Dynamic Reconfiguration in Multihop WDM Networks

George N. Rouskas
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206

Mostafa H. Ammar
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract

We consider multichannel multihop lightwave networks with stations equipped with a small number of transmitters and receivers. By assigning wavelengths to the receivers and transmitters at each station, one can define the logical connectivity of the network independently of the underlying physical topology. The advent of fast tunable optical transmitters and receivers makes it feasible to dynamically update the network connectivity to accommodate traffic demands that vary over time. Of major concern in such design is how the connectivity should react to changes in traffic patterns. The problem is formulated as a Markovian Decision Process and the properties of the optimal configuration policy are identified. These properties are then used to develop an algorithm for obtaining policies that make decisions similar to the decisions of the optimal policy. A procedure is also proposed to manage the large state space for systems with a large number of stations.

1 Introduction

Wave Division Multiplexing (WDM) is emerging as a promising technology for the next generation of multiuser high-speed communication networks. WDM divides the low-loss wavelength spectrum of the optical fiber into independent, non-overlapping channels, each operating at a data rate accessible by the attached stations. The multiple channels introduce transmission concurrency and provide a means to overcome the speed mismatch between electronics and optics. As a result, WDM networks have the potential of delivering an aggregate throughput that can grow with the number of wavelengths deployed, and can be in the order of Terabits per second.

In multihop networks each station is equipped with a small number of transceivers [1, 2]. An assignment of transmit and receive wavelengths defines an interconnection pattern independent of the underlying physical topology. Packets are relayed to their destination through, possibly, intermediate stations, undergoing conversion from the optical to the electrical domain at each hop. By properly assigning the wavelengths the connectivity can be optimized with respect to some performance parameters. Techniques have been developed to minimize the mean packet delay [3], and the maximum link flow [4], given some information about the network traffic load.

In environments where traffic demands change over time, it is desirable to have the network connectivity dynamically respond to these changes. With the advent of fast tunable optical transceivers [5], it is feasible to contemplate the design of such networks. Of major concern in such design is *when* and *how* the connectivity should react to changing traffic patterns. The approach taken by Labourdette and Acampora [6] is to reconfigure the network infrequently, and only when the traffic pattern changes dramatically or when the current connectivity cannot accommodate the traffic load. Reconfiguration is achieved through a series of branch exchange operations, whereby only one pair of transceivers is retuned at a time. At the other extreme, Auerbach and Pankaj [7] have devised a distributed algorithm to rearrange the connectivity at, potentially, the beginning of every packet burst. Their algorithm recursively tries to establish 1-hop, 2-hop, etc., paths, and can handle concurrent requests.

These approaches suffer from two problems. First, no attempt is made to model the effect of the reconfiguration phase on the overall network performance. The transition from one connectivity to another incurs some cost due to packet loss, the control resources involved in transceiver retuning, and the features of each reconfiguration scheme; this cost is not taken into account in the design process. A long reconfiguration phase of branch exchange operations results in outdated routing tables at all stations, and, consequently, misrouted packets, congestion and more packet loss. Auerbach and Pankaj's scheme requires the execution of a very complex algorithm for every packet burst. Secondly, the issue of when to reconfigure the network has been decided upon *a priori*, without investigating alternative solutions or considering the trade-offs involved.

In this paper we start by modeling the effect of the reconfiguration phase on network performance in terms of packet loss. We then take this reconfiguration penalty into account in the design of reconfiguration policies. Therefore, the reconfiguration policy to be used and, consequently, the frequency of reconfiguration is determined by the extent of packet loss.

Following the introduction we present a model of the network and of the reconfiguration phase. In Section 3 we introduce the concept of a configuration policy and in Section 4 we formulate the problem as a Markovian Decision Process. Section 5 presents the properties of the optimal configuration policy, obtained for a small network. In Section 6 we develop an algorithm to obtain good configuration policies and Section 7 describes our approach to managing the state and decision space explosion. Finally, Section 8 contains some concluding remarks.

2 Network Model

We consider a network of N stations, each equipped with a small number, p, of transceivers attached to a broadcast optical medium that can support $C = pN$ wavelengths (see Figure 1). In a network with tunable transmitters and fixed receivers (TT-FR), each receiver is assigned a unique receive wavelength, while the transmitters can tune over the entire range of wavelengths; similarly for a fixed-transmitter, tunable-receiver (FT-TR) network. An assignment of transmit and receive wavelengths defines a logical connectivity. The tuning delay is defined as the time it takes a transceiver to tune from one wavelength to another, and can be different for different transceivers and/or wavelength pairs. For our purposes, knowledge of Δ_{min} and Δ_{max}, the minimum and maximum tuning delays in the network, respectively, is sufficient.

We define a *template* as a logical diagram that provides at least one path between any pair of stations [1]. For a given network (i.e., for a given N and p), a large number of different templates is possible. In general, the set of templates, $\mathcal{T}$, that we will consider will be a subset of the set of all templates, and will be derived using information about the traffic characteristics. At any time instant the connectivity will be described by a template $\tau \in \mathcal{T}$. The connectivity can be changed to a new template $\tau' \in \mathcal{T}$ by assigning different wavelengths to (retuning) all or some of the transceivers.

[1]For this paper we assume that if a receiver of station i is tuned to a transmitter of station j, then a receiver of j is also tuned to a transmitter of i; this, however, need not be true in general.

Communication in the network is connection oriented; a connection must be established prior to any data been transferred between any two stations. Connections are established by issuing *connect* requests. A *disconnect* request is issued at the conclusion of a session. A connection, c, is identified by two end-point stations and its duration follows an exponential distribution with mean $\frac{1}{\mu_c}$. The time between the termination of connection c until it is requested again is exponentially distributed with mean $\frac{1}{\lambda_c}$.

2.1 The Reconfiguration Phase

In general, reconfiguration of the network connectivity from one template to another will be triggered by the occurrence of an *event* (what constitutes a valid event will be defined formally later). When such an event occurs, several actions must be taken:

1. A new connectivity (template) must be determined, based on the current connectivity and the information carried by the triggering event.
2. The decision to reconfigure, as well as the new connectivity must be communicated to all the stations, not just those that will have to retune their transceivers, since the routing tables may need to be updated.
3. Finally, the actual transceiver retuning must take place.

The rest of the paper addresses the problem of determining what the new connectivity should be. In this section we focus on the remaining two issues.

One option for reporting a reconfiguration triggering event would be to have a dedicated station detect the occurrence of events, process them and compute the new connectivity, and inform all other stations. A distributed version would require each station to detect local events and report them, possibly on a common control channel employing TDMA. In the latter case, the station reporting an event may also compute and transmit the new connectivity. A problem may arise due to concurrent events arriving at different parts of the network. A solution would be to have the stations report only the occurrence of events; at the end of each TDMA cycle on the control channel all stations would use the same algorithm to determine the new connectivity based on the events that took place during the last cycle.

Let t_r be the time a reconfiguration triggering event, e, takes place. The event will be detected by one or more stations and will be reported to the network, possibly by one of the mechanisms discussed above. Regardless of the specific implementation, the net effect is that station i will find out about the occurrence of e at time $t_r + T_i(e)$; $T_i(e)$ is the delay introduced by the event reporting mechanism. This delay is a function of the event e (if i detects e then $T_i(e) = 0$, otherwise $T_i(e) > 0$), and, in general, $T_i(e) \neq T_j(e)$ for $i \neq j$.

