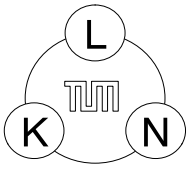

Universität Passau
Lehrstuhl für
Theoretische Informatik
Prof. Dr. Franz J. Brandenburg

Technische Universität München
Lehrstuhl für
Kommunikationsnetze
Prof. Dr.-Ing. Jörg Eberspächer

Diploma Thesis

Algorithm for Multi-Path Hop-By-Hop Routing

Jens O. Oberender

Date: August 2003

Supervisors: Prof. Dr. Franz J. Brandenburg, Universität Passau
Dipl.-Ing. Claus Gruber, Technische Universität München

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle wörtlich oder sinngemäß übernommenen Ausführungen wurden als solche gekennzeichnet. Weiterhin erkläre ich, dass ich diese Arbeit in gleicher oder ähnliche Form nicht bereits einer anderen Prüfungsbehörde vorgelegt habe.

Passau, den 27. August 2003

.....
(Jens O. Oberender)

Imprint
Jens O. Oberender (V. i. S. d. P.)
Oscar-Wilde-Str. 15
50858 Köln
EMail: mailto:jens.oberender@joooo.de
Web: http://www.joooo.de

Supervisor Contacts
Prof. Dr. Franz J. Brandenburg
Lehrstuhl für Theoretische Informatik
Universität Passau
EMail: mailto:brandenb@mail.infosun.fmi.uni-passau.de
Web: http://www.infosun.fmi.uni-passau.de/br/lehrstuhl/

Dipl.-Ing. Claus Gruber
Lehrstuhl für Kommunikationsnetze (Prof. Dr. Jörg Ebersprächer)
Technische Universität München
EMail: mailto:claus.gruber@ei.tum.de
Web: http://www.lkn.ei.tum.de

Abstract

The next generation internet provides resilient wide area networking. Resilience is the ability to resist outer influences such as link failures. During routing protocols reorganize the communication paths after a topology change, data loss can occur. Using multiple paths, network operation can continue after failure detection.

This work examines *Multi-Path Hop-by-Hop* routing where any single link failure can be locally recovered. We produce *acyclic* routing graphs for destination-based routing. Our approach results in two edge sets: active and reserve links. Active edges provide an acyclic graph embedding a spanning tree. Any failure that is not covered by redundant active edges is recovered by inserting a reserve edge. We guarantee recovery of the first link failure event and then seamlessly restore a HammockSet for the new topology.

Two similar approaches have been published. The O2-algorithm derived out of the project "Key Components for the Mobile Internet of Next Generation" [Sch01] and constructs thin HammockSets but is restricted to certain topologies. The MPA-algorithm [Nar00] succeeds on any topology, yet it cannot provide redundancy to all nodes. We specify topologies that allow stand-by-recovery to all nodes and destinations, while we construct edge-maximized HammockSets.

For evaluation we introduce *link significance*, a measure for the forwarding function of inner HammockSet nodes. A heuristic algorithm optimizes the HammockSet layout for *traffic distribution*. It restricts the number of HammockSets on one network edge, increasing the bandwidth fraction available to the participating HammockSets.

A prototype implementation has been part of this work. It constructs HammockSets for any destination node of a topology. The final chapter discusses the feasibility of implementing our approach in real-world systems. Further, we point out possibilities for future work.

Preface[1]

The information society and its economics heavily dependent on today's technologies. Lately, a 29 hour blackout on the west coast and Canada drastically demonstrated how vulnerable we have become to reliability of the power grid. Digital networks are also essential for industry, even world-wide connectivity is important. In this work we develop a new approach to provide immediate recovery of network routes in case of a single link failure. The next generation internet will be resilient to outer influences. We propose a mechanism to provide resilience.

This work is divided into eight chapters and two appendices. Network and routing definitions are introduced in the first chapter. It also covers fault recovery, redundancy, resilience, and methods to prevent disconnections.

In Chapter 2, we show that classic routing graphs cannot provide redundancy to all nodes and then formally define *extended routing graphs*, *regular routing graphs*, and *HammockSets*. We analyze the mechanisms of cycle avoidance and introduce a representation as list of strongly-connected components. The networks which embed HammockSets are examined in Chapter 3. Based on the concept of candidates we develop a description for *Complete HammockSet Graphs*. Chapter 4 formally proves the construction algorithm. Its execution alternates two phases: neighborhood discovery and candidate selection phase. Last in this Chapter we prove the bound of transitions steps necessary to process a HammockSet.

The actual operation of HammockSets is described in Chapter 5. We discuss how the packet order inside data flows can be maintained on HammockSet-operated networks. Then we examine in detail which kind of link failure sets can be automatically recovered. We find not all node failures can be recovered by HammockSets. Then we introduce a procedure, which seamlessly switches to a new HammockSet to reestablish full coverage of first link failures.

The evaluation of HammockSets is described in Chapter 6. We introduce the measure *link significance* that determines the relevance of one link for the HammockSet operation. In Chapter 7 we summarize the outcome of our exhaustive study of related work and correlate them into our context. Chapter 8 contains our conclusion, list our contributions, and suggests future work.

Appendix A provides in-depth knowledge of physical redundancy in all its aspects. In closing, Appendix B gives an insight into the *prototype implementation*. Also we suggest an heuristic approach for construction of optimum HammockSets.

[1]I would like to thank the supervisors Prof. Brandenburg and Claus Gruber; the proofreaders M. G. Berberich, Peter Blick, Mike Forster, Wolfgang Heinze, Andreas Pick, Marcus Raitner, Stefanie Scherzinger, Tom Zimmermann; the advisor Peter Erven; my parents and all people supporting me during my studies.

Contents

Chapter 1

Introduction

Computer networks transmit data between network devices. Communications between network nodes, that do not have a common link, use the forwarding feature of intermediate network devices. This is called routing. Its transmission paths are automatically configured by routing protocols.

The main focus of this work is providing stand-by reserve routes, which can be activated without network interaction in case of a link failure. We utilize multiple paths, where after a single link failure at least one path remains operational. Today routing protocols often use single paths. In case of link failure they are able to reconstruct them on their own. However, they require a few seconds or a few minutes to resume functional routing for all nodes. During this time data may be lost.

The next generation internet introduces resilience, that means the network has adequate methods to resist outer influences disturbing normal operation. The most recent resilience approach is the combination of redundant paths and fast convergence mechanisms.

1.1 Networks

We can formally describe a network topology through an equivalent graph. We replace routers by nodes and links by directed edges.

Definition 1 *A* **graph** $G = (V,E)$ *consists of a set of nodes* V *and a set of edges* $E \subseteq V \times V$*. A graph is called* **undirected***, if for any edge* $e = (u,v)$ *its reverse edge* $\widehat{e} = (v,u)$ *is included in the edge set, otherwise the graph is* **directed***.*

Network links often use bi-directional connections between two nodes. Some topologies must be bi-directional, since the routers must respond to the link availability test on the opposite-directed link.

Assumption 1 *We assume networks to be non-directed, ensuring an opposite directed edge $\widehat{e}$ exists for all edges e.*

Two routers connected by a link can exchange data. Such routers are neighbored to each other. Also distant messaging is possible, since routers can be configured to forward data onto another link.

Definition 2 *In a graph, a set of nodes U has* **neighbors** *$N(U)$, which consist of all nodes having adjacent edges with U.*

$$N(U) := \{\ u \in V \setminus U \mid \exists v \in V : (u,v) \in E\ \}$$

Definition 3 *A* **path** *from node u to node u' of length k is a sequence of vertices $p =< v_0, \ldots v_k >$, where $u = v_0, u' = v_k$, $\forall i \in \{0, \ldots k-1\} : (v_i, v_{i+1}) \in E$ and the nodes are distinct $\forall i \neq j : v_i \neq v_j$.*[1]

Since all network stations should be able to communicate with each other, the network topology must be connected. A network structure is modeled through a graph with edge capacities.

Definition 4 *A graph is called* **connected** *if paths between all nodes exist.*

Definition 5 *A* **network** *$N = (V, E, C)$ consists of a set of nodes V, a relation of edges $E \subseteq V \times V$, and a capacity function $C : E \to \mathbb{N}$. (V, E) is a connected, undirected graph. Neither parallel edges nor self-loops occur in the network graph.*

While most links have symmetric capacity into both directions, some links have different capacities $C(u,v) \neq C(v,u)$, e. g. for upstream/downstream of *Asymmetric Digital Subscriber Line* (ADSL). Network topologies are spare graphs, usually with a node degree between two and four. They provide point-to-point communications realized through routing.

1.2 Routing

Routing provides communication over network paths. The *Open System Interconnection* Reference Model abstractly describes network communication. Its network layer defines how messages are addressed between nodes. No.1 router vendor *Cisco*, defines routing as the act of moving information across an internetwork [Cis02]. Referring to the network model, routing operates the message forwarding between non-neighbored nodes.

We will now investigate two current transport layer protocols and see how routing relevant information gets enclosed in messages and how routing devices make forwarding decisions.

Asynchronous Transfer Mode (ATM) messages attach a virtual path identifier to the messages. Before usage any path must be set up. Therefore any router knows about that virtual path and can correctly forward the messages on that path. After a link failure, any path including that link is destroyed and must be manually reconstructed [Tan03].

[1] Paths in a directed graph are directed.

The **Internet Protocol (IP)** uses host addresses to identify destinations. IP routers know about the topology and can decide, which of their links has the shortest path towards destination. This is called destination-based *hop-by-hop* routing. If a link fails, the routers consolidate a global view and determine new shortest paths [Hui99].

In both cases a message field references to information locally stored at the router where to forward the message. In IP static addressing is used and routing information is available from everywhere. In ATM point-to-point connections must be individually set up, which enables Quality-of-Service. In a failure event the ATM approach is very expensive, since it requires reconstruction of many paths. With IP routing, link failures can be rapidly recovered by changing the routing tables next to the failing link. Other stations do not need to reconfigure. We focus our research on hop-by-hop routing.

1.3 Routing Tables

Destination-based hop-by-hop routing rules forward decisions using routing table information. We introduce the routing table information as relation between addresses and links, although implementations use highly optimized data structures for fast access.

Definition 6 *A* **routing device** *connects n subnets with each other. It locally stores a* **routing table** $rt : addr \rightarrow link$*, suggesting the shortest path towards* $addr$ *using next hop* $link$ *(single path).* **Multi-path** *routers use an extended routing table* $rt' : addr \rightarrow \mathcal{P}(link)$.

During forwarding operation, a router analyzes the destination address field of a received message, accesses its internal routing table, extracts the next hop link and schedules the message for transmission over that link.

IP routers provide a next hop to any network node, which is given by the routing table entry. Any sequence of nodes connected by next hops must then lead to that destination. Formally, any node must have a path towards destination.

Definition 7 *A* **spanning tree** *is a set of edges that contains paths from all nodes to destination.*

Early Internet suffered from routing problems with cycling messages, which could not reach destination neither could be dismissed from forwarding. IP then introduced the message header field Time-to-Live, which the sender initializes with maximum hop count. With each passing of a router, its value is decreased by one. If a router receives a message with a zero Time-to-Live field it will drop the message, remove it from the network. In a correctly configured routing environment, that will not happen. The most common reason, why messages reach value zero is the existence of a routing cycle, causing paths of infinite length.

Definition 8 *A* **simple cycle** $< v_0, \ldots v_k, v_0 >$ *consists of a path* $< v_0, \ldots v_k >$ *with length* $k \geq 1$ *and an additional edge* (v_k, v_0).

