Elimination of All-Optical Cycles in Wavelength-Routed Optical Networks

Jason Iness
Byrav Ramamurthy
Biswanath Mukherjee
Krishna Bala

Follow this and additional works at: http://digitalcommons.unl.edu/csearticles

Part of the Computer Sciences Commons

Iness, Jason; Ramamurthy, Byrav; Mukherjee, Biswanath; and Bala, Krishna, "Elimination of All-Optical Cycles in Wavelength-Routed Optical Networks" (1996). CSE Journal Articles. 72.
http://digitalcommons.unl.edu/csearticles/72

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Journal Articles by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
Elimination of All-Optical Cycles in Wavelength-Routed Optical Networks

Jason Iness, Student Member, IEEE, Byrav Ramamurthy, Biswanath Mukherjee, Member, IEEE, and Krishna Bala, Member, IEEE

Abstract—A transparent (wide-area) wavelength-routed optical network may be constructed by using wavelength cross-connect switches connected together by fiber to form an arbitrary mesh structure. The network is accessed through electronic stations that are attached to some of these cross-connects. These wavelength cross-connect switches have the property that they may configure themselves into unspecified states. Each input port of a switch is always connected to some output port of the switch whether or not such a connection is required for the purpose of information transfer. Due to the presence of these unspecified states, there exists the possibility of setting up unintended all-optical cycles in the network (viz., a loop with no terminating electronics in it). If such a cycle contains amplifiers (e.g., Erbium-Doped Fiber Amplifiers (EDFA’s)), there exists the possibility that the net loop gain is greater than the net loop loss. The amplified spontaneous emission (ASE) noise from amplifiers can build up in such a feedback loop to saturate the amplifiers and result in oscillations of the ASE noise in the loop. Such all-optical cycles as defined above (and hereafter referred to as “white” cycles) must be eliminated from an optical network in order for the network to perform any useful operation. Furthermore, for the realistic case in which the wavelength cross-connects result in signal crosstalk, there is a possibility of having closed cycles with oscillating crosstalk signals. We examine algorithms that set up new transparent optical connections upon request while avoiding the creation of such cycles in the network. These algorithms attempt to find a route for a connection and then (in a post-processing fashion) configure switches such that white cycles that might get created would automatically get eliminated. In addition, our call-set-up algorithms can avoid the possibility of crosstalk cycles.

I. INTRODUCTION

WAVELENGTH-ROUTED optical (wide-area) networks using wavelength cross-connects have been proposed for the purpose of providing transparent interconnection and wavelength reuse [1], [3]. Such a wavelength-routed optical network is being studied and demonstrated as a part of ARPA-sponsored programs at Bellcore (ONT and MONET). The wavelength cross-connect switches may be implemented using acousto-optic tunable filters (AOTF’s) or WDM multiplexer/demultiplexer-based technologies. The wavelength cross-connects may or may not include wavelength-translation functionality.

However, as a direct consequence of transparency, these networks can form closed cycles that contain amplifiers, e.g., erbium-doped fiber amplifiers (EDFA’s). Such closed cycles create the possibility of oscillations and amplifier saturation due to the recirculation of amplified spontaneous emission (ASE) noise from the EDFA’s. Hence, searching for and eliminating such closed cycles becomes very important; otherwise, the network will be nonfunctional.

The wavelength cross-connects considered here are $K \times K$ reconfigurable optical switching elements which can achieve any permutation of input to output ports, independently for each of $W$ wavebands, $\lambda_1, \lambda_2, \ldots, \lambda_W$. That is, each input port of a cross-connect switch is always connected to some output port of the switch whether or not such a connection is required for the purpose of information transfer. As a result, there exists the possibility of setting up unintended all-optical cycles in the network (viz., a loop with no terminating electronics in it). If such a cycle contains amplifiers [e.g., Erbium-Doped Fiber Amplifiers (EDFA’s)], there is a possibility that the net loop gain is greater than the net loop loss.

End users communicate with one another through electronic access stations which are attached to some of these cross-connects. The access stations exchange information using signals on wavelengths within the wavebands $\lambda_1, \lambda_2, \ldots, \lambda_W$. Consider the EDFA gain curve shown in Fig. 1. We define $\lambda_k$ to be the desired set of wavebands used for communication between the access stations.

A waveband may contain just one wavelength or a band of several wavelengths.

Fig. 1. EDFA gain curve.
Fig. 1. Wavelength routing using AOTF.

We define $A_0$ as the remaining set of wavelengths in the amplifier spectrum, excluding the set $A_k$.

We will examine Acousto-Optic Tunable Filter (AOTF) cross-connects [5] and other types of wavelength cross-connects for the way they treat the build up of ASE from the EDFA's.

