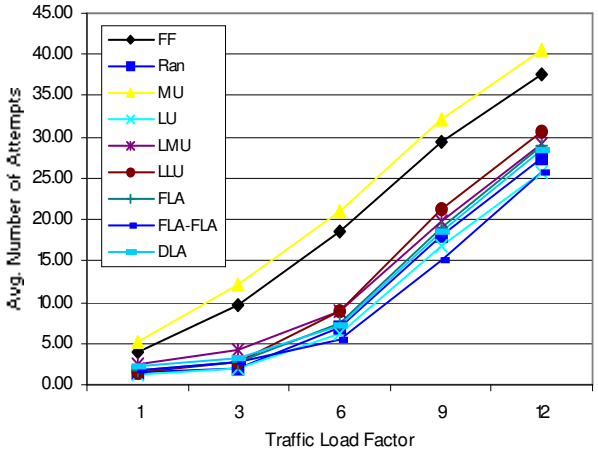


Figure 23: (W=50). Avg. Number of Attempts for the NSFNET with 4 converter nodes.

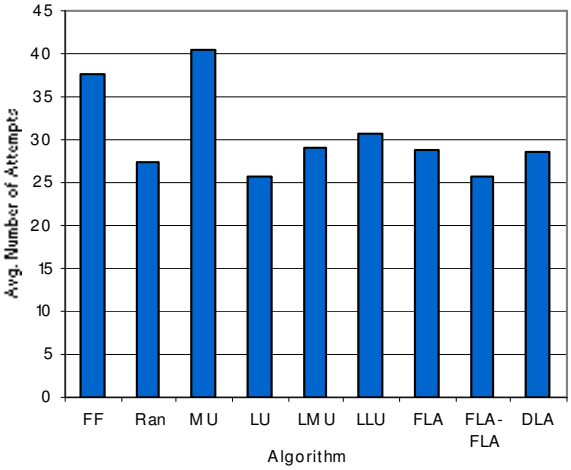


Figure 24: (W=50, At load factor 12). Avg. Number of Attempts at high load for the NSFNET with 4 converter nodes.

LLR with different WA On NSFNET

It is well known that the Least Loaded Routing (LLR) is an innovative routing technique that repeatedly produced competitive results. LLR has been defined in two differing flavors in [28] and [29]. Both versions has been mentioned or adopted repeatedly [][][]. However, the first version (although showed promising results) assumes a fully connected network[§] which isn't always feasible in long-haul and metro networks. For instance, NSFNET is not fully connected. We adopt the second version described in [29], the technique chooses the least congested path among *k* shortest paths. The least congested path is defined as the path in which it's most congested fiber link (i.e. the link that has the least number of unused wavelengths) has the largest number of unused wavelengths in all such links in all *k* shortest paths. The same paper also assumes that a least loaded wavelength assignment is to be used. However, as the paper shows that for single fiber links (which is our case) the least loaded WA reduces to a *tie* situation where it is assumed to behave as MU WA. In our simulation, we use all other WA algorithms for convenience, not just MU. In Figure 25, we see that FLA-FLA is no longer the best method in terms of blocking probability. However FLA and DLA perform the best in most loads. This admittedly gives an indication that LLR produces less blocking than FLA *routing* in this case. Figure 26 shows setup times were DLA excels at high loads and FLA performs very good to moderate.

We should point out here that LLR demands online network state information [30]. Despite the interesting setup times achieved as shown in Figure 26. LLR requires a pre-setup phase were all link state information on all *k* shortest paths is collected (either using control messages or querying a central database). Then LLR computes the minimum link capacity over each path then sorts the paths in a descending order according to their congestion. On the other hand FLA routing does NOT require network state information. Once a call has arrived the LA list (which can reside on the source node memory) already have their pre-computed probabilities.

