◆ Network Capacity Recovery and Efficient Capacity Deployment in Switching Centers

Nachi K. Nithi, Carl J. Nuzman, and Benjamin Y. C. Tang

One way for service providers to reduce their costs is through network capacity recovery. We describe the problem of recovering capacity by minimizing the switching overhead incurred by multiple switches in a switching center. We also describe the Multiple ATM Switch Configuration Optimization Tool (MASCOT), which addresses this problem. Initially developed for asynchronous transfer mode (ATM) switches, the tool applies flexibly to other devices such Synchronous Optical Network (SONET) cross connects or Multiprotocol Label Switching (MPLS) routers. New services enabled by this tool include (1) reconfiguration of an existing switching center to increase the number of available switch ports and (2) efficient green field design for migrating to new hardware. Given the set of external trunks and the traffic demands between them, MASCOT searches for a switching center topology that minimizes the traffic carried on internal trunks. MASCOT handles a wide variety of switch types with hierarchical constraints imposed by components such as interface cards, processor cards, and slots.

Introduction

Network optimization has always been an important part of the business of building and upgrading communication networks. In a capacity deployment scenario, the network operator would like to minimize capital expenditures and operating expenses, while satisfying capacity and quality of service requirements. Capacity recovery services take the complementary approach of performing network optimization as part of the maintenance of an existing communication network. Here, the capital infrastructure is constrained, and the goal is to maximize the communication capacity and quality of service within the existing network.

On the face of it, capacity recovery and capacity deployment services could be seen as competing, since a successful capacity recovery service could postpone a network provider's need for new equipment. However, they should more properly be seen as complementary and synergistic. Capacity recovery provides a beneficial service to network providers who are unable or unwilling to make significant capital purchases. Applying similar optimization tools and expertise for capacity recovery and capacity deployment can increase customers' confidence that they are receiving the greatest possible return on their capital and operational investment.

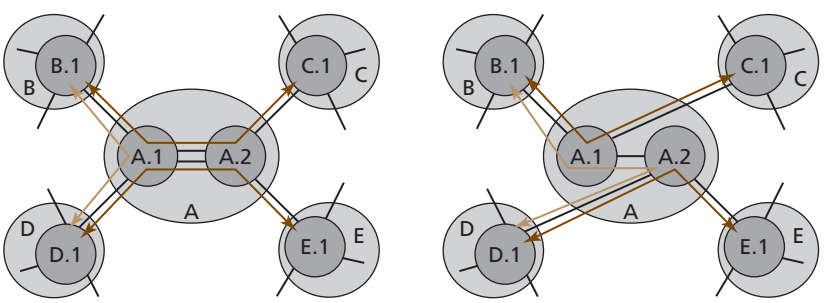
Bell Labs Technical Journal 9(4), 83–100 (2005) © 2005 Lucent Technologies Inc. Published by Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/bltj.20063

In this paper, we focus on a class of physical topology optimization that is applicable to capacity deployment and capacity recovery services: optimizing the connectivity between switches within a single switching center. The need for these services arises when network switching centers contain multiple collocated switches. Such switching centers are able to handle more traffic than any individual switch could, but the aggregate capacity is always less than the sum of the individual switch capacities due to internal switching overhead. To understand the proposed services, it is helpful to consider a simple example. **Figure 1** depicts a switching center (A) connected to four neighboring centers (B, C, D, and E) under two different physical topologies. The light gray circles represent switching centers and the dark gray circles are switches. The brown arrows represent large traffic flows, while the tan arrow represents a smaller flow. Black lines represent trunks. In Figure 1(a), switching center A contains two switches (A.1 and A.2), connected to each other by two trunks; these trunks are called *internal* trunks because both endpoints are in the same switching center. The four trunks that connect A.1 and A.2 to switches in other switching centers are referred to as *external* trunks. The brown arrows represent large aggregate traffic flows between centers B and C, and between centers D and E, which traverse both local switches and use the internal trunks. The tan arrows represent a smaller aggregate flow between centers B and D that only traverses switch A.1. In Figure 1(b), the external A-D trunk terminates on switch A.2 rather than switch A.1, and the external A-C trunk terminates on switch A.1 rather than switch A.2. In this configuration, the large flows pass through single local switches, and only the smaller tan flow must pass through both A.1 and A.2. The net effect is a reduction of load on A.1 and A.2, reduced traffic between A.1 and A.2, and the ability to remove one of the internal trunks. Other benefits could include a reduction in end-to-end average delay.

In a true switching center, the situation is much more complicated. There may be several switches of

Panel 1. Abbreviations, Acronyms, and Terms

ATM—Asynchronous transfer mode
CGI—Common gateway interface
GUI—Graphical user interface
IP—Internet Protocol
MASCOT—Multiple ATM Switch Configuration Optimization Tool
MPLS—Multiprotocol Label Switching
OC—Optical carrier; OC12, OC48, and OC192 are optical carrier digital signal rates of 622.08 Mb/s, 2.488 Gb/s, and 9.953 Gb/s, respectively, in a SONET system.
PC—Processor card
PIC—Physical interface card
POS—Packet over SONET
PVC—Permanent virtual connection
SDH—Synchronous Digital Hierarchy
SONET—Synchronous Optical Network
UBR—Unspecified bit rate



(a) Before optimization (b) After optimization

Figure 1.
Switching center configurations.

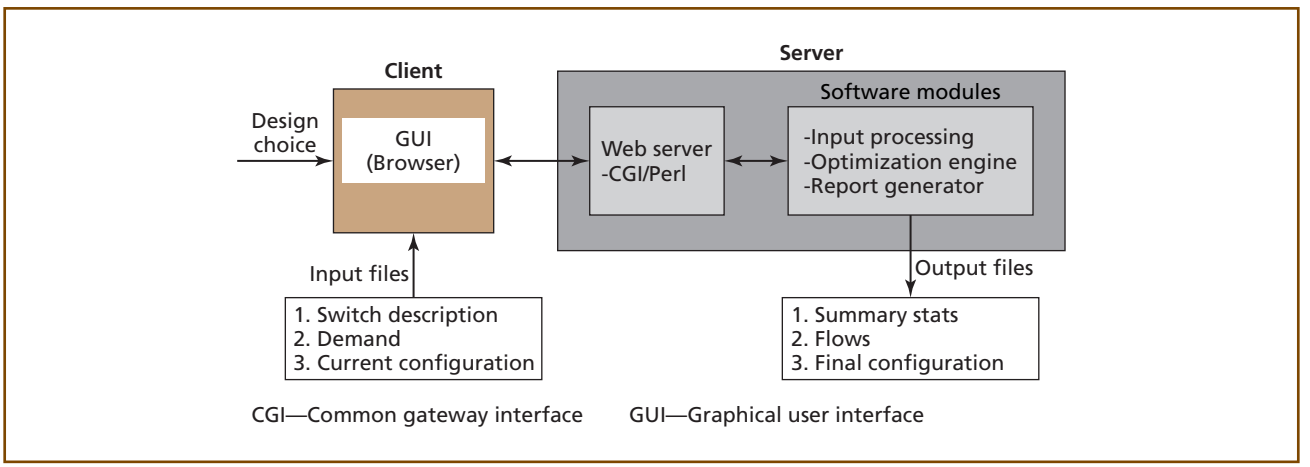


Figure 2.
MASCOT software architecture.

different types terminating dozens of external trunks of many types, which in turn carry thousands of flows. We have developed a switching center optimization tool for solving problems of this type. Depending on the service, the optimization can be used to make an existing switching center more efficient, to deploy an efficient switching center in the first place, or to explore the consequences of different switch designs.