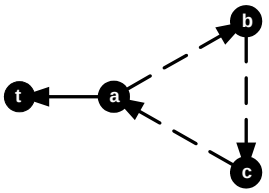

Figure 1.1: Routing Cycle

Assume we have a four-node network (Fig. 1.1). There is a cycle of three, with one node having a second path to destination. Messages that repeatedly cycle between a, b, c could be lost during transmission, as they reach zero in their Time-to-Live field. We see here several disadvantages: messages are repeatedly transferred between routers, increase the link usage and messages are delayed in transmission to their destination, or get dropped before reaching it. Therefore we strongly avoid cycles in routing graphs.

In 1974, McQuillan proposed the principle of hierarchical routing [McQ74], which is the predecessor to routing graphs. Such graphs consist of all next-hop edges to one destination.

Definition 9 *A* **routing graph** $T = (V, E, D, t)$ *consists of a network* (V, E)*, a subset of active edges* $D \subseteq E$ *and a destination node* t*. The set of active edges is cycle free and includes a spanning tree to the destination node.*

Such a routing graph guarantees network operation, i. e. messages can be transmitted from and to all nodes in the network. The spanning tree property verifies, that any node has a path towards destination and the cycle-free property ensures the promptly delivery without message droppings.

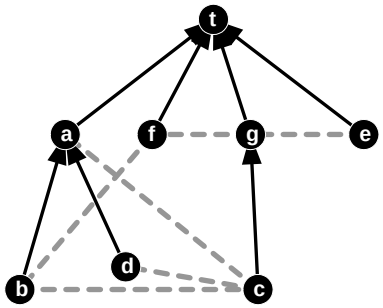

Figure 1.2: A Routing Graph towards *t*

In some ways routing graphs are orthogonal to routing tables. The overall routing information can be sliced down by destination node, which results in routing graphs (Fig. 1.2). And they can be dismantled to a first-hop view per node, which are routing tables (Tab. 1.1). We see that either a complete set of routing tables or routing graphs describe the routing behavior in a network.

Destination	Outgoing Link
a	t-a
b	t-f
c	t-g
d	t-a
e	t-e
f	t-f
g	t-g

Table 1.1: Routing Table at Node *t*

1.4 Routing Protocols

The router configuration is maintained by routing protocols. This is done in a distributed process, where each router receives information about the network topology and locally builds its own routing table.

Definition 10 *A* **routing protocol** *formally defines the message format that is used to establish and maintain routing table information. The concept also includes the algorithm that creates routing tables.*

There are various routing protocols available, which differ in algorithm simplicity and converging performance. In 1988, John Moy developed a taxonomy that highlights the different approaches of routing protocols [Moy88][2]. Besides analyzing the construction of routing information, Moy also looks how fast routes are deleted when they become unavailable. Most interesting in our context is the examination of robustness and reliability. How does the network react on lost status messages, link failures and disconnection of subnets? Does it converge a stable routing and how fast does it converge? Routing protocols focus on a few aspects and fit very well for some networks, but not for all topologies.

The **Routing Information Protocol** (RIP) uses a Distance-Vector-Protocol to create routing information [Hed88]. The routers examine incoming distance vectors and remember what link provides the minimum distance to some destination. With a change of distance information each router distributes its internal distance vector to its neighbors. This procedure guarantees cycle free routes through shortest path trees (compare Bellman [Bel57] and Ford and Fulkerson [FF62]).

However, the resulting routing information uses single paths only. If a link along such a path fails, routing is disabled until renewed routing information. RIP's resilience is low, because it is based on network propagation of routing information ([Hui99] p. 117).

Open Shortest Path First (OSPF) aims to discover the complete network topology [MRR78]. OSPF transmits Link-State-Vectors, which contain enough information to construct a global network view at all stations. From such a view several paths can be picked, one

[2] Byrnes gives an overview of Moys theses in [Byr00].

of them being the shortest path. The Equal-Cost Multi Path (ECMP) variant provides several paths with same cost, if available [Hop00].

OSPF is widely used in large networks. Its convergence duration is between seconds and minutes, depending on the topology and preset timeout values.

While RIP performs best in simple networks, OSPF gives a good deal for a routing protocol with distributed failure recovery. However, this does not fulfill strong resilience criteria, since propagation of topology information through several routers may delay convergence. We note that routing protocols use a distributed process to reestablish routing, which is too slow. Our proposed method must be faster than that and avoid network interactions.

1.5 Fault Recovery

We are going to review network recovery methods and learn why they fail to meet today's requirements. Fault recovery methods resolve failures in a network without user interaction. Its objective is to automatically establish network connections under any external influence. It success is measured through the availability ratio.

Definition 11 *The* **availability ratio** *describes the fraction between online and offline duration of network components.*

The *ARPANET* had been designed to survive a partially network destruction, especially when nodes have been destroyed. With this assumption, the ARPANET did not require alternative paths. The introduced strategies are not optimized for fast recovery after link destruction. Since the Internet still uses many ideas from its ancestor ARPANET, there is no really fast recovery of link failures available. This is a design issue, the current implementations provide no solutions against short communication disruptions.

Characteristics of global networks have drastically changed. Most Internet service providers own high redundancy equipment and node failures got very rare. Today, networks are rather threatened by failing links. In wide area networks the availability ratio is still below $1000 : 1$[TN94]. After the examination of network failures a geographic researcher summarizes: "The ability for networks to re-route, re-connect and have redundancy is clearly important for the survival of the Internet" [Gru02]. Re-routing means the routing devices should establish new routing after nodes or links have been lost. Re-connecting describes the client application behavior, when they switch to other servers and retransmit their requests. Providing redundancy can be gained through physically independent lines without any common point of failure. Refer to Appendix A for a full survey.

Fault recovery influences the resilience and availability of networks. Single-path routing protocols do not provide methods to recover failing links without network interaction. Many topologies contain a second path, where outgoing traffic can be switched to in case of failure events.

Our approach is going to enrich routing graphs with multiple paths. Then a router can rapidly recover routing with local mechanisms.

1.5.1 Fault Resilience

Fault resilience summarizes everything that gains high availability of a network. It involves the topology, its physical properties, as well as routing behavior, the routing protocol abilities and the behavior after converging.

Definition 12 *A network is* **fault resilient***, if it provides methods to detect and remedy failures so that normal operation resumes promptly.*

Resilience requires early detection of network problems. Link availability can be tested between neighbored routers. They frequently exchange messages that reset timeout counters. The timeouts occur if no messages have been received lately, which indicates a link loss. All destinations served over that link become unreachable. Now, a mechanism needs to recover routing very fast, to minimize data lost through buffer overruns. These occur if data cannot be forwarded since the shortest-path link is defective and router queues become fully stuffed.

We distinguish two converging types.

Routing protocols distributed recover routing. This involves propagation delays and therefore the time to convergence scales with the topology size.

Preplanned recovery strategies provide alternative paths so that a route remains after any single link failure. They recover immediately, since usually no network interaction is required to recover. They have better resilience.

Our approach provides preplanned recovery through multiple paths.

1.5.2 Redundancy

In the network context redundancy is ”a method to ensure that if there is a failure of a network device, there are back up units available to keep the network running” [Cab02].

Definition 13 *A connection between two nodes is* **redundant***, if two physically independent paths exist.*

From a graph theoretic view, we assume all edges to be physically independent from each other. In practice many facts must be taken into account (for a complete survey see Appendix A).

1.6 Multi-Path Routing

If single path routing is highly vulnerable by link faults, we can increase the resilience through providing several paths between two nodes. If we want to provide redundancy to all nodes in a destination-based routing environment, then this requires two paths from any node.

Definition 14 *Two paths* $p_1 =< u, x_1, \ldots x_n, t >, p_2 =< u, y_1, \ldots, y_m, t >$ *are said to be* **redundant in their first hop** *if they utilize different first links, i. e.* $x_1 \neq y_1$.

Providing redundant paths will be challenged with cycle avoidance. We are going to provide first-hop redundant paths to any inner node.

Chapter 2

HammockSets

Resilient network operation can be ensured with redundant network paths. In this chapter we are going to extend routing graphs into HammockSets, a special graph type that provides destination-based multi-path hop-by-hop routing.

2.1 Maximum Routing Graphs

In Chapter 1, we introduced routing graphs and defined their minimum requirements. Now we examine whether such graphs can be enhanced towards resilience and if not, we will analyze what prevents full resilience.

The definition of a routing graph (Definition 9 on page 7) demands the subset of active edges to contain a spanning tree and to be cycle free. This definition allows additional edges as long as they do not conflict with routing graph properties.

Definition 15 *For a* **maximum** *routing graph $T = (V,E,D,t)$ there does not exists a routing graph $T' = (V,E,D',t)$ with a superset of active edges $D' \supset D$.*

Fig. 2.1 (A) shows an eight-node topology and a routing graph towards node t. It consists exactly of the spanning tree edges. The grey dashed lines are links in the topology, which remain unused towards this destination. Fig. 2.1 (B) is a maximum routing graph, which edges are a superset of the spanning tree in Fig. 2.1 (A). It cannot be enlarged, because all edges can be used in at most one direction, since bi-directional use creates cycles.

We note that the underlying topology has bidirectional links, but a routing graph utilizes only either direction or none. Besides, we see that the graph layout is such that all edges point upwards to destination. We identify both properties as abstract principles of relations.

The edge set of a network has been formalized as a binary relation $E \subseteq V \times V$ (def. 5) and so is D the set of active edges. We now introduce relation properties that we require for a

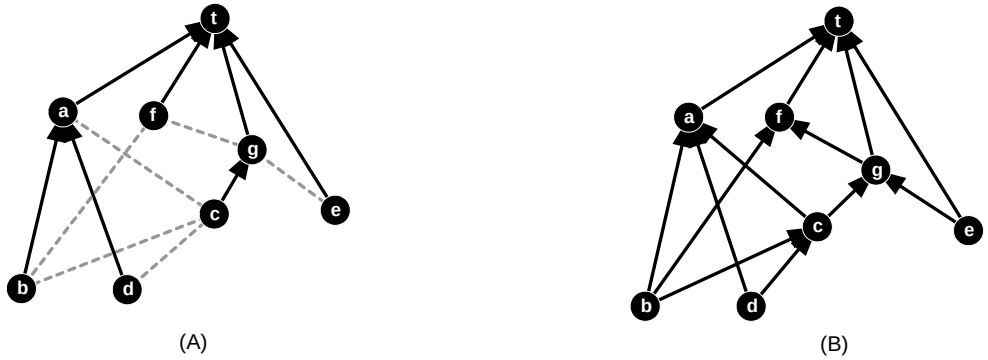

Figure 2.1: Enriched Routing Graph

classification of routing graphs. Relations obtain extra attributes, if all their elements hold such specified properties. These attributes can be used to define abstract requirements onto sets and order relations.

Definition 16 *A relation is* **reflexive** *if* $a\,R\,a$ *holds for any* a.

Definition 17 *A relation is* **transitive** *if* $a\,R\,b \wedge b\,R\,c \Rightarrow a\,R\,c$.

Definition 18 *A relation is* **antisymmetric** *if* $a\,R\,b \wedge b\,R\,a \Rightarrow a = b$.

We apply these criteria now on routing graphs. The relation R expresses whether there exists a directed path between two nodes. Routing graphs are reflexive, since any node has a zero-length path to itself. Paths also provide transitivity, since paths can be concatenated to longer paths, if the middle nodes match. Third, routing graphs are antisymmetric, because nonzero-length paths between $< a, \ldots b >$ and $< b, \ldots a >$ would be a cycle, which must not exist in routing graphs.

Definition 19 *A* **partially ordered** *relation is reflexive, transitive, and antisymmetric.*

Intuitively, a partially ordering requires three relation properties. First, the ability of finding an identic element. Second, comparing elements to each other using the relation. Third, comparisons can be deduced from other comparisons over the same set. A partial order is incomplete, meaning that for some pairs x,y neither $x \leq y$ nor $y \leq x$ holds.

