A Parallel Fault-tolerant Routing Algorithm for Real-Time Media Transmission

Roman Messmer, FernUniversität in Hagen, Faculty of Mathematics and Computer Science, 58084 Hagen, Germany
Jörg Keller, FernUniversität in Hagen, Faculty of Mathematics and Computer Science, 58084 Hagen, Germany

Abstract

Networks based on multimedia switches which are designed to carry live media streams (such as video or audio) are the most recent products used in facilities for television production and distribution. They transport the live signal from a single source like a camera or microphone to multiple dedicated sinks like video monitors, loudspeakers and transmission lines, by using multicast or point-to-multipoint technology, and are sensitive to node or line failures. In previous work, we have introduced a sequential algorithm to overcome single and multiple node or link failures by re-computation of routes. As current SMP and multicore processors allow parallel computations, we present a parallel version of this algorithm to achieve real-time behavior in larger graphs with complex structure. We apply our algorithm to example graphs and present some preliminary experimental results to demonstrate its efficiency.

1 Introduction

Media switches which are capable to switch multimedia signals from low-bandwidth MIDI-signals up to HDTV-signals at 3 Gbit/s are the most recent development in the media industry concerning high quality signal distribution. Because of the high bandwidth requirements fiber-optic connections are standard. The signals are coded into Time-Division-Multiplexer timeslots and the whole signal is converted optical-electrical-optical (OEO) at each switch hop. Failures of nodes or lines in those packet switched networks often lead to unacceptable delays or breakdowns of data transport. In [10] we proposed a sequential algorithm (ZirkumFlex) to calculate a bypass route in a short amount of time. An implementation would provide (almost) real-time fault-tolerance to fibre-optic switched network platforms and other infrastructure. Rerouting on these platforms can be started on signalled errors on the physical layer (like fibre loss-of-light) but could also be commenced on detecting logical errors like CRC mismatch. Though the algorithm is meant to support the above-mentioned fibre-optic infrastructure, it also could be used generally as a methodology to achieve fault tolerance in network topology.

Here, we propose a parallel version of the ZirkumFlex algorithm which can speed up calculation substantially, together with some refinements of the sequential algorithm. On occurrence of failures, the algorithm basically performs BFS graph searches over surrounding graph nodes starting at predecessors and successors of the failing node/line to calculate a bypass. Additionally, there is a detection mechanism for pathological (no longer supported) links so that unused links can be removed from the local and/or central routing table after a failure. The parallelization allows real-time restoration after failures, i.e. restoration in less than 100 milliseconds for larger or complex graphs. This property is very important as frozen video images or hickups in audio streams are very unwanted events in live transmissions. At the same time the required overhead is lower than in completely redundant network structures. We illustrate the efficiency of our parallelization by applying it to example graphs.

The remainder of this article is organized as follows. In Section 2 we discuss related work. In Section 3 we present the parallel version of the ZirkumFlex algorithm, and in Section 4 we apply our algorithm to example graphs and provide some preliminary experimental results. We conclude in Section 5.

2 Related Work

Earlier work like [1, 2, 6] propose distributed algorithms to restore point-to-point connections after single network failures. This work does not cope with point-to-multipoint (multicast) connections. Associated flooding techniques as well as necessary bidirectional message transmission stages consume a lot of time (hundreds of milliseconds up to several seconds, depending on the underlying infrastructure). Our proposed algorithm is intended to achieve “real-time” property in fibre-optic networks. In broadcast terms this generally means restoration time well below 100 milliseconds. In case of a failed multicast route, the construction of a minimum spanning tree between the predecessor of the failed node and all successors should be approached to optimize the number of necessary restoration nodes and therefore reduce costs. To the best of our knowledge, this unique feature of our algorithm has not been published elsewhere.