In order to eliminate inconsistencies in routing tables and minimize the need for synchronization among the network stations, the reconfiguration phase must be as short as possible. We, therefore, require that all stations retune their transceivers “simultaneously”. For the distributed environment under consideration, in which the stations do not share a common clock, “simultaneously” should be interpreted as “as soon as they find out about the reconfiguration triggering event”. In particular, station i's actions at time $t_r + T_i(e)$ for each of its transceivers that needs to be retuned are as follows:

1. Complete the transmission (reception) of the current packet, if any.
2. Retune the transceiver to the new wavelength. During retuning (which takes time anywhere between Δ_{min} and Δ_{max}), update the routing tables to reflect the new connectivity.
3. Start transmitting (receiving) packets as soon as retuning is complete.

If a transceiver does not need to be retuned, its operation is not affected.

2.2 The Effect of the Reconfiguration Phase on Network Performance

We are interested in the effect of the reconfiguration phase on packet loss. Lost packets have to be retransmitted, increasing the average delay experienced by an application. Also, some loss-sensitive applications may not tolerate excessive packet loss. In this section we show how to compute the packet loss incurred during the reconfiguration phase for a TT-FR network. The analysis for the case of tunable receivers is very similar and is omitted [2].

One point of the network is taken as the reference point, RP. RP has the property that the optical signal passing through it is the combination of the signals of all the transmitters in the network. Depending on the physical topology, RP would be the hub (for a star network), the bend (for a D-bus), or the root (for a tree network). The propagation delay from station i to RP is given by d_i.

[2] If the receivers are tunable, packet collisions are not possible. Packets can still be lost, however, if they reach the intended receiver while the latter is in the process of retuning, or they may be received by the wrong station if reconfiguration has taken place during their flight (recall that propagation delays dominate in high-speed environments).

In Figure 2 we show the occurrence of a reconfiguration event that causes the transmitter of j (denoted by X_j) to retune to wavelength λ, used previously by the transmitter of i (which now will retune to a new wavelength). The vertical axis shows the distance of the two stations from RP, while the horizontal axis represents time. The figure shows a worst case scenario, in the sense that (a) at time $T_r + T_i$ when i is informed about the upcoming reconfiguration, it has just started a packet transmission on wavelength λ and has to delay the retuning of its transmitter until the transmission is completed, and (b) j starts retuning its transmitter at the earliest possible time, $t_r + T_j$, its tuning delay is equal to Δ_{min}, and it has a packet to send immediately after tuning to wavelength λ. The first bit of j's packet will arrive at RP at time $t_r + T_j + \Delta_{min} + d_j$, while the last bit of i's packet will arrive at RP at time $t_r + T_i + T_P + d_i$; T_P is the packet transmission time. As a result, there is a time period of length

$$\text{Collision Interval} = \begin{cases} d_i - d_j + T_i - T_j + T_P - \Delta_{min}, & \text{if } d_i - d_j + T_i - T_j + T_P - \Delta_{min} > 0 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

during which, packets by either i or j arriving at RP may collide; for a worst case scenario, we may assume that *all* packets arriving at RP within this time interval will collide.

Observe that in some cases no packets will collide (for example, if in Figure 2 we interchange the positions of i and j relative to RP). Also, in the case of an ATM switch [8] when all stations would be within the same room or building, we may have $d_i \approx d_j, T_i \approx T_j, \forall\ i, j$, and the collision interval can be as short as $\max\{0, T_P - \Delta_{min}\}$. Another way to reduce packet loss is to delay transmissions from station j in Figure 2 by a time δ_j such that

$$d_j + T_j + \delta_j \quad \geq \quad \max_i\{d_i + T_i\} \quad (2)$$

provided that j has enough buffer capacity to store packets arriving during a time interval equal to δ_j.

In general, packet loss cannot be altogether eliminated. Our model can then be used to identify limitations in the network size and frequency of reconfiguration (more on this shortly), or the buffer requirements so that packet loss be kept within acceptable levels.

3 Configuration Policies

The state of the network is defined as a tuple $(\mathbf{v}, \tau)$. $\mathbf{v}$ is a connection state that describes the established connections; it can be described by a bit vector in which a 1 (0) in the c-th bit denotes

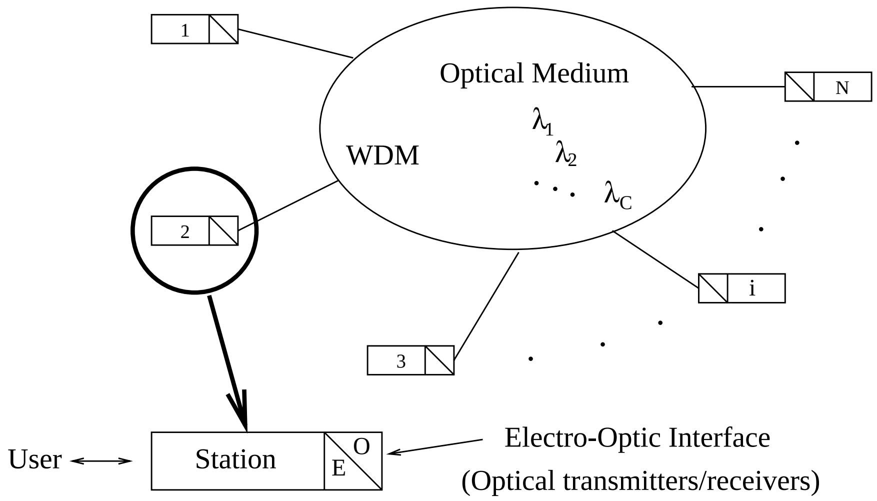

Figure 1: A Lightwave WDM Network

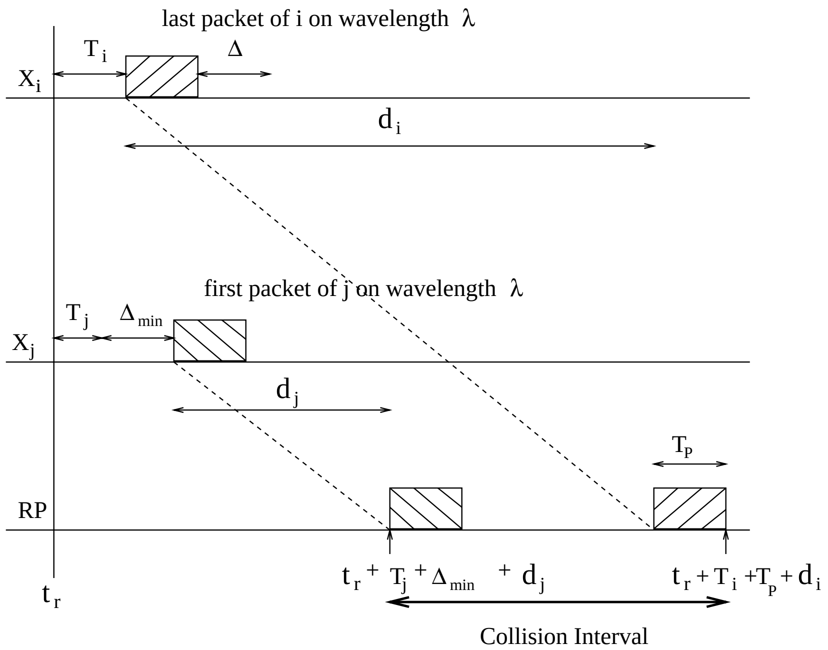
last packet of i on wavelength λ
T_i
Δ
X_i
d_i
first packet of j on wavelength λ
T_j
Δ_{min}
X_j
d_j
T_P
RP
$t_r + T_j + \Delta_{min} + d_j$
$t_r + T_i + T_P + d_i$
t_r
Collision Interval

Figure 2: Reconfiguration cost for Tunable Transmitters - Fixed Receivers

that connection c is on (off). $\tau \in \mathcal{T}$ is a template representing the current network connectivity. Changes in the network state occur at connect and disconnect request instants. Since we define the connect and idle times to have exponential distributions, our system is Markovian. We will refer to $\mathbf{\Omega}$, the set of all possible connection states, and $\mathcal{T}$, the set of all templates, as the *state* and *decision* spaces, respectively.

