Reprint of paper 15.7 in Proceedings of IEEE ICC '98, Altanta, June 7-11, 1998. pp. 537 - 543 Patents Pending

Cycle-Oriented Distributed Preconfiguration:
Ring-like Speed with Mesh-like Capacity for Self-planning Network Restoration

Wayne D. Grover, *Senior Member IEEE*, Demetrios Stamatelakis, *Member IEEE*
TRLabs c/o Dept. of Electrical and Computer Engineering, University of Alberta

ABSTRACT: Cycle-oriented preconfiguration of spare capacity is a new idea for the design and operation of mesh-restorable networks. It offers a sought-after goal: to retain the capacity-efficiency of a mesh-restorable network, while approaching the speed of line-switched self-healing rings. We show that through a strategy of pre-failure cross-connection between the spare links of a mesh network, it is possible to achieve 100% restoration with little, if any, additional spare capacity than in a mesh network. In addition, we find that this strategy requires the operation of only two cross-connections per restoration path. Although spares are connected into cycles, the method is different than self-healing rings because each preconfigured cycle contributes to the restoration of more failure scenarios than can a ring. Additionally, two restoration paths may be obtained from each pre-formed cycle, whereas a ring only yields one restoration path for each failure it addresses. We give an optimal design formulation and results for preconfiguration of spare capacity and describe a distributed self-organizing protocol through which a network can continually approximate the optimal preconfiguration state.

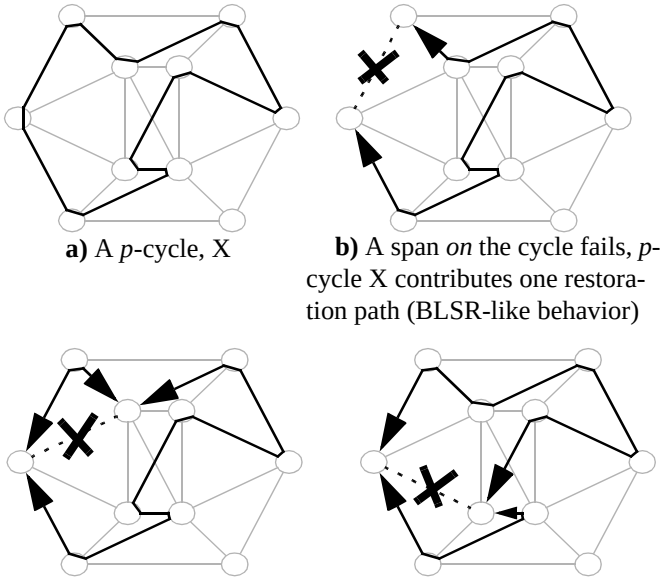
1. Introduction

Protocols for the real-time restoration of mesh transport networks have been studied for several years. Investigators generally agree that it is feasible for such protocols to *compute* a set of replacement paths for span restoration in under 2 seconds, which is fast enough to avoid large scale consequences within the network [1,2]. However, there is still a significant motivation to increase the speed of mesh-based restoration schemes [3,4]. Indeed, some network operators insist that ring-like restoration speeds (50 - 150 ms) are needed for ATM services.

With this motivation, we have developed a strategy for network operation that should make it possible to achieve ring-like restoration times while retaining the desirable capacity efficiency of the mesh-restorable alternative. The method is based on the formation of pre-configured cycles, called *p-cycles*, formed out of the previously unconnected spare links (e.g., STS-1s or STS-3s) of a mesh-restorable network. Despite similarity to rings in the use of a cycle-topology, *p*-cycles emulate ring behavior for a certain class of failures, but also go beyond the functionality of rings, to more widely protect the network as a whole, as in a mesh restoration scheme. The most important real-time advantage is that only two DCS nodes have any real-time cross-connection workload for any given failure. Also, the end nodes only have to effect end-node traffic-substitution connections which are functionally equivalent to switching in rings.

A. The Cycle Preconfiguration Concept

Cycle-oriented preconfiguration remains fundamentally a *mesh* restorable network technology in terms of its capacity efficiency and in its functional differences from self-healing rings. Figure 1 illustrates the way in which an individual *p*-cycle may be used for restoration, in ways that are not possible with rings. In Fig. 1(a) an example of a *p*-cycle is shown. In (b), a span *on* the cycle breaks and the surviving arc of the cycle is used for restoration. This action is like a unit-capacity bi-directional line-switched ring (BLSR). In (c) and (d), however, we see how the *p*-cycle can also be accessed for restoration of working paths that are *not* on the cycle. In fact, cases (c) and (d) are the more advantageous in general, because *two* restoration paths are obtained from each *p*-cycle for such failures. In contrast, a ring can provide at most one restoration path to any failure and protects only against failures on the spans of the same ring. Further inspection of the example in Fig. 1 shows in fact that this particular *p*-cycle provides a single restoration path for nine *on*-cycle failures and *two* restoration paths, each, for *ten* other "straddling" span failures (the latter are for each of those spans which are not on the cycle, but straddle it).