In [7], parts of the network are analyzed for backup routes which could be used in case of a single node or link failure. This approach extends the resource reservation protocol (RSVP, RFC2208) to improve resource utilization and achieve a higher call acceptance rate and perform a better quality of service (QoS). The level of fault tolerance of each connection can be controlled separately. This approach requires precalculation and reservation of links which in broadcast technology are expensive. Holding the reservation information gets more complicated within extensive networks. Our algorithm does not need any reserved path and no communication overhead during the restoration path calculation is necessary, which crucially simplifies the restoration process.

In [3] several techniques and hints for designing efficient parallel graph algorithms are given. While most of the mentioned papers treat link failures, [8] also analyzes node failures. Our algorithm can even cope with multiple link and/or node failures. [13] discusses classical network resiliency in optical networks. [5] and [9] show a different fault tolerant approach to achieve fault-tolerance in optical networks using p-cycle reservation technology. In [11] dual link failures are surveyed. [12] examines a multiple segmented backup scheme for dependable real-time communication, but also relies on redundant reservation and intrinsic costs and complexity.

In most approaches the restoration bases on network flooding and some kind of path route monitoring. Both lead to a massive communication overhead and slow down the routing process which leads to latency times of some hundred milliseconds and more depending on the size of the network. Our concept suggests to calculate the restoration path exclusively on the predecessor of the (first) failed node. Depending on the number of failed nodes (because of the error notification time of more distant node failures) the whole restoration concept consisting of accessing current network layout, computing the backup route and finally switching the rerouting paths meet the mentioned real-time requirements.

3 The Fault-Tolerance Concept of ZirkumFlex

3.1 Preconditions

Our approach does not assume any reserved or active redundancy paths. Instead it assumes that there is additional bandwidth available in the network that can be used for a failure-triggered routing. The algorithm can cope with *Fail-stop* and *byzantine* failures, which can both be discovered by the network, as well as *intermittent* failures w.r.t. the usage of a Synchronous Time Division Multiplex (TDM) or Dynamic Synchronous Transfer Mode (DTM) protocol with repeating time-slots and CRC-coded information in the data stream.

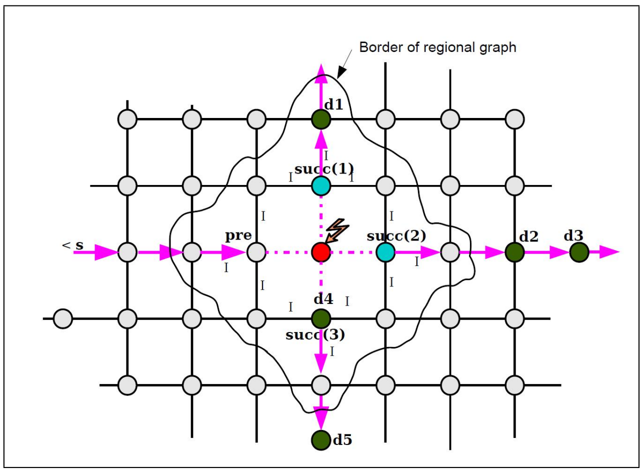

Figure 1: Example grid graph, during algorithm run

We assume that a failure can be detected and broadcast immediately by the node succeeding the failed node or link in the multicast tree. Then the computation of the bypass routes starts automatically. Restoration of the previous routing after a switch or line repair must be done manually if necessary.

When the failure is detected, the local switch databases remove the failed node/link from their network graphs instantly so that it is no longer used by the ZirkumFlex algorithm as a potential bypass node or link.

Figure 1 depicts the ZirkumFlex algorithm operating on an example network graph. Within the image, the set of nodes visited during the algorithm runtime is outlined. Together they represent the so-called *regional graph*.