In the next section, we describe the optimization tool. In subsequent sections, we give examples of how this tool has already been used to support capacity deployment services and describe how it could be used in capacity recovery applications.

Switching Center Optimization Tool

The Web-based client-server tool we have implemented to address the problem of switching center topology optimization is the Multiple ATM Switch Configuration Optimization Tool (MASCOT). The core optimization engine uses a local search heuristic to try to minimize the total traffic on internal trunks or, equivalently, to minimize the total switching load. The tool can also compute the hardware costs associated with any configuration and choose the least-cost solution among candidates produced by the core optimization engine. The software architecture of the system consists of a Web browser-based user interface client and a back-end server consisting of a Web server and software modules. The back-end software modules comprise a core optimization engine, an input parsing and validation module, and a report generator. On the server side, common gateway interface (CGI)-driven Perl language scripts are used to glue the back-end software modules to the Web server. **Figure 2** illustrates the MASCOT architecture.

There are two types of optimizations that MASCOT performs: a green field design, to design a new switching center, and a brown field design, to perform re-optimization of an existing center. Apart from specifying the optimization, the client graphical user interface (GUI) also allows the user to upload input data files and to browse and download optimized configurations. For a green field design, MASCOT requires two input files: a *Switch Specification* file and a *Demand* file. In case of re-optimization, an additional input file, called *Current Configuration*, specifying the existing switch configuration is needed. MASCOT produces outputs in three files: the *Summary* file, the *Flows* file, and the *Final Configuration* file.

Input and Output Data

The input and output files, which are ASCII text based, are described in detail in the subsections below.

Switch model. The *Switch Specification* file is used to define the types of switches available in the switching center and how they may be used. Specifically, the file must define which interfaces are supported by the

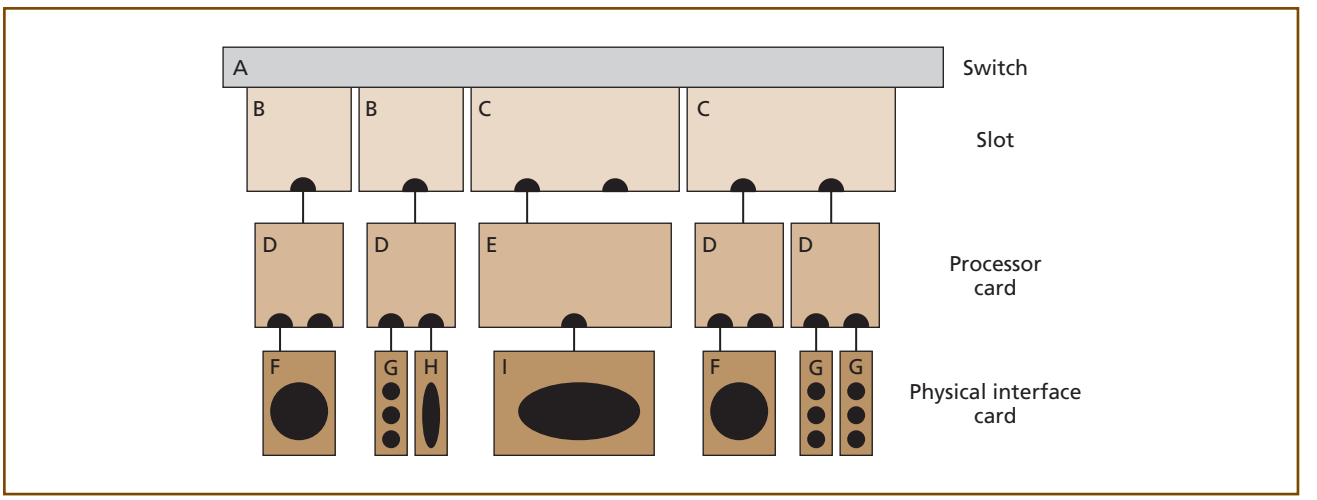


Figure 3.
Hierarchy of switch components.

switch, what combinations of interface ports are possible, and what minimal cost is required to support a particular combination of ports. Rather than requiring an explicit description of the set of valid combinations of ports, the tool uses a hierarchical model to implicitly define this set. A hypothetical switch described by the model is illustrated in **Figure 3**. At the bottom of the hierarchy are physical interface cards (PICs), each of which contains a fixed number of ports of a single interface type. In Figure 3, there are seven PICs of four types. The PICs of type G have three ports, while the others each contain one port; the varying sizes and shapes of the ports indicate different data rates, protocols, and so on. At the next level are processor cards (PCs). Each PC can service one or more PICs from a specified set. Each PC has a fixed limit on the total number of PICs it can take, as well as a limit on the total data rate of all ports connected to it. In Figure 3, a PC of type E can accept one PIC of type I, while each PC of type D can accept up to two PICs of types F, G, and H, in various combinations. At the top level of the model, the switch consists of a number of slots of various types. The number of slots of each type is considered to be fixed for any given switch type. Each slot can accept processor cards from a given set, up to a specified maximum number of cards. In Figure 3, a switch of type A comes with two slots of type B and two slots of type C. Slots of the latter type can accept PCs of type E or D, while type B slots are restricted to PCs of type D. The information needed to specify the model parameters is simple and usually readily available from the manufacturer. By contrast, an explicit description of the set of all possible port combinations can be quite complicated.