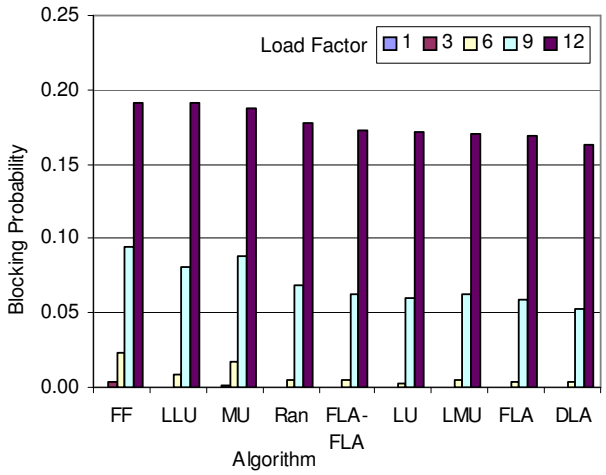


Figure 25: (LLR) Blocking Probability for the NSFNET with 4 converter nodes.

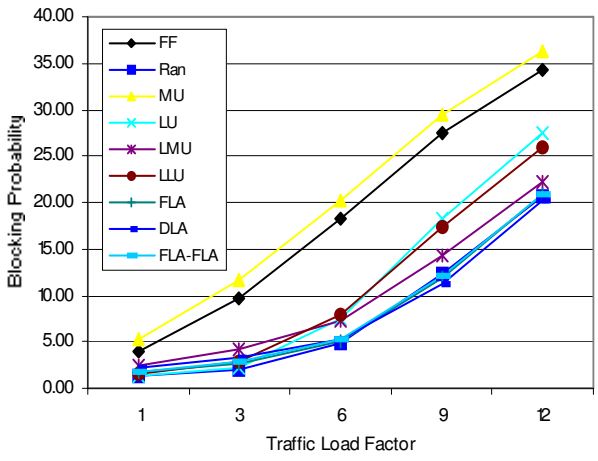


Figure 26: (LLR) Avg. Number of Attempts for the NSFNET with 4 converter nodes.

[§] Each source to destination pair of nodes must have a direct link between them.

LLR with different WA On NSFNET with homogeneous capacities (W=50)

As in FAR case in section #.#, Figure 27 illustrates how MU and FF are two of the best methods in reducing the blocking rates for homogeneous networks. However, MU and FF are not designed to reduce setup time as shown in Figure 28 were they require a very high number of call attempts on the average. In contrast, FLA and DLA which fall just behind MU and FF with marginal differences in terms of blocking probability, do not exhibit such high setup time requirement.

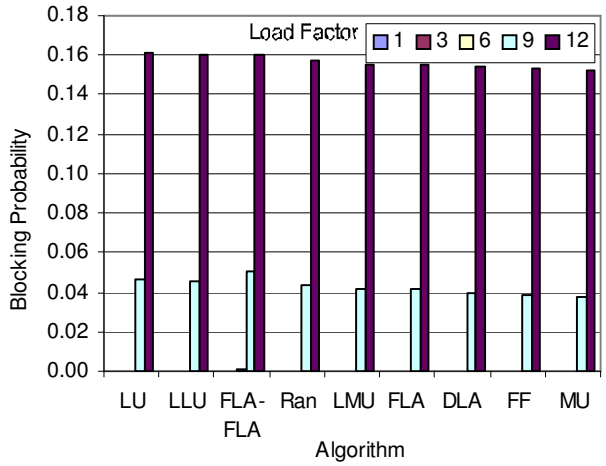


Figure 27: (LLR, W=50) Blocking Probability for the NSFNET with 4 converter nodes.

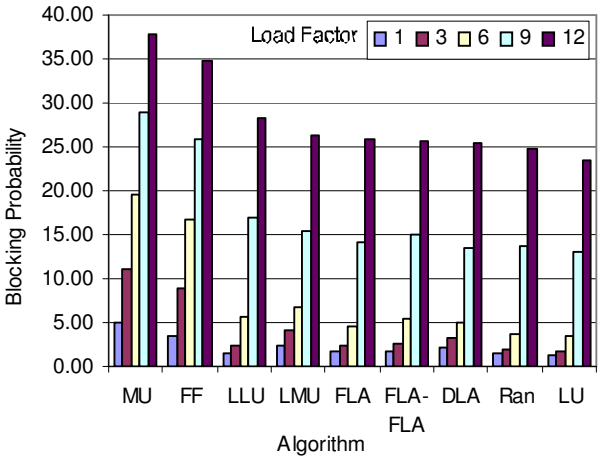


Figure 28: (LLR, W=50) Avg. Number of Attempts for the NSFNET with 4 converter nodes.