A. Wavelength Cross-Connect Switches

The transmission properties of the AOTF cross-connect are depicted in Fig. 2(a) and 2(b). As a first approximation, let us assume that this device impresses (on its incident signal spectrum) piecewise-flat, transparent passbands for wavebands $\lambda_1$ through $\lambda_w$, separated by opaque stopbands. Fig. 2(a) illustrates the device's transmission spectra when operated in the BAR state for all wavebands, i.e., when no RF control signals are applied. The spectrum above output port 1 shows that, in the BAR state, the device is perfectly transparent from input 1 to output 1 at all wavelengths; the spectrum shown at output port 2 shows that, in this BAR state, the device is opaque at all wavelengths from input 1 to output 2. Fig. 2(b) shows transmission spectra when the device is in the CROSS state for wavebands 1 and 2 (physically achieved by applying the appropriate RF drive frequencies), and in the BAR state for all the other wavebands. In this case, the transmission from input 1 to output 1 is then simply the complement of the lower spectrum. The device's transmission properties when illuminated at port 2 are directly obtainable from the above by symmetry. The above description summarizes the physical behavior of the AOTF under the idealized assumption that its passbands are piecewise flat. Multiple RF control signals can be applied simultaneously to the AOTF with the result that many wavebands may be switched independently and simultaneously. Note that, by default, the AOTF is always in the BAR state for all wavelengths (entire set $A_0$ and set $A_k$), but it can be put in the CROSS state only for wavelengths in the set $A_k$. Fig. 2(c) shows an example of a $2 \times 2$ AOTF in CROSS state for the waveband $\lambda_1$ and in the BAR state otherwise (the notation is self-explanatory). Larger-size AOTF cross-connects have the same properties as the $2 \times 2$ shown here. Other types of wavelength cross-connects (mux/demux-based) also possess characteristics similar to the ones described above.

There are two important observations that we can make from the above discussion.

1) Wavelength cross-connects can result in unspecified states for some wavelengths, i.e., wavelengths may pass through the cross-connects, uninterrupted, in the BAR state, without being set up for this purpose.

2) Setting up a connection on a wavelength between a single input port and a single output port may result in an unspecified connection between another pair of ports. For example, in Fig. 2(b), say we wish to create a connection only from input 1 to output 2 on waveband 5. Since the cross-connect is symmetric, setting up a connection from input 1 to output 2 on waveband 5 automatically results in an undesired and unspecified connection between input 2 and output 1 on waveband 5.

Based on the above observations, by considering multiple cross-connects connected to one another via fiber to form an arbitrary physical topology, we can show that the corresponding optical network can result in closed optical cycles due to unspecified states in the cross-connects.

When the recirculating wavelengths include the set $A_0$, we call the cycle a $A_0$ cycle. When the recirculating wavelengths exclude the set $A_0$ but include the set $A_k$, we call the cycle a $A_k$ cycle.

In the presence of ASE noise from the optical amplifiers, such feedback cycles can saturate the amplifiers and can cause the ASE noise to oscillate. An example of a wavelength-routed network using AOTF's and EDFA's is shown in Fig. 3. The ASE from the EDFA’s passes through the AOTF’s in the BAR state and forms closed feedback loops. Hence, such cycles must be eliminated as a first principle.

This paper builds up on previous work [2], and proposes network-layer solutions to this problem. Furthermore, the presence of crosstalk in the cross-connects can exacerbate the problem. Our algorithms can eliminate crosstalk as well.

In this paper, we examine the problems that arise in the simple scenario where only a single waveband is present, amplifiers are perfect with flat gain curves, etc. We then develop a corresponding solution. Issues related to multiple wavelengths and precise device models have not yet been examined, and are open research problems.

Section II lists our assumptions for this study. Section III describes a number of different modules which comprise our
solution, while Section IV provides the details of the corresponding algorithms used in each module. Some illustrative examples are presented in Section V. Section VI concludes the paper.

II. NETWORK ASSUMPTIONS

Our assumptions for this study are listed below.

1) The physical network consists of (electronic) access stations, wavelength cross-connects (switches), and physical fiber links. A fiber link connects either an access station to a switch or a switch to another switch to form a fiber network, e.g., see Figs. 5 and 6. Fig. 5, which is a subgraph of the NSFNET T3 backbone network, employs five switches and five access stations, with one access station per switch, while Fig. 6, which is a random network generated by one of our algorithms, consists of eight switches, but only four of them have an access station attached.

2) Presently, we consider only a single waveband in the network.

3) The in-degree (number of input ports) of any switch is equal to its out-degree (number of output ports). The same is true for an access station as well. Since we are considering a single waveband, an access station's in-degree and out-degree will be further constrained to unity.