For graphs one relation is being-connected-through-a-directed-path. In the maximum routing graph T of Fig. 2.1 (B) nodes a and f have no relation to each other, since there exists neither a path from $< a, \ldots f >$ nor $< f, \ldots a >$ in T. The graph nodes are partially ordered.

Theorem 1 *A Routing graph is a* **directed acyclic graph** *(abbrev. dag).*

Proof. Routing graphs are directed and they do not contain cycles by definition. □

Theorem 2 *A dag is partially ordered.*

Proof. Let $T = (V,D)$ be a directed acyclic graph. Let $P := \{(s,t) \in V \times V \mid < s, \ldots t > \text{ path in } T\}$. We show that the set of edges D with the relation connected-through-a-path P is

reflexive, transitive, and antisymmetric.

- P includes paths of length zero. Therefore any node u has a path to u, written $u\ P\ u$ which guarantees to be reflexive.
- Paths can be concatenated. If a path $< a, \ldots b >$ continues with path $< b, \ldots c >$ then there is also a concatenated path $< a, \ldots b, \ldots c >$ in D and therefore $(a,c) \in P$, which makes P transitive.
- Antisymmetry is shown by contradiction. Assume $< x, \ldots y >$ a non-zero length path. Since the edge set is cycle free, there cannot be any path $< y, \ldots x >$, if not $x = y$. Therefore the path $< x, \ldots y >$ has length zero and $x = y$. Thus, P has antisymmetry. □

Now we also know that routing graphs are partially ordered.

Definition 20 *A partially ordered relation has a* **total order** *P, if it includes either relation between any of its elements. Totally ordered sets can be written as a list $(x_1, \ldots x_n)$, where $\forall i < j : x_i\ P\ x_j$.*

Theorem 3 *A partial order can be extended to a total order.*

Proof. Let $P = P'$ be a partially order. We show that we can insert additional tuples into P' resulting in a total order.

For any a, b that are incomparable, i. e. neither $a\ P\ b$ nor $b\ P\ a$ hold, we define $a\ P'\ b$ through $(a,b) \in P'$. And P' is still a partial order, since neither transitive, reflexive or antisymmetry have changed. Then for any pair a, b either $a\ P'\ b$ or $b\ P'\ a$ holds, which identifies a total order. □

A routing graph can be totally ordered. Total order means that there is a node sequence where all edges go from a high node to a lower node. We say the edge direction complies with the total order.

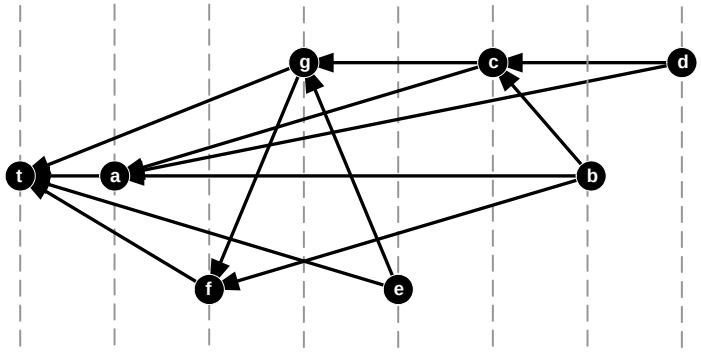

Figure 2.2: Routing Graph enhanced to Total Order

For the example in Fig. 2.2 the partial order of routing graph of Fig. 2.1 had been extended to a total order by including additional edges:
$P' = P \cup \{(a,f),(a,g),(a,e),(f,e),(f,c),(f,d),(g,b),(e,b),(e,c),(e,d),(b,d)\}$. Fig. 2.2 still

shows the original edge set P, not P'. However, its node layout complies with P', which introduces a definite node sequence. Note that all edge directions point towards the destination node, which is the only node with incoming edges only.

Regarding a routing graph, a node is redundantly connected, if it has two distinct paths to destination. We show now that any maximum routing graph has nodes, which are not redundantly connected to desination.

Lemma 1 *Any routing graph has one non-redundant node.*

Proof. Let $T = (V,E,D,t)$ be a routing graph. Since T is a dag, it is also partially ordered through its edges D. Find a compatible node sequence $S = (v_1, \ldots v_n)$ providing a total order P'. The second node v_2 cannot have first-hop redundant paths to the destination node. Since all edges comply with the totally ordered node sequence, only the edge (v_2, v_1) can be outgoing from that node. Networks do not include parallel edges, and therefore v_2 has only a single outgoing edge and is not redundantly connected. □

Fig. 2.2 visualizes the idea of Lemma 1. In this totally ordered routing graph, node *a* becomes second element. This means that outgoing edges are only allowed to nodes more left in the total order, namely the destination node only. Without parallel edges, then no two outgoing edges are possible.

If we cannot gain full redundancy with routing graphs in general, we have to seek for another approach to grant redundant paths for all nodes.

2.2 Full Redundancy Routing Graphs

In Chapter 1 we quoted a failure analysis in a wide area network. We showed that the occurrence of single link failures is likely.

Assumption 2 *We assume the occurrence of a single link failure over the total network.*

We imply that all used links are physically independent as defined in Chapter 1 and Appendix A. This will have an considerable effect on network resilience, since the method covers any single link failure. In Chapter 5.2 we will analyze in detail which failure sets can be recovered.

2.2.1 Reserve Links

Redundancy is given by two independent network paths. Some protocols divide traffic onto multiple paths. Messages can also be sent on either one of them – during normal operation. In case of a failure, the existence of a second path allows for immediate recovery. However before failure occurrence, a backup path can remain completely unused. We use this knowledge to merge network reserve capacities.

Definition 21 *A* **reserve link** *is a stand-by resource, which is not used during normal operation.*

Any failure not covered by multiple active links, can be recovered using one reserve link. Therefore we do not use several reserve links simultaneously. Schollmeier, et. al. propose in their KING project [Sch01] the O2-Routing concept, using joker links [SCK+03]. Their algorithm produces a routing graph with one joker link, that is used in case of failure in either direction.

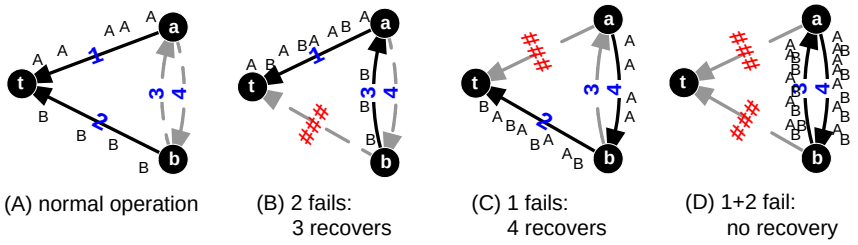

Figure 2.3: How Shared-Reserve-Links provide fallback routing

Let us review a topology of three nodes like Fig. 2.3 (A). There are three links between nodes t, a, b. Fig. 2.3 visualizes transmitted messages on the network to destination t. ”A” are messages being sent from a to t, resp. ”B” being transferred from b towards destination node t. During normal operation node a uses the direct link 1 to destination, while b uses the distinct link 2 (Fig. 2.3 (A)). If link 2 fails, router b utilizes link 3 to transmit messages to node a, that forwards them to t. In this case, link 1 carries both messages from a and b (Fig. 2.3 (B)). Because of the symmetric topology link 4 provides recovery for failing of link 1 (Fig. 2.3 (C)), where link 2 carries both ”A” and ”B” messages.

From the viewpoint of router a, the routing table towards t list two links: $\{1,4\}$, where link 1 is active and used during normal operation and link 4 is a reserve. With local recovery routing converges extremely fast, since no routing information has to be propagated through the network.

Any single link failure (Assumption 2) can be recovered by activating at most one reserve link. Fig. 2.3 (D) explains what can happen if two neighbor routers enable their reserve links 3, 4 simultaneously. Then router a forwards its traffic towards t onto link 4, where it is resent into a loop back to a. In this situation links 3, 4 form a cycle and will drop messages. Again, operation is guaranteed with at most one failing link.

Formally, such pairs of reserve links are called Shared-Reserve-Link, since they share link capacity reserved for recovery purposes. They remain unused during normal operation and either direction will be activated.

Definition 22 *A* **Shared-Reserve-Link** *(SRL) consists of a pair of opposite-directed links* (u,v) *and* (v,u).

We have now seen an example graph where a Shared-Reserve-Link provides recovery. The previous definition of a routing graph does not allow Shared-Reserve-Links since they consist of

a cycle. An obvious approach is to extend the routing graph with a set of reserve edges and tests whether none of them introduces a cycle into the routing graph.

Definition 23 *An* **extended routing graph** *T'=(V,E,D,R,t) consists of a routing graph* $T = (V,E,D,t)$ *and reserve edges* $R \subseteq E$ *(i) which are disjoint with the set of active edges. (ii)* $D \cup R$ *consists of two outgoing edges for any node. (iii)* $(V,E,D \cup \{r\},t)$ *is routing graph for any* $r \in R$*.*

Now we are able to test, whether a set of reserve edges correctly extends a routing graph. This is rather expensive since we construct $n = |R|$ routing graphs and search for cycles. Here the verification time extends linear with the amount of Shared-Reserve-Links.

An edge is either active, reserve or unused for that routing graph (i). If the set of active edges with any single reserve edge is a routing graph (iii) then it is also free of cycles. Criteria (ii) provides first-hop redundant paths to destination, since the routing graph includes a spanning tree and two edges from any node.

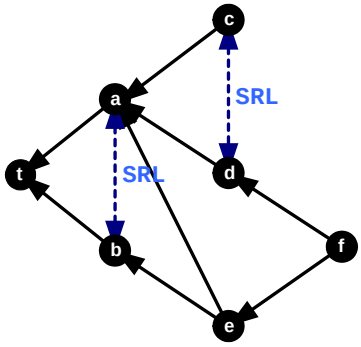

Figure 2.4: An extended routing graph

Fig. 2.4 contains an extended routing graph for destination node t. Here we find two inner node types: either nodes that have two active outgoing edges (nodes e, f), or nodes with one active and one reserve edge: a,b,c,d. Note that the latter always occurs in a pair of nodes, connected through a Shared-Reserve-Link. Every node is adjacent to at most one SRL. With a trick we can nicely arrange the nodes: coupling every two nodes that are connected by an SRL (nodes a,b and c,d). This covers all reserve edges. The remaining graph of active edges is acyclic and partially ordered. Fig. 2.4 is totally ordered into a sequence S, so that all active edges point from right to left. Cycles of reserve edges cannot be layouted like this, therefore we stack them vertical, gaining both nodes vertical into the same distance from destination.

Corollary 1 *An extended routing graph provides first-hop redundant paths for all inner nodes.*

Proof. Let $T = (V,E,D,R,t)$ be an extended routing graph with a node $v \in V \setminus \{t\}$. We show that node v has two first-hop redundant paths $< v,x_1,\ldots x_n,t >, < v,y_1,\ldots y_m,t >, x_1 \neq y_1$ in D to destination.

Criteria (ii) of an extended routing graph guarantees two distinct outgoing edges from any inner node. Let x_1, y_1 be the end nodes of two outgoing links from v. Because T includes a spanning tree, paths $< x_1, \ldots x_n, t >$ and $< y_1, \ldots y_m, t >$ exist. The paths $< v, x_1, \ldots x_n, t >, < v, y_1, \ldots y_m, t >$ are first hop redundant since $x_1 \neq y_1$. □

Corollary 2 *An extended routing graph cannot be destroyed by a single link failure: all inner nodes have redundant paths towards destination.*

Proof. Let T be a extended routing graph with $f \in E$ a single destroyed link. We show that any node v has a path towards destination.