A network in state $(\mathbf{v_1}, \tau_1)$ will enter state $(\mathbf{v_2}, \tau_2)$ if a connection request or termination causes the connection state to change form $\mathbf{v_1}$ to $\mathbf{v_2}$. Implicit in the state transition is that the system makes a decision to reconfigure into template τ_2. In order to completely define the Markovian state transitions associated with our model we need to establish *next template* decisions. The decision is a function of the current state and the next event and is denoted by $d[(\mathbf{v}, \tau), e]$. Setting $d[(\mathbf{v}, \tau), e] = \tau_{next}$ implies that if event e occurs while the system is in this state, the network should be reconfigured into template τ_{next}. Note that τ_{next} can be the same as τ, in which case the decision is not to reconfigure. A decision needs to be defined for each possible system state and for each valid event. Disconnect requests for existing connections and connect requests for new connections are the only valid events.

The set of decisions for all network states defines a *configuration policy.* A given configuration policy in conjunction with the rates $\{\lambda_c\}$ and $\{\mu_c\}$ completely defines a continuous time, discrete state Markov process. Such a process, depending on the configuration policy, might have multiple chains and/or transient states.

Configuration policies can be:

- *Blocking* or *non-blocking.* With a blocking policy connection requests may be blocked. A non-blocking policy guarantees that any connect request can be satisfied at any time.

- *Rearranging* or *non-rearranging.* With a rearranging policy, an ongoing connection may be rerouted over different paths. This is not allowed by a non-rearranging policy.

Since, by definition, a template provides full network connectivity, our policies will be *non-blocking.* Insisting on a non-rearranging policy would mean that template changes are only allowed when there are no on-going network connections; an uninteresting proposition. We, therefore, allow our policies to be *rearranging.* Rearrangement of the path of an existing connection may cause some packets to be lost. The extent of packet loss will be a factor determining the particular configuration policy to be used. Recovery from lost packets is assumed to take place via some higher level (probably end-to-end) protocol.

Finally, it is important to emphasize that this work is concerned with policy selection and not with the mechanisms by which a policy can be implemented.

4 Markov Decision Process Formulation

Our objective is to obtain a configuration policy such that the "cost" of running the network is minimized. We now formulate the problem as a Markovian Decision Process (MDP). There are two ways in which an MDP incurs cost:

1. *Transition Cost*, which is incurred in a lump sum when a state transition occurs, and
2. *State Occupancy Cost*, which is directly proportional to the time spent in each state.

The transition (i.e., *reconfiguration*) cost from state $(\mathbf{v_1}, \tau_1)$ to state $(\mathbf{v_2}, \tau_2)$ is a function of the two templates τ_1 and τ_2 and is incurred due to the packet loss and the control resources involved in transceiver retuning. Let $\nu(t)$ be the number of times the template had to be changed up to time t under some policy, z. Let $r_k, k = 1, \ldots, \nu(t)$, be the number of packets lost during the k-th reconfiguration, and $l(t)$ be the number of packets generated in the network up to time t. We define the average reconfiguration cost, $\mathcal{R}_z$, incurred by policy z, as:

$$\mathcal{R}_z = \lim_{t\to\infty} \inf \frac{\sum_{k=1}^{\nu(t)} r_k}{l(t)} \tag{3}$$

R_z is the fraction of packets lost during the operation of the network under policy z.

We consider a state occupancy cost that is proportional to the distance travelled by a packet, referred to as *hop* cost. Let $(\mathbf{v}(t), \tau(t))$ be the network state at time t, and $h_c(\tau)$ be the distance travelled by packets of connection c when the connectivity is described by template τ. The average "hop" cost incurred by policy z is then given by [3]

$$\mathcal{H}_z = \lim_{t\to\infty} \inf \frac{1}{t} \int_0^t \frac{\sum_{c\in\mathbf{v}(t)} h_c(\tau(t))}{\sum_{c\in\mathbf{v}(t)} 1}\, dt \tag{4}$$

We define the *total cost* for policy z as:

$$\mathcal{A}_z = \alpha\mathcal{H}_z + \beta\mathcal{R}_z \tag{5}$$

where α and β are weights assigned to the costs.

[3] We will use $c \in \mathbf{v}$ to denote that connection c is "on" in the connection state $\mathbf{v}$.

The basic idea is to use these weights to appropriately define a total performance measure. Consider for example the case when the important performance measure is average packet delay. Let α be $1/v$, where v is the speed of light in the optical medium. Let β be the average time-out interval. Then $\beta\mathcal{R}_z$ is the extra delay experienced by packets that are lost and have to be retransmitted, and $\mathcal{A}_z$ gives the average packet delay. On the other hand, for some loss-sensitive applications the only performance measure may be packet loss, in which case we may set $\alpha = 0, \beta = 1$.

Howard [9] has developed a *policy-iteration* algorithm which is guaranteed to produce a configuration policy that minimizes $\mathcal{A}_z$ for our model. A difficulty in applying Howard's algorithm is that its complexity is directly proportional to the number of network states and events, which grows very rapidly with N and $|\mathcal{T}|$ (see Appendix A for a description of this algorithm and a discussion on its complexity). In general, it is not possible to apply Howard's algorithm to obtain the optimal configuration decisions. Our approach is to apply the algorithm to a small system and identify the properties of optimal configuration policies. These properties are then used to develop techniques to obtain configuration policies for larger systems.

5 Properties of the Optimal Configuration Policy

We now consider a network with $N = 4$ and $p = 2$. There are 6 different connections for this network, which can be operating in any of the 3 interconnection patterns (templates) shown in Figure 3. Note that when $p = 2$, for any N, the stations will be connected as a ring. The valid connections and the numbers we will use to refer to them are shown in Table 1. Table 2 lists, for sixteen of the connection states, the template(s) that provide the minimum total hop cost. The table will help us interpret the decisions of the optimal configuration policies.

connection	connection No	connection state
(1,2)	1	(0,0,0,0,0,1)
(1,3)	2	(0,0,0,0,1,0)
(1,4)	3	(0,0,0,1,0,0)
(2,3)	4	(0,0,1,0,0,0)
(2,4)	5	(0,1,0,0,0,0)
(3,4)	6	(1,0,0,0,0,0)