Effect of number of converters on RWA performance

In this section we vary the number of converter nodes in the network to see how adaptable FLA and DLA are. LLR routing is used on NSFNET with the high load factor of 12. First the simulation is done with no converter nodes. Then we pick a node at random to be a converter node and we run the simulation again. Then we pick another node randomly again to be a converter node (keeping the previous one as a converter) and run the simulation again. We redo this process repeatedly until we have a network with all the nodes as converters. Figures 29 shows interesting results were DLA excels (since this is high load case) and FLA competes with LU. As it is expected, for the fully convertible network case (14 converter nodes), all WA algorithms gave exactly the same blocking probability. This is not a surprise since the routing is fixed to LLR for all algorithms. Figures 30 shows results that speak for them selves regarding setup time.

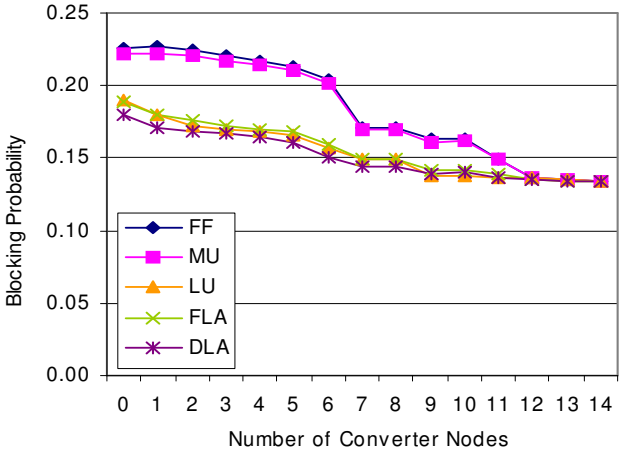


Figure 29: (LLR, Load =12, Heterogeneous capacities) Blocking Probability for the NSFNET.

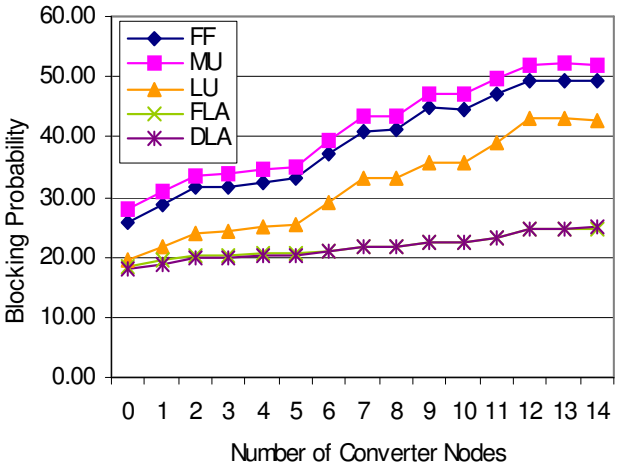


Figure 30: (LLR, Load =12, Heterogeneous capacities) Avg. Number of Attempts for the NSFNET.

[1] Anwar Al-Yatama, “Dynamic Routing and Wavelength Assignment Using Learning Automata Technique”. Globcome 2004.

[2] Jun Zhou, Xin Yuan, “A Study of Dynamic Routing and Wavelength Assignment with Imprecise Network State Information”.

[3] HUI ZANG, JASON P. JUE, BISWANATH MUKHERJEE , “A Review of Routing and Wavelength Assignment Approaches for Wavelength Routed Optical WDM Networks”. Optical Networks Magazine, January 2000.

[4] Dinh Thi Thuy Nga, Minho Kang, “An Online Wavelength Assignment Algorithm for Optical Network”

[5] Xuehong Sun, Yunheo Lit, Ioannis Lambadarist and Yiqiang Q. Zhao. “Performance Analysis of First-Fit Wavelength Assignment Algorithm in Optical Network”. 7th International Conference on Telecommunications - ConTEL 2003.