Node s represents the source. The flash symbol marks a failed node within the graph. If several failures appear concurrently, it is left to the implementation whether neighboring failed nodes are treated as two independent failures launching two runs of the algorithm or whether they are consolidated into one calculation. This split permits the algorithm to work locally, not on the whole graph. Destination nodes $d_1, \ldots, d_5$ are shaded in dark grey. Because the multicast tree is loop-free by definition, there is only one single predecessor per failed node. We begin at the already established multicast path and assume that the hop-distance to the source node over the multicast path is known at each multicast transporting client. Every node holds the multicast call number (to distinguish between several multicast routings per node), its own number, source distance d, and finally rendezvous fields containing encountered predecessor and/or successor numbers for each multicast path passing the node. A node list holds node numbers not yet connected to any current or earlier predecessor w.r.t. the multicast path.

3.2 Procedure

After a failure has been detected, the predecessor and the successors of the failed node are determined. A following test of the predecessor node (pre) being a member of a unicast or multicast routing determines if any rerouting activity is necessary. If the local node indeed is a member, we propose to first start a relaxation (see Subsection 3.3.3), which reduces the number of potential bypass links and increases algorithm performance. Depending on the network topology and the failure position, this part might not be performed in parallel.

In the sequential Zirkumflex algorithm, breadth first searches (BFS) are started now at the predecessor(s) and at the successor node(s) alternating[1] and spread within the regional graph. These calculations are only executed on the predecessor node, which possesses the necessary network information. After a BFS reaches a node which already has been visited by another BFS a link between the two BFS-start nodes is calculated and the path is added to an emerging recovery path.

For the parallel version we assume that the nodes are capable of parallel calculations, use shared memory and consist of p processors. The calculations again are executed on the single predecessor node only. Graph breadth first searches (BFS) for predecessors and successors of failed node(s) are computed *concurrently*. The BFS runs reach the neighboring nodes step by step in parallel. There is no synchronization process necessary. Because of being seperated by the failed nodes the first search steps will not produce any data dependencies. Upon the first detection of BFS-node meetings the emerging bypass graph is modified. In most cases and infrastructures the BFS runs of two nodes meet at most once and so a write-lock can be easily applied to handle the data dependencies.

As long as there are unreached successors do the following:

- If the number of predecessors $|P|$ plus the number of successors $|S|$ are less than or equal to the number of the p available processors: Do BFS-runs (only for direct neighbors at a time) starting on the predecessor node and on the successor nodes (for each successor and the predecessor one BFS is started on one processor in parallel) and mark the linked list of the reached nodes with the path to the origin, i.e. the first detected node gets the predecessor or successor node name in its linked list, respectively. In the next run there are two entries in the list and so on. If $|P| + |S| > p$, then the above procedure has to be executed using a round-robin technique where predecessors and successors visit their direct neighbors only, then the processing is switched to the following predecessors or successors.

- If a previously predecessor-BFS-marked node is reached by a successor-BFS, the successor node is marked as reached and the path from the predecessor to the affected successor is taken into the emerging *regional graph*. A simple concatenation of the reached node's path lists (linked list) and the node-BFS's path lists in inverted visit order directly delivers the path. Predecessor-BFS are stopped when all of the successor-BFS are stopped. Successor-BFS are stopped when the regarded node-BFS finds a connection to its or an earlier predecessor (BFS-node list). Their corresponding node names are taken out of the node list. Already visited nodes/edges are then excluded from further BFS-runs.

- If another successor-BFS-marked node has been reached, the path between both successors is also added to the regional graph. Already visited nodes/edges are again excluded from further BFS-runs.

[1] By alternating we mean, that for each node, we consider all neighbors at a certain distance before switching to the next node.

