

# MöbiPack: Optimal Hitless SONET Defragmentation in Near-Optimal Cost

Swarup Acharya    Bhawna Gupta    Pankaj Risbood    Anurag Srivastava

Network Software Research Department, Optical Networking Division

Bell Laboratories, Lucent Technologies, Inc.

{acharya,bhawna,risbood,anurag}@research.bell-labs.com

**Abstract**— We study the problem of bandwidth fragmentation in links that comprise rings and meshes in SONET networks. Fragmentation is a serious challenge for network operators since it creates “holes” in the transport pipe causing new demands to be rejected, in spite of sufficient bandwidth being available. Unlike the well-studied, general fragmentation problem, link defragmentation is “hard” and novel due to some unique constraints imposed by the SONET standard. Since a defragmentation operation typically occurs on a network carrying *live* traffic, in addition to the “quality” of the output, any link defragmentation algorithm has to avoid traffic hit and also optimize the “cost” of reorganizing circuits. We propose an algorithm called *MöbiPack* that is optimal in its defragmentation quality. Moreover, we demonstrate via extensive simulations, that it also achieves its goal in a hitless manner with only marginal additional cost over the optimal, making it extremely attractive to use in practice.

## I. INTRODUCTION

Telecom service providers have been lately facing a serious crunch in their capital budgets. As the data traffic continues to grow and their capital investments fail to keep up (and, even shrink), providers are increasingly seeking network engineering tools to extract higher utilization from their existing infrastructure. The SONET (or Synchronous Optical Networking) infrastructure is a good candidate for engineering since it is the most widespread technology for optical transport<sup>1</sup>.

SONET uses *time division multiplexing* (TDM) to efficiently subdivide the bandwidth of an optical channel into smaller usable fragments called “time slots”. In such TDM systems, bandwidth “fragmentation” is a serious problem. Bandwidth gets fragmented, as circuits are added and deleted, leaving behind “holes” in the transport pipe. Over time, the network efficiency is lowered as the free bandwidth is split into multiple non-contiguous slots, thereby causing new demands to be rejected. Thus, minimizing fragmentation is one of the key engineering challenges in improving network efficiency.

Fragmentation is an old, well-studied problem going back over 30 years [1]. In the very early computing systems prior to the use of paging, the system memory got fragmented as different-sized programs got loaded and exited the system. The most common example of fragmentation is seen on hard disks and tools to perform disk defragmentation are widely available. Thus, it may come as a surprise that there exists

an instance of the fragmentation problem that has not already been addressed in the prior literature. It turns out that the bandwidth fragmentation problem in SONET is indeed one such case. As we highlight in the paper, the novelty arises due to some unique constraints imposed by the SONET standard and key operational requirements on the field.

In this paper, we explore one specific instance of bandwidth fragmentation, namely, the *Link Defragmentation (LDF)* problem, that arises in links that make up SONET ring and mesh networks in the transport core. Consider a simple case of an *LDF* operation. Figure 1(a) represents the frame for an STS-48 link between two nodes with each slot representing bandwidth for STS-1 circuit. The set of circuits currently on the link are marked — for example, STS-3c circuit A on slots <1-3> and STS-12c circuit C occupying slots <13-24>. If a new request is received for another STS-12c circuit it will be rejected due to lack of 12 contiguous free slots on this link.

However, there does exist sufficient bandwidth — it is simply fragmented. Consider an alternate defragmented slot assignment in Figure 1(b) for the same set of demands. The circuits are now “pushed to the wall” that we shall refer to as the *PW* layout. In this layout, a new request for a STS-12c request would be satisfied on slots <37-48>. Thus, defragmentation helps satisfy demands that would otherwise be rejected. In the long term, these improvements in the network utilization can lead to significantly lower operating costs.

Since the defragmentation operation is performed on a live network, a key operational requirement is that *it should not cause any service disruption*. In other words, the defragmentation should be *hitless* and not require tearing circuits down. Thus, in the above example, a link defragmentation algorithm (*LDA*) has to also output a step-by-step transition sequence to migrate circuits from their current slot to a new one. Thus the *PW* layout in Figure 1(b) can be reached in seven steps, namely, circuits D, E, F, G, H, I, J and K have to move to new slots in that order. To enable the transition to be hitless, a slot has to be free before a circuit can be moved there. Thus, D has to move to slot <4-6> first before K can take its place. Clearly, without such a sequence, any *LDA* would have little merit in practice.

While the *PW* layout is an obvious one that works in other environments, it is not necessarily the most appropriate in the SONET context. Consider the layout in Figure 1(c),

<sup>1</sup>While the paper is written in the context of SONET, this work is equally valid for SDH (Synchronous Digital Hierarchy), the prevalent transport standard outside the USA.

computed by an *LDA* called *MöbiPack*. We refer to it as the *MP* layout and even though it is not immediately obvious, *MP* is actually *optimally* packed (as is the *PW* layout). But, more importantly, *MP* can be reached from the original layout by moving *only four circuits (A, B, G and H)*! This makes it *cheaper* by three moves compared to *PW*! Thus, a non-obvious solution such as *MP* is clearly preferred to the more logical *PW* layouts.

In fact, in some cases it may not be possible to reach the optimal packed layout at all without causing service disruption. Consider Figure 1(d) that we refer to as *optimal design (OD)*. This is the optimal way to design the layout if all demands were known *a priori* — assigning slots in the decreasing order of granularity. However, one can see that it is not possible to move from the original layout to *OD* without either deleting circuits first and then adding them later or, moving some circuits more than once. As we point out in the next section, both of these options may be unacceptable in many environments.

In the rest of the paper, we will explore these requirements of optimality and circuit migration. In particular, we will highlight why a solution such as *MP* is indeed acceptable and how it may be derived. Suffice it to say that on a SONET infrastructure carrying live traffic, minimizing the cost of defragmentation, i.e., the number of circuit moves, is as critical as the quality of defragmentation. In fact, this requirement is one of the key factors that makes this problem so unique.

*LDF* has come to the fore with the advent of grooming cross-connect switches that enable an incoming signal passing through a node to be switched to any other time slot (Time Slot Interchange or TSI) [2]. While SONET networks have been traditionally engineered as rings, grooming switches have enabled mesh topologies consisting of these switches interconnected by high-speed line systems. Since TSI meshes (or, rings) enable a circuit to take any slot along its path, each link can be defragmented independently to locally improve bandwidth efficiency.

#### A. Contributions and Outline

To the best of our knowledge this is the first paper that addresses the *LDF* problem in SONET/SDH links, incorporating all the constraints as imposed by SONET standards listed in Section II-A. We formally define the *LDF* problem and provide insights into the unique nature of this problem. We present a very efficient *LDA* called *MöbiPack* that provides optimal packing with near-optimal defragmentation cost. We posit that achieving optimal packing in optimal cost is a likely hard problem to solve. The key contribution is that *MöbiPack*, with adequate support from the network element, can perform hitless defragmentation and thus, is extremely well suited as a practical solution.