Because the model relates to the actual architecture of most switches, it is flexible enough to apply to a wide range of switch types. **Panel 2** illustrates the syntax used to specify the model parameters for each switch type in the *Switch Specification* file, and **Panel 3** gives an example of such a file. The file consists of an optional *User_Defined_Interface* object, followed by one or more *SWITCH* objects. A number of common generic interface types and their corresponding transmission rates are predefined in MASCOT (e.g., OC12 and 622 Mb/s). The *User_Defined_Interface* object is used to define any other interfaces that are needed. Each PIC defined in a SWITCH object is given an interface type, and each trunk defined in the *Demand* file (described below) is also assigned an interface type. During optimization, trunks can only be assigned to ports with a matching interface type. In Panel 3, for example, interfaces called OC12POS and OC12ATM are defined in order to distinguish between ports designed for packet-over-SONET (POS) transmission and those designed to use asynchronous transfer mode (ATM). The user-defined interfaces could also be used, for example, to distinguish between short-range and

Panel 2. Syntax of *Switch Specification* Objects

```
Object User_Defined_Interface
    {<InterfaceName string>}+
end

{ Object  SWITCH
 <Vendor      string>
 <SwitchName  string>
 <Capacity    real>
 <SwitchCost  real>
 <NumSlots    integer>
 <LoadFactor  real>

 Object PIC
 {<PIC_Label PIC_ID NumberOfPorts CapPerPort Cost>}+
 end

 Object PC
    {<PC_Label PC_ID MaxNumPICs PICAllowed MaxProcessingCapacity Cost>}+
 end

 Object SLOT
    {<SLOT_Label SLOT_ID MaxNumPCs PCAllowed Cost>}+
 end
end }+
```

long-range optical interfaces on SONET equipment, or between native ATM ports and frame relay ports on an ATM switch.

Each *SWITCH* object consists of a *Basic Switch Information* section followed by a *PIC* object, a *PC* object and a *SLOT* object. Each line of the *PIC* object defines a different type of physical interface card, including the interface type, number of ports, and cost for each card. Similarly, the *PC* object defines the parameters of each PC type. Because each switch comes with specific slots built in, the *SLOT* object includes one line for each slot provided on the switch, rather than just for each slot type; multiple slots may have the same parameters, in which case we may say that they are of the same slot type. Panel 3 provides an example Switch Specification file defining a router with eight slots, two types of processor cards, and four types of physical interface cards. This router model will be referred to again in the "Capacity Deployment Services" section below.

Traffic flows. The *Demand* file is used to list the flows that pass through or terminate in a given switching center and to specify which external trunks they are routed on. An example file is depicted in **Panel 4**. The first section lists the flows: for each flow, the file identifies a source trunk, a destination trunk, and a bandwidth. Each trunk has a text label and an integer identifier (e.g., "ATM-APOP1" and "101"). Multiple flows with identical parameters can be included on one line using a count field; in Panel 4 each flow is listed separately with a count of one. The aggregate bandwidth demand between each pair of external trunks is determined by summing over appropriate flows. The second section of the file defines the trunks for the switching center, which fall into three categories. External trunks are those that carry traffic to and from remote switching centers, while internal trunks connect two switches within the local switching center. Access trunks are used to carry flows that are dropped locally in the switching center.

Panel 3. Example *Switch Specification* File

```
Object UserDefinedInterfaces
# label      DataRate(Mpbs)
OC12POS      622
OC12ATM      622
end

Object Switch
# Base switch information
SwitchName   A-Router
Capacity     320.0
SwitchCost   200
NumSlots     8
LoadFactor   1.0

Object PIC
# label            ID   NumPorts    Interface    Cost
PIC-1pOC192POS     1    1           OC192        200
PIC-4pOC48POS      2    4           OC48         300
PIC-4pOC12POS      3    4           OC12POS      50
PIC-2pOC12ATM      4    2           OC12ATM      25
end

Object PC
# label     ID   MaxNumPICs    PICAllowed    MaxProcCap   Cost
PC-fast     1    4             1,2           40           100
PC-slow     2    4             3,4           16           25
end

Object Slot
# label     ID   MaxNumPCs     PCAllowed     Cost
Slot-1      1    1             All           0.0
...
Slot-8      8    1             All           0.0
end

end
```

External trunks are part of the problem definition, and each trunk is defined on its own line. The line identifies the trunk by the same integer identifier used in the previous section of the file and specifies an interface type and fill factor. The fill factor is a multiplier applied to nominal trunk capacity to obtain the available capacity; a factor of 0.9, for example would force the optimization tool not to use more than 90% of the nominal capacity. Because internal and access trunks are created as part of the solution to the problem, these trunks are not explicitly listed in the *Demand* file. Instead, two lines are used to specify the interface types and fill factors to be used when creating internal and access trunks.

Switching center configuration. The *Current Configuration* file contains a description of the existing switch configurations and is used as the starting point for the brown field optimization runs. It contains an explicit

Panel 4. Example *Demand* File

```
# List of flows between external trunks
# from    trunkID  to trunkID   bandwidth count comment
ATM-APOP1   101   CORE1 1     1.03667e+08 1     NA
ATM-APOP1   101   CORE2 2     1.03667e+08 1     NA
ATM-APOP2   102   CORE3 3     1.03667e+08 1     NA
ATM-APOP2   102   CORE4 4     1.03667e+08 1     NA
ATM-APOP2   102   CORE5 5     1.03667e+08 1     NA
ATM-APOP3   103   CORE4 4     1.03667e+08 1     NA
ATM-APOP3   103   CORE5 5     1.03667e+08 1     NA
...
IP-APOP1    501   CORE3 3     2.49259e+07 1     NA
IP-APOP1    501   CORE4 4     2.49259e+07 1     NA
IP-APOP3    503   CORE5 5     2.49259e+07 1     NA
IP-APOP3    503   CORE6 6     2.49259e+07 1     NA
IP-APOP4    504   CORE1 1     2.49259e+07 1     NA
IP-APOP4    504   CORE2 2     2.49259e+07 1     NA
IP-APOP4    504   CORE3 3     2.49259e+07 1     NA
...
CORE1       1     CORE5 5     2.06667e+08 1     NA
CORE3       3     CORE4 4     5.01250e+08 1     NA
...

#Specify interface to be used for internal and access trunks
# Type      ID    Interface   FillFactor
Trunk I     All   OC48        1.0
Trunk A     All   OC12POS     1.0

#Define the interfaces for the external trunks
# Type      ID    Interface   FillFactor
Trunk E     1     OC192       1.0
...
Trunk E     6     OC192       1.0
Trunk E     101   OC12ATM     1.0
Trunk E     102   OC12ATM     1.0
...
Trunk E     501   OC12POS     1.0
Trunk E     502   OC12POS     1.0
...
Trunk E     635   OC12POS     1.0
```

list of all of the equipment in use, including switches, slots, PCs, PICs, and trunks. In addition, the file specifies how all of the pieces are connected to each other. The syntax of the seven sections in the file is shown in **Panel 5**, and an example configuration, with three instances of a single switch type, is shown in **Panel 6**. Each component is described by an identifier and a type. The type is a reference to the different object types defined in the switch specification file in Panel 3 above. The identifier uniquely specifies each component and its relationship to other components. For example, S1 refers to a particular switch, S1.1 refers to the first slot on that switch, S1.1.2 is the second PC on S1.1, and so on. Assuming that components are numbered from left to right in Figure 3, the PIC of type H would have the identifier S1.2.1.2, as it is the