4) So, if switch \( i \) is attached to \( D(i) \) other switches in the physical graph, then switch \( i \) must be of size \([D(i) + m(i)] \times [D(i) + m(i)]\), where \( m(i) \) is the number of access stations attached to switch \( i \). Note that a switch may possibly have no access stations attached to itself, i.e., \( m(i) = 0 \), as in switches 5, 6, 7, and 8 in Fig. 6.

5) Each input port of a switch always connects to one output port of the switch, independent of whether such a port-to-port connection is intended or not.

6) At this point, we are assuming nearly-perfect filters. Crosstalk problems are addressed by our algorithms, but we have not addressed the problems created by switching multiple wavelengths on a fiber.

7) We may design the network's physical connections such that "\( \Lambda_0 \) cycles" are eliminated when switches are in default (BAR) state [2]. Alternatively, the network must have been previously designed this way.

III. OVERVIEW OF SOLUTION/ALGORITHMS

In this section, we provide an outline of our network-layer solution approach to eliminate all-optical cycles in a wavelength-routed optical network (see Fig. 4). The solution approach consists of six modules. High-level descriptions of these modules are provided below, and details of the specific algorithms that they employ are provided in Section IV.

Network Generator: This module generates a random network consisting of access stations and switches. It is not needed when a physical description of the actual network to be studied is already available.

Switch Port Labeler: This module labels the input and output ports of each switch according to whether the port carries an active connection or not.

\( \Lambda_0 \) Cycle Eliminator: This module removes any \( \Lambda_0 \) cycles in the default state of the network (when all switches are in BAR state). When a random network is generated, it may contain \( \Lambda_0 \) cycles when the switches are in BAR state. These cycles need to be removed only once, i.e., when designing the network.

Connection Setup: This module handles each new connection request that needs to be established. The new connection is attempted without disturbing the current routing (i.e., switch settings) of existing connections. This module outputs new settings of the switches to accommodate the requested connection, if possible.

\( \Lambda_k \) Cycle Eliminator: This module attempts to eliminate both \( \Lambda_k \) cycles and crosstalk in the network while establishing the requested connection.

(Static) Network Configuration Analyzer: This module is independent of the other modules. It is used off-line to
statically assess the properties of a given network, e.g., number of connections achievable at any one time in the network without the occurrence of $A_0$ and $A_k$ cycles and crosstalk. In particular, given that a $d \times d$ switch can be in one of $d!$ states, how many of the network’s states are good or bad, given that each of the network switches can be in any state.

Fig. 4 gives an overview of how the different modules in this solution approach interact with one another. All of these modules are described in detail in the following section.

IV. DETAILS OF ALGORITHMS

For each of the modules outlined in the previous section, we now provide its goal, required inputs, the output that it generates, method/alGORITHM used, and possible enhancements wherever applicable. Readers who are interested in only the overview of these modules/algorithms, but not their details, may skip this section and go directly to Section V.

A. Network Generator (Module 1)

Goal: Generate a random network for testing various algorithms. (This module is not needed if the physical network’s connectivity pattern is known.)

Input: Number of access stations; Number of switches; Minimum and maximum size of each switch (switch size currently is uniformly distributed between this minimum and maximum); and Boolean indicating possibility of more than one access station per switch.

Output: File containing number of access stations; number of switches; links connecting the stations and switches of the network; and switch settings. Switch settings are output from this module with switches in BAR state. (If a known physical graph is to be analyzed, its connectivity information should be provided according to the format output by this module.)

Methods/Algorithms used: The program first randomly determines the size of each switch, which is currently based on an uniform distribution between the minimum and maximum size of switches. This switch size calculation is easily modifiable to accommodate other distributions (including a different specification for each switch). The program then randomly connects access stations to switches, and switches to other switches.

The program then “corrects” the following three problems that may occur in a randomly-generated network:

- a self loop (i.e., a switch connected to itself);
- a switch that is connected to another switch more than once; and
- a disconnected network (i.e., inability of a station to reach all other stations or switches). The corrections are performed in a random fashion in order to maintain the true random nature of the network.

Possible Enhancements: Placements may be based on input parameters to examine propagation delay of a link (currently, the program just indicates which physical links exist, not the distance of a link; physical location of a link; etc.) Additional of more input parameters will allow flexibility in the types of random networks generated (e.g., server stations, connectivity requirements, etc.)

B. Switch Port Labeler (Module 2)

Goal: To analyze the current network configuration and label all the ports on each switch as either white, black_used, or blackUnused. A switch port, either input or output, is referred to as black if it is NOT part of an all-optical cycle; it is referred to as black_used if it is carrying a (active, intended) connection; otherwise, it is called blackUnused.