The rest of the paper is as follows. In the next section, we define the *LDF* problem and list some of the constraints imposed by the SONET standard that make this problem unique. Following that, we outline the previous research in the area. In Section IV, we address the requirements in meeting the

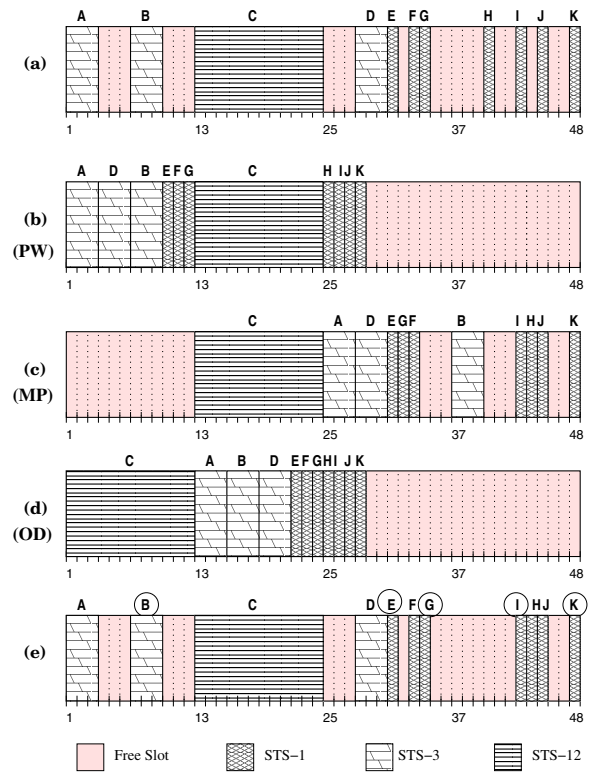


Fig. 1. Link Defragmentation on STS-48 frame

two optimality metrics, namely, optimal packing and optimal migration. In Section V, we describe the *MöbiPack* algorithm. Following that, we present some performance experiments and finally, conclude.

## II. LINK DEFRAGMENTATION PROBLEM

### A. SONET/SDH Ground Rules

Before we formally describe the *LDF* problem, we highlight some of the constraints that any solution must handle. These are primarily derived from the SONET/SDH standard and from operational requirements on the field.

The requirements that arise from the SONET/SDH specifications [3] include:

- *Demand Granularity*: We assume that the circuits follow SONET STS hierarchy i.e., they are one of  $STS-1, 3c, 12c, 48c, 192c$  granularity with each  $STS-nc$  carrying  $n$  times the capacity of a  $STS-1$  signal. We also assume *contiguous concatenation*, namely that all the slots required to satisfy a demand be contiguously assigned<sup>2</sup>.

- *Alignment property*: SONET also imposes an alignment property, namely that a  $STS-nc$  circuit can only use slots  $s$  to  $(s+n-1)$  such that  $s \equiv 1 \pmod{n}$ . In other words, a  $STS-3c$

<sup>2</sup>A new protocol called *Virtual Concatenation* proposed for Ethernet data traffic no longer enforces this constraint. However, while it will obviate fragmentation for data traffic, the problem still remains for voice traffic on all the legacy equipment in the field today.

circuit can be provisioned starting only at slots 1,4,7,10,... and a STS-12c only at 1,13,25 and so on. Surprisingly, in spite of its criticality, this alignment property has never been considered in the vast body of prior research on SONET network design [4], [5], [6].

- *Time Slot Interchange (TSI)*: We assume that the link being defragmented forms part of a TSI ring or, mesh network consisting grooming cross-connects. This essentially means that time slot assignment can be done independently on every link or span along the path.

- *Bridge-n-Roll*: Most SONET/SDH network elements today support what is called a Bridge-n-Roll functionality [7], that enables the circuit to be first replicated or *bridged* on to a new slot and then *rolled* over. A bridge-n-roll operation does cause some disruption but it is typically less than the 50ms SONET restoration time. Thus, even though there is some data loss, it is within the acceptable limits for most applications. Therefore, for all practical purposes the bridge-n-roll operation can be considered “hitless”. We assume support for such functionality from the underlying network elements.

### B. Unique LDF Requirements

In this section, we list some of the necessary criteria that any LDA has to meet in order to be effective in an operational network. It is these requirements in conjunction with the SONET constraints that makes this problem fundamentally more challenging.

- *Optimal Defragmentation*: The output of any LDA should be a link that is optimally “packed” at the end of the defragmentation operation. We will formally define optimality in Section IV-A. However, suffice it to say that the algorithm should produce the best possible output one can achieve.

- *Hitless Defragmentation*: As motivated earlier, any LDA needs to provide a migration sequence for circuits to move to new slots. In conjunction with the availability of Bridge-n-Roll support in the network element, this enables the defragmentation operation to be hitless.

One may argue that disk defragmentation is also required to follow a “hitless” analogy, i.e., it is not possible for a disk defragmentation solution to delete some files completely and later reinstate them back. However, there is a fundamental difference — a file on disk can be moved one page at a time to its new location. On the other hand, *all slots* of a SONET circuit have to be Bridge-n-Rolled together as one unit. This makes the problem significantly more complex since sufficient number of *contiguous slots* have to be free at the destination for a hitless transition.

- *Minimize Migration Cost*: Any LDA will be judged on two metrics — a) whether it can optimally defragment a link and b) the *cost* to migrate to that optimal state. In this context, the cost of defragmentation is the number of circuit moves in

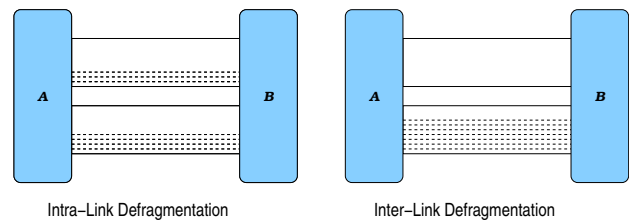


Fig. 2. Inter- and Intra-Link Defragmentation

terms of Bridge-n-Roll operations to get to the optimal layout. Minimizing this number is critical for many reasons. Firstly, a failure in the Bridge-n-Roll process can bring down the circuit being rolled. Secondly and more importantly, a Bridge-n-Roll operation temporarily uses up twice the circuit bandwidth (at the old and new slot location). Failure at this time can reduce the available bandwidth of the link and potentially impact restoration capability if there is a second failure. Thus, in Figure 1, even though both the PW and MP layouts were optimally packed, MP is clearly the preferred choice due to four fewer circuits requiring to move.

It is this migration cost that separates this problem from prior work on memory and disk defragmentation where the only metric of interest was to achieve optimality in space [1]. In fact, since there are potentially multiple optimal layouts (e.g., PW, MP, OD), the requirement is to find the layout that imposes the least migration cost. The MöbiPack algorithm proposed in this paper attempts to do exactly that.