a) A *p*-cycle, X

b) A span *on* the cycle fails, *p*-cycle X contributes one restoration path (BLSR-like behavior)

c) A span *off* the *p*-cycle fails, *p*-cycle X contributes two paths (mesh-like)

d) A span *off* the *p*-cycle fails, *p*-cycle X contributes two paths (mesh-like)

FIGURE 1. Use of *p*-cycles in restoration

Table 1 summarizes these and other comparative aspects of the *p*-cycle concept, as distinct from rings. *p*-cycles are formed from individual spare links (or channels) of the point-to-point OC-n systems present, whereas SONET rings commit a full OC-n's worth of working and spare capacity to the same cycle. As shown in Fig. 1, each *p*-cycle can contribute one or two paths to a wider range of restoration scenarios than a ring. While rings have a structural association between the working demands which they protect and the protection bandwidth in the same ring, *p*-cycles are formed only within the spare capacity layer of

contact: Wayne Grover phone 403 441-3815 , email: grover@trlabs.ca *Patents applied for*

the network, leaving the working paths to be routed freely on shortest paths, or any other route desired. In other words, the working demands may be provisioned as if in a point-to-point mesh network and the *p*-cycles are formed in the sparing layer to adapt for maximal protection of this working path layer. A deployed *p*-cycle design may also be easily modified by the DCSs used to form and sustain it. In contrast, a ring, once deployed, is essentially hardwired in place within the network.

TABLE 1: Comparison of the *p*-cycle and Ring Technologies

Attribute	*p*-cycles	SONET rings
Modularity	One spare link per span	OC-n or OC-n modularity
Protection Yield	Up to two useful restoration paths per *p*-cycle	One restoration path unit per protection channel
Protection Flexibility	*p*-cycles contribute to the restoration of working links on the cycle and all cycle-straddling links	Rings only protect working links in spans contained within the ring
Routing and provisioning of working paths	Proceeds without regard to structures formed in the sparing layer	Working path routing must be by a succession of intra-ring and inter-ring traversals
Total Network Redundancy	Essentially just that of a span restorable mesh network	Over 100% investment in spare capacity. Up to 300%.

In the last row of Table 1, we allude in advance to one of the key findings of this study: that a *p*-cycle spare capacity design takes little or no more capacity than a corresponding span-restorable mesh network. As is well known, a mesh restorable network can have substantially less than 100% capacity redundancy while BLSRs typically embody well over 100% spare to working capacity ratio. The simplicity of the real-time switching for restoration with *p*-cycles, combined with virtually the capacity efficiency of a mesh design, is the reason we say this is a path to "the speed of rings, with the efficiency of mesh."

The *p*-cycle concept addresses the primary speed limitation of mesh-based restoration. In the *p*-cycle approach, all the time needed for calculating and connecting restoration paths (in the form of *p*-cycles) is invested by a self-organizing process in non-real time, before any failure occurs. Real time restoration speed is then determined solely by the time needed for the two end-node DCS to do signal bridging and receive selection operations that are no more complex than in a BLSR. Moreover, as will be seen, each node learns in advance exactly which port-to-port connection will be needed for each prospective failure. Also, there is no (fundamental) need for real-time signalling between end nodes to effect the restoration switching.

B. Outline of paper

We proceed as follows: The rest of Section 1 defines some needed terminology. Section 2 then presents theory for optimal spare capacity design with *p*-cycles and compares the total capacity results to equivalent conventional mesh restorable designs. Section 3 then addresses the desire to avoid dependence on centralized control for deployment and maintenance of a the *p*-cycle state for a network. This is done through a distributed self-planning protocol, which we outline, followed by performance results obtained to date with OPNET simulation.

C. Definitions

We use *link* to denote an individual bidirectional digital carrier signal between adjacent nodes. In general the link is whatever unit of capacity the DCS manipulates for transport management and restoration. For instance, a DS-3 or STS-1. A *span* is the set of all working and spare links in parallel between adjacent nodes, whether on one or several OC-n systems in parallel. A *useful path* is a path segment contained in a *p*-cycle which is related to a failure span in a manner that allows it to contribute to restoration of a given failure.