Assume that before a link failure, node v has been connected through path $< v, x_1, \ldots x_n, t >$ to destination. Either f is an edge in the path or not. If it is, then there is a latest node x_i with a working path $< v, x_1, \ldots x_i >$. Since T is an extended routing graph, its node x_i has first-hop redundant paths. One is $< x_i, \ldots x_n, t >$, but there is another $< x_i, y_j, \ldots, y_m, t >$. This cannot include edge f because that would form a cycle. Therefore $< v, x_1, \ldots x_i, y_j, \ldots, y_m, t >$ is a working path after edge f fails. □

Now we have seen that extended routing graphs provide redundant routing and any single link failure can be recovered.

2.2.2 Cycle Avoidance

The last Section presented extended routing graphs. Their definition requires a test for any reserve edge in the graph, to proof the extended routing graph property. Now we will refine these and find a representation, which can be tested in one run. Then we introduce a new subtype with restricted cycle length: HammockSets.

Definition 24 *If any reserve link in an extended routing graph is part of an SRL, the routing graph is called* **regular**. *The regular routing graph* $L = (V, D \dot{\cup} R, t)$ *refers to the united edge set of active and reserve links of an corresponding extended routing graph* $T = (V, E, D, R, t)$.

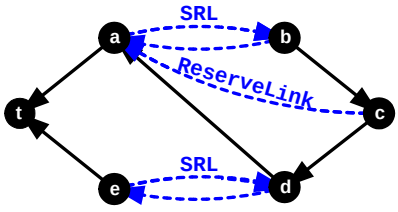

Figure 2.5: Non-regular Routing Graph

Fig. 2.5 is a non-regular routing graph, because of the edge (c, a). This edge cannot be removed, since it provides a first-hop redundant path for node c. But inserting another reserve (a, c) is

also not possible, since it would close a cycle with a 1-reverse-edge (a,c,d,a). For this topology there exists no regular routing graph.

To check all cycles in one run, we union the set of active edges with the reserve edges. The join introduces cycles, for which we will enforce certain cycle properties.

Corollary 3 *In a regular routing graph an edge e with opposite-directed edge $\widehat{e}$ is reserve, while an edge e without $\widehat{e}$ is an active link.*

Proof. Because any reserve edge is part of an SRL in an regular routing graph, it can be detected through its opposite-directed edge. □

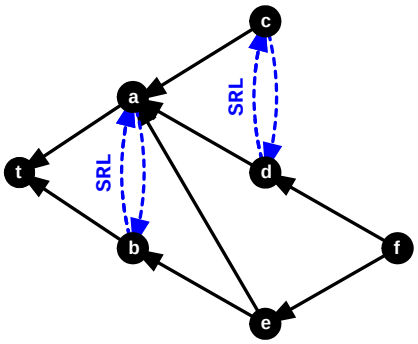

Figure 2.6: A Regular Routing Graph

The regular routing graph in Fig. 2.6 is described by a single set of edges (dashed lines only shown for orientation). Cycles included in that graph obtain properties deriving from an extended routing graph.

Definition 25 *A simple cycle with edges $C =< v_0, \ldots v_{n+1} >$ has k-***reverse edges** *in a graph $G = (V,E)$, if there are k different edges $e_i = (v_i, v_{i+1})$ such that the reverse edges $\widehat{e_i}$ are part of the graph. Cycles with 2-reverse edges are called* **single-fault-blocking cycles**.

Note, that Fig. 2.6 contains only single-fault-blocking cycles. This states, that all these cycles contain two reserver links, that avoid an activated cycle after one link failure.

Theorem 4 *If a regular routing graph includes cycles, then all cycles are single-fault-blocking.*

Proof. Let $L = (V, D \cup R, t)$ be a regular routing graph to an extended routing graph $T = (V,E,D,R,t)$. Let $C =< e_1, \ldots e_n >$ be a cycle of edges $e_i \in D \cup R$. Each cycle has at least length two since the network does not contain self-loops. C cannot consist purely of active edges D, because these are cycle free. Also, an extended routing graph with any single reserve edge does not contain a cycle – property (iii). Therefore at least two edges of any cycle must be reserve edges. Any reserve edge is part of a Shared-Reserve-Link and has a reverse edge. Therefore cycles contained in such a graph have two edges with reverse edges and are then single-fault-blocking. □

Single-fault-blocking cycles will not be active assuming one global link failure. In detaile, no cycle will be active if any single-fault-blocking cycle has at most one link failure. Above Fig. 2.6 is able to recover routing if one of $(a,t),(b,t)$ and one of $(c,a),(d,a)$ fail.

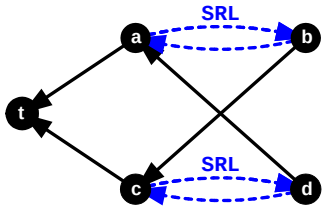

Figure 2.7: Regular Routing Graph with a 4-Cycle

Fig. 2.7 shows an regular routing graph with two Shared-Reserve-Links (a,b) and (c,d). Any cycle in this graph is single-fault-blocking. The cycle (a,b,c,d,a) consists of four edges and includes two distinct SRLs.

Definition 26 *For an regular routing graph $L = (V, D\dot{\cup}R, t)$ the union $H(t) = D\dot{\cup}R$ is called* **HammockSet**[1]*, if no cycle longer than two is included in $H(t)$.*

The HammockSet definition introduces further restrictions. Fig. 2.5 is an extended routing graph but not regular, while Fig. 2.7 is regular but not a HammockSet. Since Fig. 2.6 does not include cycles longer than two, it is a HammockSet graph.

Corollary 4 *Two Shared-Reserve-Links have no common nodes in a HammockSet graph.*

Proof by contradiction. Assume two SRLs (a,b) and (b,c) have a common node b. Then the simple cycle (a,b,c,b,a) with length four occurs which is not allowed in a HammockSet. □

Summarizing, we have found that HammockSets are a special type of extended routing graph which include an active spanning tree. It is regular, therefore all occurring reserve links are part of a SRL, and cycles are single-fault-blocking with maximum length of two.

2.2.3 Components

The active edges of an extended routing graph are partially ordered. We can find this property embedded in strongly-connected HammockSet components. Until otherwise stated all components reviewed in this thesis are strongly connected.

Definition 27 *Two graph nodes are* **strongly-connected** *if they are reachable from each other.*

Corollary 5 *Only nodes in a common cycle are strongly connected to each other.*

[1]The name HammockSet derives from real hammocks, that usually consists of ropes with knots. If a rope wrenchs, other ropes keep the hammock. The network model rests upon the same idea.

Proof. Nodes x,y in a cycle have paths $< x,\ldots y >$ and $< y,\ldots x >$. Then x,y are strongly connected. Otherwise, if two nodes x,y are not in a cycle, at most one path $< x,\ldots y >$ or $< y,\ldots x >$ is included in the HammockSet. But then x and y are not strongly connected. □

Corollary 6 *In a HammockSet, only SRL-connected nodes are strongly connected.*

Proof. The set of active edges is cycle free. Only Shared-Reserve-Links include cycles in a HammockSet. □

Corollary 7 *Strongly-connected components of a HammockSet contain one or two nodes.*

Proof. Only SRL-adjacent nodes merge into a component. As we have shown above, SRLs have no common nodes, therefore the maximum component size is two. Nodes without adjacent Shared-Reserve-Link have a component size of one. □

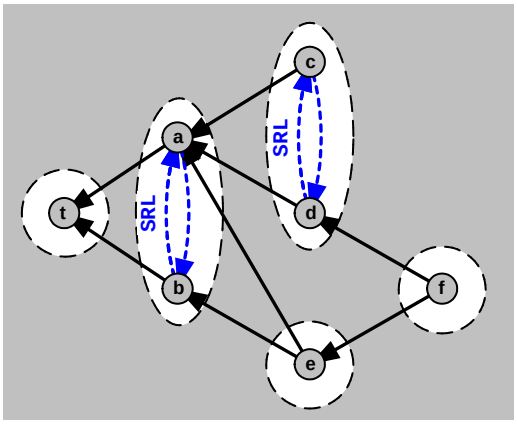

Figure 2.8: A HammockSet and its components

See Fig. 2.8 for an example of HammockSet components. It shows the HammockSet of Fig. 2.6 decomposed into its strongly-connected components. Note that all edges between components have been active edges of the extended routing graph.

Theorem 5 *The graph of strongly-connected HammockSet components is partially ordered.*

Proof. Let D' be the relation of directed egdes between HammockSet components. We show that D' equals the set of active edges D of the extended routing graph.

Edges of a HammockSet consist of the union of active and reserve edges. The components swallow all reserve edges, since they represent Shared-Reserve-Links. Then only active edges from the Extended Routing Graph occur in the component graph. The set of active edges is a directed acyclic graph and therefore is partially ordered. □

The nodes in Fig. 2.8 include a partial order $\{(a/b,t),(c/d,a),(e,a/b),(f,c/d),(f,e)\}$. It is not a total order, since the components e and c/d have no direct path in either direction. There is neither a path $< e,\ldots c/d >$ nor $< c/d,\ldots e >$ (see Definition 19).

Definition 28 *In a list of strongly-connected HammockSet components, a component C_i is* **nearer to destination** *as C_j, iff $i < j$.*

Theorem 6 *The partial order of a strongly-connected HammockSet component graph is extensible into a totally ordered list of components.*

Proof. Let $H(t)$ be a HammockSet to an extended routing graph $T = (V,E,D,R,t)$ and C_i strongly-connected components of $H(t)$. We construct a list $C = (C_1,\ldots C_n)$ by extending the partial order relation induced from active edges. Since the graph consists only of acyclic active edges. List C introduces additional tuples that relate non-connected nodes of the previous edge set. □

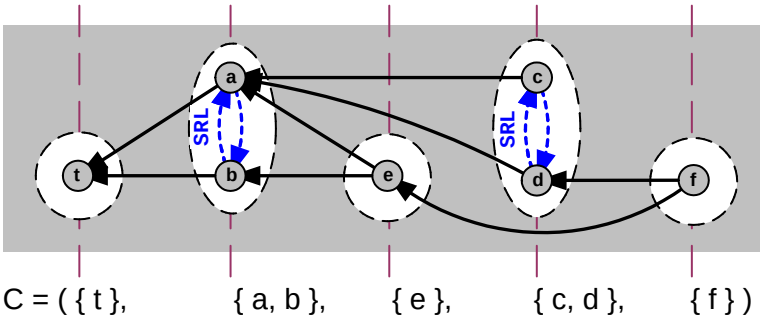

Figure 2.9: Total-ordered HammockSet Component List

Fig. 2.9 shows a component list for the HammockSet in Fig. 2.6. The first list entry always consists of the destination node of the corresponding HammockSet, since it has no outgoing edge. $C_2 = \{a,b\}$ is the candidate node pair for $\cup_{i<2}C_i = \{t\}$. The node subset $\cup_{i<3}C_i = \{t,a,b\}$ has two candidates: $\{e\}$ and $\{c,d\}$. Here we choose the single note candidate first, to minimize the amount of total used Shared-Reserve-Links. Note that the HammockSet edges do not relate the components $(\{e\},\{c,d\})$. In Fig. 2.9 $\{e\}$ is nearer to destination than $\{c,d\}$, which incorporates a total order. The pair $(\{c,d\},\{e\})$ would result in another component list $C' = (\{t\},\{a,b\},\{c,d\},\{e\},\{f\})$. The total order complies with the partial order, introduced through the set of active edges D, since all edges are directed towards destination node component. Note that each inner component has two outgoing edges to other components.

Theorem 7 *All inner components of a strongly-connected HammockSet have two outgoing edges.*

Proof. Let $C = \{u\}$, resp. $C = \{u,v\}$ be an inner component, i. e. not the destination node component. We show that the component has at least two edges to nodes nearer to destination.