- *Intra- and Inter- Link defragmentation*: While defragmenting a link is a useful operation, the more likely scenario is to defragment an entire trunk of multiple parallel links that connect two nodes in a network. Figure 2 provides a schematic of *intra- and inter-link defragmentation*. In the first case, the existing circuits (shown as dotted lines) are moved to slots at one end of each link while in the latter case they are moved all the way into one of the links. Inter-link defragmentation is a useful tool to clear one or more links in order to bring down network elements for maintenance. Clearly, a single algorithm that handles both these cases would be preferable.

- *Nailed Down Circuits*: Service providers often have high priority circuits whose disruption imposes serious penalties. Thus, any algorithm should also be able to account for “nailed down” circuits that need to be left untouched in the defragmentation process. Note that the presence of nailed down circuits makes both the PW and OD approaches ineffective. Figure 1(e) shows the optimal defragmentation of the STS-48 frame in Figure 1(a), if the five circled demands were nailed down. Sections IV and V describe how this can be achieved.

- *In-space Defragmentation*: The defragmentation operation does not have access to additional *scratch* space as a temporary

staging ground for circuits prior to moving to their final destination.

- *Move-only-once Requirement*: A bridge-n-roll operation does causes some loss of data (however, marginal) that an operator would like to avoid. Thus, from the operational viewpoint a solution that moves a circuit only once during a defragmentation operation would be preferred.

- *Bi-directional Circuits*: In this paper, we focus only on bi-directional circuits. The defragmentation problem with uni-directional circuits will be addressed elsewhere.

### C. $\mathcal{LDF}$ Problem Definition

We now formally define the Link Defragmentation Problem as follows:

**$\mathcal{LDF}$  Definition:** Given a link of capacity STS- $n$  (having  $n$  STS-1 slots), a set of bi-directional circuits  $D = \{d_1, d_2, \dots, d_k\}$  and a map  $\psi = \{d_i \rightarrow \{t_i, g_i\}\}$  where  $t_i$  is slot number at which demand  $d_i$  of granularity  $g_i$  starts.

The  $\mathcal{LDF}$  problem is to find a new map  $\psi' = \{d_i \rightarrow \{t'_i, g_i\}\}$  such that:

- The *alignment property* (Section II-A) is satisfied, and,
- The link is optimally packed as defined in Section IV-A (*Optimal Packing*), and,
- A circuit migration sequence is produced to go from  $\psi$  to  $\psi'$  in the minimum number of moves (*Optimal Migration*). ■

Clearly, a good  $\mathcal{LDA}$  aims to achieve optimality on two fronts — in quality of the packing and in the length of the migration sequence. We will refer to these two requirements as *Optimal Packing* and *Optimal Migration* respectively.

## III. RELATED WORK

Fragmentation is a well studied problem going back to the sixties. [1] explores fragmentation extensively in the context of memory and storage allocation. The problem exhibits itself when different sized chunks of memory are allocated and de-allocated from a common memory area. A two-pronged approach has usually been taken to tackle this problem — a pro-active *avoidance* approach and a *re-active* garbage collection with memory compaction. The fragmentation avoidance schemes include strategies like *Best Fit* and *First Fit* at the time of allocation. Most systems today use one of these avoidance strategies in conjunction with a garbage collection/replacement strategy such as LRU (Least Recently Used).

Fragmentation avoidance techniques are also applicable in the context of  $\mathcal{LDF}$ . There is a vast body of prior research in this area, involving modeling of request arrivals to devise optimal slot assignment strategies with the goal of reducing the rejection rate [8], [9].

To the best of our knowledge this is the first work that presents a reactive approach to bandwidth fragmentation on links and provides a very efficient defragmentation algorithm that accounts for the constraints imposed by the SONET/SDH

standard. In a prior work [10], we addressed the analogous problem of bandwidth fragmentation on SONET BLSR rings.

## IV. OPTIMAL $\mathcal{LDF}$ SOLUTION

In this section, we explore key characteristic properties of any optimal solution. As motivated earlier, one can aim for optimality on both the packing and movement fronts. We will address both these issues in order.

### A. *Optimal Packing*

We define optimality of packing for link defragmentation as follows:

**Optimal Packing Definition:** A SONET link or, a set of links is optimally packed if the layout of demands on the link(s) is such that the link(s) can be filled completely to capacity with the fewest number of new circuit demands.

Based on this definition, one needs to determine the fewest set of demands that would completely fill up a link. Stated differently, any  $\mathcal{LDA}$  needs to create free space to fit each of these demands. We refer to this set of demands and therefore the layout as the *optimal layout of free space* ( $\mathcal{OLS}$ ). In this section, we state how one may determine this  $\mathcal{OLS}$  set. We will denote the  $\mathcal{OLS}$  as an ordered set of numbers, one corresponding to each SONET STS rate in increasing order of granularity. Thus, a link with  $\mathcal{OLS}$  of  $\{2, 2, 1\}$ , implies that on defragmentation, this link can satisfy two STS-1, two STS-3c and one STS-12c circuits.  $O_j$  refers to the value of  $j^{th}$  granularity in the  $\mathcal{OLS}$  set.

We state below a general formula to calculate the  $\mathcal{OLS}$ . However, if a link has no nailed down circuits, calculating the  $\mathcal{OLS}$  is reasonably straightforward. First determine the total number of free STS-1 slots, say  $S$ . Divide  $S$  by the highest SONET granularity possible on the link that is less than  $S$ , to get the number of circuits of that granularity in the  $\mathcal{OLS}$  set. Repeat this step with the remaining free slots with the next lower granularity and so on. Thus, with 20 free slots in Figure 1(a), the  $\mathcal{OLS}$  set would be 1 STS-12c, 2 STS-3c (using remainder 8) and 2 STS-1 (using remainder 2).

Once the  $\mathcal{OLS}$  is known, it can be used to verify if a link is indeed optimally packed. Thus, in the example in Figure 1(a), since both the  $\mathcal{PW}$  and  $\mathcal{MP}$  packing can satisfy the  $\mathcal{OLS}$  of  $\{2, 2, 1\}$ , both are optimally packed.

If some circuits are nailed down i.e., cannot be moved, a more general formula for the  $\mathcal{OLS}$  is required. In this case, availability of twelve free STS-1 slots does not imply that they can always be made contiguous to free a STS-12c slot. For example, if all the circuits in Figure 1(a) were nailed down, the same 20 free slots would not be able to produce a STS-12c free slot. Thus, the context of the free slots should also be noted. The formula below computes the  $\mathcal{OLS}$  for a link taking into account the nailed down circuits. This formula can also be used for links of different sizes. We present the formula below and then highlight the non-obvious steps.