2. Optimal Design of *p*-Cycle Restorable Networks

A linear integer program (IP) was formulated [5,6] for the design of *p*-cycle based restorable networks. First, the set of all simple distinct cycles up to some limiting size is generated from the network topology. The IP then generates an optimal *p*-cycle plan by choosing the number of copies of each elemental cycle on the network graph, to be configured as a *p*-cycle. Two variations were developed and tested [6]: the first designs a *p*-cycle plan within an existing mesh network spare capacity plan. It attempts to maximize the *p*-cycle restorability with a given amount and placement of spare capacity, such as from an existing mesh restorable design. The second formulation designs a fully restorable *p*-cycle spare-capacity plan while minimizing the total amount of spare capacity. The latter formulation is the one used for the present results, and is detailed in the Appendix.

We studied both of these problems in the test networks detailed in Table 2. In Table 3, the efficiency of optimal *p*-cycle

TABLE 2: Properties of the Test Networks

Net	Working Links	Spare Links	Number of Nodes	Spans
Net1	142	48	10	22
Net2	1404	800	15	28
Net3	4369	3124	20	31
Net4	27522	25232	30	59
Net5	2191	2049	53	79

spare capacity designs for these networks is compared to corresponding optimal spare capacity designs for conventional span

restorable mesh networks using the methods from [7]. "Excess Sparing" is the percentage, if any, of total spare capacity that the *p*-cycle design required above the conventional mesh spare capacity design. Both types of design in Table 3 are 100% restorable against any single span cut.

TABLE 3: Spare Capacity required for fully restorable *p*-cycle designs, relative to span-restorable mesh

Net	% Excess Sparing	total # of *p*-cycles (# distinct p-cycles)
Net1	9.09	5 (5)
Net2	3.07	88 (10)
Net3	0.0	250 (10)
Net4	2.38	2237 (27)
Net5	0.0	161 (39)

The data in Table 3 drive home the point that despite using a cycle as the graph theoretic building-block, as with rings, the *p*-cycle principle is essentially mesh-like in its efficiency. Thus, at least with an optimally designed set of *p*-cycles, we can put the network in a fully restorable pre-configured state, with little or no additional spare capacity than in a conventional mesh-restorable network.

3. Self-organization of the *p*-cycle state

Here we give an overview of the self-organizing strategy we have developed for the autonomous deployment and continual adaptation of the network cycle preconfiguration state. The *Distributed Cycle PreConfiguration* (DCPC) protocol is an adaptation of the statelet processing rules of the Selfhealing Network (SHN^{TM}) protocol [1,2]. A statelet is embedded on each spare link and contains a number of state fields. Each logical link, as viewed by a node attached to it, has an incoming statelet and outgoing statelet. An incoming statelet arrives at a node on a link, and originates from the adjacent node connected through the link.

As in the SHN, each outgoing statelet has an incoming statelet which forms its precursor. An incoming statelet is a precursor to an outgoing statelet if the incoming statelet was cause, under the protocol rules, for the creation of the outgoing statelet. One incoming statelet can be the precursor for many outgoing statelets but each outgoing statelet can have only one precursor.

As a family of statelets is broadcast through a network, it forms a statelet broadcast tree which, at each node in the tree, is rooted at the precursor port from which the outgoing statelets are propagated. The particular chain of causal events from the Sender through to the present node is called the statelet route.

There are only two node roles in the DCPC. A combined sender / chooser role called a "Cycler" and a Tandem node. The Cycler sources and later receives parts of the statelet broadcast pattern it initiates. Each node adopts this role in a round-robin fashion. While in this role it is temporarily in charge of the cycle-exploration process within the network as a whole. When not in the cycler role, each node plays a Tandem-node role which mediates the statelet broadcast competition, as in the SHN, but with a new decision criterion. At a high level of description, the DCPC first allows each node to explore the network for *p*-cycle candidates that are discoverable by it. After completion of its exploratory role as cycler (detailed below), it hands off to the next node in order by a simple "next-node hand-off" flood-notification. After all nodes have assumed the role of the cycler once, each "posts" its best found cycle in a distributed network-wide comparison of results. In this step all nodes hear the metric, and other details, of the globally best *p*-cycle candidate discovered by any of their peers. The competition flood expands through the network as each node locally relays the statelet with the best cycle metric, or asserts its own best if and while the latter is still superior to anything else it has received notice of yet. Eventually, the globally best cycle candidate dominates everywhere. Upon thus learning of the winning candidate, the Cycler node who discovered this *p*-cycle, goes on to trigger its formation as a *p*-cycle. All nodes on the *p*-cycle update their local tables of restoration switching pre-plans to exploit the new *p*-cycle. The whole process then repeats, spontaneously, without any central control, adding one *p*-cycle per iteration until a complete deployment of near-optimal *p*-cycles is built. Thereafter, it continually adapts the *p*-cycle set to changes in the working capacity layer.