Table 1: Connections and corresponding connection states for $N = 4$

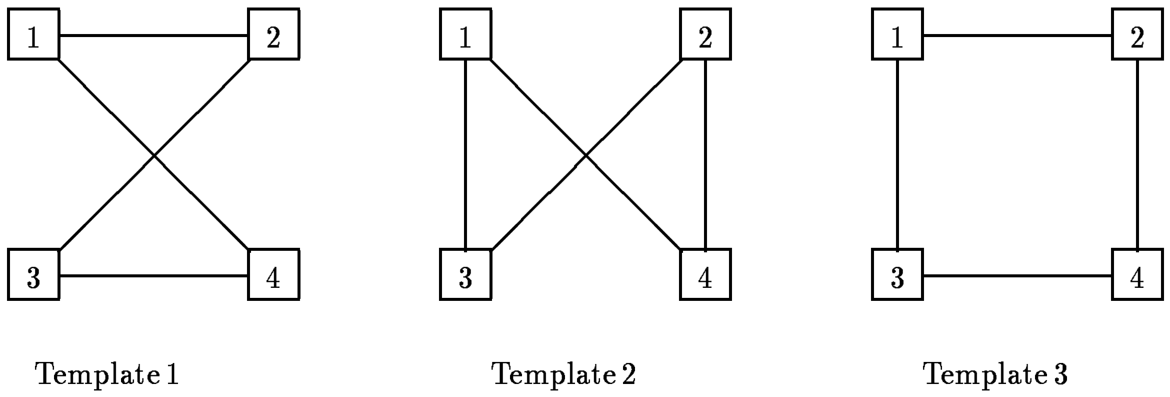

Figure 3: Templates for $N = 4$ and $p = 2$. Each link is bidirectional

For this network we were able to obtain the optimal configuration policy using Howard's algorithm, but only after setting $\lambda_6 = \mu_6 = 0$ (connection 6 was never used). For the results presented here and in the following sections we have made the following simplifying assumptions. First, the distance between any pair of stations was taken to be equal to 1. Secondly, we assume that a connection is always routed over a minimum distance path in the current template. Finally, instead of (3) we used

$$\mathcal{R}_z = \liminf_{t\to\infty} \frac{1}{t}\sum_{k=1}^{\nu(t)} s_k \qquad (6)$$

where s_k denotes the number of transceivers retuned in the k-th reconfiguration instant. We feel, however, that our conclusions about the relative performance of the various policies are not affected by these simplifications (the effect of different levels of packet loss was captured by adjusting the value of β).

The next template decisions of the optimal policy for different values of α and β are shown in figures 4 - 11, where we show what the next template will be if the network is at the *current* connection state and makes a transition to the *next* connection state. For ease of presentation, we only show results for connection states 0 to 15 that do not involve connections 5 and 6. Very similar results have been obtained for the states not shown here.

In Figures 4 - 6 we show the next template decisions when the network is operating in any of the templates 1, 2 or 3, and $\alpha = \beta = 1$. For all connections we assume that $\lambda_c = \mu_c = 1$ [4].

[4]Note that $\frac{\lambda_c}{\lambda_c+\mu_c}$ is the percentage of time that connection c is "on". The higher this value, the more the hop cost the network will incur due to connection c.

connection state	optimal templates	hop cost
0 = (0,0,0,0,0,0)	1,2,3	0
1 = (0,0,0,0,0,1)	1,3	1
2 = (0,0,0,0,1,0)	2,3	1
3 = (0,0,0,0,1,1)	3	2
4 = (0,0,0,1,0,0)	1,2	1
5 = (0,0,0,1,0,1)	1	2
6 = (0,0,0,1,1,0)	2	2
7 = (0,0,0,1,1,1)	1,2,3	4
8 = (0,0,1,0,0,0)	1,2	1
9 = (0,0,1,0,0,1)	1	2
10 = (0,0,1,0,1,0)	2	2
11 = (0,0,1,0,1,1)	1,2,3	4
12 = (0,0,1,1,0,0)	1,2	2
13 = (0,0,1,1,0,1)	1	3
14 = (0,0,1,1,1,0)	2	3
15 = (0,0,1,1,1,1)	1,2	5

Table 2: Optimal templates and hop costs for connection states

Observe that in all cases the next template decision depends only on the next connection state: decisions are the same along a horizontal line. Let us consider decisions out of template 3 (Figure 6). We see that the network either remains at the same template or reconfigures to template 2. Reconfiguration takes place only if the next connection state incurs lower hop cost at template 2. However, for some next connection states, the network does not reconfigure to the template that provides lower hop cost for this next state (for example, see the decisions when the next connection state is 5,9 or 13). Similar observations can be made for the decisions when at template 1 (Figure 4).

It is interesting to see that when at template 2, the decisions are not to reconfigure. Therefore, regardless of which template it is started at, the network will eventually be operating at template 2. Similar results have been obtained by increasing the value of β, and can be explained as follows. For this set of values for $\{\lambda_c\}$ and $\{\mu_c\}$, the average hop cost is minimized when the network is at template 2. Since the importance of the reconfiguration cost is relatively high, the network tends to enter template 2 and stay there, thus incurring zero reconfiguration cost (see (3) or (6)).

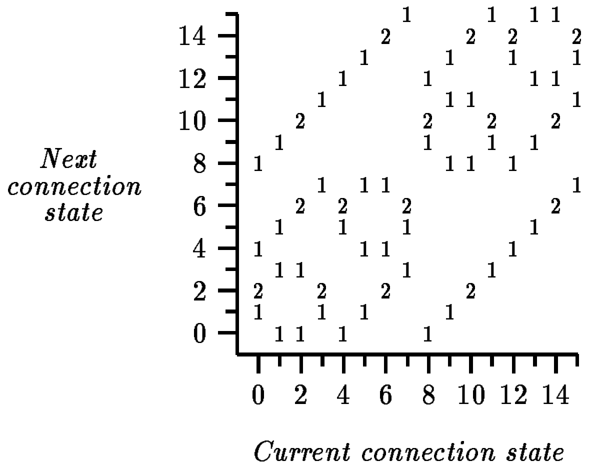

Figure 4: Next template decisions at template 1, $\alpha = \beta = 1, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

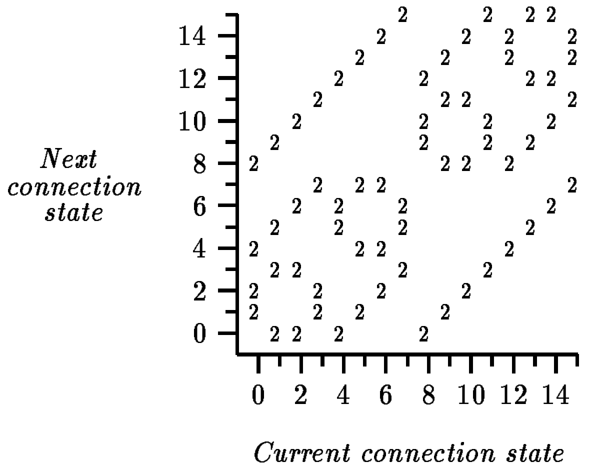

Figure 5: Next template decisions at template 2, $\alpha = \beta = 1, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

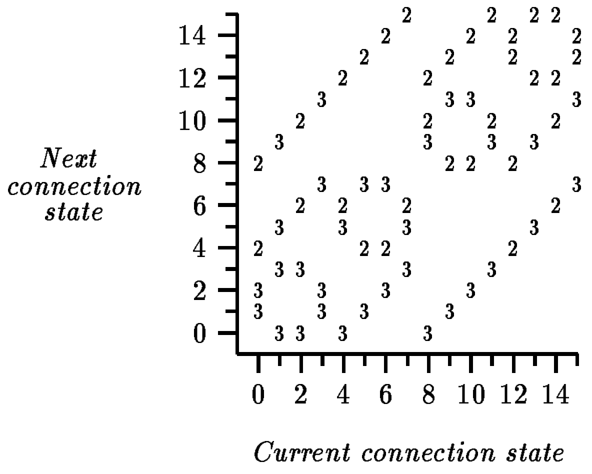

Figure 6: Next template decisions at template 3, $\alpha = \beta = 1, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

We then increase the relative importance of the hop cost by setting $\alpha = 5$ and keeping all other parameters the same. The next template decisions are shown in Figures 7 - 9. The next template is always a template in which the next connection state incurs the minimum hop cost (if this template is different than the current template, the decision is always to reconfigure). Similar results have been obtained for larger values of α. We can see that when the hop cost is important, the network tends to reconfigure to templates that favor the next connection state.