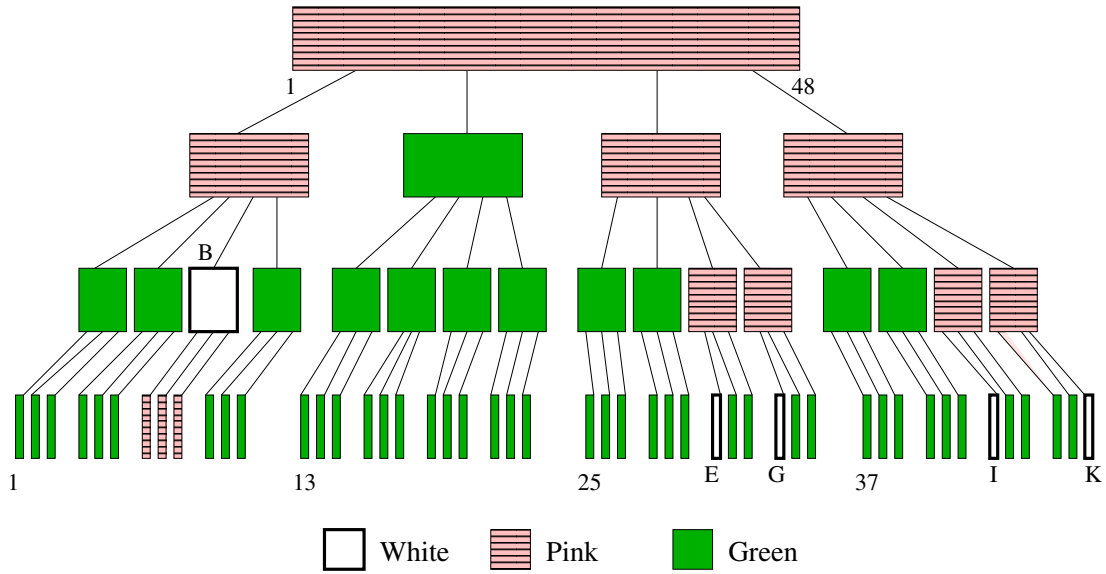


Fig. 3. Effect of Nailed Down Demands

Let,

- $G =$  Ordered set of SONET circuit rates STS- $x$  in increasing granularity order
- $g_i =$  SONET circuit rate with index  $i$  in  $G$
- $h =$  Total number of SONET circuit rates
- $g_h =$  Highest granularity SONET circuit rate
- $D_i =$  Number of movable demands of granularity  $g_i$
- $R_{x,y} = g_x/g_y$
- $U_i =$  Maximum number of  $g_i$  rate circuits that the link can support after accounting for the nailed-down circuits
- $O_i =$  The number of  $g_i$  circuits in  $OLS$
- $F =$  Total number of free STS-1 slots on link

Then,

$$O_h = \min(F/g_h, U_h - D_h) \quad (1)$$

$$A_h = F - O_h * G_h \quad (2)$$

$$O_i = \min(A_{i+1}/g_i, U_i - D_i - \sum_{x=i+1}^h (O_x + D_x) * R_{x,i}) \quad \forall i < h \quad (3)$$

$$A_i = A_{i+1} - O_i * g_i \quad \forall i < h \quad (4)$$

Consider the OC-48 link in Figure 1(a). Let STS-3c demand B and four STS-1 demands E, G, I and K be nailed down and cannot be used for defragmentation. We use this example to calculate the  $OLS$  using the formula. The set  $G$  is  $\{1, 3, 12\}$ ,  $D$  is  $\{1, 2, 3\}$  and  $F$  is 20. To compute the set  $U$  consider Figure 3<sup>3</sup>. Each level of the tree shows the slots at one STS granularity starting with OC-48 on top and reducing top-down. The children of a node are the slots of next lower granularity that can be accommodated in that slot.

<sup>3</sup>This figure is best viewed in color. For grayscale views, the legend may be used to identify the colored slots.

Clearly, if a demand uses a node in this tree, all its children are not usable. Moreover, no parent of this node up to the root, can be used by a demand. For example, the presence of an STS-3c demand A in slot 1, precludes any STS-1s in slots  $< 1 - 3 >$  or, a STS-12c or a STS-48c starting at slot 1. The white nodes in the tree show the position of the nailed down demands from Figure 1(a). The pink nodes in the tree show the nodes that cannot support demands due to the effect of these demands. Thus, all the remaining nodes, that are marked green, can support movable demands and are the ones used for defragmentation. These form the set  $U$  and for this example, the set value is  $\{41, 11, 1\}$ . This sets the upper bound on the maximum number of demands of each granularity that may be provisioned on the link, after accounting for all the nailed down demands.

The  $OLS$  is calculated as follows. As noted earlier, a demand of granularity  $g_i$ , in addition to using the slots of granularity  $g_i$ , also eats up slots of granularity lower than  $g_i$ . Therefore while calculating the free available slots of a granularity  $g_i$ , the slots used up due to demands of granularity higher than  $g_i$  also have to be considered. Therefore the upper limit on the number of free available slots of granularity  $g_i$  that can be created in the link is  $U_i - D_i - \sum_{x=i+1}^h (O_x + D_x) * R_{x,i}$ . Clearly for the highest granularity, this upper limit is  $U_h - D_h$ . Therefore the number of free slots of highest granularity  $O_h$  that can be created is  $F/g_h$  bounded by  $U_h - D_h$  as captured by Eqn(1).  $A_i$  represents the number of free STS-1 slots remaining after removing  $O_i$  from the current free slots. Eqn(2) captures this for the highest granularity and Eqn(4) for all other granularities. The free slots that can be created for any other granularity are computed by dividing the remaining free STS-1 slots by the granularity subject to the upper bound discussed above and is captured by Eqn(3). Using these equations, the  $OLS$  computes to  $\{5, 5, 0\}$ . Figure 1(e)

represents the optimal packing with nailed down circuits. As shown, *only one* demand  $H$  needs to move to achieve the optimal packing!

In case of multiple links, the  $OLS$  can be computed by independently determining the sets  $D$  and  $U$  for each link as described above and then adding them up. Total free slots can be computed across all links and the resulting  $OLS$  gives the optimal defragmentation across all links.

### B. Optimal Migration Cost

A key operational requirement in  $LDF$  is to minimize the number of bridge-n-roll operations required to defragment a link. Unfortunately, determining the optimal layout that requires the minimal number of bridge-n-roll operations to reach, is not a straightforward problem. In fact, we posit the following hardness hypothesis:

**Optimal Migration Hypothesis:** *Determining the optimal circuit migration sequence to optimally defragment a link is NP-hard.*

In other words, packing a link optimally in optimal number of moves is likely a hard problem to solve. While we believe that this hypothesis is true, there is no obvious prior work that can be leveraged to justify this claim one way or the other. However, it is well known that the discontinuous property imposed by the alignment constraint is a very difficult optimization constraint to model. Thus, we propose it as an open problem.

### C. Necessary Conditions for Optimal Movement

While the hardness of the theoretical problem may be unknown, there are some characteristics that can be proven about the shortest migration sequence. Indeed, these are necessary (but not sufficient) conditions required to achieve Optimal Migration. The formal proofs for these conditions are beyond the scope of this paper and are available in [11]. Suffice it to say, that each of them can be proved by contradiction.

Given a migration sequence where one or more of these conditions are not met, an alternate shorter migration sequence can be found that meets these conditions and achieves an equally good (or, better) packing.