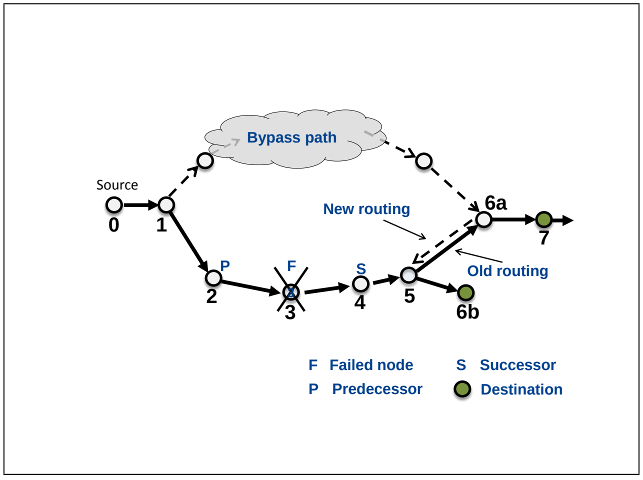

Figure 2: Falling node swap

This procedure puts up a local subgraph of the network graph (regional graph) and delivers a spanning tree consisting of shortest paths between successors and predecessor in a short amount of time. [4] showed that BFS searches find a shortest path between two nodes of a graph.

While a small number of BFS steps is sufficient in dense graphs, some network structures lead to a surprising effect. It may occur that a network link carries a normal multicast link and a bypass link in opposite directions as seen in Fig. 2. Supporting nodes which lie closer to the failed node than the bypass node meeting the original path, in the example between nodes 5 and 6, means wasting bandwidth. To support nodes e.g. by using a prior calculated and per-node marked source-distance information, routing from a higher node number to a lower one means that the old routing has to be terminated and a reverse link direction has to be applied for this graph edge when applying the new routing.

The major advantage of this model is that the successor nodes and further on the destination nodes get the source stream as quickly as technically possible and nodes that are not affected directly are not disconnected for rerouting purposes.

3.3 Data dependencies

3.3.1 Relaxation part

The proposed data structure for an appropriate availability graph - which is the fundamental graph our algorithm is intended to work on - is an adjacency list representing available links for the current multicast path bandwidth in the local network area. Another adjacency list which holds the predecessors for quick access is recommended. It can easily be calculated offline from the main list in $O(k)$ time, where k denotes the size of the (limited) network part known by the calculating predecessor. Alternatively an adjacency matrix can be used to determine successors and predecessors, the latter by accessing the matrix in reverse direction. We recommend using the list version because the underlying fibre-network represents a sparse network graph. We assume that synchonized access to the list entries is possible by read-write-locking or other appropriate data structures. Two different kinds of processes for predecessor and successor nodes are provided.

In Fig. 3 the relaxation procedure, which slightly differs from the sequential algorithm version, is detailed. There is no more additional call of the process if there are links left to be removed. Starting from the failed predecessor and successors each process takes away unnecessary links from the network graph one by one until a branch is reached. Therefore, prior to removing a link the process has to know if the indegree and outdegree of a neighbor equal one. This information has to be calculated and provided separately. In the example in Fig. 3 the F_2-process deletes the dashed marked link. No other process will access this link by writing because all delete processes will stop at any branch of the network. Therefore, no write-locking is necessary here. If, on the other hand, the deletion process of F_1 reaches P_2 after several steps, it finds P_2 as indeg=outdeg=1 and proceeds deleting the following links. For a *successor*, the deletion process ends at a destination node or branch. A *predecessor* run stops at a detected branch or source node. On finding a destination node the effective starting node is moved (corrected) until a branch is detected, but to support the destination node no further link will be deleted.

The number of simultaneous failures to be treated depends on the implementation target: faster computation or larger failure count. For big graphs, we propose the former version, which assumes the determination of nodes to be collected in sets and assigned to groups of failures.