Panel 5. Syntax of *Configuration* Files

```
# SWITCH id description
{<SWITCH_ID Type>}+

# SLOT configuration description
{<SLOT_ID Type>}+

# PC configuration description
{<PC_ID Type>}+

# PIC configuration description
{<PIC_ID Type>}+

# External Trunk (E-Trunks) configurations
{<TRUNK_ID Type Port_ID Fill_Factor>}+

# Access Trunk (A-Trunks) configurations
{<TRUNK_ID Type Port_ID Fill_Factor>}+

# Internal Trunk (I-Trunks) configurations
{<TRUNK_ID Type Port_ID_1 Port_ID_2 Fill_Factor>}+
```

second PIC on the first PC on the second slot of the first switch. The trunks are described using additional information such as the ports to which they are connected and their fill factor. For example, in Panel 6, external trunk E502 is connected to port id S2.2.1.1.2, meaning the second port on PIC S2.2.1.1. Each internal trunk uses a switch port on two local switches while the external and access trunks use only one local port. The same format is used to describe the state of the system after optimization in the *Final Configuration* file.

Results. At the end of a run, MASCOT produces outputs in three different files. A high-level summary of the design is produced in the *Summary* file. This includes information such as the number of switches needed for the final configuration and their type; a count of the number of external, access, and internal trunks; ports needed on each switch; a list of PCs, PICs and slots required; cost information; aggregate load on switches; and statistics on switch overhead. The *Final Configuration* file, whose format is same as that of the *Current Configuration* file, is generated to give detailed information about how various elements on each switch should be configured. The *Flows* file is the third output file generated by MASCOT. Its format is similar to the *Demand* file, except that the *Flows* file also specifies a route that each flow should take through the switching center. The route is specified by listing, in order, all of the switches and trunks that the flow passes through.

Optimization Algorithm

The problem faced in switching center optimization is essentially a partitioning problem. Given a particular set of external trunks, the goal is to form clusters of trunks with traffic in common such that the amount of traffic *between* clusters is small. Each cluster is then assigned to a different switch, so that a large amount of traffic need only pass through a single switch in the switching center. The problem can be considered to be a variant of the min k-cut problem in graph theory. In that problem, a graph has several nodes (external trunks) connected by edges, and each edge has a non-negative weight (the amount of traffic between corresponding trunks). In the min k-cut problem, the goal is to divide the graph into k disjoint clusters while minimizing the weight of inter-cluster edges (i.e., minimizing traffic on internal trunks). For

Panel 6. Example Configuration File for a Switching Center with Three Routers

```
#Switches
#ID    type
S1    A-Router
S2    A-Router
S3    A-Router

#Slots
#ID    type
S1.1  Slot-1
S1.2  Slot-2
...
S3.8  Slot-8

#PCs
#ID    type
S1.1.1      PC-FAST
S1.2.1      PC-SLOW
...
S3.7.1      PC-SLOW

#PICs
#ID    type
S1.1.1.1    PIC-1pOC192POS
S1.1.1.2    PIC-1pOC192POS
S1.1.1.3    PIC-1pOC48POS
S1.1.1.4    PIC-1pOC48POS
S1.2.1.1    PIC-2pOC12ATM
S1.2.1.2    PIC-2pOC12ATM
...
S3.7.1.1    PIC-4pOC12POS
s3.7.1.2    PIC-4pOC12POS
s3.7.1.3    PIC-4pOC12POS

#External Trunks
#ID   type     Port           fillFactor
E1    OC192    S1.1.1.1.1      1
...
E6    OC192    S2.1.1.4.1      1
E101  OC12ATM  S2.2.1.3.1      1
E102  OC12ATM  S1.2.1.1.1      1
...
E501  OC12POS  S2.2.1.1.1      1
E502  OC12POS  S2.2.1.1.2      1

#Internal Trunks
#ID   type  Port1       Port2       fillFactor
I1    OC48  S1.1.1.3.2  S2.1.1.3.1  1
...
I8    OC48  S3.2.1.1.2  S2.1.1.3.2  1
```

fixed k, the optimal solutions can be found in polynomial time, and there is a fairly simple 2-approximation algorithm [3, 5]. The key distinction in switching center optimization is that there are complex constraints specifying which combinations of nodes can fit into the same cluster. These constraints include switch capacity constraints as well as configuration constraints implied by the hierarchy of switch components.

Given a particular set of switches fitting into the hierarchical model, the problem may be formulated as an integer program and, in principal, solved to optimality. When individual flows are small relative to the trunk size, some of the integer variables can be relaxed to take continuous values. Even so, this approach does not scale well to problems with several switches and tens to hundreds of external trunks. The MASCOT algorithm instead uses a heuristic technique similar to the search algorithm of Lin and Kernighan [4]. The latter algorithm was used to partition large graphs having limits on the number of nodes in each cluster. In its simplest form, that algorithm involved sequentially swapping pairs of nodes between clusters, choosing at each step the pair of nodes that produces the maximum benefit. The MASCOT algorithm also makes a sequence of incremental perturbations, choosing in each step to swap a pair of external trunks or transfer a single external trunk from one switch to another. It is straightforward to compute the benefit of each such perturbation, but relatively difficult to determine if a given perturbation satisfies all switch constraints. The key to making the algorithm scalable is to make this determination as fast as possible. This involves pre-processing the hierarchical switch constraints in order to extract relatively simple approximate constraints to be used in interior loops. As with any greedy algorithm, the core optimization algorithm is prone to becoming trapped in local minima. To find better local minima and increase the likelihood of finding the global minimum, the algorithm is repeated for several random initial configurations, and the best solution is retained.

Pseudo-code for an implementation of the MASCOT algorithm is shown in **Panel 7**. The core of the algorithm is the local optimization subroutine *improve_config()*. This subroutine takes a switching center configuration as input, generates a set of candidate perturbations of the configuration, and computes the improvement that would result from each perturbation. Here improvement is measured in terms of a reduction in switching load. The feasibility of perturbations is checked in order of decreasing benefit, and the first feasible perturbation is applied to create an improved configuration. The process of generating and evaluating perturbations continues until no feasible and beneficial perturbation is found, at which point *improve_config()* returns the improved configuration and its cost. Depending on the user's preference, the cost can be the switching load or the total hardware cost of the configuration. The overall algorithm is embodied in the *optimize_configuration()* routine, which controls the optimization process based on the design choice selected by the user. In case of the brown field design (improving an existing design), the existing switching center configuration is used as the starting point for *improve_config()*. In the case of a green field design, random initial designs generated by the *gen_random_config()* are fed into the *improve_config()* subroutine. The best configuration seen so far and its cost are updated after each call to *improve_config()*. The program halts when a terminal condition is reached: either the total number of iterations exceeds a limit or the number of successive iterations without improving the best solution exceeds a limit. The *gen_random_config()* routine begins with a single switch and attempts to assign all trunks to it. If this fails, additional switches are added one at a time, and all external trunks are randomly divided between the switches, in proportion to the switch capacity. The process of adding switches and assigning external trunks continues until a feasible initial assignment is found.