To count the number of white cycles in the network and list switches which each white cycle passes through.

To compare the intended and the established interconnection patterns of stations and report any discrepancy. This informa-
tion is later utilized by the algorithm which eliminates these white cycles in Modules 3 and 5.

**Input:** Number of access stations; Number of switches; Current settings of each switch; Intended interconnection pattern of stations; and Physical links in the network.

**Output:** Number of white cycles; List of switches in the path of each white cycle; For each switch in the network,

- Label of each port (white, black_used, black_unused);
- Number of the white cycle, if any, passing through each port; Settings information; and Physical links in the network.

**Method used:** Each switch and each access station is broken into as many nodes as there are input and output ports in it. An adjacency matrix is formed for the graph consisting of these 'port' nodes. The physical link information and the settings information is used to form edges in this graph. Starting from any node in this graph, we traverse a path, which is guaranteed to return to the starting node. (Since this graph is Eulerian, it can be broken into edge-disjoint cycles.) We then examine if this cycle is white (goes through only switch ports) or black (goes through at least one access station). This is done until all the edges in the graph have been traversed. The interconnection pattern of stations is found out by starting from the node corresponding to the output port of each access station and keeping note of the destination. A comparison is made between the active connections intended and those established; any discrepancy results in an error message.

**C. \( \Lambda_0 \) Cycle Eliminator (Module 3)**

**Goal:** To create a network that contains no \( \Lambda_0 \) cycles when the switches of the network are in BAR state.

**Input:** Number of access stations; Number of switches; Size of stations (BAR state by default) of each switch; Physical links in the network; and Labels of the switch ports — white, black_used, or black_unused. [At this point (physical network design stage), there will be no port labeled black_used since no connections have been set up yet.]

**Output:** File containing full network specifications (number of access stations, number of switches, size and settings (BAR state) of each switch, physical links in the network) for the input network converted such that it contains no \( \Lambda_0 \) cycles.

**Methods/Algorithms used:** The algorithm "visits" each switch (in order) that currently contains an input port that is labeled white. It then attempts to find another input port that is labeled black_unused at that switch. If one is found, it relabels all ports that were part of that white cycle to black_unused and "swaps" the switch settings of those two input ports such that the white labeled input port now goes to where the black labeled input port used to go, and the black labeled input port now goes to where the white labeled input port used to go. This combines the two cycles (one black, one white) into one larger black cycle (black since this larger cycle still contains at least one access station that caused the smaller original black cycle to be black). If no black_unused port at that switch is found, then go to the next switch that contains a white port and repeat the process. If, after going through all switches containing white cycles, some switches still contain white cycles, repeat process until no switch contains any white cycle. If there is any way to eliminate white cycles, this method is guaranteed to break them. Due to the Eulerian property of this network, we are guaranteed that there is always a way to break white cycles at this point, so we can say that this program always succeeds in breaking white cycles. After the cycle breaking is complete, the switches will be in a specified state. We want to achieve this network configuration but with the switches in BAR state. Therefore, move the physical output port connections to achieve this network when the switches are in the BAR state. As an example, if for a particular switch, input port 0 is connected to output port 3 of that switch, connect output port 0 to what output port 3 used to be connected to. This will finally result in a network that has no \( \Lambda_0 \) cycles when the switches are in the BAR state.

**Possible Enhancements:** Make the algorithm more efficient. There are a few ways that the algorithm can be made to run in time \( O(\text{number of white ports}) \), but these improvements have not been implemented yet.

**D. Connection Setup (Module 4)**

**Goal:** To establish a new connection in the network by creating new settings for the switches, without changing the paths of the existing connections.

**Input:** A structure containing the current state of the network as given by: Number of access stations; Number of switches; Current settings of each switch; Intended interconnection pattern of stations; Physical links in the network; and a list of new connections to be added, given by <Source station, Destination station> pair.

**Output:** SUCCESS, if a switch setting can be found to satisfy the new connection, or FAILURE, if no such path can be found to establish the new connection because of unavailability of network resources such as fiber links and switch ports.

In case of SUCCESS, the algorithm returns: Number of white cycles; List of switches in the path of each white cycle; For each switch in the network, Label of each port (white, black_used, black_unused), Number of the white cycle, if any, passing through each port, Settings information; and Physical links in the network.