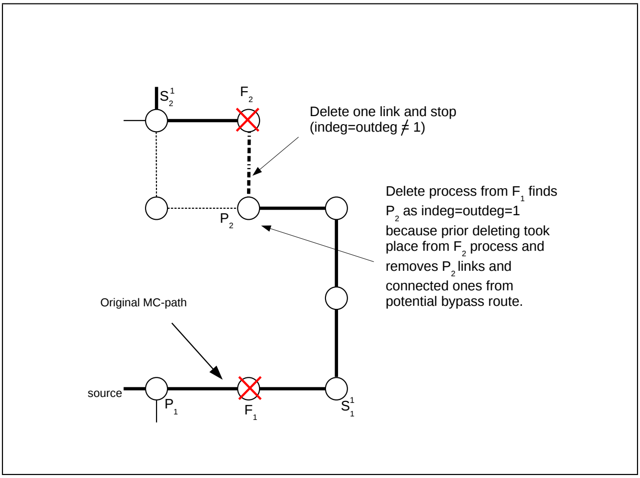
S_2^1
F_2
Delete one link and stop (indeg=outdeg $\neq$ 1)
Delete process from F_1 finds P_2 as indeg=outdeg=1 because prior deleting took place from F_2 process and removes P_2 links and connected ones from potential bypass route.
P_2
Original MC-path
source
P_1
F_1
S_1^1

Figure 3: Relaxation details

3.3.2 BFS part

Also for the BFS part, we engage the adjacency list we introduced in the relaxation part and use read-write-locking per entry within the list. In the first part of the algorithm, the parallel processes, executed only at the predecessor node, begin from different nodes (predecessor and successors) per definition and perform read-and-write access to the referring node entries within the list. As long as new nodes are being detected, they are untouched by other processes; thus, collisions are not conceivable and write-lock will be efficient. When finding an already touched node, each process writes its own path information into the BFS-node list. In parallel, the path information in the node is concatenated and the result is stored in the emerging ZirkumFlex-graph. Compared to the number of visited nodes, conflicts in the graph construction process are relatively rare hence the read-write-locking is a practicable way for handling dependencies. After finishing the construction of the bypass path a DFS over the remaining graph makes sure that potential loops are eliminated.

3.3.3 FNDS part

After establishing the bypass path, the assembly of the routing commands has to be checked for routings which lead opposite to the original direction. The old routing links have to be removed to drop the reservation of unnecessary bandwidth. This part as secondary priority can be done while or even after sending the routing commands because the former direction cannot be necessary for the bypath path[2]. As a prerequisite, the distances to the source

[2]The bypass path is loop-free and directing the signal on a link in both directions would presume a loop.

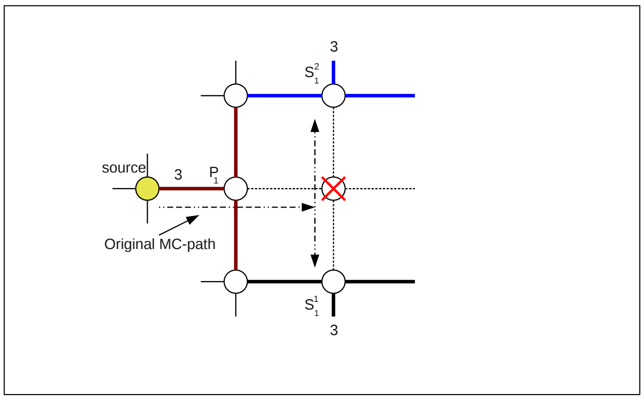

Figure 4: Predecessor BFS touching successor-BFS

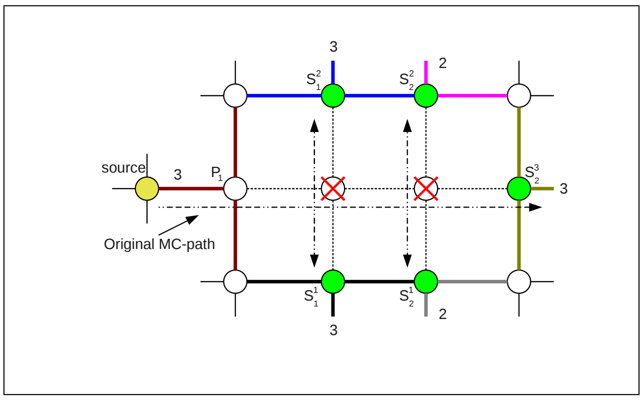

Figure 5: Two failed nodes and BFS propagation

node for each multicast call is marked into a node. The routing commands can be analyzed for routing from higher to lower marked nodes. In that case, the old routing direction has to be deleted.