These necessary conditions are as follows:

- **GoodMove Condition:** *The slots freed up by a circuit moving out (to a new slot) is taken up by a circuit of higher granularity, or, the space cleared is left empty.* This follows from the goal of defragmentation to create longer contiguous blocks of free slots. Thus, an STS-12c circuit never moves out to make space for a STS-3c circuit. If so, an alternate migration sequence would instead move the STS-3c circuit to the slot where the STS-12c circuit was destined and leave the STS-12c where it is. One can demonstrate that this alternate sequence would require one less move (the STS-12c remains put) and produce an equivalent packing. We refer to a move that follows this rule as a *GoodMove*.

- **Corollary:** *Every circuit moving out causes its neighbors to also move out so that enough contiguous free slots are formed to fit a new demand of granularity higher than all the circuits moving out.* This follows from the last requirement that every slot vacated is filled by a circuit of higher granularity. In other words, three neighboring STS-3c circuits will either all move out to create a hole of size STS-12 (or, more) or, none of them will.

- **MoveOnce Condition:** *A demand moves only once in the optimal migration sequence.* As in the other cases, one can prove this condition by contradiction. The condition has a key implication, namely that there cannot be a cyclic dependency in the transition sequence between any demand and the demand(s) that currently occupy the slots it is expected to move into.

## V. MöbiPack ALGORITHM

In this section, we highlight the *MöbiPack* algorithm. The algorithm aims to achieve optimal packing while attempting to minimize the number of circuit moves. While the likely hardness conjecture of  $LDF$  precludes delivering an optimal migration sequence in all cases, we show via extensive experiments (Section VII) that *MöbiPack* is indeed within a few extra moves of the optimal. We provide a general algorithm that also accounts for the cases wherein some demands may be nailed down. We first focus on defragmenting a single link (*Intra-link defragmentation*). In Section V-E, we highlight how one may extend this algorithm to defragment multiple links (*Inter-link defragmentation*).

The algorithm works in four phases:

- **Phase I [Initialization]:** Initialize appropriate data structures.
- **Phase II [ $OLS$  Calculation]:** Compute  $OLS$  for the link. This provides for each granularity  $g_i$ ,  $O_i$  number of slots to be freed to achieve optimal packing.
- **Phase III [Free Slot Selection]:** Identify requisite  $O_i$  slots for each granularity  $g_i$  and the corresponding circuits that require to move.
- **Phase IV [Migration Sequence]:** For these identified circuits, find new slots for them to move into. In addition, output the migration sequence so that the moves can be done in a bridge-n-roll manner.

### A. Phase I: Initialization

In this section, we describe some of the one-time steps to improve the efficiency of the *MöbiPack* algorithm. As pointed out in Section II-A, a SONET demand can be provisioned starting only at very specific slots. For example, a STS-12c demand can start only at slots 1,13,25 and so on. However, this mapping is not straightforward if there are nailed down circuits. For example, if there is a nailed down STS-1 circuit on slot 14, then slot 13 cannot host an STS-12c circuit. We use the following notation to denote this mapping for each granularity  $g_i$ :

$S_i =$  Set of valid starting slots for circuits of granularity  $g_i$

This set  $S_i$  can be created in one sweep of all the slots. The algorithm also uses a bit mask  $B_i$  corresponding to demands of granularity  $g_i$ . A bit is set corresponding to all slots used by that demand to indicate that no new demands of lower granularity can be assigned to that slot.

### B. Phase II: $\mathcal{OLS}$

The next phase involves computation of  $\mathcal{OLS}$  for the link based on the formula provided in Section IV-A. If present, any nailed down circuits are also accounted for.

### C. Phase III: Free Slot Selection

In this phase, for each circuit granularity, the algorithm identifies as many slots to free as dictated by the  $\mathcal{OLS}$  requirements for that granularity. Thus, for the example in Figure 1 with an  $\mathcal{OLS}$  of  $\{2,2,1\}$ , this would mean identifying sufficient numbers of circuits to move out of their current locations such that one STS-12c and two STS-3c sized free slots are created. As hypothesized earlier, the inherent hardness of the problem arises in this step – identifying which specific slot to “clear” such that the number of moves is minimized, does not admit an obvious answer. Thus, in this phase, *MöbiPack* uses a greedy strategy to identify slots to clear and the pseudocode is given in Figure 4.

*MöbiPack* uses a heuristic to choose slots that require the lowest up-front cost to clear (Step 5). For example, in order to create a free STS-12c slot in Figure 1(a), it chooses the first twelve slots. Doing so would require two circuit moves (A and B) as opposed to four each for the other possibilities, namely slots  $\langle 25-36 \rangle$  and  $\langle 37-48 \rangle$ . Slots  $\langle 12-23 \rangle$  are ignored since it is already occupied by a STS-12c and little is gained by moving it to free a slot of the same size (*GoodMove* condition).

Note that the circuits identified for moving (A and B) also need to find new slots to move into. These “spill over” circuits are accounted for by incrementing  $\mathcal{OLS}$  values for their respective granularities (Step 9). Thus, because of them four STS-3c slots have to be cleared now in the iteration for that granularity (2 originally from  $\mathcal{OLS}$  and two more for A and B).

The mask  $B_i$  for granularity  $g_i$  identifies the slots in which no circuit of granularity lower than  $g_i$  can be moved into. This ensures that circuits identified later do not invalidate the work done in previous iterations and in effect, fragment a higher granularity circuit that was cleared.

Since the algorithm uses a greedy approach, it does not guarantee optimal movement. The sub-optimality comes from the fact that local decision of marking circuits to move at one level does not take into account the number of circuits of lower granularity that will be required to move to create space for these marked circuits. For example, let there be two STS-12c slots, one with two STS-3c and one with three STS-1 circuit. The algorithm will prefer to clear the first since it will require

### MöbiPack Algorithm Phase III

- *Input:*
  - 1) STS- $N$  link with  $N$  slots.
  - 2) Circuit set  $D$  of  $G$  different granularities and demand to slot map  $\psi$ .
  - 3)  $\mathcal{OLS}$  set  $\{O_1, \dots, O_h\}$ . Let  $g_h$  be the highest granularity in the set.
- *Output:*
  - 1) List  $L$  of demands to be moved
  - 2) Bit mask  $B_i$  corresponding to each  $g_i \in G$
- *Algorithm:*
  - 1) Let  $B$  be a bit mask of size of  $N$  initialized to 0.
  - 2) Set  $i = h$ , the highest granularity in the  $\mathcal{OLS}$  set.
  - 3) For each circuit  $d \in D$  of granularity  $g_i$  or higher find  $\psi(d) = (t, g_r)$  and set all bits from  $t$  to  $t + g_r - 1$  as 1 in  $B$ .
  - 4) For all valid slots  $s$  in  $S_i$ 
    - a) Initialize new list  $T$ .
    - b) If bit corresponding to  $s$  in  $B$  is not set
      - i) Find the number of demands in slots from  $s$  to  $s + g_i - 1$ . These demands will have to be moved if these slots are chosen to be cleared.
      - ii) Add the count of these demands corresponding to  $s$  to a list  $T$
  - 5) Sort  $T$  and find first  $O_i$  elements. These are the  $O_i$  slots that will be cleared.
  - 6) Mark the demands currently using these slots and append to  $L$ .
  - 7) Set  $B_i = B$ .
  - 8) Set bits in  $B$  corresponding to the slots identified in Step 5 to 1.
  - 9) For each demand of granularity  $g_k$  identified to be moved in Step 6, increment  $O_k$  in  $\mathcal{OLS}$  by 1.
  - 10) Set  $i = i - 1$
  - 11) If  $g_i$  is 1 (i.e., STS-1), set all bits corresponding to slots that are in use and save mask at  $B_1$  and exit Phase II.
  - 12) Go to step 3.