In Figure 10 we show the decisions out of template 2 when $\beta = 0$ (the actual value of α is not important as long as $\alpha > 0$). The next template is again one that provides the minimum hop cost for the next connection state. In particular, although the current template may provide this minimum cost, the decision sometimes is to reconfigure, as for example in the transition from state 6 to state 2. This, of course, is due to the fact that there is no reconfiguration cost involved.

Finally, Figure 11 shows the decisions when the network is at template 2 and $\alpha = \beta = 1$. In this case however, the value of $\frac{\lambda_c}{\lambda_c+\mu_c}$ is equal to 0.5 for connection 1 and is equal to 0.1 for all other connections. Figure 11 (which is identical to Figure 8) should be compared to Figure 5 for which the value of $\frac{\lambda_c}{\lambda_c+\mu_c} = 0.5$ for all connections. In the new network, connection 1 incurs more hop cost per unit time than any of the other connections, because of its longer average duration. For this set of values for $\{\lambda_c\}$ and $\{\mu_c\}$ no template is favored, in terms of the hop cost incurred when the network is operating in it. Thus, the network keeps changing template (decisions out of templates 1 and 3 are the same as in Figures 4 and 6). This example shows how different values for $\{\lambda_c\}$ and $\{\mu_c\}$ influence the decisions taken by the optimal configuration policy.

Based on the above experiments and from various common sense arguments it can be surmised that the basic pattern followed by an optimal configuration policy is as follows:

> When the reconfiguration cost is heavily weighted compared to the hop cost, the decisions most of the time are not to reconfigure. Usually, a template that provides the minimum average hop cost is preferred: if the network enters this template, it will stay there forever. As the relative weight of the hop cost is increased the network tends to reconfigure to templates in which it incurs lower hop cost at the expense of incurring some reconfiguration cost. When the weight of the hop cost exceeds a certain threshold, the network, at each transition, reconfigures to one of the templates that provide the minimum hop cost for the next connection state.

The policies at the two ends of the policy "spectrum" (the *no reconfiguration policy* and *configure for minimum hop cost* policy) can be easily determined. However, the points at which these policies become optimal are not easy to determine as they depend on traffic parameters $\{\lambda_c\}$ and $\{\mu_c\}$. In

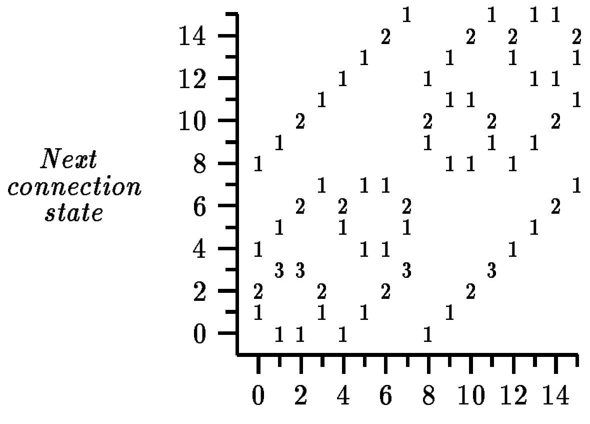

Figure 7: Next template decisions at template 1, $\alpha = 5, \beta = 1, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

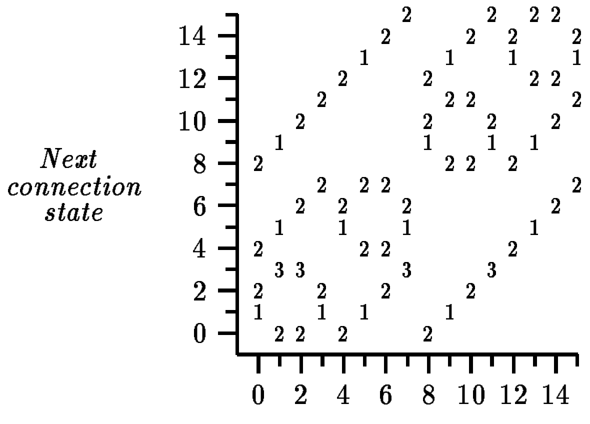

Figure 8: Next template decisions at template 2, $\alpha = 5, \beta = 1, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

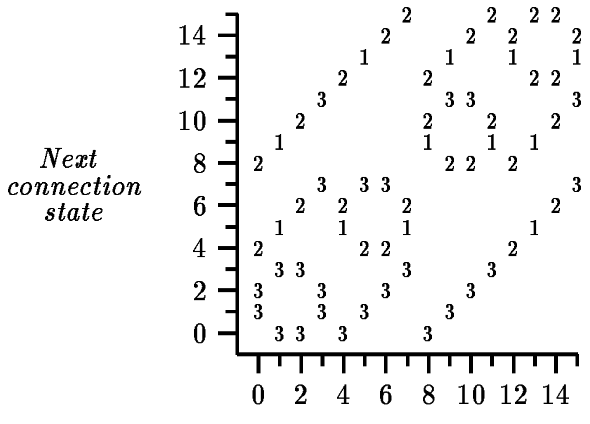

Figure 9: Next template decisions at template 3, $\alpha = 5, \beta = 1, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

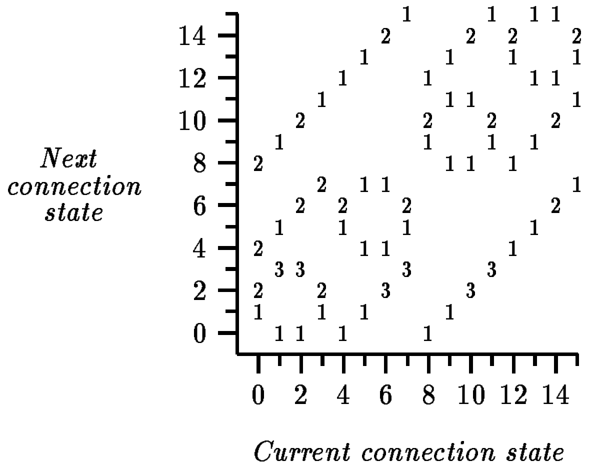

Figure 10: Next template decisions at template 2, $\alpha = 1, \beta = 0, \lambda_c = \mu_c = 1, c = 1, \ldots, 5, \lambda_6 = \mu_6 = 0$

what follows we concentrate on a class of policies for which the decision as to which template to use upon entering a new connection state is only a function of that state, or

$$d[(\mathbf{v}, \tau), e] \quad = \quad \tau_{next} \quad = \quad f(\mathbf{v_{next}}) \tag{7}$$

Our examination of this class of policies is motivated by two factors. First, it is relatively straightforward to compute the cost of such policies (partly because they induce an ergodic Markov process). Secondly, this type of policy has been observed in our experiments for a wide range of parameters.