4 Example Graphs

The efficiency of the ZirkumFlex parallelization has been tested by several carefully chosen example graphs. When looking at Fig. 4, we see the predecessor BFS reaching its three incident nodes and the successor nodes which both also reach three nodes and connect together to the shortest bypass route. In theory, there are only three process steps necessary when parallel processes are used, or nine steps sequentially. This results in a speedup of 3.

In Fig. 5, we consider two failed nodes in an example grid graph. This graph exemplifies the other extreme of the speedup-range. There are five successors, one predecessor and 16 links to visit sequentially. Parallel processes need no more than three links concurrently when

$$Speedup = \frac{no.\ of\ visited\ nodes}{no.\ nodes\ handled\ per\ processor} = \frac{16}{3} = 5.3$$

This graph demonstrates another fact: The maximum speedup is directly proportional to the number of successors plus predecessors. Concerning grid graphs the maximum number of successors per failure equals three like already seen in Fig. 4. The handling of several failures allows an accordingly larger number of parallel processes leading to a accordingly larger speedup.

In a tested grid graph 8 neighboring failed nodes on a unicast path lead to a regional graph of about 200 nodes. A series of failed nodes on a unicast path represents the worst case for our algorithm within grid graphs. In this case, the parallel version of our algorithm reaches a speedup of 2. The BFS calculation can be split and starts from predecessor and successor in parallel. As a result of examples with several graphs of different sizes with an arbitrarily chosen number of failures and the experience with the

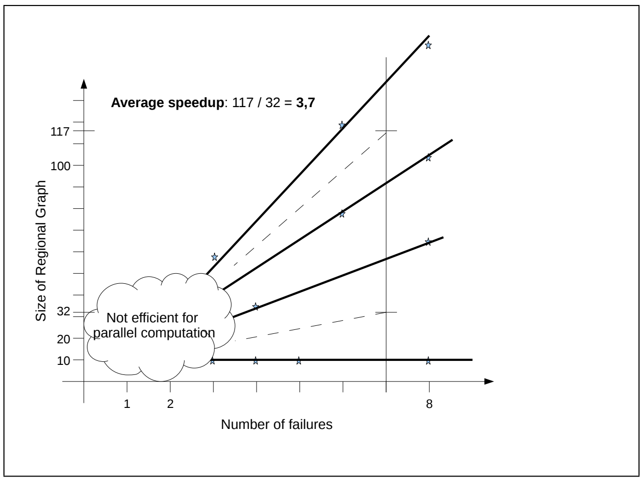

Figure 6: Regional graph size

limitation concerning the relation of successors and processes the diagram in Fig. 6 has been set up. It displays the upper and lower limits of the runtime with sequential as well as parallel computation on different topologies on the basis of failed node number against regional graph size. It can clearly be seen that the size of the regional graph grows *linearly* with the number of failures and the average speedup over the tested topologies is about 3.7. There is no overlapping of the areas and therefore no topology and failure layout where the parallel version would lose performance. Since in practical applications there is some overhead in splitting tasks to processes, the exact resulting speedup value will also depend on the number and position of failures relatively to the multicast path and on the graph topology itself.

5 Conclusion and Outlook

We have presented a parallel version of the fault-tolerance algorithm Zirkumflex, together with some refinements of the sequential algorithm itself. Our preliminary experiments indicate that within grid graphs the parallel implementation speeds up linear with the number of multicast path-successors and therefore the usable number of processors. Even in the worst case (several failed nodes in a series on a unicast path within a grid graph) we still achieve speedup 2. Starting from predecessors and successors the BFS propagation synchronously sweeps in circles upon existing links until they meet. Within general graphs the efficiency will not be as high because there are no underlying symmetries which allow a more or less equal runtime of the processes. The minimum speedup will always be above 2, yet. As in the sequential version of the algorithm there is no need of redundant paths, path reservations and no communication overhead. All surveyed alternatives suffer at least from one of the mentioned drawbacks, communication overhead or bandwidth use. After a failure of one or more neighboring nodes recalculation of the multicast path is done automatically and in a very short time.