There are two cases. Either $C = \{u\}$ is a single node. Since u is part of a HammockSet, it has first-hop redundant paths to destination, and therefore has two distinct outgoing edges. Otherwise the component is $C = \{u,v\}$ and both HammockSet nodes u,v provide an active path towards

destination (routing graph criteria). (u,v) is a SRL and consists of reserve links, both u,v must have at least one active outgoing edge. □

We have seen how HammockSets transmute into strongly-connected components, which are partially ordered and where any component has two outgoing edges. Turning down these features allow us to construct HammockSets.

Chapter 3

Network Topology

To enable resilient communications, HammockSets for all nodes must exist. Fig. 3.1 shows such a Complete HammockSet Graph (CHG). In this Chapter we are going to describe qualities of such graphs. A CHG topology enforces a minimum count of edges for the network graph.

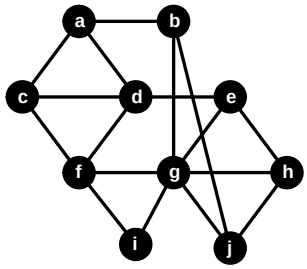

Figure 3.1: Complete HammockSet Graph

Definition 29 *A network topology is called* **Complete HammockSet Graph** *(CHG), if any network node has a HammockSet covering all topology nodes.*

3.1 Neighbor Nodes

A HammockSet incorporates neighbors, which associate the edge-relationship between nodes.

Definition 30 *The empty and the full node subsets are* **trivial**.

Definition 31 *A graph* (V,E) *is called n-***connected***, if any two-partition* $U \dot\cup U' = V$ *has at least n edges between* U *and* U'.

Theorem 8 *In a 1-connected graph, any non-trivial node subset has a neighbor.*

Proof. Let $G = (V,E)$ be a 1-connected graph and U a non-trivial node subset of V. We show, that one edge from U into $V \setminus U$ exists. Because U is non-trivial, both U and $V \setminus U$ contain elements. 1-connectivity states that any partition $U \dot\cup U'$ is connected by at least one edge (u,v). With $U' := V \setminus U$ this edge proofs the existence of a neighbor $v \in V \setminus U$ for node subset U. □

Corollary 8 *Networks are 1-connected.*

Proof. Let $N = (V,E,C)$ be a network and $U \dot\cup U'$ a two-partition of the node set. Since (V,E) is connected, there exist paths $P =< u,\ldots,v,v' \ldots u' >$ for any node pair $u \in U, u' \in U'$. Therefore some edge $(v,v') \in U \times U'$ connects U with U' and (V,E) is 1-connected. □

3.2 Candidates

We now extend the neighbor concept to find a suitable HammockSet description. This requires not only neighbors to a node set, but also neighbors with redundant first hop edges.

Definition 32 *Let $N = (V,E,C)$ be a network. Let $U \subseteq V$ be non-trivial node subset. Two connected neighbor nodes $x,y \in N(U)$ are called* **candidate node pair** *for U, if they have two edges $(x,u),(y,v)$ into the set $u,v \in U$. A neighbor node $x \in N(U)$ is called* **candidate node** *for a node set U, if it has two distinct edges $(x,u_1),(x,u_2)$ into the set $u_1,u_2 \in U, u_1 \neq u_2$.*

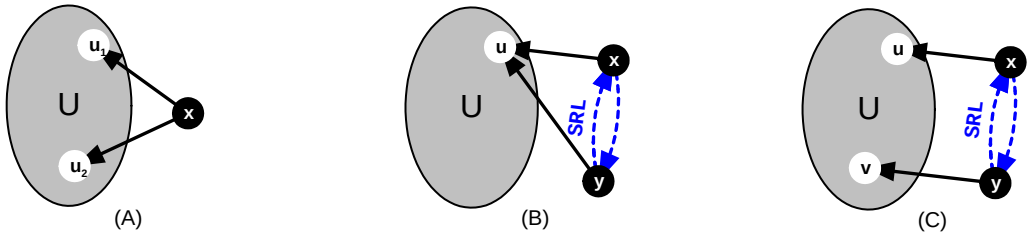

Figure 3.2: Candidate Node and Candidate Node Pair

Candidates extend the neighborhood relation towards redundant connections. There exist two generic candidate types: a candidate node x that has two edges to nodes $u_1,u_2 \in U$ (Fig. 3.2 (A)). The edges $(x,u_1),(x,u_2)$ must be distinct, and since the network does not model parallel edges, we can follow that $u_1 \neq u_2$. The other type is a candidate node pair (x,y). Both nodes have an edge into U, which can either touch the same node $u \in U$ (Fig. 3.2 (B)) or two different nodes $u,v \in U, u \neq v$ (Fig. 3.2 (C)) . The node pair is connected through a network edge that will be used as Shared-Reserve-Link, to grant first-hop redundancy to both nodes.

Candidates are closely related to HammockSets. If U contains a HammockSet $H(t)$, then a candidate of U can be redundantly connected to U and therefore attached to $H(t)$. Before we show that candidates procreate HammockSets, we show the reverse direction, that HammockSets consist of candidates.

Theorem 9 *In a Complete HammockSet Graph, there exists a candidate for any non-trivial node subset.*

Proof. Let $N = (V,E,C)$ be a Complete HammockSet Graph and U a non-trivial node subset of V. By definition, there exists a HammockSet $H(t)$ for any node $t \in U$. Let $C = (C_1, \ldots C_n)$ be a total-ordered list of strongly-connected HammockSet components for $H(t)$ (see Theorem 6).

Now we use the HammockSet $H(t)$ in its component representation to proof the existence of a candidate for the subset U. To do so, we mark all nodes of U in the component list. By this we gain three types of component marking: fully marked, nothing marked and partially marked. Since components have size one or two (see corollary 7), the latter can only appear in components of size two, where one node is part of U, while the other is not included in U. From this state, we determine the minimum j-value, so that C_j is a component that is not entirely marked, i. e. covered by U. For such a j all nearer components $C_i, i < j$ are fully marked. Furthermore C_j consists either of a single node not in U or a pair of nodes, where at least one is not in U. We show, that the unmarked fraction of C_j is a candidate for set U.

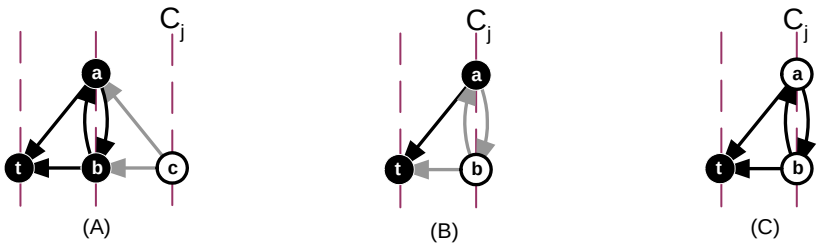

Figure 3.3: Candidates of a node subset

- If C_j is a single node component (Fig. 3.3 (A)), then its node x is not marked. Theorem 7 states that any component C_j has at least two edges to earlier components C_i, C_k with $i,k < j$. All these edges provide first-hop redundant outgoing links to the HammockSet destination. This makes x a candidate node.

If C_j is pair component, then two cases can occur: either both nodes are unmarked or one is marked, while the other is unmarked.

- With only one unmarked node x (Fig. 3.3 (B)) this has one edge inside the component and another from the active spanning tree property to a component that is nearer to destination. Both edges together guarantee first-hop redundancy.
- A node pair (x,y) of unmarked nodes in the minimum C_j component is neighbor to U with at least two edges into U. Each node has an active spanning tree to destination guaranteeing a single outgoing edge to a nearer component. The nodes (x,y) must have a common edge, since they are in the same HammockSet component. This makes (x,y) a candidate node pair for set U.

Summarizing, a CHG has candidates for any non-trivial node subset. □

The three different reasoning cases are shown in the next figure. First, the node subset $U = \{t, a, b\}$ is highlighted in the HammockSet $H(t)$ (Fig. 3.3 (A)). $H(t)$ includes an additional component C_2 consisting of a single node. Since C_2 is the left most not entirely marked component, $c \in C_2$ is a candidate for U. The other two examples show pair components, which are not completely covered by U. In Fig. 3.3 (B) the minimum j-component contains a, b, where node a is marked. Then b is candidate node, as it has two outgoing edges into U. Fig. 3.3 (C) shows an candidate node pair (a, b), where the minimum j component C_j is complete unmarked. No single node a or b can alone fulfill the candidate criteria for set U, since they lack a second edge. But united into a candidate node pair $\{a, b\}$ with their connecting edge (a, b), they gain two edges into U.

3.3 Hammock Topology

Next we are going to find, that existing candidates for all subsets indicate a Complete HammockSet Graph. We therefore use the representation of HammockSets in strongly-connected components.

Lemma 2 *Complete HammockSet Graphs have a candidate for any non-trivial node subset.*

Proof. We show that (1) any topology having candidates is a CHG and that (2) any CHG has candidates.

(1): Let $N = (V, E, C)$ be a topology, where any non-trivial node subset has a candidate and $t \in V$ a destination node. We construct a list of strongly-connected HammockSet components by iteratively adding candidates. By definition, the united set of already found nodes $U = \cup_i C_i$ has a candidate. The nodes of each candidate are together transmuted into a newly component C_i, which is then trailed to the component list.

We proof by induction, that $(C_1, \ldots C_n)$ is a list of strongly-connected HammockSet components. Base case is the $C_1 = \{t\}$, which forms a HammockSet of a destination node without any edges. In the induction step $n \rightarrow n+1$, we assume $(C_1, \ldots C_n)$ to form a valid HammockSet of nodes $U = \cup_i C_i$ and proof that $(C_1, \ldots C_n, C_{n+1})$ also fulfills HammockSet properties, where C_{n+1} has been candidate for U. We distinguish two cases: single node candidates and candidate node pairs.

- Let x be a single candidate node. It is going to be appended as a new component $C_{n+1} = \{x\}$. We can guarantee first-hop redundant paths, since each candidate has two edges into U, which already proofed HammockSet properties. Both edges are used outgoing only, this allows expansion of the active spanning tree and does not introduce any cycle.
- Let (x, y) be a candidate node pair, which is to be appended as new component $C_{n+1} = \{x, y\}$. The component-inner link (x, y) is used as Shared-Reserve-Link and the two outgoing links $(x, u), (y, v), u, v \in U$ as active links, then both nodes have first-hop redundant

paths to destination and the active spanning tree is extended up to x and y. The SRL (x,y) introduces a new cycle of length two, which is allowed for HammockSets.

Summarizing, in a topology where any non-trivial subset has a candidate, HammockSets for all nodes can be constructed, which proofs the CHG properties.

(2): A Complete HammockSet Graph provides candidates for any node subset. This has been shown in Theorem 9. □

See Fig. 3.2 for a draw of given candidate properties. Fig. 3.2 (A) holds a single candidate and has directed edges that allow its HammockSet membership. For candidate node pairs, Fig. 3.2 (B,C) displays the construction with a Shared-Reserve-Link.

Testing is still very expensive: $O(2^n)$, but we did not found another sufficient criteria yet. Two-connectivity and triangles are necessary criteria that can be easily tested cannot verify the existence of a Complete HammockSet Graph.

However, we found an adequately description for Complete HammockSets Graphs: candidates of non-trivial subsets. The network topologies describes by CHG allow resilient routing with fast local recovery procedures. Besides the two description of this graph class, we are going to apply Theorem 9, subset candidates in hammock topologies, as a founding Theorem for an HammockSet construction algorithm.

Chapter 4

HammockSet Construction

Many algorithms make use of partial solutions, i. e. solve a problem subset and then extend this solution to solve the whole problem. This scheme can also be applied for HammockSet construction.