6 Near-Optimal Policies

Our objective is to find, within the class of policies described by (7), a dynamic configuration policy with low cost. Our approach is to start with the optimal policy in the case of $\beta = 0$ (i.e., the reconfiguration cost is not considered) and modify it to make decisions similar to those of the optimal policy for $\beta > 0$. When $\beta = 0$ the optimal policy dictates that the network be reconfigured to the minimum hop cost template for the new connection state. Such a policy obviously falls into the class defined by (7).

For the class of policies defined by (7) the Markov process consists of a single chain and there are no transient states. We can then compute the hop and reconfiguration costs as follows.

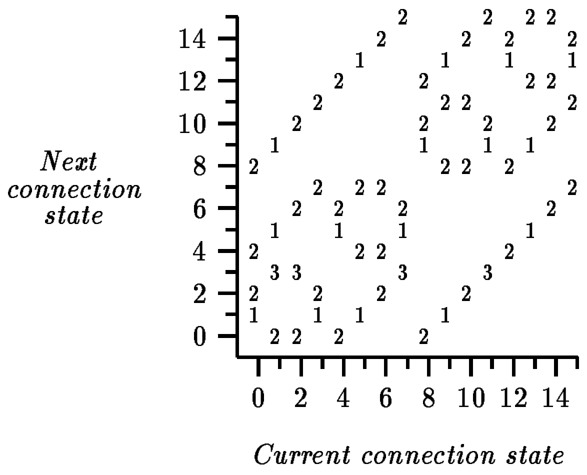

Figure 11: Next template decisions at template 2, $\alpha = 1, \beta = 1, \lambda_c = 1, c = 1, \ldots, 5, \mu_1 = 1, \mu_c = 9, c = 2, \ldots, 5, \lambda_6 = \mu_6 = 0$

$$\mathcal{H}_z = \sum_{\mathbf{v} \in \mathbf{\Omega}} P(\mathbf{v}) \;\; HopCost(\mathbf{v}, f(\mathbf{v})) \tag{8}$$

$$\mathcal{R}_z = \sum_{\mathbf{v} \in \mathbf{\Omega}} P(\mathbf{v}) \;\; ReconfCost(\mathbf{v}, f(\mathbf{v})) \tag{9}$$

$$ReconfCost(\mathbf{v}, \tau) = \sum_{\mathbf{u} \in \mathbf{\Omega}} \gamma_{\mathbf{vu}} \;\; RCost(\tau, f(\mathbf{u})) \tag{10}$$

$$P(\mathbf{v}) = \left(\prod_{c \in \mathbf{v}} \frac{\lambda_c}{\lambda_c + \mu_c} \right) \left(\prod_{c \notin \mathbf{v}} \frac{\mu_c}{\lambda_c + \mu_c} \right) \tag{11}$$

$P(\mathbf{v})$ is the probability that the network is in connection state $\mathbf{v}$, $HopCost(\mathbf{v}, \tau)$ is the hop cost and $ReconfCost(\mathbf{v}, \tau)$ is the reconfiguration cost that the network incurs when at state $\mathbf{v}$ and template τ, $RCost(\tau_i, \tau_j)$ is the cost to reconfigure from template τ_i to template τ_j, and $\gamma_{\mathbf{vu}}$ is the transition rate from state $\mathbf{v}$ to state $\mathbf{u}$ [5].

Our approach to obtaining a good configuration policy is described by the following heuristic.

[5] $\gamma_{\mathbf{vu}}$ is equal to λ_c or μ_c for some connection c, or zero if no single connect/disconnect request can take the connection state from $\mathbf{v}$ to $\mathbf{u}$.

Heuristic 1

1. *Optimal Policy for $\beta = 0$.* For each connection state $\mathbf{v}$ let τ be the template for which the hop cost of $\mathbf{v}$ is minimized. Set $f(\mathbf{v}) = \tau$.

2. *Local Improvement.* For each $\mathbf{v}$ consider all $\tau \in \mathcal{T}$ as possible candidates for $f(\mathbf{v})$. Let τ' be a template such that

$$\alpha HopCost(\mathbf{v}, \tau') + \beta ReconfCost(\mathbf{v}, \tau') = \min_{\tau \in \mathcal{T}} \{\alpha HopCost(\mathbf{v}, \tau) + \beta ReconfCost(\mathbf{v}, \tau)\}$$

   Set $f(\mathbf{v}) = \tau'$. Repeat for all $\mathbf{v}$ until no further cost reduction is possible.

3. *Template Removal.* For each $\tau \in \mathcal{T}$ do the following: for each $\mathbf{v}$ such that $f(\mathbf{v}) = \tau$, set $f(\mathbf{v}) = \tau' \in \mathcal{T} - \{\tau\}$ and τ' is selected as in Step 2. If the new policy incurs lower cost set $\mathcal{T} = \mathcal{T} - \{\tau\}$, otherwise restore the old policy. Repeat for the new $\mathcal{T}$ until no further cost reduction is possible.

After producing a policy optimal for $\beta = 0$, Step 2 of the heuristic goes through the state space and modifies the decisions at each state (using information only about the state and the transitions out of it) to improve the initial policy. Step 3 goes through the decision space and removes templates (i.e., the final policy does not consider them as decision alternatives [6]) if the cost of making a transition to these templates is high. The degree by which the final policy differs from the initial and intermediate policies depends on the relative importance of the hop and reconfiguration costs (the relative values of α and β).

6.1 Numerical Results

Heuristic 1 was applied to a network with $N = 5$ stations and $p = 2$ transceivers per station. Results for two sets of values for $\{\lambda_c\}$ and $\{\mu_c\}$ are presented in Tables 3 and 4. The costs incurred by the initial policy and the policies after Steps 2 and 3, as well as the number of templates active for the final policy are shown; the value of α was set to 100 and we varied the value of β. The costs for all policies were computed using (5) and expressions (8) - (11).

The costs presented in Tables 3–6 can be interpreted as follows. Since we have assumed unit distances among network stations, the hop cost $\mathcal{H}_z$ is a measure of the total number of hops ongoing connections are routed over. Also, according to (6), the reconfiguration cost $\mathcal{R}_z$ is the average number of transceivers retuned per unit of time. If we think of α as the average packet

[6]We say that a template τ is "active" if $\exists \mathbf{v} \in \mathbf{\Omega} : f(\mathbf{v}) = \tau$; otherwise, τ is "removed" in Step 3.

delay (propagation plus processing plus queueing) per hop, and of β as the extra delay per retuned transceiver introduced in a unit of time as a result of reconfiguration (e.g., by means of packet loss), then $\alpha\mathcal{H}_z + \beta\mathcal{R}_z$ is the average total delay in the network.

From the tables we observe that for a given value of $\beta > 0$, Steps 2 and 3 of Heuristic 1 improve on the cost of the policy produced by the previous step. As β increases the final policies incur higher hop cost and lower reconfiguration cost; this is desirable as the importance of the reconfiguration cost increases with β. Also, templates for which the reconfiguration cost is prohibitively high are not considered by the policies for high β values; when β exceeds a certain threshold the best policy is to choose one template and never reconfigure.