A. Statelet Format

The DCPC statelet format has 5 main fields:

- *index*: Each statelet belongs to an index family. Any outgoing statelet has an index value that is inherited from the incoming statelet which is currently its precursor.
- *hopcount*: As a statelet is relayed from node to node, a count of the number of hops it has taken is maintained.
- *sendNode*: All statelet broadcast trees originate from only one node at a time. This is the current cycler node, which asserts its name in this field.
- *numpaths*: This is the accumulating figure of merit for prospective *p*-cycles that are represented within a statelet broadcast. It contains the apparent number of useful paths which the *p*-cycle candidate, contained in a given statelet, can provide (details follow in this section.)
- *route*: This field contains the route, originating at the Cycler node, which a certain branch of a statelet broadcast tree represents between the Cycler and the current node.

B. The Tandem Node

The bulk of the processing in the DCPC algorithm takes place in the Tandem nodes. The Tandem node rules determine what *p*-cycle candidate the Cycler node will discover in a given round of global cycle comparison and formation. A Tandem node will broadcast each incoming statelet to the largest extent warranted by the statelet's *numpaths* score within the context of the available outgoing link resources and other statelets cur-

rently present. If all outgoing spare links on a span are occupied, a new incoming statelet can displace an outgoing statelet if it has a *numpaths* score better than the precursor with the lowest current score. Also, statelets on a given index can only be forwarded to adjacent nodes which are not already present in the accumulating route of the corresponding precursor. The single exception to this rule is that a statelet may be broadcast from a Tandem to the Cycler node, which is present in all route fields. Figure 2 shows an example of this behavior, which limits the cycle exploration and formation process to consider only simple cycles. Additionally, at most one outgoing statelet of a given index may appear on a span. If multiple incoming statelets, of like index exist at a node, then the statelet with the best *numpaths* score becomes precursor for all outgoing statelets of that index. The emergent effect of these rules is that, shortly after triggering the process, with a sender primary flood, the Cycler receives incoming statelets whose route fields trace out cycles which begin and terminate at the Cycler node.

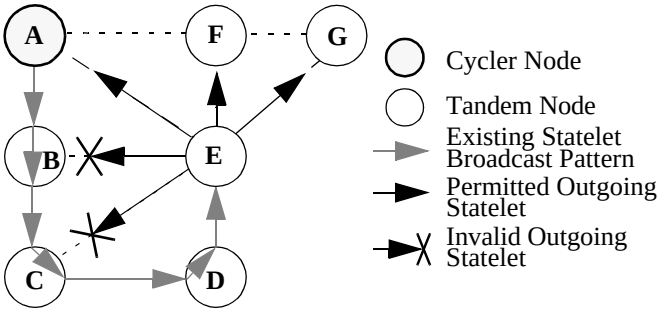


FIGURE 2. An Example of the Tandem Node Broadcast Rule that Limits the Cycles Generated to Simple Cycles

Now we cover the Tandem node statelet competition rules. The idea is to identify the best prospective *p*-cycles. However, the Tandem node view is local only to the links directly connected to itself. Thus, a propagating metric of some type needs to be embedded and updated in each statelet so that the side-effect of Tandem node competition is the generation of "good" *p*-cycle candidates. The metric or score that is used is intended to represent the potential of an incoming statelet's route to form a *p*-cycle with a high ratio of useful paths to spare links consumed. The conundrum, however, is that the Tandem nodes must try to assess this metric before any complete cycle route has actually been formed.

To do this, the Tandem node rules operate on the presumption that any index-tree branch may eventually succeed in closing again with the Cycler, and evaluate it for useful paths on this basis. A statelet's score is $s = (numpaths)/(hopcount)$ where *numpaths* is the number of useful paths that would be provided by a cycle formed from the union of the incoming statelet's route and an imaginary direct span joining the tandem node to the cycler node. *Hopcount* is the number of spans so far traversed in the statelet's route. The number of useful paths, *numpaths*, is updated incrementally by each Tandem node as illustrated in Fig. 3. For each span on the route, *numpaths* is increased by one. *Numpaths* is increased by *two* for each node that appears in the route list and which the current Tandem node has a direct span connection, other than the span on which the current statelet has arrived. In other words, for spans that would have a straddling relationship to the prospective *p*-cycle.