Fig. 4. MöbiPack Algorithm: Phase III

only the two STS-3c to move (saving one move over the other case). However, if two or more STS-1 circuits had to move in order to create space for the two STS-3c circuits, this choice would be suboptimal (since the three STS-1s could be moved with no additional overhead). Unfortunately, doing such look ahead increases the complexity exponentially. It is instructive to note that *MöbiPack* will be optimal if there are only two distinct levels in granularity.

### D. Phase IV: Circuit Migration Plan

In this phase, we find the circuit migration plan. It is clear from Phase III that certain circuits of lower granularity have to be moved before circuits of higher granularity. The pseudocode for this phase is in Figure 5.

The non-obvious step is Step 2. Unlike the specific transition order imposed by Step III, this step essentially moves circuits starting from the lowest granularity onwards. Obviously, this is a weaker condition and much simpler to implement. In the

### MöbiPack Algorithm Phase IV

- *Input:*
  - 1) List  $L$  of circuits to be moved
- *Output:*
  - 1) Migration sequence  $M = \langle l, s_o, s_n \rangle, \forall l \in L, s_o$  and  $s_n$  being the old and new slots respectively.
- *Algorithm:*
  - 1) Sort  $L$  in ascending order of granularity.
  - 2) For each demand  $d$  of granularity  $g_j$  in  $L$ 
    - a) Find first valid slot  $s$  in  $S_j$  which bit mask in  $B_j$  is 0.
    - b) Append  $\langle d, s_d, s \rangle$  to  $M$  where  $s_d$  is  $d$  current slot
  - 3) When done, space corresponding to optimal contiguous breakup has been freed.

Fig. 5. MöbiPack Algorithm: Phase IV

next section, we show that it suffices to move demands in order of their granularity without regard to their dependency.

#### E. Inter-Link Defragmentation

In this section, we describe how we can extend the *MöbiPack* algorithm to multiple parallel links across a given pair of nodes, i.e., the inter-link defragmentation problem. One can do this by simply concatenating the different links into one big “megalink” whose size is the sum of all the individual links. The megalink creates its initialization variables in Phase I ( $S$  array) from each of the constituent links. Beyond Phase I, each of the the other three stages remain the same.

The reason one can simply slap together multiple links into a single megalink is due to the alignment property (Section II-A) in SONET systems that limits circuits of a given granularity to start at only specific slots. The consequence of this property in conjunction with the values in the  $S$  array for each slot prohibits a circuit from spanning multiple links. Thus, with this simple transformation, one can scale *MöbiPack* to work on links of different sizes and each with its own set of nailed down circuits.

### VI. MöbiPack ANALYSIS

In this section, we establish complexity and correctness results about the *MöbiPack* algorithm. We first argue that *MöbiPack* meets both the necessary conditions to achieve a optimal migration sequence as highlighted in Section IV-C.

*Theorem 1: MöbiPack meets the GoodMove condition.*

*Proof:* In the Phase IV of *MöbiPack*, all circuits are moved in ascending order of their granularity. Since a circuit of higher granularity will always be moved after all circuits of lower granularity have been moved, all moves are indeed *GoodMoves*. ■

*Theorem 2: MöbiPack moves a circuit only once in a defragmentation operation.*

*Proof:* In Phase III of *MöbiPack*, once a demand is selected to be moved to accommodate a demand of higher granularity, the corresponding slots are masked to avoid them from being selected again. Thus every demand is selected to be moved exactly once. In Phase IV of *MöbiPack*, the list  $L$  of circuits to be moved is traversed in the ascending order of granularity and each circuit in the list is moved exactly once. Hence no circuit is moved more than once in the entire defragmentation operation. ■

*Theorem 3: MöbiPack achieves OLS.*

*Proof:* It is obvious from the computations in Phase III that if all identified circuits are moved then the link will be optimally defragmented and will result in  $OLS$  of free slots. Hence for correctness, we need to show that all identified circuits can be moved in the manner suggested in Phase IV. In other words, for every circuit identified to be moved in Phase III, Phase IV creates enough free space for them to move into in one step. We prove this requirement below.

Let the number of free slots of STS-1 granularity be  $S$  and the corresponding  $OLS$  is given by  $\{O_1, O_2, \dots, O_h\}$ . for granularities  $\{g_1, g_2, \dots, g_h\}$ . Then,

$$S = O_h g_h + \dots + O_2 g_2 + O_1 g_1 \quad (1)$$

Consider an arbitrary iteration step in Phase III of *MöbiPack* for granularity  $g_j$ . Consider Step 9 where “spill over” circuits from one step are added to the  $OLS$  requirement to clear in a later step. Let  $O_i^j$  be the number of spill-over circuits of granularity  $g_i$  identified to be moved in the iteration to create space for granularity  $g_j$  as dictated by the  $OLS$ . Thus,  $O_i^j = 0$  if  $j \leq i$  (Condition *GoodMove*). Adding this spillover, the number of circuits of any granularity  $g_i$  to be moved is given by

$$O_i + \sum_{j=i+1}^h O_i^j \quad (2)$$

Further at end of iteration  $k$  the new  $OLS$  is

$$O_{h-k} + \sum_{j=h-k+1}^h O_{h-k}^j, \dots, O_1 + \sum_{j=h-k+1}^h O_1^j \quad (3)$$

Now let us consider the first iteration clearing  $O_h$  slots of highest granularity  $g_h$  which would create spill over circuits of granularity lower than  $g_h$ . The total number of STS-1 slots required to satisfy all these circuits is

$$O_{h-1}^h g_{h-1} + \dots + O_2^h g_2 + O_1^h g_1 \quad (4)$$

Also, moving these spill over circuits will in turn create a free space of  $O_h g_h$  slots. Recall that each iteration also masks the cleared space to prevent lower granularity circuits from moving in at a later iteration. However, in doing so, some free slots which existed in this masked space originally are no longer usable. This can be determined as the difference between the slots freed ( $O_h g_h$ ) and the slots taken up by the

spill over circuits (Equation (4)). Thus, the “lost” slots after the first iteration is

$$O_h g_h - (O_{h-1}^h g_h + \dots + O_2^h g_2 + O_1^h g_1) \quad (5)$$

or equivalently

$$O_h g_h - \sum_{i=1}^{h-1} O_i^h g_i \quad (6)$$

Thus, the number of free slots in un-masked region for granularity  $g_{h-1}$ , obtained by subtracting Equation (6) from (1) is

$$\sum_{i=1}^{h-1} (O_i g_i + O_i^h g_i) \quad (7)$$

Continuing this argument to level  $l-1$  where we need to clear  $O_{h-1} + O_{h-1}^h$  slots of granularity  $g_{h-1}$ , we get number of free slots outside mask for  $g_{h-2}$  as

$$\sum_{i=1}^{h-2} (O_i g_i + O_i^h g_i + O_i^{h-1} g_i) \quad (8)$$

Hence, in general, we can write that number of free slots outside the mask for granularity  $g_j$  is

$$\sum_{i=1}^j (O_i g_i + \sum_{k=j+1}^h (O_i^k g_i)) \quad (9)$$

Consider expression for granularity  $g_1$  which is.