6.2 Further Improvement of the Final Policy

It is possible to further refine the final policy of Heuristic 1 to obtain a lower cost policy. This can be done, if there are at least two templates that have not been removed, by noting the following. Suppose the network is in state $\mathbf{v}$ and template $\tau = f(\mathbf{v})$ when an event causes a transition to state $\mathbf{u}$ for which $f(\mathbf{u}) = \tau' \neq \tau$. If $HopCost(\mathbf{u}, \tau) \leq HopCost(\mathbf{u}, \tau')$, it is better for the network to remain at template τ than to reconfigure to template τ'. Obviously, it will incur no greater hop cost in τ. But also, the reconfiguration cost will be decreased since the network will occur no cost for this state transition. A fourth Step can be introduced in Heuristic 1 that considers all states and active templates to set

$$d[(\mathbf{v}, \tau), e] \quad = \quad \tau \quad \text{if } HopCost(\mathbf{v_{next}}, \tau) \leq HopCost(\mathbf{v_{next}}, f(\mathbf{v_{next}})) \tag{12}$$

The new policy will incur lower cost than the policy at the end of Step 3. Unfortunately, this policy is not in the class of policies defined by (7), as its next template decisions are based on both the next connection state and the current template, and we do not yet have an efficient and accurate method for computing its cost.

7 Configuration Policies for Large Systems

As the number of states and alternatives per state grows exponentially with N and $\mid \mathcal{T} \mid$, Heuristic 1 becomes inefficient even for networks of moderate size since it operates on the whole state and decision spaces. We now propose a way to manage the state and decision space explosion.

Managing the Connection State Space. The first component of our approach deals with defining a set of "important" connection states. We, therefore, restrict our attention to a small

β	Policy for $\beta = 0$	Policy After Step 2			Policy After Step 3			Active
	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	Templates
0	668.03	6.68	2.90	668.03	6.68	2.90	668.03	12
5	682.52	6.68	2.09	678.47	6.68	2.09	678.47	12
10	697.01	6.68	2.08	688.89	6.68	2.08	688.89	12
20	725.99	6.69	2.02	709.05	6.85	0.77	700.77	3
40	783.95	6.89	0.40	704.84	6.93	0.17	699.34	4
50	812.93	6.91	0.22	701.84	6.92	0.17	700.66	5
80	899.87	6.91	0.21	707.85	6.93	0.16	705.38	4
100	957.83	6.91	0.21	712.04	6.93	0.16	708.51	3
110	986.81	6.91	0.21	714.13	7.01	0.08	709.86	2
150	1102.73	6.91	0.21	722.42	7.10	0.00	710.06	1

Table 3: Results for $N = 5$, $\alpha = 100$, $\lambda_c = 0.1$ and $\mu_c = 0.01 * c^2, c = 1, \ldots, 10$

β	Policy for $\beta = 0$	Policy After Step 2			Policy After Step 3			Active
	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	Templates
0	438.11	4.38	3.24	438.11	4.38	3.24	438.11	12
5	454.30	4.38	2.74	451.80	4.38	2.74	451.80	12
20	502.87	4.38	2.73	492.75	4.38	2.73	492.75	12
30	535.25	4.39	2.63	518.19	4.59	1.78	512.11	5
40	567.62	4.43	2.34	536.20	4.67	1.24	516.58	3
50	600.00	4.45	2.21	554.87	4.71	1.00	521.15	4
60	632.38	4.45	2.17	575.35	5.04	0.00	504.41	1

Table 4: Results for $N = 5$, $\alpha = 100$, $\lambda_c = 0.1, c = 1, \ldots, 10$ and $\mu_c = 0.0111, c = 1, \ldots, 5, \mu_c = 0.9, c = 6, \ldots, 10$

subset, $\mathbf{\Omega}_\mathcal{P}$, of the connection state space. To this end we use algorithm ORDER-II [10] to efficiently enumerate the most probable connection states until a desirable degree, $\mathcal{P}, 0 < \mathcal{P} \leq 1$, of coverage of the state space, is obtained. The main justification for doing this lies in the fact that the network will be operating in one of the "important" states most of the time. In addition, the number of these states will in general be a very small fraction of the total number of states.

Managing the Decision Space. Secondly, we only consider a small number, M, of templates. These templates may be selected randomly. However, since we are interested in minimizing the cost the network will occur while in the connection states in $\Omega_\mathcal{P}$, we can select a set of templates that optimize the hop cost for these states as follows: (a) Partition $\mathbf{\Omega}_\mathcal{P}$ in M sets $\zeta_1, \ldots, \zeta_M$, and (b) for each set ζ_k find a template τ_k that maximizes the one hop traffic for the connection states in the set. Finding such a template is similar to the Connectivity Problem in [4], a transportation problem that can be solved using a specialized version of the Simplex algorithm.

Heuristic 2 describes our approach to managing the large state and decision spaces. By adjusting the values of $\mathcal{P}$ and M we can trade the quality of the final policy for speed.

Heuristic 2

1. Given $\mathcal{P}$, use ORDER-II [10] to produce $\mathbf{\Omega}_\mathcal{P}$.
2. Given $\mathbf{\Omega}_\mathcal{P}$ and M obtain a set of templates, $\mathcal{T}$, such that $\mid \mathcal{T} \mid = M$.
3. Apply Heuristic 1 to obtain $f(\mathbf{v}) \in \mathcal{T}$ for all states $\mathbf{v} \in \mathbf{\Omega}_\mathcal{P}$.
4. For each $\mathbf{v}$ and $\tau \in \mathcal{T}$, if event e takes the network to $\mathbf{u} \notin \mathbf{\Omega}_\mathcal{P}$, set $d[(\mathbf{v}, \tau), e] = \tau$.

Heuristic 2 optimizes the decisions for the states in $\mathbf{\Omega}_\mathcal{P}$ in which the network will be operating most of the time. In addition, Step 4 ensures that when the network makes a transition to a connection state not in $\mathbf{\Omega}_\mathcal{P}$ the decision is not to reconfigure, and no reconfiguration cost is incurred. The hop cost experienced while in states not in $\mathbf{\Omega}_\mathcal{P}$ is not expected to constitute a significant part of the total cost, as the network will spend only a small amount of time in these states. An upper bound on this extra cost (not included in the cost of the policy produced in Step 3) is $(1 - \mathcal{P})HopCost_{max}$, where $HopCost_{max}$ is the highest hop cost incurred by any state.

7.1 Numerical Results

In this section we apply Heuristic 2 to a network with $N = 16, p = 2$ and traffic parameters as in Tables 5 and 6. Using algorithm ORDER-II we obtain a 90% coverage of the state space by considering the 2047 most probable connection states, only a tiny fraction of the total number of states, which is equal to 2^{120}. The results presented in the two Tables are for $M = 15$, and two different sets of templates; for the first set the templates were chosen randomly, while for the second they were selected so as to minimize the hop cost of the connection states in $\Omega_{\mathcal{P}}$. Again, α was fixed at 100 and only the value of β was varied.

Regarding the properties of the policies produced as the value of β increases, we can make observations similar to the ones for tables 3 and 4. In addition, we note how the particular set of templates affects the quality of the policies. A comparison of Tables 5 and 6 reveals that the the policies for the second set of templates outperform the corresponding policies for the first set. Although when operating on the second set of templates the network incurs slightly higher reconfiguration cost, the lower hop cost more than makes up for the difference.

8 Concluding Remarks

We have considered multichannel multihop networks with stations equipped with a small number of tunable transceivers, and we have studied the problem of updating the network connectivity in response to changes in the traffic pattern. The problem has been formulated as a Markov Decision Process. Two costs have been considered: the fraction of packets lost as the network reconfigures from one interconnection pattern to another, and the distance that connections are routed over. Associated with each state transition in our model, is a decision to reconfigure the network, defining a configuration policy. Although an algorithm to obtain the optimal configuration policy exists, it can not be applied to networks of practical interest due to the state and decision space explosion. We have used this algorithm to identify the properties of the optimal policy, based on which we have developed heuristics to obtain policies that make decisions similar to the decisions of the optimal policy.