**Method used:** The algorithm used for setting up the new connection is Dijkstra's Shortest Path Algorithm. First, we eliminate the active links (links in the path of existing connections) from the graph. Then, we add new edges from all idle input ports of a switch to all its idle output ports. All edges in this new graph are given equal weight (value = 1). If no edge exists between two stations, the corresponding entry in the weight array is set to \( \infty \). We run Dijkstra's algorithm on this graph. The resulting path, if any, is used to change the switch settings. The labeling of ports in this graph is performed as described in Section IV-B.

**Possible Enhancements:** The Shortest Path Algorithm may not always be the best one. By taking the shortest path for this connection, we might decrease the chances for setting up a new connection later, without white cycles and crosstalk. (See example discussed later in Section V.)
We should allow teardown of existing connections. We need to study the blocking probability of any connection after running through a random sequence of connection setups and teardowns.

E. \(A_k\) Cycle Eliminator (Module 5)

**Goal:** To configure the switches to achieve a network that contains no \(A_k\) cycles after certain network connections are set up.

**Input:** Number of access stations; Number of switches; Size and current settings of each switch; Physical links in the network; and Labels of the switch ports – white, black_used, or black_unused. For a connection that was just requested, the path it uses will appear as a black_used path to this program.

**Output:** Structure containing all possible settings (number of access stations, number of switches, size and settings (BAR state) of each switch, physical links in the network) for the input network converted such that it contains no \(A_k\) cycles.

**Methods/Algorithms used:** The algorithm “visits” each switch (in order) that currently contains an input port that is labeled white. It then attempts to find another input port that is labeled black_unused at that switch. (NOTE: Here, a white cycle cannot be combined with a black_used port because, if that happened, a node would be transmitting traffic that entered this switch twice and we would have a problem with crosstalk.) If one is found, the program relabels all ports that were part of that white cycle to black_used and “swaps” the switch settings of those two input ports such that the white labeled input port now goes to where the black labeled input port used to go, and the black labeled input port now goes to where the white labeled input port used to go. This combines the two cycles (one black, one white) into one larger black cycle (black since this larger cycle still contains at least one access station that caused the smaller original black cycle to be black). If no black_unused port at that switch was found, then go to the next switch that contained a white port and repeat the process. If, after going through all switches containing white cycles, some switches still contain white cycles, repeat the process until no switches contain white cycles. If there is any way to eliminate white cycles, this method is guaranteed to break them. This procedure is not guaranteed to succeed at this point due to the crosstalk limitations. If crosstalk were not a concern, we could combine the white ports with either black_unused or black_used ports and this procedure could be guaranteed to succeed.

**Possible Enhancements:** Make the algorithm more efficient. There are a few ways that the algorithm can be made to run in time \(O(d!d)\), but these improvements have not been implemented yet.

F. (Static) Network Configuration Analyzer (Module 6)

**Goal:** To place all the switches in the network in all possible settings and determine:
- total number of states of the network,
- number of GOOD states (no white cycles and no crosstalk in any active connections), and
- number of permutation connection patterns of access stations which are achievable in this network.

**Input:** Number of access stations; Number of switches; Size of switches; and Physical links in the network.

**Output:** Total number of states in the network. Total number of permutation connections in which none of the access stations is idle.

Number of settings which establish these patterns:
- with possible white cycles and crosstalk,
- without white cycles but with possibility of crosstalk, or
- without both white cycles and crosstalk. Number of such connections which are impossible without white cycles or crosstalk.

A list of these impossible connections.

A file with as many lines as there are states giving the information for each state such as:
- Interconnection pattern established,
- Number of white cycles, and
- Presence/absence of crosstalk.

**Method used:** A \(d \times d\) switch can be placed in any of \(d!)\) settings. Suppose there are \(m\) switches, each of size \(d \times d\). Then, the total number of states of the network is \((d!)^m\). We exhaustively place the network in each of these states and then find out the interconnection pattern that has been established. The number of white cycles in the network is also counted. An active path is one between an access station and another station. All self-connections are assumed to be idle connections. In each of the active paths, we check to see if we encounter the same switch more than once. If so, then there is crosstalk in this path. For each state, this information is output in a single line in a specific format.

V. ILLUSTRATIVE EXAMPLES

In this section, we discuss some illustrative examples which demonstrate how our algorithms (modules) presented in the previous sections may be employed to operate, or to learn about the properties of a wavelength-routed optical network. In Section V, we consider the “dynamic” case in which, given an optical network, calls need to be set up without the possibility of white cycles and crosstalk. In Section V, we study the “static” properties of a wavelength-routed optical network, viz. its proportion of good states, etc. Modules 1 through 5 from Section IV are used for the “dynamic” study, while Module 6 is employed for the “static” analysis.