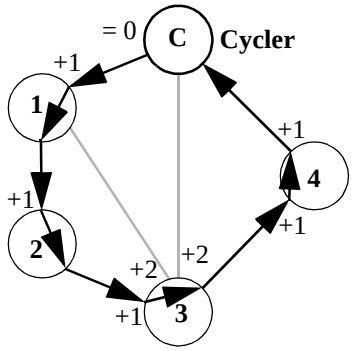


FIGURE 3. Evaluation of *numpaths* by the Tandem Node

C. The Cycler Node Role

All statelet family broadcasts originate at the cycler node. To initiate the cycle-exploration process, the cycler places an outgoing statelet on one spare link in each span at its site. Each of these primary statelets has a unique index number. After the primary statelet broadcast, the cycler node invests a pre-determined time in sampling of the returning statelets. As returning statelets arrive, the cycler maintains a record of the received statelet (and statelet route) with the best score, *s*, as above. The Cycler persists in observing the incoming signatures because a cycle tends to provide a higher number useful paths as it is allowed to evolve under the collective interactions of the Tandem nodes. Usually it grows in size as it improves its score, until hopcount limiting effects stabilize the pattern of cycle-candidates formed. Fig. 4 is an illustration, from simulation, of how one prospective *p*-cycle evolves, usually outward, improving its score with time.

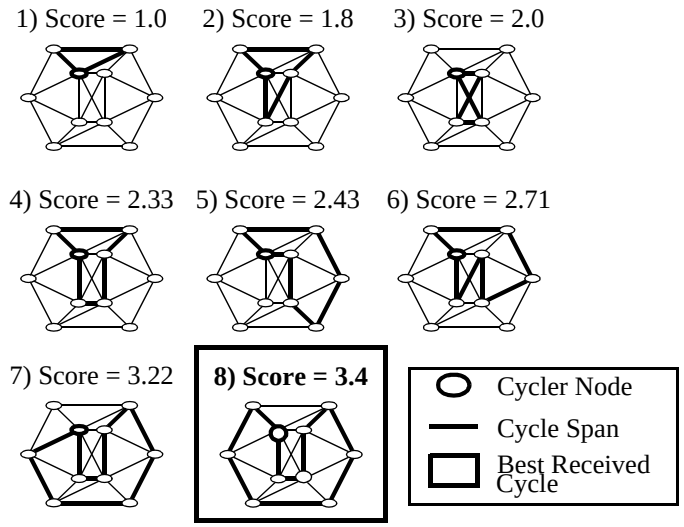


FIGURE 4. Evolution of the *p*-cycle candidate score.

The sampling periods in our simulations are at most 1/3 of a second. But even if a few seconds was allowed for the best *p*-cycle candidates to emerge in a large network, there is little issue because the process is running in non-real-time. It is running in anticipation of a span failure, not in response to one, so

there is no problem with investing this time to observe the evolution of the cycle metrics under the Tandem node actions. When the sampling time runs out, the cycler suspends all primary statelet broadcasts, terminates its role as the Cycler, and emits a Cycler hand-off flood (a statelet with op-code "hand-off" and the node name, on one link of each span.) The hand-off flood is relayed (once only by all nodes, without link persistence, with one copy in each span). When node *n* hears "hand-off flood, *n-1*" it knows that it is its turn to become cycler.

D. Construction of the Best Candidate p-Cycle

After a complete round of cycler action by every node, the last node in the sequence (i.e., when *n* equals the number of network nodes) knows all nodes have assumed the role of the cycler and are ready to take part in a network wide comparison of results. The purpose is to find the one globally best *p*-cycle candidate found by any cycler node. This is done automatically by initiation of a global comparison flood by the last node in the sequence of cycler exploratory phases. The initiating node broadcasts a single statelet, containing the node's name and its best cycle's score, on each span. When adjacent nodes receive such a statelet they compare the received best score to their local best score and relay the better of the two into all spans, along with the name of the node who is reporting the better cycle. If scores are equal, precedence is based on ordinal rank of the node names involved. Rapidly, only the single best score is present everywhere, and the node which found this candidate will proceed to initiate its construction.

To deploy the *p*-cycle that has emerged from the all-nodes comparison of results, the node associated with the winning candidate cycle examines the route field of that cycle and identifies the node adjacent to itself which appears first in the route vector. It then finds a spare link on the span to that node and places a statelet with a "construct-cycle" op-code, followed by the route vector. The adjacent node makes a cross-connection between the incoming spare link bearing this statelet, and a spare link in the direct span going to the next node in the route vector. It then forwards the cycle-constructing statelet on that spare link; subsequent nodes effect a similar connection and relay the construction command in a similar manner. As each node along the route makes its cycle-constructing cross-connection, it also updates its local list of uncovered working links, and notes all of the working links for which the current *p*-cycle can be used for restoration (i.e., any working link from this site to any of the other nodes listed in the route vector). These considerations ready the node to use the *p*-cycle immediately for restoration. They also are reflected in subsequent cycle-exploring iterations so that future *numpaths* measured are scored accurately, given the reduction in uncovered working capacity that each constructed *p*-cycle creates.