Simulation

The degree of optimization obtained and the number of iterations required can vary greatly from one scenario to another. Results from two representative scenarios are shown in **Figure 4**. The first scenario involved ATM switches connecting 16 external trunks under a dense traffic matrix derived from a proposed

Panel 7. Switching Center Optimization Algorithm

```
optimize_configuration (design, demand, switch_spec, current_config) {
   gain = 0;
   if (design == brown_field) then {
      C_i = current_config;
      (cost, best_C) = improve_config(C_i);
      }
   else {/* Green field design */
       cost = infinity;
       do
            C_i = gen_random_config(switch_spec, demand);
            (t_cost, t_C) = improve_config(C_i);
            if (t_cost <= cost) then {
                cost = t_cost;
                best_C = t_C;
            }
       until (term_condition);
     }
     gen_reports(cost, best_C);
}

sub improve_config(CC) {
    do
       no_further_improvement = 1;
       T_I = gen_trades(CC);
       for each S_i in T_I { U_i = compute_gain(S_i); }
       sort (U_i, S_i) in order of decreasing U_i;
       for each U_i such that U_i>0 {
           if feasible(S_i) then {
              CC = apply_trade(CC,S_i);
              no_further_improvement = 0;
              break; /* to beginning of do..until */
            }
         }
   until (no_further_improvement);
return(cost_config(CC),CC);
}
```

national network design. The second scenario involved optical cross connects connecting over 100 trunks, using a sparse traffic matrix taken from an operating switching center. In each scenario, the *improve_config* subroutine was run 80 times, using randomly generated initial conditions. The total switching load was recorded after each perturbation step to obtain an improvement profile for each run. We refer to the lowest switching load observed in 80 trials to be the supposed optimal. Figure 4 shows the average switching load observed in each step, expressed as percentage excess relative to the supposed optimal. The ATM example improves by about 5%, coming very close to the supposed optimal on average within 10 perturbation steps. In the optical example, the excess is reduced from 40% to 5% on average. The number of perturbations required is much larger, simply because the number of trunks is larger. Generally speaking, the difference between random and optimal designs is largest when the elements of the trunk traffic matrices

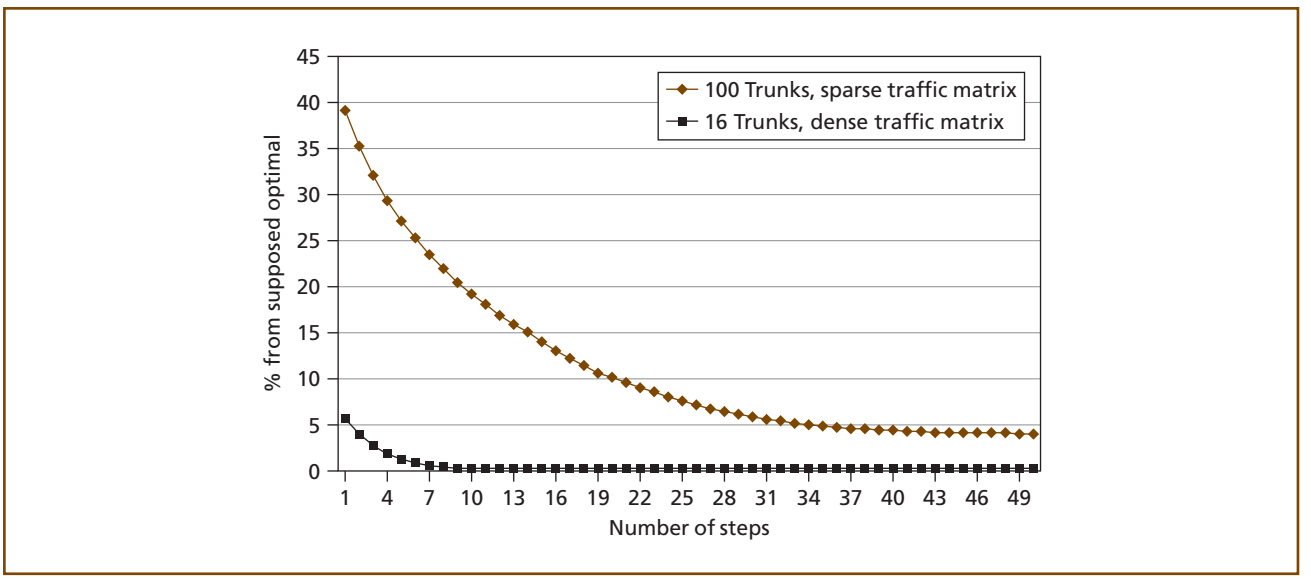


Figure 4.
Progress of MASCOT optimization algorithm.

are highly variable and is smallest when the traffic matrices are relatively uniform.

Capacity Deployment Services

In capacity deployment scenarios, internal overheads or constraints are sometimes overlooked in the initial design phase, resulting in unrealistic switching center designs. In later design phases, in the absence of a switching center optimization tool, there is a tendency to over-engineer the switching center. In this section, we first illustrate how MASCOT can be used to accurately find a feasible and efficient design between these two extremes. Secondly, we show how this accuracy was used in a network evolution study to compare the cost of deploying different routers.

Improved Accuracy Compared with Simple Estimates

In one Multiprotocol Label Switching (MPLS) network design scenario, there was a need for a switching center that connected to the core network via six OC192 POS trunks and that also connected to regional networks using 166 OC12 POS trunks and 28 OC12 ATM trunks. The traffic matrix primarily consisted of traffic flows between the six core trunks and the regional trunks, with a total bandwidth of 108 Gb/s over all flows. **Table I** illustrates the results of designing the switching center via three methods: an optimistic estimate, a conservative estimate, and the MASCOT design tool. The optimistic estimate is based on ignoring switching overhead and internal trunks; equivalently, it assumes that all flows can pass through the switching center using only a single switch. To make the optimistic estimate, one first computes the number of PICs and PCs needed to terminate all of the external trunks, and then computes M_1, the number of switch chassis needed to terminate the set of PCs previously computed. Next, M_2 is computed by rounding up the ratio of total traffic bandwidth to the switch capacity. Finally, the number of chassis needed is the maximum of M_1 and M_2. In the present example, $M_2 = 1$ since the total traffic load of 108 Gb/s would easily fit onto one router with 320 Gb/s capacity. On the other hand, $M_1 = 2$ since the trunk terminations require two fast processor cards and fourteen slow processor cards, hence 16 slots and two routers.