A. Dynamic Analysis

1) Application of Algorithms on Example Physical Networks: To study the application of our algorithms on some example physical networks, we first consider a random network generated by Module 1, with switch ports labeled by Module 2, as shown in Fig. 6. Note that this network does contain \(A_0\) cycles in the initial state of the network when all the switches are in the BAR state. These \(A_0\) cycles are indicated by the dashed lines in Fig. 6. The result of eliminating these \(A_0\) cycles by Module 3 is shown in Fig. 7.

Then, the following two connections are set up: (A) from Station 2 to Station 4; and (B) from Station 4 to Station 3.
Iness et al.: Elimination of Cycles in Networks

The state of the network after establishing these connections is shown in Fig. 8. The links carrying traffic are indicated by thick continuous lines.

Now, we would like to set up a new connection (C) from Station 3 to Station 1. The path suggested by the Shortest Path Algorithm is indicated by the thick dashed lines in Fig. 8. Notice, however, that this routing choice results in a \( \Lambda_k \) cycle which is indicated by thin dashed lines. To avoid this \( \Lambda_k \) cycle, the \( \Lambda_k \) Cycle Eliminator Algorithm (Module 5 in Section IV) is employed. It changes a few switch settings, and the resulting network is shown in Fig. 9. Notice that none of the paths for the existing connections is altered.

2) Performance Analysis of Dynamic Algorithms: To study the performance of the dynamic algorithms, we assume the following. 1) Calls (i.e., connection requests, also referred to as messages) arrive at each station according to a Poisson process, with an intensity (load) of \( \alpha \) per unit time at each station, and the holding time for each call is exponentially distributed with a mean of 100 time units. 2) A call arriving at each station is equally likely to be destined to any of the other stations. 3) Each station can source at most one call at a time, i.e., calls are not buffered, or a call arrival is allowed only if the station's transmitter is idle.

A call may be blocked due to unavailability of resources, viz., 1) if the destination is busy, or 2) if there is no free path (through fiber links and switches) to the destination even if the destination is available. This is referred to as resource blocking.

A call that is not resource-blocked may still be considered to be blocked if its establishment would have introduced crosstalk. This is referred to as crosstalk blocking.

Calls that are blocked due to either of the above blocking phenomenon are immediately lost.

Now, consider Fig. 5, which has five access stations, 17 links, and five switches, and which is a subset of the NSFNET T3 backbone. Calls arrive to this network according to the above model. For this network, we would like to study its throughput and call blocking performance when the call-set-up algorithms outlined in earlier sections are used.

We would also like to study, for the same number of access stations and physical links, what is the effect of reducing/expanding the number of switches. In one extreme, we consider a network of two switches, since a network of just a single switch is not very meaningful with our assumptions—our assumption that an access station has in-degree = out-degree = 1 would mean that, if the network has \( N \) access stations and \( L \) links, then the single giant switch must have \( N - L \) external (and useless) fiber links connecting one of its output ports to one of its input ports. In the other extreme, we consider the network to consist of as many \( 2 \times 2 \) switches as possible; thus, the largest number of switches turns out to be \( \lfloor L/2 \rfloor \). For a particular number of switches, we find that the network demonstrates the best performance if all switches are as equally-sized as possible. Thus, in our network construction, we maintain the difference between the largest-sized and the smallest-sized switches to be unity; also, whenever possible, access stations are attached to the larger-sized switches in order to achieve better blocking performance.²

² We use these principles for generating the numerical results in Figs. 10–15; however, these principles are not followed in the random graph in Fig. 6.
Expanding the number of switches would mean reducing the sizes of (some or all of) the switches (for the same number of fiber links). Smaller switches, in turn, would reduce the number of possible network states (as determined by all possible combinations of switch settings), so that resource blocking would increase. When the number of switches is reduced, the sizes of the switches will increase, in general, for the same number of links. However, now, a larger number of network states will also be unusable due to crosstalk, as we shall observe in the static analysis in Section V.

The dynamic performance of the five-station, 17-link network in Fig. 5 has been studied, along with that of its derivatives with an expanded/contracted number of switches M as outlined in earlier paragraphs. The loading of calls is varied from $\alpha = 0.01$ to $\alpha = 2$, and the corresponding performance results for different numbers of switches are shown in Figs. 10–12. These results were obtained by simulating calls for 100,000 time units, collecting the pertinent statistics, and then averaging the statistics over five runs (i.e., five different graphs) for each switch size. In general, as expected, the higher the offered load, the higher is the probability of call blocking due to unavailable resources and crosstalk, and the lower is the fraction of unblocked calls. One anomaly we would like to comment on is the value of crosstalk blocking for $M = 6$ and $M = 8$. Due to the small number of access stations, it is fairly easy to get all of the stations communicating with one another. For $M \leq 6$, five stations would be prevented from talking to one another simultaneously due to crosstalk. In the 8-switch case, five stations would usually not be able to talk to one another simultaneously due to lack of a path. So, in effect, the probability of crosstalk blocking is going down. Also, note that a call is considered resource-blocked first and crosstalk-blocked second, and resource blocking is the dominant phenomenon (the absolute value for resource blocking is generally over an order of magnitude higher than crosstalk blocking).