$$O_1 g_1 + \sum_{k=2}^h (O_1^k g_1) \quad (10)$$

Since  $g_1$  is 1, we can write it as

$$O_1 + \sum_{k=2}^h O_1^k \quad (11)$$

This is exactly the number of circuits of granularity  $g_1$  identified to be moved from Expression 2. In other words, the number of free STS-1 (i.e.,  $g_1$ ) slots available after the iteration for  $g_2$  is the same as the number of spill over STS-1 circuits that would need new slots. Hence, there exists enough space to move all circuits of lowest (STS-1) granularity and they can always be moved since alignment (Section II-A) is not an issue at this level.

From Phase III of *MöbiPack*, it is clear that the ability to move a circuit only depends on moving circuits of lower granularities. Thus, after moving all demands of granularity  $g_1$  space has already been created for movement of all identified circuits of next higher granularity  $g_2$  and this process can continue on for higher granularities. Thus, all demands identified in Phase III of *MöbiPack* can be moved in a hitless manner. Hence *MöbiPack* achieves *OLS*. ■

From the observations above, it is clear that *MöbiPack* satisfies all the necessary conditions of an optimal solution as listed in section IV-C. *MöbiPack* is optimal in defragmenting the bandwidth but is not necessarily optimal in the number of

circuit moves. In the next section, we show that in practice, *MöbiPack* is indeed very close to optimal in terms of number of moves.

#### A. Complexity

We next analyze the space and time complexity of the *MöbiPack* algorithm. For a SONET STS- $n$  link, the highest granularity  $g_h$  is equal to  $n$ . The complexity of both Phase I and Phase II is  $O(n)$ .

Lets consider the processing involved in Phase III for any arbitrary granularity  $g_j$ . Clearly there are  $g_h/g_j$  possible slots for demands of granularity  $g_j$ . The number of demands to be moved for each such slot can be found in one sweep over the entire link, i.e.,  $O(g_h)$ . The next step involves sorting the  $g_h/g_j$  slots based on this number, and hence has the complexity of  $O(g_h/g_j \log(g_h/g_j))$ . Updating the bit mask and copying also require  $O(g_h)$  operations. Thus, the complexity for one iteration loop in Phase III is

$$O(g_h) + O(g_h/g_j \log(g_h/g_j)) \quad (12)$$

The loop in Phase III is executed  $h$  times, the number of levels in hierarchy. Hence, the complexity of entire phase is

$$O\left(\sum_{j=1}^h g_h + g_h/g_j \log(g_h/g_j)\right) \quad (13)$$

Substituting the smallest value for  $g_j$  as 1 and ignoring the  $O(n)$  terms we find complexity as  $O(nh \log(n))$ .

In Phase IV, sorting complexity in step 1 is  $O(n \log(n))$  and for step 2 the complexity is  $O(nh)$ . Therefore the overall complexity of the *MöbiPack* algorithm is  $O(nh \log(n))$ . The amount of space required is the original given space for  $\psi$  which is  $O(n)$ . In addition, at every level in hierarchy we save a bit mask of  $O(n)$  space. Hence, the total space complexity is  $O(nh)$ .

## VII. PERFORMANCE ANALYSIS

As highlighted in the last section, the *MöbiPack* algorithm results in optimal packing of circuits but does not guarantee minimum number of circuit moves to achieve it. Hence the key performance metric is the number of moves taken to achieve optimal packing. However, we first need a benchmark to compare against.

We use as the benchmark, a brute-force procedure that computes the optimal number of moves by exhaustively searching the search space for the shortest migration path. The exhaustive search procedure exploits properties of optimal solution as presented in Section IV-C to reduce the computation complexity of the search. The scheme takes the solution of *MöbiPack* as current known best solution and tries to improve up on it. The procedure is very similar to Phase III of *MöbiPack* except that instead of greedily making the local choice in Step 5, it explores all possible combinations at each granularity level by recursively invoking the same function. Any exploration that incurs more moves than the current best, is abandoned. It also exploits the fact that *MöbiPack* procedure