Theorem 10 *Let $N = (V, E, C)$ be a network, t a destination node, for which some HammockSet exists covering all nodes. Let $V' \subset V$ be a subset of vertices. Then a HammockSet $H'(t)$ covering nodes in V' can be extended to a HammockSet $H(t)$ for complete V-coverage, so that it includes all the edges from $H'(t) \subset H(t)$.*

Proof. By definition, the topology has a HammockSet to destination t. Therefore Theorem 9 applies stating that a candidate C for subset V' exists. The candidate can be used to extend the HammockSet $H'(t)$. This can be repeatedly applied for any other subset $V' \subset V$, until the last candidate completes V. Then the HammockSet $H(t)$ including all edges from $H'(t)$ has been created. □

For a construction algorithm, we must just identify candidates and add these into the HammockSet with their corresponding edges. We do so in two alternating phases: after altering the destination set, discovery phase processes the closure of neighborhood nodes. Then a normalized state is reached, the next candidate is selected, and moved into destination set. These steps are repeated, until all nodes have been connected to the destination node.

4.1 Discovery Phase

By definition, candidates are a subset of destination set neighbors. We now provide a scheme to classify neighborhood nodes of the destination set. We do so by maintaining states for any node. These keep record about the number of recognized edges from destination set to a neighborhood node. We will use these states to identify candidates.

State	Node	Description
V	topology	exists in the network topology
B	border	one edge into destination has been found
R	redundant	at least two edges into the destination set exist
T	destination	has been added into the HammockSet

Table 4.1: List of Node States

Nodes can have one of four states (see tab. 4.1): all nodes are initialized with state V. If the first edge has been found from destination set, then the corresponding neighbor node will be marked as border node, indicated by B. With the second edge, the state changes to R marking the node as redundant. Consumed candidates are identified through state T, meaning that they now belong to the destination set.

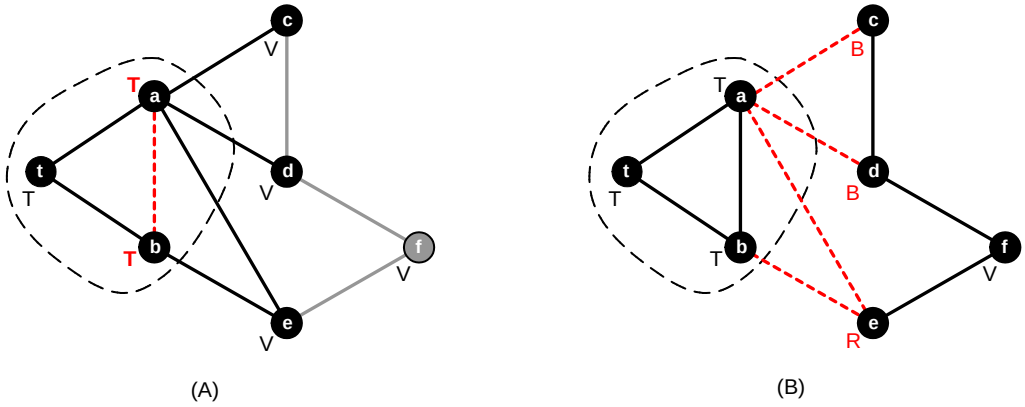

Figure 4.1: Neighborhood Discovery

Let us investigate the node state evolution with an example (Fig. 4.1 (A)). Here nodes a, b reached state T. Now, new neighbors must be discovered, namely the nodes that have common edges with a, b. All other neighbors of $T \setminus \{a, b\}$ have already been discovered before. Three edges are connected with node a: $(a,c),(a,d),(a,e)$. The opposite nodes $\{c,d,e\}$ shift from V into state B, meaning that one edge has been found from destination set. Primarily, we search for redundant nodes and these must have two edges to become candidate. With processing edges from node b, a second edge to node e is found. At this time e is already in state B and is switched to R (Fig. 4.1 (B)). With that, all edges from the destination set had been processed. Such a state is called normalized.

Note that we also have modified the state of visited edges: from E to state U (tab. 4.1). The third state, H for HammockSet, can be reached through candidate selection. As the topology is represented by non-directed edges, the procedure will find either the edge or the reverse edge with swapped nodes, i. e. (u,v) instead of (v,u). However, when some transformation turns an edge into the HammockSet, then the edge direction becomes relevant. Also note, that the displayed states in Fig. 4.1 (B) are reached independently of the edge processing order. To proof

State	Edge	Description
E	topology	waiting for discovery
U	unused	processed
H	HammockSet	part of the HammockSet

Table 4.2: List of Edge States

this, we must first formalize state transitions.

Definition 33 *The* **processing state** *is an eight-tupel $S = (E,U,H,V,B,R,T,C)$, where the first three correspond to the edge states E,U,H, the next four to the node states V,B,R,T, and C is a list of strongly-connected HammockSet components.*

In the processing state tupel, nodes and edges are distinguished by their state into node sets resp. edge sets. Since each node and edge has one state at a time, the sets are disjoint.

We are now going to formally define the transition rules that alter the processing state in a well-defined fashion. Such rules contain a number of state conditions and state transitions. A rule may only be applied if any single term can be fulfilled with the current state.

Definition 34 *A* **state transition** $[S \xrightarrow{u} S']$ *is successful, if a node u with a previous state S is switched to state S'.*

Definition 35 *A* **state transition** $[S \xrightarrow{(u,v)} S']$ *is successful, if either an undirected edge (u,v) resp. (v,u) has state S can be successfully transferred as directed edge (u,v) into state S'.*

Definition 36 *A* **transition rule** $TR = [c, st_1, \ldots st_n, b_1, \ldots b_m]$ *can be applied if all conditions b_i match and any contained state transition st_i can be successfully executed under one variable mapping. Then c is either empty, or gives a candidate node or node pair that is appended to the component list.*

The neighborhood discovery can now be formalized into three transition rules. TR1 describes the conversion of neighbor nodes v into B, where an edge from T must exist that has not yet been processed. TR2 turns border nodes u into redundant nodes, if another edge $(u,v), v \in T$ can be found. TR3 collects additional edges for nodes that already turned into border nodes. Note that during discovery phase, no candidates are added into the component list.

Transition Rule 1 $[\ \emptyset, E \xrightarrow{(u,v)} U, V \xrightarrow{v} B, u \in T\ ]$

Transition Rule 2 $[\ \emptyset, E \xrightarrow{(u,v)} U, B \xrightarrow{v} R, u \in T\ ]$

Transition Rule 3 $[\ \emptyset, E \xrightarrow{(u,v)} U, u \in T, v \in R\ ]$

Definition 37 *Initial state for a topology $N = (V,E,C)$ is $S_0 = (E,\emptyset,\emptyset,V,\emptyset,\emptyset,\emptyset,[\,])$. The transition $[\ \{t\}, V \xrightarrow{t} T\ ]$ launches the HammockSet construction for destination node t.*

Definition 38 *In a* **normalized state** *the destination set neighborhood has been completely discovered.*

Corollary 9 *A state is normalized, iff none of TR1, TR2, TR3 applies.*

Proof. All three transitions require an unmarked neighborhood edge which connects a destination set node with an external node having state V, B, R. If none exists, all neighborhood edges have been discovered and a normalized state has been reached. □

Now we refine the neighborhood node states. This knowledge will be used in the upcoming phase, where candidates are identified by state.

Theorem 11 *Invariant for normalized states: $B \dot{\cup} R$ contains all neighbor nodes of the destination set T.*

Proof. Let S be a normalized state, successor of the initial start state and its destination node launch. Since S is normalized, all edges to neighbored nodes have been processed. Then all neighborhood node states are up to date, meaning that they left V and arrived in either B or R. □

Theorem 12 *The normalized state is independent of the edge processing order during discovery phase.*

Proof. Let S, S' be two succeeding normalized states, derived from one common state. We show that $S = S'$. Since in both S, S' the destination set neighborhood had been completely discovered, the same amount of edges had been processed. Regarding to a single neighborhood node, the same count of edges into the destination set is examined for any processing order. Since advancing from V to B to R depends only on the amount of discovered edges, the resulting node state is independent of the edge processing order. Also all neighborhood edges are going to be shifted from E to U, independent of the order. The component list remains unchanged in discovery phase. Therefore $S = S'$ and the upcoming normalized state is definite. □

In this Section we found that the discovery phase is deterministic and results in a state, where all R-nodes have two or more edges into the destination set, while B nodes are neighbored with exactly one edge. There exist several sequences to the next normalize state, but all of them result in the same state. However, during candidate selection we will find a degree of freedom that can result in differing normalized states.

4.2 Candidate Selection Phase

After discovery phase set up correct states for all neighborhood nodes, the selection phase utilizes these state information to find candidates. Single node candidates are R-marked, while candidate node pairs only occur in node set B.

Fig. 4.2 shows a network with destination set $\{t, d, f, h\}$. Because this topology contains HammockSets, Theorem 9 says that candidates can be found for any node subset. Nodes b, e, g has

been discovered as redundant nodes, meaning that discovery phases found at least two distinct edges from destination set to these nodes.

The candidate selection phase has two transition rules only: one seeks for single candidate nodes, while the other deals with candidate node pairs.

Transition Rule 4 $[\ \{x\}, R \xrightarrow{x} T, \forall u_i : U \xrightarrow{(u_i,x)} H, u_i \in T\ ]$

Rule four shifts a redundant node x into the destination set and sets the direction for all edges from x to the destination set. Note, that this rule initializes all edges between x and T.

Fig. 4.3 (A) shows a situation, where TR4 applies. Node x is marked redundant, because of three discovered edges into the destination set: $(x,u_1),(x,u_2),(x,u_3)$. Since $\{u_1,u_2,u_3\} \subset T$ are redundantly connected nodes, node x can also gain redundant paths to destination (see Theorem 10). Node x is trailed to C and its state shifted to T.

Some normalized states have no redundant marked nodes (Fig. 4.4). Then there are no nodes where transiton rule 4 could apply. But there exists a second type of candidates: pairs of two nodes, connected by an edge. The algorithm must search for these pairs. In Fig. 4.4 five edges between border nodes $(b,d),(b,f),(d,g),(e,f),(f,g)$ exist. Each of them is a possible candidate node pair at this state. Note, that all such nodes are marked as border nodes, since they have one edge into the destination set.

Transition Rule 5 $[\ \{x,y\}, B \xrightarrow{x,y} T, U \xrightarrow{(x,u),(y,v)} H, E \xrightarrow{\widehat{(x,y),(y,x)}} H, R = \emptyset, u,v \in T\ ]$

Transition rule five moves both nodes from B to T and shifts the corresponding edges into H, while (u,v) is turned into a Shared-Reserve-Link. Note, that this rule can only apply, if the set of redundant nodes is empty. A scenario for TR5 is shown in Fig. 4.3 (B). Here the connected node pair x,y provides a candidate for destination set T. The HammockSet can be extended by nodes x,y, since both have two outgoing edges and do not introduce a cycle with a length greater than two.

Fig. 4.6 lists a sequence of normalized transition steps during HammockSet construction. TR4 selects one node per tranisiton and redundantly connects them to the HammockSet destination. After the transitions in (A) and (D) the set of redundant nodes is empty, prohibiting TR4. Instead, TR5 applies with an edge inside the border node set. The edge between the two nodes will be transformed into a Shared-Reserve-Link. In Fig. 4.6 (G), the last added node i gains three outgoing edges with TR4: $(i,c),(i,g),(i,h)$.

The other Fig. 4.5 contains the list of strongly-connected HammockSet components, where each step relates to the chosen candidates of Fig. 4.6. Each component has two outgoing edges to nodes nearer to destination. We have seen in Theorem 7 that this always holds for totally ordered component lists.