β	Policy for $\beta = 0$	Policy After Step 2			Policy After Step 3			# of Active
	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	Templates
0	1790.03	17.90	1.22	1790.03	17.90	1.22	1790.03	15
50	1850.83	17.90	1.18	1848.71	17.90	1.16	1848.42	14
100	1911.72	17.90	1.18	1907.47	17.92	1.14	1905.51	13
150	1972.61	17.90	1.18	1966.23	17.95	1.11	1961.23	12
200	2033.50	17.90	1.18	2024.99	18.16	0.98	2011.67	8
250	2094.38	17.90	1.18	2083.76	18.24	0.94	2058.79	7
300	2155.27	17.90	1.17	2142.17	19.01	0.53	2060.57	6
500	2398.83	17.91	1.17	2374.31	20.37	0.07	2069.90	2
600	2520.60	17.92	1.16	2488.22	20.71	0.00	2071.31	1

Table 5: Results for $N = 16$, $\mathcal{P} = 0.9$, $M = 15$, $\alpha = 100$, $\lambda_c = 0.194, \mu_c = 0.1, c = 1, \ldots, 11$ and $\lambda_c = 0.1, \mu_c = 99.9, c = 12, \ldots, 120$ (first set of templates)

β	Policy for $\beta = 0$	Policy After Step 2			Policy After Step 3			Active
	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	$\mathcal{H}_z$	$\mathcal{R}_z$	$\alpha\mathcal{H}_z + \beta\mathcal{R}_z$	Templates
0	1222.12	12.22	1.46	1222.12	12.22	1.46	1222.12	15
50	1294.68	12.22	1.41	1292.54	12.22	1.40	1292.51	14
100	1367.44	12.22	1.41	1363.16	12.28	1.33	1361.43	11
150	1440.20	12.22	1.41	1433.78	12.32	1.30	1427.64	9
200	1512.95	12.22	1.41	1504.38	12.37	1.27	1490.81	7
250	1585.71	12.22	1.41	1575.00	12.55	1.19	1553.03	6
300	1685.47	12.22	1.41	1645.66	13.03	0.96	1591.95	4
500	1949.50	12.26	1.39	1921.24	15.98	0.00	1597.85	1

Table 6: Results for $N = 16$, $\mathcal{P} = 0.9$, $M = 15$, $\alpha = 100$, $\lambda_c = 0.194, \mu_c = 0.1, c = 1, \ldots, 11$ and $\lambda_c = 0.1, \mu_c = 99.9, c = 12, \ldots, 120$ (second set of templates)

References

[1] A. S. Acampora. A multichannel multihop multihop local lightwave network. In *Proceedings of GLOBECOM '87*, pages 1459–1467. IEEE, November 1987.

[2] B. Mukherjee. WDM-Based local lightwave networks Part II: Multihop systems. *IEEE Network Magazine*, pages 20–32, July 1992.

[3] J. A. Bannister, L. Fratta, and M. Gerla. Topological design of the wavelength-division optical network. In *Proceedings of INFOCOM '90*. IEEE, 1990.

[4] J-F. P. Labourdette and A. S. Acampora. Logically rearrangeable multihop lightwave networks. *IEEE Transactions on Communications*, 39(8):1223–1230, August 1991.

[5] C. A. Brackett. Dense wavelength division multiplexing networks: Principles and applications. *IEEE Journal on Selected Areas in Communications*, SAC-8(6):948–964, August 1990.

[6] J-F. P. Labourdette, A. S. Acampora, and G. W. Hart. Reconfiguration algorithms for rearrangable lightwave networks. In *Proceedings of INFOCOM '92*. IEEE, May 1992.

[7] J. Auerbach and R. Pankaj. Use of delegated tuning and forwarding in WDMA networks. Technical Report RC 16964, IBM Research Report, 1991.

[8] J-F. P. Labourdette and A. S. Acampora. Logical clustering for the optimization and analysis of a rearrangeable distributed atm switch. In *Proceedings of INFOCOM '93*. IEEE, March 1993.

[9] R. A. Howard. *Dynamic Programming and Markov Processes*. M.I.T. Press, Cambridge, 1960.

[10] Y. F. Lam and V. O. K. Li. An improved algorithm for performance analysis of networks with unreliable components. *IEEE Transactions on Communications*, COM-34(5):496–497, May 1986.

A Howard's Policy-Iteration Algorithm

Consider an ergodic, continuous-time, discrete-space Markov process with rewards. Let K be the total number of states of the process, and let l_i be the number of alternatives when the system is at state i. We call θ_{ij}^m the transition rate from state i to state j under alternative $m, 1 \leq m \leq l_i$, and r_{ij}^m the reward (or cost) of making a transition from state i to state j under alternative m; similarly, r_{ii}^m is the reward earned (or cost incurred) per unit time by the system while at state i. Howard's algorithm [9] can be used to develop a policy, i.e., a set of alternatives, one for each state, that maximizes the long term rewards (or minimizes the cost) of the system.

Initially an arbitrary policy is specified from which all state transition rates are determined. The first stage of Howard's policy-iteration algorithm, the *Value-Determination Operation*, uses θ_{ij} and Q_i to solve the set of equations

$$A \;=\; Q_i \;+\; \sum_{j=1}^{K} \theta_{ij}\; V_j, \qquad i = 1, \ldots, K \tag{13}$$

V_j is a measure of the cost of occupying state j, A is a relative measure of the long term average system cost, and Q_i is the expected immediate reward for state i, given as $Q_i = r_{ii} + \sum_{j \neq i} \theta_{ij} r_{ij}$; there is no need for a superscript m in these expression, because the establishment of a policy has determined the rates and rewards for the system. In the second stage of Howard's algorithm, the *Policy-Improvement Routine*, we use the V's obtained from the first stage and obtain a new configuration policy, i.e., a new alternative m' for each state, and therefore new state transition rates θ_{ij}, such that

$$Q_i^{m'} \;+\; \sum_{j=1}^{K} \theta_{ij}^{m'}\; V_j \;=\; \min_{m=1,\ldots,l_i} \left\{ Q_i^m \;+\; \sum_{j=1}^{K} \theta_{ij}^m\; V_j \right\} \qquad i = 1, \ldots, K \tag{14}$$

The new values for θ_{ij} are used in the next iteration of the algorithm. The two stages are repeated until the policy remains unchanged for successive iterations. At this point the algorithm has converged and the policy is optimal with respect to minimizing A. Note that Howard's algorithm is guaranteed to converge [9].

Expression (13) requires the solution of a set of K linear equations, while expression (14) considers l_i alternatives per state. For our model, a state is described by $(\mathbf{v}, \tau), \mathbf{v} \in \mathbf{\Omega}, \tau \in \mathcal{T}$; since $\mid \mathbf{\Omega} \mid = 2^N$, then $K = 2^N \mid \mathcal{T} \mid$. There are $\frac{N(N-1)}{2}$ valid events (connect/disconnect requests) per state, and for each event any template can be chosen as τ_{next}, resulting in $\mid \mathcal{T} \mid^{\frac{N(N-1)}{2}}$ alternatives per state. Thus, the complexity of the algorithm is determined by (13) and (14) as $O\left(\left(2^N \mid \mathcal{T} \mid\right)^3 \;+\; \left(2^N \mid \mathcal{T} \mid\right) \mid \mathcal{T} \mid^{\frac{N(N-1)}{2}}\right)$ *per iteration*, and is impractical to apply in this form even for $N = 5$.