To further increase re-routing speed on a larger number of sequential failures the regional graph could be restricted. Meeting the criterion for this purpose will be the subject of further work.

References

[1] C. Edward Chow, Steve McCaughey, and Sami Syed. Rreact: A distributed protocol for rapid restoration of active communication trunks. In *Proc. 2nd IEEE Network Management and Control Workshop*, 1993.

[2] E. Chow, J. Bicknell, S. McCaughey, and S. Syed. A fast distributed network restoration algorithm. In *Twelfth Annual International Phoenix Conference on Computers and Communications*, pages 261–267, 1993.

[3] Guojing Cong and David Bader. Techniques for designing efficient parallel graph algorithms for SMPs and multicore processors. In Ivan Stojmenovic, Ruppa Thulasiram, Laurence Yang, Weijia Jia, Minyi Guo, and Rodrigo de Mello, editors, *Parallel and Distributed Processing and Applications*, volume 4742 of *Lecture Notes in Computer Science*, pages 137–147. Springer Berlin / Heidelberg, 2007.

[4] B. A. Crane. Path finding with associative memory. *IEEE Transactions on Computers*, C-17(7):691–698, July 1968.

[5] Hamza Drid, Samer Lahoud, Bernard Cousin, and Miklós Molnár. A topology aggregation model for survivability in multi-domain optical networks using p-cycles. In *Proc. 6th IFIP International Conference on Network and Parallel Computing*, pages 211–218, 2009.

[6] W.D. Grover. *Selfhealing networks: a distributed algorithm for k-shortest link-disjoint paths in a multigraph with applications in real time network restoration.* PhD thesis, University of Alberta, 1990.

[7] P. Krishna Gummadi, Jnana Pradeep Madhavarapu, and C. Siva Ram Murthy. An efficient primary-segmented backup scheme for dependable real-time communication in multihop networks. *IEEE/ACM Trans. Netw*, 11(1):81–94, 2003.

[8] H. Komine, T. Chujo, T. Ogura, K. Miyazaki, and T. Soejima. A distributed restoration algorithm for multiple-link and node failures of transport networks. *Proceedings GlobalCom '90*, 1990.

[9] C. Liu and L. Ruan. p-cycle design in survivable WDM networks with shared risk link groups (SRLGs). *Photonic Network Communications*, 11(3):301–311, 2006.

[10] R. Messmer and J. Keller. Real-time fault-tolerant routing in high-availability multicast-aware video networks. In Felix C. Freiling, editor, *Sicherheit 2010*, Lecture Notes in Informatics, GI Edition, pages 49–60, 10 2010.

[11] Srinivasan Ramasubramanian and Amit Chandak. Dual-link failure resiliency through backup link mutual exclusion. *IEEE/ACM Trans. Netw*, 16(1):157–169, 2008.

[12] G. Ranjith and C. Siva RamMurthy. A multiple segmented backups scheme for dependable real-time communication in multihop networks. In *Proc. 17th International Parallel and Distributed Processing Symposium*, page 121, Nice, France, April 2003. IEEE Computer Society (Los Alamitos, CA).

[13] Lena Wosinska, Didier Colle, Piet Demeester, Kostas Katrinis, Marko Lackovic, Ozren Lapcevic, Ilse Lievens, George Markidis, Branko Mikac, Mario Pickavet, Bart Puype, Nina Skorin-Kapov, Dimitri Staessens, and Anna Tzanakaki. Network resilience in future optical networks. In Ioannis Tomkos, Maria Spyropoulou, Karin Ennser, Martin Köhn, and Branko Mikac, editors, *COST Action 291 Final Report*, volume 5412 of *Lecture Notes in Computer Science*, pages 253–284. Springer, 2009.