Theorem 13 *After launching processing, C contains a valid list of strongly-connected HammockSet components with edge set H.*

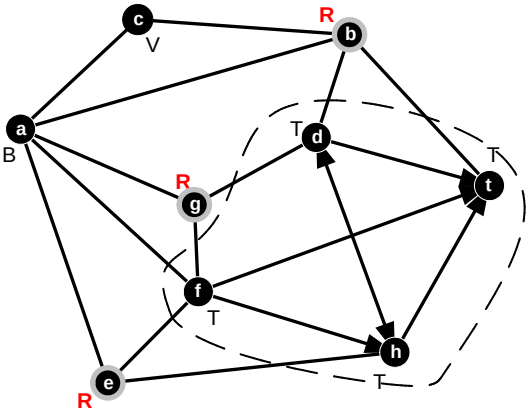

Figure 4.2: Topology with Annotated Nodes in a Normalized State

Figure 4.3: Candidate Nodes

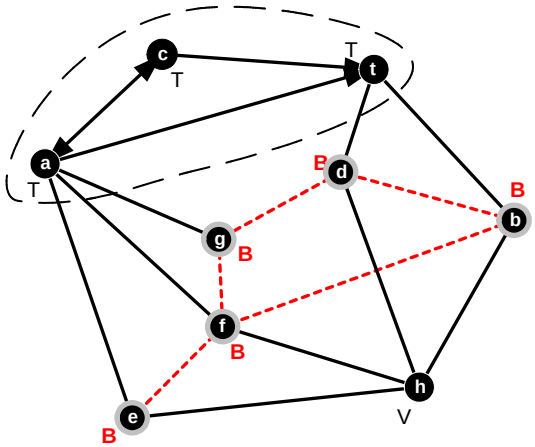

Figure 4.4: State without Single Candidate Nodes

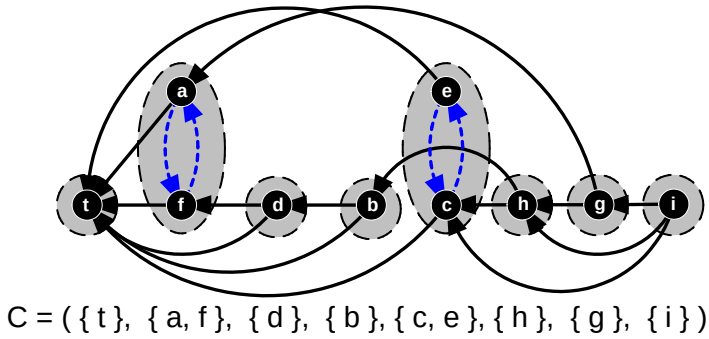

Figure 4.5: List of Strongly-Connected HammockSet Components

Proof. Base case: the launching transition introduces the destination node as first list element $\{t\}$. The edge set is empty at that state. The discover phase finishes after complete neighborhood discovery into a normalized state.

In the induction step, we assume a valid HammockSet component list C and show that TR4, TR5 both correctly extend the HammockSet with candidates. TR4 inserts a redundant node with at least two neighbor edges, which are all shifted to H and will be used as active outgoing edges (Fig. 4.3 (A)). This correctly extends the active spanning tree and no cycle is introduced. Node x also gains first-hop redundant paths for destination.

For TR5, a Shared-Reserve-Link will be constructed between the pair nodes, and with these border nodes and the additional reserve links, two first-hop redundant paths are provided (Fig. 4.3 (B)). The generated cycle has length two and is allowed in HammockSets. With the two newly created edges, each node has two outgoing links, where one is the part of the Shared-Reserve-Link and the other is active. Therefore both nodes of the candidate node pair are redundantly connected to the destination set.

Summarizing, C contains a HammockSet component list of all nodes that have been added yet. □

The last Theorem gives the instructions to implement Lemma 2. Next we will show that we always construct maximized HammockSets.

Definition 39 *A HammockSet $H(t)$ is* **maximized**, *if no enlarged HammockSet $H'(t) \supset H(t)$ exists.*

Theorem 14 *The algorithm described by TR1-5, constructs maximized HammockSets.*

Proof by contradiction.
Assume, the algorithm would result in a HammockSet $H(t)$, which does not include edge e nor $\widehat{e}$, but $H'(t)$ is a valid HammockSet involving the edge e. We know, the discovery phase

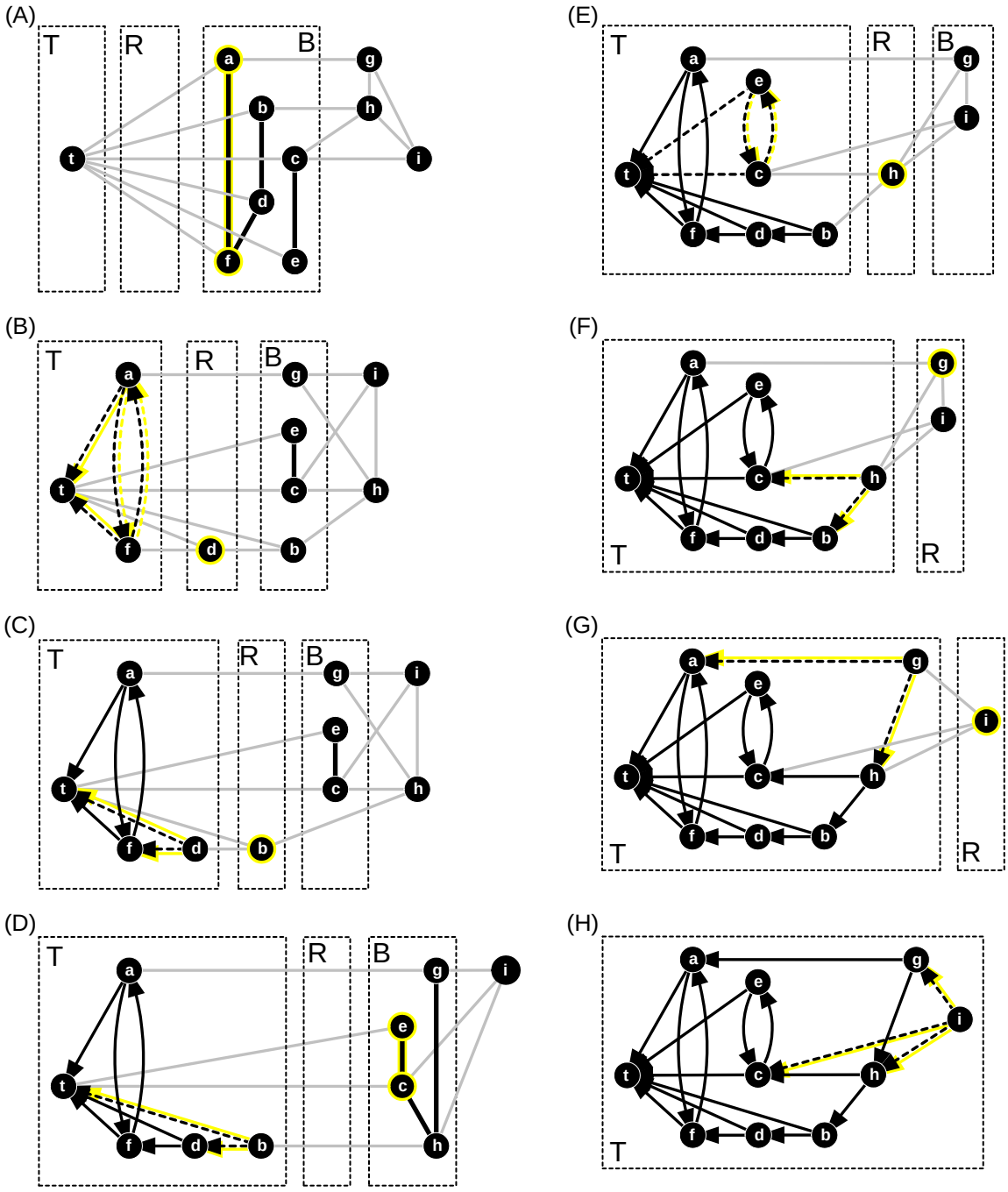

Figure 4.6: Normalized States of a Transition Sequence

finishes after complete neighborhood coverage (Definition 38). Thus edge *e* is discovered at the right time, when one of its end nodes had been processed. A selection of that node with TR4 must then use all links including *e*. Therefore TR4 includes all possible edges, covering *e*. TR5 is a minimized case. There exist only two border nodes, that are singely connected into the destination set. However, all edges to be mentioned during TR5 will be included into the HammockSet. No edge can be lost. Summarizing, the construction always results in maximized HammockSets. □

The guarantee to construct maximized HammockSet for a topology is very important to provide quality HammockSets. However, there could be another maximized HammockSet deriving from other candidate choices.

Theorem 15 *For a normalized state, either TR4 or TR5 applies.*

Proof. TR4 shifts a node from *R*, while TR5 can only succeed with an empty set *R*. Therefore an ambiguous choice cannot appear between TR4 and TR5. □

Theorem 16 *In a normalized state, several candidates may compete to be processed first.*

Proof. If two or more nodes are marked *R*, any of them could be processed first with transition rule 4. In case of rule 5, any two border nodes connected through an edge can be selected as candidate. If the border node set has multiple edges included, then any of them could be chosen. □

Then we examine, whether the candidate processing order can result in various HammockSets.

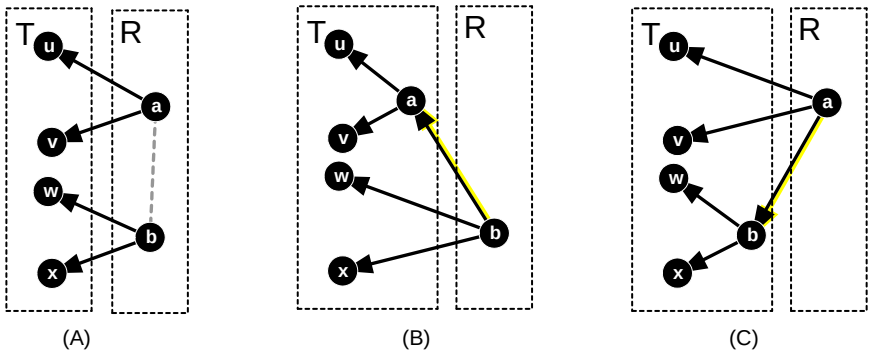

Figure 4.7: Candidate Processing Order influences Edge Direction

Corollary 10 *The direction of a HammockSet-included link can depend on the candidate processing order.*

Proof. First we consider single candidate nodes with TR4. Let a, b be two candidate nodes for a normalized state (Fig. 4.7 (A)). Selection of candidate *a* results in the state shown in Fig. 4.7 (B), while using *b* first results in Fig. 4.7 (C). The resulting HammockSets differ in the direction of edge (a, b).

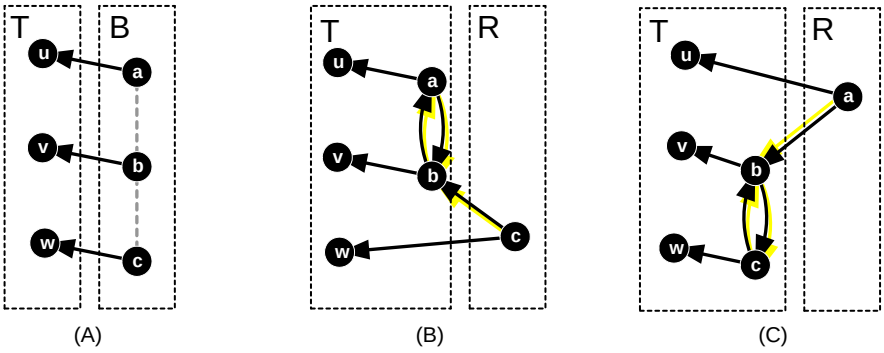

Figure 4.8: Candidate Processing Order influences SRL position

Fig. 4.8 (A) explains similar circumstances for candidate node pairs. Again, we can choose between several candidates. Selection of (a,b) at first, makes an active direct link (c,b) and SRL (a,b) (Fig. 4.8 (B)), while starting with (b,c) creates link (a,b) and SRL (b,c) (Fig. 4.8 (C)). Summarizing, the choice of connected candidates influences some edge directions of the resulting HammockSet. □

The above corollary identifies one freedom degree in HammockSet construction. If we want to find the optimum HammockSet, then we must find the right decisions, i. e. choose the right candidates that lead to the edge directions required for the optimum traffic distribution. Note that such a degree of freedom depends on the topology. If the neighborhood has exactly one redundant node, or the broder node includes exactly one edge, then the next normalized state is also definite.