The effect of the number of switches $M$ is interesting. At light loads, when $M$ is increased, the probability of
blocking increases (and the fraction of unblocked calls decreases) because, now, a call has to find its way through several small switches. When the loading is heavy, each station's receiver is busy most of the time, and this dominates the probability of resource blocking, which becomes nearly independent of \( M \). Correspondingly, the fraction of unblocked messages in nearly independent of \( M \) for heavy loading.

When \( M \) is small, crosstalk blocking increases with increasing load. When \( M \) is increased, for light loading, crosstalk blocking increases, since most calls are not resource-blocked.

We repeated these experiments on a larger network, viz., the 14-station, 56-link, NSFNET T1 network that existed until early 1992, before it was upgraded to the NSFNET T3 network. For this example, we studied not only the capabilities of the 14-switch network, but also those of its related constructions employing \( M = 2 \) through \( M = 26 \) switches. The corresponding performance results are shown in Figs. 13–15. Now, each simulation was run for 2000 time units which is not very long but which seemed to provide reasonably-accurate results (also, each run took quite a long amount of computer time due to the larger network size). As before, statistics were compiled and averaged over five different graphs for each value of \( M \). The results in Figs. 13–15 reinforce our earlier observations on the smaller five-station network. However, now, the value of \( M \) seems to play a more dramatic role with respect to increasing blocking (and decreasing the fraction of unblocked calls) for higher values of \( M \).

B. Static Analysis

Consider Fig. 5, which has five access stations and five switches. In one extreme, when all five stations are busy, we have a complete permutation connection pattern. Notice that the receiving stations form a permutation of the sending stations. The number of such permutation connection patterns formed from \( N \) stations is given by \( \chi(N) \), e.g., \( \chi(5) = 44 \). Now, among all the network states which support such a permutation connection pattern, we keep track of how many are unaffected by cycles and crosstalk. In fact, we keep count of the number of states which achieve each such pattern without cycles and crosstalk. If we do not find even a single state to support a pattern, then that particular connection pattern is \textit{impossible} to achieve in the network with the given constraints. At the other extreme, when all five stations are idle, the network could be in any of its states with all of its transmitters being idle. Between these two extremes, we have the case of exactly \( k \) stations being idle, where \( 1 \leq k \leq 4 \). We are interested in finding out if each of these connection patterns is achievable on this network. In other words, does there exist at least one network state which supports each of these connection patterns?

Assume that a permutation routing pattern has to be set up without white cycles or crosstalk, i.e., we need to set up five connections, with each access station being the source of a connection, as well as the sink of a (another) connection, i.e., there are no self-loops. The following were the results obtained after an exhaustive search through all possible switch settings of the network shown in Fig. 5.

- Number of Access Stations = 5
- Number of Switches = 5
- Number of Links = 22

Total number of switch settings (network states): 124416.
Total number of connection permutations (with five stations): 44.

- # of network states allowing the permutations of connections
  - With cycles and crosstalk: 9696.
  - Without cycles but with crosstalk: 6528.
  - Without cycles and crosstalk: 84.
- # of possible connection patterns (without cycles and crosstalk): 36.
- # of impossible connection patterns (due to cycles and crosstalk): 8.

The following eight configurations were the set of connections which were impossible to establish due to white cycles and crosstalk. (Notation: Each group represents a complete set of five connections. The sources of the connections are 1
through 5 from left to right. The respective destinations are given below.

<table>
<thead>
<tr>
<th>M</th>
<th>Settings</th>
<th>Impossible Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>124416</td>
<td>62208 (50.00%)</td>
</tr>
<tr>
<td>6</td>
<td>15552</td>
<td>10016 (64.40%)</td>
</tr>
<tr>
<td>7</td>
<td>3456</td>
<td>2456 (71.06%)</td>
</tr>
<tr>
<td>8</td>
<td>768</td>
<td>310 (60.36%)</td>
</tr>
</tbody>
</table>

Now, consider all possible connection patterns with five stations. The number of connection patterns of five stations with different numbers of idle stations are the following:

* No idle stations = 44
* 1 idle station = 265
* 2 idle stations = 320
* 3 idle stations = 130
* 4 idle stations = 20
* 5 idle stations = 1

Total = 780.