[6] George N. Rouskas, “Routing and Wavelength Assignment in Optical WDM Networks”

[7] Wenhao Lin, Richard S. Wolff, Brendan Mumey, “A Markov-based Reservation Algorithm for Wavelength Assignment in All-optical Networks”

[8] Yuhong Zhu, George N. Rouskas, Harry G. Perros, “A Comparison of Allocation Policies in Wavelength Routing Networks”

[9] What is WDM technology? http://www.spie.org/web/oer/november/nov00/wdm.html.

[10] http://www.tycotelecom.com/AboutUs/HistoryTelecom.asp

[11] http://www.burtongroup.com/promo/columns/column.asp?articleid=67&employeeid=56

[12] Ahmed Mokhtar, and Murat Azizoğlu, "Adaptive Wavelength Routing in All-Optical Networks"

[13] John Strand, Robert Doverspike, Guangzhi Li, "Importance Of Wavelength Conversion In An Optical Network"

[14] Abdelhamid E. Eshoul and H. T. Mouftah, Fellow, IEEE, "IFF: A NOVEL WAVELENGTH ASSIGNMENT SCHEME FOR WDM OPTICAL NETWORKS"

[15] Cheng Xiao-fei, Lin Mian-feng, Gu wan-yi, "Wavelength Assignment for Dynamic Traffic in WDM Optical Networks". Journal of Optical Communications 2004.

[16] Djuana Pigford Lea, "SOFT COMPUTING APPROACHES TO ROUTING AND WAVELENGTH ASSIGNMENT IN WAVELENGTH-ROUTED OPTICAL NETWORKS".

[17] I-Shyan Hwang, I-Feng Huang, Shin-Cheng Yu, "Dynamic RWA scheme using fuzzy logic control (FLC RWA) on IP over GMPLS with DWDM networks"

[18] Ralf Hülsermann, Monika Jäger, Sven O. Krumke, Diana Poensgen, Jorg Rambau, Andreas Tuchscherer, "DYNAMIC ROUTING ALGORITHMS IN TRANSPARENT OPTICAL NETWORKS"

[19] J. Yates, M. Rumsewicz and J. Lacey, "Wavelength Conversion in Networks with Differing Link Capacities," IEEE GLOBECOM"

[20] Y. Zhu, GN Rouskas, and HG Perros, "A Comparison of Allocation Policies inWavelength Routing Networks".

[21] Esa Hyytiä, Jorma Virtamo, "Dynamic Routing and Wavelength Assignment Using First Policy Iteration"

[22] Bo Wen, Krishna M. Sivalingam , "Routing, Wavelength and Time-Slot Assignment in Time Division Multiplexed Wavelength-Routed Optical WDM Networks"

[23] Xi Wang, Hiroyuki Morikawa, and Tomonori Aoyama, "Priority-based Wavelength Assignment Algorithm for Burst Switched WDM Optical Networks".

[24] S. Subramaniam, M. Azizoglu, and A. K. Somani, “Connectivity and Sparse wavelength Conversion in Wavelength-Routing networks”, in IEEE/ACM Journal of Networking, Vol. 4, No. 4, August 1997.

[25] S. Subramaniam, A. K. Somani, M. Azizoglu, and R. A. Barry, “The Benefits of Wavelength Conversion in WDM Networks with Non-Poisson Traffic”, in IEEE Communications Letters, 1998, Vol. 3, No. 3, March 1999.

[26] Ahmad, J. Zaidi, S.M.H. Nawaz, S. “Dynamic routing in wavelength convertible WDM networks”.

[27] Xiaowen Chu, Bo Li, Zhensheng Zhang , “A Dynamic RWA Algorithm in a Wavelength-Routed All-Optical Network with Wavelength Converters”

[28] A. Birman, “Computing approximate blocking probabilities for a class of all-optical networks”. INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. March 1999.

[29] E Karasan, E Ayanoglu, “Effects of Wavelength Routing and Selection Algorithms on Wavelength Conversion Gain in WDM Optical Networks”. IEEE/ACM Transactions on, 1998.

[30] G. Maier, A. Pattavina, L. Barbato, F. Cecini, M. Martinelli, "Routing Algorithms in WDM Networks under Mixed Static and Dynamic Lambda-Traffic". Photonic Network Communications Journal, Volume 8, Number 1 / June, 2004

[31] Link to Hegons simulator. http://www.solostuff.net/hegons/