When the sequence of relays that constructs the *p*-cycle returns to the initiating Cycler node, that node makes a final crossconnection to the first spare link on which it began the cycle-building process, completing the *p*-cycle. Once deployed, any node on the *p*-cycle may use the cycle for restoration. The only further special role for the custodial node for this cycle is to apply and maintain a statelet into it that repeats the route vector. The *p*-cycle is thus put into storage with a holding statelet on it that support continual self-checking of the continuity and correctness of the cycle route while in storage by the nodes on it. To use any *p*-cycle for restoration, any node on the cycle must only first test for in-use status (marked on the holding statelet), assert its own in-use indication (assuming its free), and bi-directionally substitute the affected signal to go on this *p*-cycle. Thus, the real time restoration procedure, once *p*-cycles are in place, reduces to being identical in this regard to the SONET BLSR standard.

E. OPNET DCPC Simulation: Functional Operation

The DCPC protocol was developed and tested by OPNET [8] simulation in the five test networks of Table 2. Each simulated network was run with statelet insertion delays for a 64 kb/s SONET line-overhead byte, propagation delays of 0.7c, and a nodal processing time of 1 ms per statelet event. The sampling intervals used in the DCPC protocol were 0.1, 0.2, 0.2, 0.25 and 0.3 seconds for Nets 1 to 5, respectively. Fig. 5 shows, for illustration, the five *p*-cycles created by the process in Net1. These five *p*-cycles comprise a complete and near-optimal restorability plan for this network.

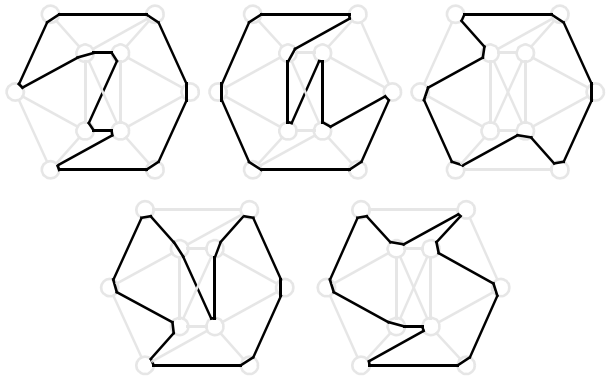


FIGURE 5. Example of self-organized cycles in Net1

Fig. 6 is a simulation trace portraying overall operation of the DCPC process, using Net 1 for its simplicity of illustration. The plot shows the score of the best *p*-cycle candidate seen by any cycler, versus simulated real time. Six main regions appear in the plot, corresponding to the 5 *p*-cycles formed by DCPC for Net 1, plus a sixth iteration to realize the halting criterion. Each of the larger regions correspond to the all-nodes cycler search, comparison, and construction of a single *p*-cycle. Inside each region, there are 10 individual cycler node explorations and next-node hand-off floods (Net 1 has 10 nodes).

The apparent dead time between these main regions is when all nodes are involved in the flood comparison of their individual results, to see which node will construct the *p*-cycle for this iteration. As the process goes on, the best scores found by any cycler decreases as uncommitted spare link counts reduce and as the coverage level rises, making it harder to discover high score *p*-cycles. In the last region of the plot, no node finds any feasible *p*-cycle candidates and the protocol terminates. In real

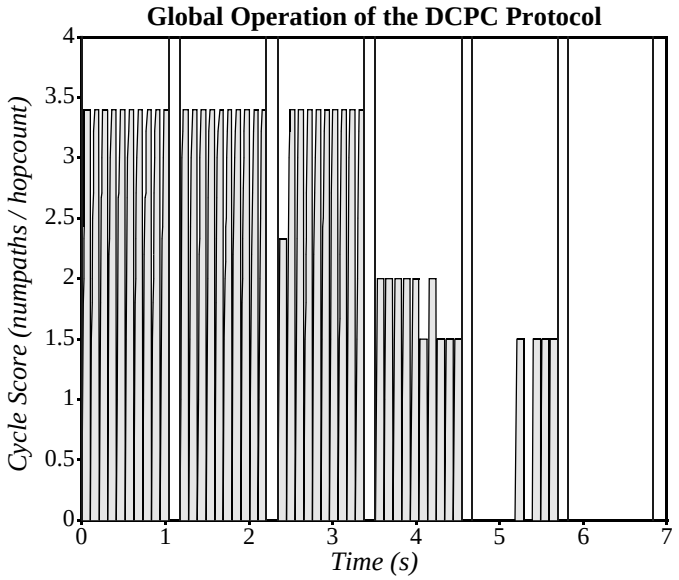


FIGURE 6. Best cycle score vs. time in Net1

operation, the protocol would then repeat continually to confirm its set of *p*-cycles or enter a mode of successive refinement and adaptation using very similar procedures.