A conservative estimate is made by assuming that each of the flows may need to pass through two switches in the switching center. This immediately doubles the total switching load to 216 Gb/s in this case. The extra switching load of 108 Gb/s, carried on OC48 internal trunks with transmission rate 2.5 Gb/s,

Table I. Results of designing switching center via three methods.

		Optimistic estimate	MASCOT design	Conservative estimate
PICs	OC192 POS	6	6	6
	OC48 POS	0	6	11
	OC12 POS	42	42	42
	OC12 ATM	14	14	14
PCs	PC-fast	2	4	5
	PC-slow	14	15	14
Chassis		2	3	3
Cost	Absolute ($M)	6.9	8.9	10.1
	Relative error	−22%	—	13%
Switching load	Absolute (Gb/s)	108	159	216
	Percentage error	−32%	—	35%

would require 11 four-port physical interface cards, which in turn would require 3 additional fast processor cards, 3 additional slots, and create the need for an additional switch chassis. Note that the conservative and optimistic estimates differ by a factor of two in switching load, and in this case by a factor of 1.5 in terms of hardware cost. The design produced by MASCOT finds a feasible design that in this case is approximately halfway between the optimistic and conservative estimates. Since trunks are clustered intelligently on the switches, only 6 OC48 PICs are necessary, and the switching overhead is reduced to about 51 Gb/s. Note that the hardware costs used to compute values in Table I are not the same as the costs depicted in Panel 3. Although the "optimal" design falls roughly halfway between the two estimates in this example, in general it could vary in a wide range. While the optimistic estimate is a true lower bound, in fact, the conservative estimate may fail to be an upper bound, particularly when the number of switches is large. For all of these reasons, an accurate design tool is preferable to the simple estimates.

Router Comparison in Network Evolution Study

In this section, we illustrate the important role of switching center optimization as part of a network evolution study. The study was done for a North American service provider, currently having separate ATM and Internet Protocol (IP) networks, which is planning to migrate to next-generation MPLS and is faced with the challenge of selecting the most cost-effective solution from a number of available MPLS evolution scenarios.

A network modeling and business case study was conducted to quantify the cost of ownership—i.e., capital expenditures and operational expenses—associated with these alternative evolution scenarios over five years. At the heart of this study was a green field network design that determined a least-cost capacity deployment for the MPLS network. Here MASCOT was used to accurately quantify least-cost switching center designs. This accuracy was especially important in evaluating one of the evolution scenarios containing a converged MPLS core where an MPLS switching center carries large converged ATM and IP traffic flows, as well as in the later years of the evolution period when yearly traffic growth added up to a high traffic load in the switching center. In both cases there was a need to deploy a large number of switches with potentially high overhead that needed to be minimized realistically. In this example, we focus on the efficiency of two different router

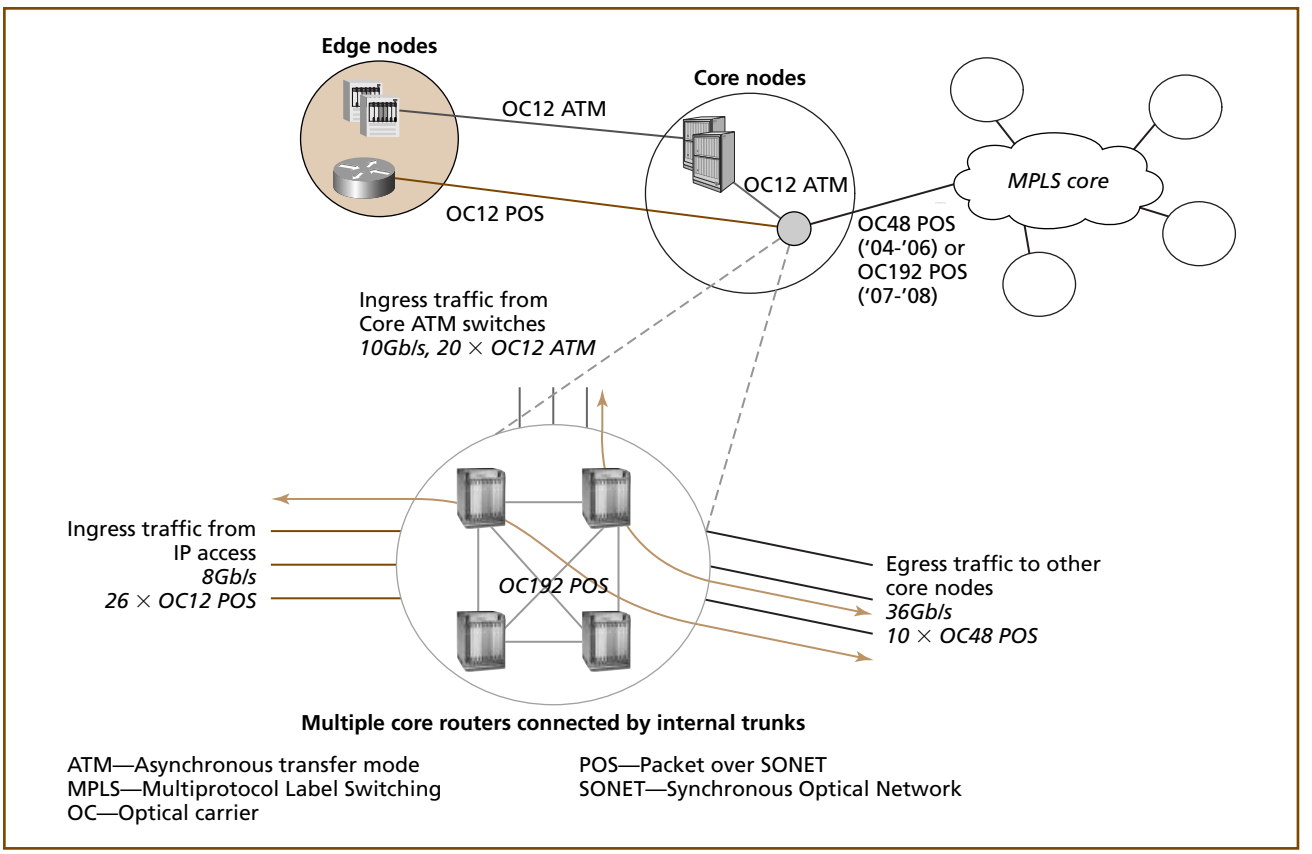
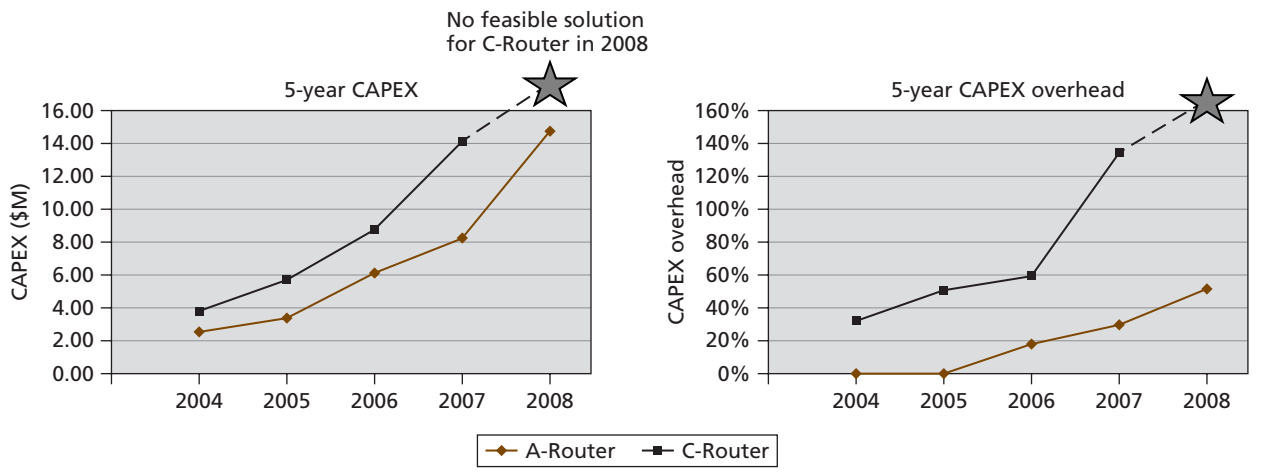