We would like to study the capability of the network in Fig. 5, as well as its derivatives with an expanded/contracted number of switches (as was studied in Section V), to support the available connection patterns without white cycles and crosstalk. In this regard, we could study networks only for \( M \geq 5 \), since for smaller values of \( M \), the number of network states was too large for our computing facilities to handle. Below, for each value of \( M \) between 5 and 8 corresponding to an example graph, we show the total number of states, number (and percentage) of states without cycles, and number (and percentage) of states without crosstalk. We also show the number of connection patterns (out of 780 total) that could not be supported because of resource blocking, cycles, and crosstalk in Table I.

The above results indicate the following:

1) The total number of states in the network, as well as the number of good states (i.e., states without white cycles and crosstalk), decreases as the number of switches increases.

2) However, the percentage of good states increases as we go from \( M = 5 \) to \( M = 7 \), and then decreases for \( M = 8 \). We have observed similar phenomena for other graphs with the same values of \( M \). This may partially explain the anomalous results in crosstalk blocking that we alluded to earlier in Fig. 12 while discussing the dynamic network performance.

3) The number of impossible connection patterns increases drastically when we go from \( M = 6 \) to 7 to 8, since the corresponding number of good states also reduces drastically.

We have not been able to perform static analysis of the 14-station network because the state space explodes. However, we believe that a sampling technique may be applicable to get a reasonable idea of the fraction of network states that are good (or bad). That is, we need not enumerate all states, but pick each state with a certain probability in order to obtain reasonable statistics in a reasonable amount of time. However, this is still an open problem.

VI. CONCLUSION

We considered the problem of setting up unintended all-optical cycles in a transparent, wavelength-routed, optical network employing wavelength cross-connect switches. If such a cycle contains amplifiers so that the net loop gain is greater than the net loop loss, the ASE noise from the amplifiers can build up, oscillate in the loop, and saturate the amplifiers. We examined algorithms that can set up new transparent optical connections upon request while avoiding the creation of such cycles in the network. These algorithms attempted to find a route for a connection and then (in a post-processing fashion) configured the switches such that white cycles that might get created automatically got eliminated. In addition, our call-set-up algorithms avoided the possibility of crosstalk. We also developed mechanisms for statically analyzing the capabilities of a network, viz., the absolute value and the fraction of good (and bad) states, and the corresponding connection patterns the network can (and cannot) support.

A large number of open research problems still exist. First, algorithms need to be developed to allow multiple wavebands. Also, problems arising out of interactions between physical layer issues and network routing problems need to be examined closely, such as precise characteristics for amplifiers and cross-connects.

ACKNOWLEDGMENT

The authors would like to thank Dr. C. Brackett of Bellcore for introducing this problem to us. We also thank Dr. J. Heritage of U. C. Davis for several valuable discussions which have improved our understanding of the problem.

REFERENCES


REFERENCES


Biswa Mukherjee (S'82-M'87) received the B.Tech. (with honors) degree from Indian Institute of Technology, Kharagpur, India, and the Ph.D. degree from the University of Washington, Seattle, in 1980 and 1987, respectively.

In 1987, he joined the University of California, Davis, where he has been a Professor of Computer Science since 1995. His research interests include lightwave networks, high-speed local and metropolitan area networks, and network security. At Washington, he held a GTE Teaching Fellowship and a General Electric Foundation Fellowship. He is a co-winner of paper awards presented at the 1991 and the 1994 National Computer Security Conferences.

Dr. Mukherjee serves on the editorial boards of the IEEE/ACM TRANSACTIONS ON NETWORKING and Journal of High-Speed Networks. He also served as Technical Program Chair of the IEEE INFOCOM'96 Conference.

Krishna Bala (S'91-M'92) for a biography, see this issue, p. 1066.

Jason Iness (S'90) received the B.S. degree in computer science and engineering with highest honors from the University of California, Davis, in 1992. He is currently working toward the Ph.D. degree in computer science at the University of California, Davis.

He has been conducting research in WDM-based lightwave networks and ATM networks. At Davis, he has been a teaching assistant (Fall '92, Winter '93) and a research assistant (Spring '93-present). He also received the U.C.D. fellowship in the 1992-93 and 1995-96 academic years.

Byrav Ramamurthy was born in Madras, India. He obtained the Bachelor of Technology degree in computer science and engineering from the Indian Institute of Technology, Madras, and his Master of Science degree from the University of California (UC), Davis, in 1993 and 1995, respectively. He is currently working toward the doctoral degree in the Department of Computer Science at UC Davis.

His research interests include optical network design, multi-media and distributed computing. He is a recipient of the Indian National Talent Search Scholarship and is currently a fellow of the Professors for the Future program at UC Davis.