F. OPNET DCPC Results: Restorability Performance

DCPC protocol performance is assessed in terms of the restorability level achieved within a theoretically minimal spare capacity environment. To do this, the test networks were provisioned with working capacity on each span (w_i) as generated by shortest-paths routing of the demand matrix. Spare capacity was then placed on each span (s_i) according to the IP solution for optimal *p*-cycle design. This means that the trials of the DCPC protocol were undertaken in the presence of the absolute minimum of spare capacity within which any method can achieve 100% restorability through *p*-cycles. Because the DCPC protocol is a self-organizing approximation to the strictly optimal *p*-cycle design, it cannot be expected to show 100% restorability in all cases under these most stringent of theoretical test conditions. Nonetheless, at this early stage in development, the *p*-cycle restorability levels being achieved, solely through self-organizing network action, are quite high relative to the centralized IP optimal solution. This is the data of Table 4.

TABLE 4: Performance in minimal-spare test networks

Network	*p*-cycle Restorability(%)	2-step Restorability (%)
Net1	100	100
Net2	100	100
Net3	90.94	97.16
Net4	89.16	97.68
Net5	83.75	95.44

Table 4 indicates that even in the worst of these tests against a stringent theoretical benchmark, one would obtain ring-like restoration speed for 83.75% or more of affected demands and, by then triggering a follow-up real-time restoration protocol such as the SHN, a final restorability level of 95 to 100% would be reached. This is the "2-step restorability" referred to above. This is the final restorability level achieved if, after first exploiting all useful *p*-cycles, one follows up with conventional execution of the Selfhealing protocol (or any k-shortest paths restoration process) on-demand for the remaining unrestored demands. This step assumes using any non *p*-cycle residual spares and breaking up other *p*-cycles as needed. The percentage figures in Table 4 are the average restorability over all possible span cuts. Most individual span cuts would still be seeing 100% restorability. These levels are all high enough, given the stringency of this test regimen, to suggest in practice that only small additional amounts of spare capacity are needed to bring the operational *p*-cycle restorability to 100%. Characterization of these top-ups, above the strictly optimal design, as well as further development of the DCPC itself, are areas of ongoing work. We think, however, that based on the results to date, the conceptual viability of a completely self-organizing 100% *p*-cycle restorable network is well demonstrated.

In practice, modularity is one source of additional spare link quantities, if we view the unused headroom in the n^{th} OC-n module on a span as additional spare link capacity. To see how much this effect alone might have on DCPC restorability levels, we modularized the total (w_i+s_i) link quantities in each of the test networks to the nearest OC-12, -24 or -48 module size. The module size for each network was that which minimized the mismatch to average (w_i+s_i) span quantities. The additional sparing amount $s_i^* = k_i n - (w_i+s_i)$ is effectively added to the restoration design for each span by this process, where k_i is the number of OC-n modules placed on span *i*. The change in the results simply due to modularity effects are shown in Table 5.

TABLE 5: DCPC in modularized minimal networks

Network	Modularity	*p*-cycle Restorability (%)	2-step Restorability (%)
Net3	OC-24	91.53	98.49
Net4	OC-48	100	100
Net5	OC-12	95.07	100

4. Concluding Discussion

The practical significance of this work is that one may be able to effect mesh restoration with the speed of a BLSR. The key is the realization of a way to plan for the placement of spare links in unit-capacity cycles on the network graph, so that 100% restorability is attained. Restoration is performed by simply breaking into these cycles and substituting traffic at failure time. This drastically simplifies the restoration protocol since only the

end nodes of a failed span need to act to substitute traffic, no real time signalling is required to or amongst other network nodes, and the end nodes know in advance exactly which port-to-port switch-overs are needed for each prospective failure. For these reasons, it may be possible to get DCS-based restoration switching times down to the level of 50 -100 ms, for ring-like speed. At the same time, however, this remains categorically a mesh restoration technology, as evidenced by the spare capacity results which are only slightly greater than the sparing in an optimal span mesh-restorable network. This simplicity of restoration switching, combined with capacity efficiency, arises because a *p*-cycle can be accessed like a ring but, unlike a ring, it can contribute up to 2 restoration paths per *p*-cycle. Continuing work is improving the network-level self planning performance of the DCPC. In sum, cycle-oriented preconfiguration of spare capacity may be a technological enabler for restoration with the speed of rings while retaining the capacity efficiency of a span restorable mesh network. This work is subject to patent pending [9].