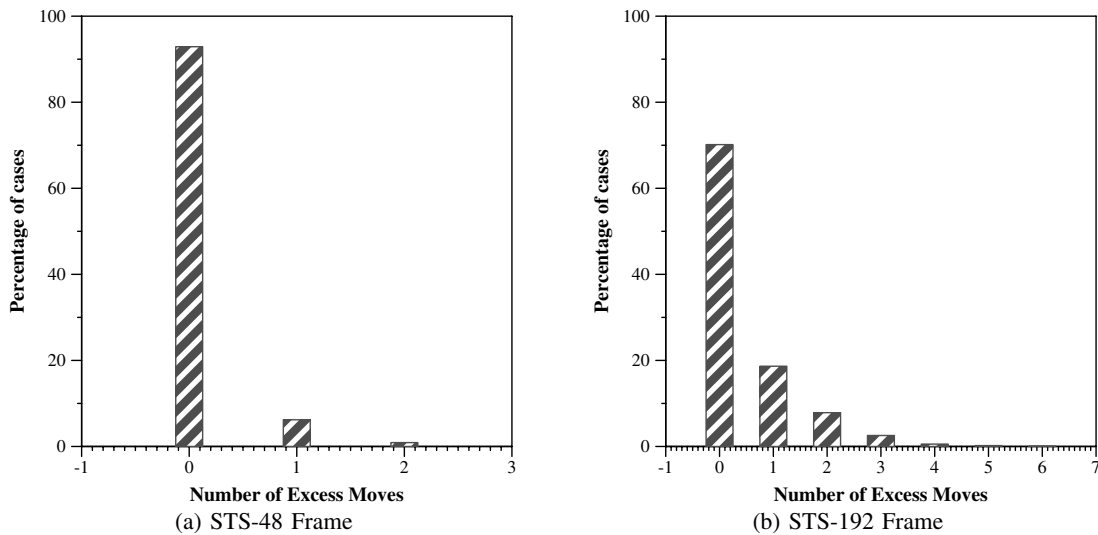


Fig. 6. Cost comparison of MöbiPack with optimal

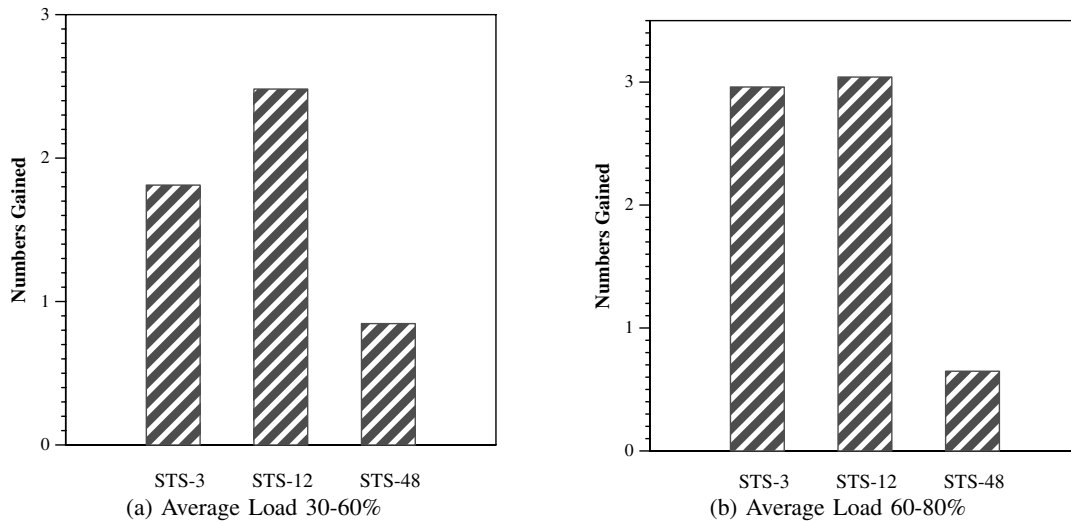


Fig. 7. Average defragmentation gains on a OC-192 link

is optimal for last two levels of granularity in each exploration. The circuit movement step is same as Phase IV of *MöbiPack*. Clearly, this procedure cannot scale very well beyond OC-192. However, for STS-48 and STS-192 links, one can derive the optimal migration sequence in reasonable time and we use these link sizes as the benchmark comparison.

#### A. *MöbiPack* Cost

Figure 6(a) and (b) compare the performance of *MöbiPack* algorithm against the exhaustive optimal procedure for link sizes of STS-48 and STS-192 respectively. The simulation is based on 2000 randomly generated link frames. The experiments highlight that performance of *MöbiPack* is extremely good, matching the optimal number of moves in 95% and 70% of the cases for STS-48 and STS-192 links respectively. For remainder of cases, it requires only a couple of extra moves more than the optimal solution. This demonstrates that

*MöbiPack* is indeed an optimal defragmentation algorithm with a near-optimal overhead.

#### B. *MöbiPack* Gains

Figure 7 shows the additional circuits *MöbiPack* can satisfy by optimally defragmenting the link. The experiment consists of randomly creating demands on a OC-192 link such that the link is filled to some load capacity. The graphs represent the average gains from 10,000 runs in terms of new STS-n demands that can be satisfied after defragmentation for a network load of 30-60% (Figure 7(a)) and 60-80% (Figure 7(b)).

The graphs show that defragmenting a link can significantly improve utilization. For example, for the 30-60% case, defragmentation allows an average of 2.5 additional STS-12c demands to be satisfied that would otherwise have been rejected. Clearly, as the load on the link is increased in Figure 7(b), the increased demands further fragment the link. Consequently,

compared to the low load case, on defragmenting, many more new free STS-3c and STS-12c slots are created (at the expense of STS-48c slots).

Converting these freed up spaces into equivalent monetary amounts will provide an estimate for the savings in capital and operating expenses for a service provider. Clearly, in the current economic environment, these results indicate that link defragmentation can indeed provide significant financial advantages and the *MöbiPack* algorithm is a cheap and efficient way to deliver them.

### VIII. CONCLUSIONS

The paper studied the heretofore unaddressed problem of link defragmentation in SONET/SDH networks. As service providers face reduced budgets, they are increasingly looking for means to improve their network utilization in a manner that does not impact service. Thus, enabling hitless bandwidth defragmentation in SONET is very attractive to network operators today.

In this paper, we presented a novel link defragmentation algorithm called *MöbiPack*. We demonstrated that unique constraints imposed by the SONET standard make this defragmentation problem fundamentally different from known analogous problems such as disk defragmentation. We posited that the theoretical problem of achieving optimal hitless defragmentation while also optimizing the cost, is a likely hard problem. The complexity is further increased by the presence of “nailed down” circuits that have to be accounted for in the defragmentation operation. In spite of this inherent hardness, *MöbiPack* achieves optimal defragmentation while requiring only marginally higher number of circuit moves than an optimal solution. In addition, *MöbiPack* has low complexity

thereby making it extremely attractive to implement in practice.

This paper opens a number of avenues for future work. Clearly, the hardness of the posed optimality problem needs to be resolved. Moreover, bandwidth fragmentation arises in every level of the SONET network – from links to rings to the end-to-end network. Creating defragmentation solutions that achieve their goal while avoiding service impact remains a critical challenge going forward.

### IX. ACKNOWLEDGMENT

We would like to acknowledge the feedback from the anonymous referees that helped improve the content of the paper.

### REFERENCES

- [1] D. E. Knuth, *The Art of Computer Programming, Fundamental Algorithms*, 2 ed. Addison-Wesley Pub Co, 1973, vol. 1.
- [2] GR-1230-CORE, “SONET Bidirectional Line Switched Ring Equipment Generic Criteria,” *Bellcore*, November 1995.
- [3] W. Goralski, *SONET*, 2nd ed. McGraw-Hill Companies, 2000.
- [4] S. Cosares and I. Saniee, “An Optimization problem related to balancing loads on SONET rings,” *Telecommunication Systems*, vol. 3, 1994.
- [5] Y. Myung, H. Kim and D. Tcha, “Optimal Load Balancing On SONET Bidirectional Rings,” *Operations Research*, vol. 45, January 1997.
- [6] A. Schrijver, P. Seymour and P. Winkler, “The Ring Loading Problem,” *SIAM Journal of Discrete Math*, vol. 11, No 1, February 1998.
- [7] M.3100, “Definition of the Management Interface for a bridge-and-roll cross-connect feature,” *Amendment 4, ITU-T*, Aug 2001.
- [8] V. Ramaswamy and K. Aswath Rao, “Flexible Time Slot Assignment-Performance study for Integrated Services Digital Network,” *ITC*, 1985.
- [9] C. Chigan, R. Nagrajan, Z. Dziong and T.G. Robartazzi, “On the Capacitated Loss Network with Heterogenous Traffic and Contiguous Resource Allocation Constraint,” *Applied Telecommunication Symposium*, April 2001.
- [10] S. Acharya, B. Gupta, P. Risbood, and A. Srivastava, “Hitless Network Engineering of SONET Rings,” *IEEE Globecom*, December 2003.
- [11] S. Acharya, B. Gupta, P. Risbood and A. Srivastava, “Hitless Link Defragmentation: Issues and Challenges,” Bell Labs, Tech. Rep., 2003.