Figure 5.
Core node in an MPLS backbone network.

designs, referred to as A-Router and C-Router, in a typical core node in the proposed MPLS backbone network. The node and its network context are illustrated schematically in **Figure 5**. Here the switching center is defined to be a group of one or more MPLS routers inside the core node. The routers connect to other core nodes via high-rate POS interfaces, to edge nodes via low rate POS interfaces, and they connect to local ATM switches via low rate ATM interfaces. Designs based on the two routers are compared over a five-year period with a yearly traffic growth rate of 15% for ATM and 80% for IP; the numbers depicted in Figure 5 are representative of the first year of the study. **Figure 6** depicts some of the results of the study.

Throughout this particular study, A-Router represents a much more cost effective solution than C-router with less equipment space, less switching overhead, and a lower total equipment cost. Although the two routers are outwardly similar in many respects, the key difference turns out to be that the port density of OC12 ATM interface cards on C-Router is much lower than that of the corresponding cards on A-Router. In 2004, for example, constraints on the OC12 ATM ports force the use of three C-Routers as opposed to a single A-Router. As the number of switches increases, the internal switching overhead increases super-linearly as more and more switch pairs require internal trunks. By the end of the five-year period, it is no longer feasible to build a C-router switching center, unless the flows are allowed to take more than two hops within the center. By using MASCOT the service provider is able to accurately measure the switching overhead inside a core node and assess the cost effectiveness and scalability of two alternative routing platforms.

	2004		2005		2006		2007		2008	
Base load (Gb/s)	36.00		51.80		78.29		123.73		202.94	
	A-Router	C-Router	A-Router	C-Router	A-Router	C-Router	A-Router	C-Router	A-Router	C-Router
CAPEX ($M)	2.56	3.82	3.34	5.66	6.07	8.76	8.19	14.08	14.70	N/A
A-Router savings	33%		41%		31%		42%		N/A	
CAPEX OH ($M)	0.00%	41.65%	0.00%	77.84%	18.11%	87.86%	29.38%	134.47%	51.31%	N/A
SW OH -Load	0.00%	54.50%	0.00%	68.80%	31.00%	76.40%	50.50%	82.70%	67.70%	N/A
SW OH -OC192	0	6	0	10	4	18	8	34	20	N/A
Switch Chasses	1	3	1	4	2	5	2	10	4	N/A
Switch Racks	0.5	3.0	0.5	4.0	1.0	5.0	1.0	10.0	2.0	N/A
OC192	0		0		0		9		14	
OC48	10		14		21		0		0	
OC12	26		47		84		101		181	
OC12 ATM	20		23		26		30		35	



ATM—Asynchronous transfer mode CAPEX—Capital expenditure OH—Overhead SW—Switch

Figure 6.
Router comparison: five-year analysis.

Capacity Recovery Services

Although capacity recovery services have some principles in common with capacity deployment, they are also different in a number of ways. The most salient difference is that capacity recovery operates on an existing network carrying live traffic so that the adage "First, do no harm" applies. Secondly, the benefit of the capacity recovery is limited by the degree to which the existing network is inefficient. In order for the service to be beneficial, the costs of reconfiguration have to be small relative to the expected benefits. For this reason, capacity recovery services are usually targeted at the logical layer of the network. Examples include adapting optical layer wavelengths to match IP layer demands [1] or rerouting virtual circuits to relieve bottlenecks in ATM networks [2]. Switching center optimization is an exceptional case in which it can be cost effective to change the *physical* topology, because the affected switches and trunk terminations all share the same physical location. Another particular feature of capacity recovery is that the service must include an incremental series of steps to get from the existing configuration to the desired configuration. Because of the need to keep costs low, the number of steps and complexity of each step must be kept reasonably small. A fourth important feature

of capacity recovery is the need to obtain accurate data about the existing network. Network inventory services enable and can be motivated by capacity recovery services.

Procedures

Regardless of the type of network involved, a capacity recovery service based on switching center reconfiguration would be performed in the four basic steps described in the subsections below. The nature of the work involved in each step could vary depending on the underlying technology.

Data collection. The purpose of this step is to gather an accurate picture of the current state of the switching center. This includes an inventory of the relevant switching center hardware such as switches, processor cards, interface cards, and cables. The capabilities of each switch must be described in a switch configuration file. The internal and external trunks need to be identified. It is also important to list all flows traversing the center, to identify the local route of each flow, and to determine the capacity used by each flow. The definition of a flow and the difficulty of each of these items will vary depending on the underlying technology and on the network management system. For example, in a SONET/SDH network, a flow is a circuit. The capacity used by the flow is known and fixed, regardless of the amount of data actually on the circuit. In an ATM network, a flow consists of a permanent virtual connection (PVC), which may belong to one of several service categories. For a connection in the constant bit rate category, the capacity is given, but this is not the case for a connection in the unspecified bit rate (UBR) category. In practice, the vast majority of ATM connections fall into the UBR category, and hence the data collection in an ATM network must include a method for estimating the traffic load on groups of UBR connections. One such method was developed for the work described in [2].