References

[1] W. D. Grover, B. D. Venables, M. H. MacGregor and J. H. Sandham, "Development and performance verification of a distributed asynchronous protocol for real-time network restoration," *IEEE J-SAC*, Vol. 9, No. 1, pp. 112-125, January 1991

[2] W. D. Grover, "Method and apparatus for self-healing and self provisioning networks," U.S. Patent No. 4,956,835, 1990.

[3] SR-NWT-002514: "Digital cross-connect systems in transport network survivability," *Bellcore*, 1, January 1993.

[4] T.-H.Wu, H. Kobrinski, D. Ghosal, T. V. Lakshman, "The impact of SONET digital cross-connect system architecture on distributed restoration," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 1, pp. 79-87, January 1994.

[5] M. H. MacGregor, W. D. Grover, K. Ryhorchuk, "Optimal spare capacity preconfiguration for faster restoration of mesh networks," *Journal of Network and Systems Management*, Vol. 5, No. 2, pp. 159-171, 1997.

[6] D. Stamatelakis, *Theory and Algorithms for Preconfiguration of Spare Capacity in Mesh Restorable Networks*, M. Sc. Thesis, University of Alberta, Spring 1997.

[7] R. Iraschko, M. H. MacGregor, W. D. Grover, "Optimal Capacity Placement for Path Restoration in Mesh Survivable Networks," *Proc. ICC'96*, pp. 1568-1574, 1996.

[8] MIL 3 Inc., *OPNET Modeller Manuals V2.5*, 3400 International Drive NW, Washington, DC 20008.

[9] D. Stamatelakis, W.D. Grover, "Distributed Preconfiguration of Spare Capacity in Closed Paths for Network Restoration," U.S. Patent Pending, July 11, 1997.

[10] D.B. Johnson, "Finding all the elementary circuits of a directed graph," SIAM J. Comput., v. 4, 1975, pp. 77-84.

Appendix: Optimal *p*-cycle network design

Here we present an IP formulation for *p*-cycle restorable network design. The formulation determines the set of *p*-cycles which minimize the total amount (or cost) of spare capacity subject to a constraint of 100% *p*-cycle restorability. *P* is the set of distinct elementary cycles on the network graph. We used an algorithm by Johnson [10] for this step, subject to a limit on the maximum cycle length from 10 to 25 depending on the network. Cycles may cross over themselves span-wise (as seen in Fig. 4) but cycles that "figure 8" through the same node are not allowed. (The latter are implicitly treated in terms of their two elemental sub cycles.)

S is the number of network spans. s_j and w_j are the number of spare and working links on span *j*, respectively. n_i is the number of copies of cycle *i* in the *p*-cycle design. $x_{i,j}$ is the number of restoration paths that an instance of *p*-cycle *i* provides for failed span *j*. $p_{i,j}$ is the number of spare links required on span *j* for an instance of *p*-cycle *i*. c_j is the cost of a unit capacity on span *j*.

The objective function is:

$$\text{minimize} \quad \sum_{j=1}^{S} c_j s_j \qquad \text{(EQ 1)}$$

Subject to:

$$s_j = \sum_{i=1}^{|P|} p_{i,j} \cdot n_i \qquad \forall j = 1, 2, \ldots, S \qquad \text{(EQ 2)}$$

$$w_j \le \sum_{i=1}^{|P|} x_{i,j} \cdot n_i \qquad \forall j = 1, 2, \ldots, S \qquad \text{(EQ 3)}$$

$$n_i \ge 0 \qquad \forall i = 1, 2, \ldots, |P| \qquad \text{(EQ 4)}$$

The coefficients $x_{i,j}$ and $p_{i,j}$ are evaluated for each cycle in *P*, prior to the execution of the IP. $p_{i,j}$ is 1 if cycle *i* passes over span *j*; otherwise it is 0. $x_{i,j}$ can be either 0, 1 or 2 . It is zero if either of the failed span end nodes are not on the cycle. It is 1 if both of the span end nodes are on the cycle and the end-nodes of the failure are also adjacent to one another along the cycle. It is 2 if both of the failure span end-nodes are on the cycle but are not adjacent to one another on the cycle, i.e., if span *j* has a straddling relationship to cycle *i*.