Analysis. The first step in the analysis process is to choose metrics that will be used to compare the benefits obtained by proposed changes. Some natural metrics for switching center optimization include the capital costs (e.g., switches and interfaces), operating costs (e.g., power and footprint), and system load (e.g., switching capacity consumed, number of occupied ports). Next, optimization tools are used to identify new physical topologies that are close to the existing topology and that have improved performance under the chosen metrics. Currently the MASCOT tool uses switching capacity consumed as an optimization metric; many of the other performance metrics are correlated with this basic one. MASCOT also produces a sequence of steps which may be used to produce the new topology. The third analysis step consists of estimating the cost and effort required to perform each transition step and comparing these costs with the resulting benefits. The transition costs include tangible elements such as time and labor as well as intangibles such as the risk of network disruption. In the end, the analysis either produces a sequence of beneficial transition steps or concludes that no reconfiguration is warranted.

Transition. Network technicians at the switching center take a series of steps to rearrange the switch connectivity at the switching center, normally without interrupting the processing of the live traffic. At the physical level, the rearrangements involve unplugging external trunks from ports on a given switch and installing them on ports of another switch, as well as adding or removing internal trunks. At the management level, the operator must switch all of the flows on the affected trunk to alternate routes. Once the trunk has been moved and advertised to the network management system, a set of flows can be routed back onto the trunk. The number of flows moved in each step can be very large: in ATM networks for example, there may be tens of thousands of flows on a given trunk. For this reason, and to avoid errors, automated tools for moving connections are required.

Evaluation. The final step is to quantify the results of the reconfiguration. The data collection procedures are repeated to determine the new network state, and performance metrics are recomputed and compared with the initial and predicted values. For example, this step would confirm that certain switch ports have been freed, that the switch loads have been reduced by a given amount, and that congestion or other undesirable conditions have not been introduced.

Switching Center Optimization and Global Routing

In this work we have taken a local view of switching center optimization. This approach helps to keep the complexity of the optimization problem and data collection manageable and scalable. Complexity aside, it would clearly be preferable to take the entire network into account and to use a model that allows global rerouting of flows as well as switching center reconfiguration. Consider for example the network reconfiguration depicted in Figure 1, and assume that all flows initially used least-weight routing in the network depicted on the left side. Suppose also that the weight of each link has a fixed component in addition to any distance-based component—i.e., internal trunks have a non-negligible weight based on the real costs of the switch ports they use. Under the new configuration, the cost of the tan flow's route increases because it now traverses an internal link at center A. However, it may be possible to move it onto another route, not involving switching center A, whose cost is less than the new route (B.1, A.1, A.2, D.1), though necessarily still greater than the cost of the original (B.1, A.1, D.1). Conversely, some flows which originally did not use switching center A may now find it advantageous to do so because of the reduced cost of the path (B.1, A.1, C.1) relative to the original cost of the path (B.1, A.1, A.2, C.1). The overall effect in this example is that the match of the traffic to the physical topology is further increased.

One scalable approach to switching center optimization in a network context would be to alternate switching center steps with global rerouting steps. In the first step, current routing information provides the input data for improving the topology of a single switching center. In the second step, the routing is improved to take advantage of the new topology. The two steps are iterated, potentially cycling among different sets of switching centers, until no further improvement is possible. Convergence can be assured by requiring that a single metric (e.g., the total load on all network switches) is improved at each step. Note that the iteration procedure would be applied as part of the analysis phase. In the transition phase, the goal would be to move to the final solution as economically as possible, without necessarily visiting the intermediate stages generated by the analysis.

Conclusions

Switching center optimization can be used to improve the existing capacity of existing switching centers and to design new centers. The MASCOT switching center optimization tool has been used successfully in design applications, and it shows promise for capacity recovery applications.

The analysis phase of switching center capacity recovery requires good information about the current hardware and traffic, while the implementation phase requires the ability to safely move traffic flows from one trunk to another. For this reason, network inventory services and capacity recovery services based on rerouting can be thought of as prerequisites to switching center capacity recovery. By the same token, switching center optimization can help to extract the full benefit from those services.

Acknowledgments

We would like to acknowledge valuable contributions by Iraj Saniee and Mohcene Mezhoudi from Bell Labs and Naeem Asghar, Colin Corcoran, Joseph Karwisch, Stephen Zlatos, and Michael Siesta from Lucent Worldwide Services.

References

[1] S. Acharya, Y. Chang, B. Gupta, P. Risbood, and A. Srivastava, "Architecting Self-Tuning Optical Networks," Proc. European Conference on Optical Commun., (Copenhagen, Den., 2002).

[2] N. Asghar, R. Bhatia, R. Chandwani, C. Corcoran, F. Hao, J. Karwisch, P. Koppol, T. V. Lakshman, S. Zlatos, and M. Siesta, "iOptimize: A Software Capability for Analyzing and Optimizing Connection-Oriented Data Networks in Real Time," Bell Labs Tech. J., 9:4 (2005), 67–81.

[3] O. Goldschmidt and D. S. Hochbaum, "A Polynomial Algorithm for the Kappa-Cut Problem for Fixed Kappa," Mathematics of Operations Research, 19:1 (1994), 24–37.

[4] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," Bell Sys. Tech. J., 49:2 (1970), 291–308.

[5] H. Saran and V. V. Vazirani, "Finding k-Cuts Within Twice the Optimal," SIAM J. Computing, 24:1 (1995), 101–108.

(Manuscript approved September 2004)

NACHI K. NITHI is a member of technical staff in the

Mathematics of Networks and Systems Department within the Mathematical Sciences Research Center at Bell Labs in Murray Hill, New Jersey. He earned a B.E. in electrical engineering from Madras University in Chennai, India, an M.E. in computer science from Anna University, also in Chennai, and a Ph.D. in computer science from Colorado State University in Fort Collins. Dr. Nithi's current interests are in design tools and algorithms for network optimization and switching center design. He is also interested in simulations, data analysis, and Web technologies.

CARL J. NUZMAN is a member of technical staff in the

Mathematics of Networks and Systems Department at Bell Labs in Murray Hill, New Jersey. He received B.S. degrees in electrical engineering and mathematics from the University of Maryland in College Park and a Ph.D. in electrical engineering from Princeton University in New Jersey. Dr. Nuzman has broad research interests in the areas of network optimization, optical network modeling, traffic characterization, and stochastic processes.

BENJAMIN Y. C. TANG is a distinguished member of

technical staff in the Data and Broadband Access Network Modeling Group at Bell Labs in Holmdel, New Jersey. He has a B.S. degree from the National Taiwan University in Taipei, an M.S. from the University of Florida in Gainesville, and Ph.D. from Purdue University in West Lafayette, Indiana, all in electrical engineering. Currently, Dr. Tang's work focuses on next-generation packet core, broadband access and converged network solutions, network evolution planning, ATM, and IP/MPLS network design and optimization. Many of his works involve joint study and development of network evolution strategy for major carriers in China, Asia Pacific, and North America. His areas of interest include all aspects of data networking, broadband access, network design and optimization algorithms, and economic analysis. ◆