4.3 Complexity

Theorem 17 *The discovery phase terminates into a normalized state within $|E|$ transitions.*

Proof. Let S be an algorithm processing state. Any discovery phase transition reduces the number of E-edges by one. At most $|E|$ nodes are neighbored to the destination set. Therefore $\leq |E|$ steps are required to shift them into U and reach a normalized state. □

With that we can gain information about the total running time. For each edge from a newly added destination node $u \in T$ either one of the three transition rules applies. Each edge is processed exactly once over all discovery phases. The overall count of transition steps is therefore the amount of topology edges.

Theorem 18 *The total number of transition steps is bounded by $|V|+|E|$.*

Proof. We examine both phases separately. Discovery phase iterates above all edges: $|E|$ transitions. For candidate selection we investigate both transition rules 4 and 5. TR5 requires fewer steps than TR4, since it transforms two nodes at once. Rule 4 only removes one single redundant

node at a time, therefore the maximum step count during candidate selection is limited by the number of nodes $|V|$. □

State Transition	Running Time
TR1	$k\ log\|E\|$
TR2	$k\ log\|E\|$
TR3	1
TR4	k
TR5	$log\|E\|$

Table 4.3: Running time of Transition Rules

Next we proof the algorithm time complexity. Table 4.3 gives an overview by transition rule. The implementations must maintain a search tree of border node edges and hash map of edges within the border node set.

Theorem 19 *TR1 and TR2 have complexity $O(k\ log|E|)$, where k is the maximum node outdegree.*

Proof. Parameter to these transitions is the edge to be discovered. They perform two state changes: the edge (u,v) is turned into U, and the end node state is switched to B (resp. R for TR2). Then the search tree of border node edges must be updated. TR1 as it procduces a new border node x, examines all neighbors of x and insertes the edge into the search tree, if another border node can be found. This requires the check for at most k edge end node states and evtl. insertion into the search tree: $O(k \cdot log|E|)$. After that, TR1 initializes a hash map to allow $O(1)$ lookup of the edge to x. TR2 does the reverse operation. It shifts a node x out of B and investigates all k edges, whether they are to be dropped from the border edge search tree. At most k edges must be inserted into or dropped from the search tree, which is cost $O(k \cdot log\ |E|)$. □

Theorem 20 *TR3 has complexity $O(1)$.*

Proof. TR3 consumes an edge by changing its state from E to U. As the edge is parameter from discovery phase, the complexity is $O(1)$. □

Theorem 21 *TR4 has complexity $O(k)$, where k is the maximum node outdegree.*

Proof. Transition rule 4 gets a node x from candidate selection phase and moves it into the HammockSet. To do so, it must initialize the edge directions of all edges between destination set and x. A maximum of k edges must be checked, whether the opposite directed edge has state T to swap edge direction if required. This can be done in $O(k)$ for all adjacent edges.

Theorem 22 *TR5 has complexity $O(log|E|)$.*

Proof. Transition rule 5 remove one edge (u,v) from the search tree $O(log|E|)$. With the hash map maintained through TR1, we have fast access to the two edges into the destination set. Their direction is initialized and they are moved into H. Then edge (u,v) is turned into a SRL. The total complexity is still $O(log|E|)$. □

Theorem 23 *The time complexity of HammockSet construction is bounded by* $O(|E| + k \cdot |V|\ log|E|)$*, where k is the maximum node outdegree.*

Proof. TR1 and TR2 both apply at most once per node, gaining total cost of $O(k \cdot |V|\ log|E|)$. TR3 then adds contant time for further edges $O(|E|)$. All discovery phases together require running time of $O(|E| + k \cdot |V|\ log|E|)$. TR4 is bounded by $O(k \cdot |V|)$ for all nodes and TR5 drops edges $O(|V|\ log|E|)$. These sum up to $O(|E| + k \cdot |V|\ log|E|)$. □

In this chapter we sketched the mechanics of an HammockSet construction algorithm that runs in $|V| + |E|$ steps, while its total running time complexity is $O(|E| + k \cdot |V|\ log|E|))$. The used representation is a list of strongly-connected HammockSet, for which we have shown that it is fully equivalent to the edge set of a HammockSet. The algorithm does not produce deterministic results, since it can include ambiguous steps, where one of multiple candidates can be chosen.

Chapter 5

HammockSet Operation

Until now, we have examined transmission of single messages only. HammockSets also transport data flows where continuity and packet order are relevant. Then, we formalize what failure events can be recovered by HammockSets and what possibilities exist to restore resilience after a first failure.

5.1 Flow Handling

Routers supporting multi-path usually have two modi related to flows: the flow mode forwards messages of one common flow over the same path, while the packet mode does not care about using the same paths. We will now see what guarantees can be given in flow mode.

Both modi divide outgoing traffic over available outlinks. This can be implemented with hash functions on the flow IDs, i. .e. source IP address and port. Since this is done on IP level, the flows are not announced, size and throughput is unknown at initialization time. While transmission of the first packet, each path-participating router must decide next-hops for the overall flow. Rescheduling might be required, if it comes out the previous distribution had been odd. Changing distribution parameters may discontinue flows and route them onto a new path. However, such a change only appears once in a time and then results into stable paths.

Corollary 11 *While normal HammockSet operation, the packet sequence order per flow is maintained.*

Proof. Let be a HammockSet $H(t)$, a sending node u, a data flow f and its path $P_f =< u, \ldots, t >$. Flow based routing guarantees usage of the same path for any message fragment in the flow. Routing queues do not reorder items with same priority. For one flow each router works as first-in first-out queue (FIFO). Summarizing, a flow that is sent in order, where all fragments take the same network path and are not reordered in a queue, arrive correctly ordered at their destination. □

Corollary 12 *A limited amount of incorrectly sequenced packets may occur at HammockSet failure recovery or modification of distribution keys.*

Proof. Let be a HammockSet $H(t)$, a sending node u, a data flow f and its path $P_f =< u,\ldots,t >$. Two events might influence the path: either a failing link or a router that modifies its flow distribution. In case of a link failure on path P_f the last receiving node detects the failure, and either changes its flow distribution (if there remain multiple outlinks) or recovers by activating a reserve link. This results into a new path P'_f. Since for a short period of time, both paths P_f, P'_f carry messages of one flow, the sequence at destination can differ from original packet order. □

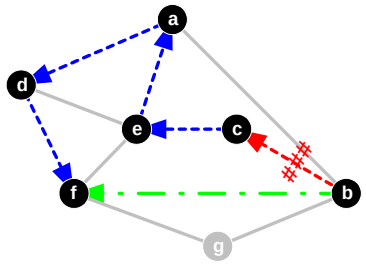

Figure 5.1: Path Switching of (b, f)-Flow

Note, that a high amount of incorrectly ordered packets occur, if the new path P'_f is very short, so that many packets arrive earlier than the packets over P_f, which passed the failing link just before failure. See Fig. 5.1 for an example: the flow f had been routed over (b,c,e,a,d,f) before the link (b,c) failed. Then router b switched the traffic of flow f onto the path $< b,f >$. Then further data takes this one-hop path and arrives, before router queues c,e,a,d forwarded all flow fragments. Since router b exclusively uses one path at a time for the flow, the total amount of incorrectly ordered packets is restricted to the sum of queue buffer sizes along the path P_f.

Also note, that some packets go lost, from the time that the link (b,c) is physically unavailable until detection and recovery. With a 628 Mbits link, a link failure that has not been detected for 20 ms loses up to 1.5 Mb messages. The failing data will be requested from the transport layer as first transmission failed. The switching speed of HammockSet suffices to catch the next transmission.

5.2 Link Failures

HammockSets are specialized to recover single link failures very fast. Now we investigate what kind of failure sets are covered through HammockSets.

Theorem 24 *A HammockSet recovers any single failure.*

Proof. A HammockSet is a specialized extended routing graph, that requires two first-hop redundant paths from each node to destination. If an active outgoing link fails for a node, the first-hop

redundancy grants a second path. In case of an reserve link, that must be activated first. However, the connection can be recovered within tens of milliseconds after link disruption. □

The two end nodes of a failing link have both first-hop redundant paths to all destinations. If the link fails, the second path takes over.

Theorem 25 *In a HammockSet each inner node has at least two edge-disjoint paths to destination.*

Proof. Let $H(t)$ be a HammockSet and u some inner node. Then two edge-disjoint paths $P =< u, x_1, \ldots x_n, t >, P' =< u, y_1, \ldots y_m, t >$ exists. Since u has first-hop redundant paths, $(u, x_1) \neq (u, y_1)$ are distinct. It is possible, that P and P' have more common nodes than u and t. Let $c_1, \ldots c_k = \{x_1, \ldots x_n\} \cap \{y_1, \ldots y_m\}$. Since all nodes c_i have first-hop redundant paths, there exist disjoint edges from c_i in P and P'. □

Definition 40 *The* **distance between links** *is the edge count of the shortest network path.*

Definition 41 *A set of link failures is called* **isolated**, *if its minimum distance between edges is two.*

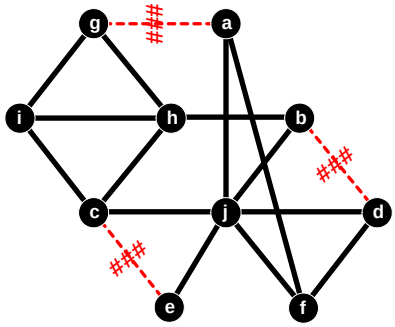

Figure 5.2: Topology with three Failures with Minimum Distance two

Theorem 26 *For an isolated link failure set, any HammockSet can recover.*

Proof. Let $H(t)$ be a HammockSet on topology N. We show that $H(t)$ recovers routing for any failure set with a minimum distance of two. Then it is impossible, that both sides of a Shared-Reserve-Link are activated together, since that can only happen with link failures of distance one. □

Fig. 5.3 (A) shows an isolated failure set. Both links (b,a) and (f,e) fail, but the operation is still guaranteed. A second failure with distance one, i. e. (d,c) (Fig. 5.3 (B)) would activate both reserve links of the SRL, causing a cycle between (b,d) and disconnects these nodes from destinations. However, failing of other links below distance two, i. e. (f,d) (Fig. 5.3 (C)) would not wound that HammockSet, but possibly a HammockSet for another destination node.

Note, that the isolation concept does not include any other assumptions about what HammockSets are used. It verifies the recovery for all possible HammockSets.

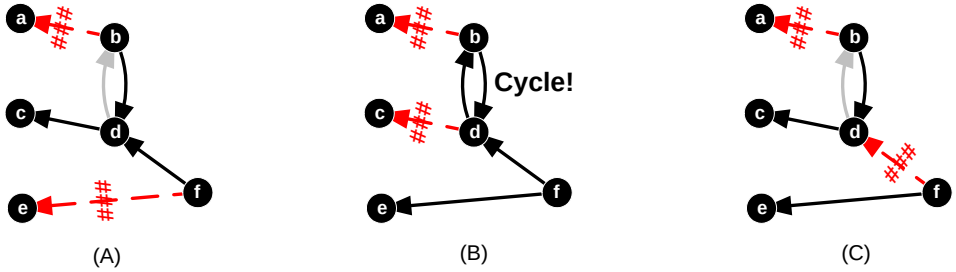

Figure 5.3: An Isolated Failure Set

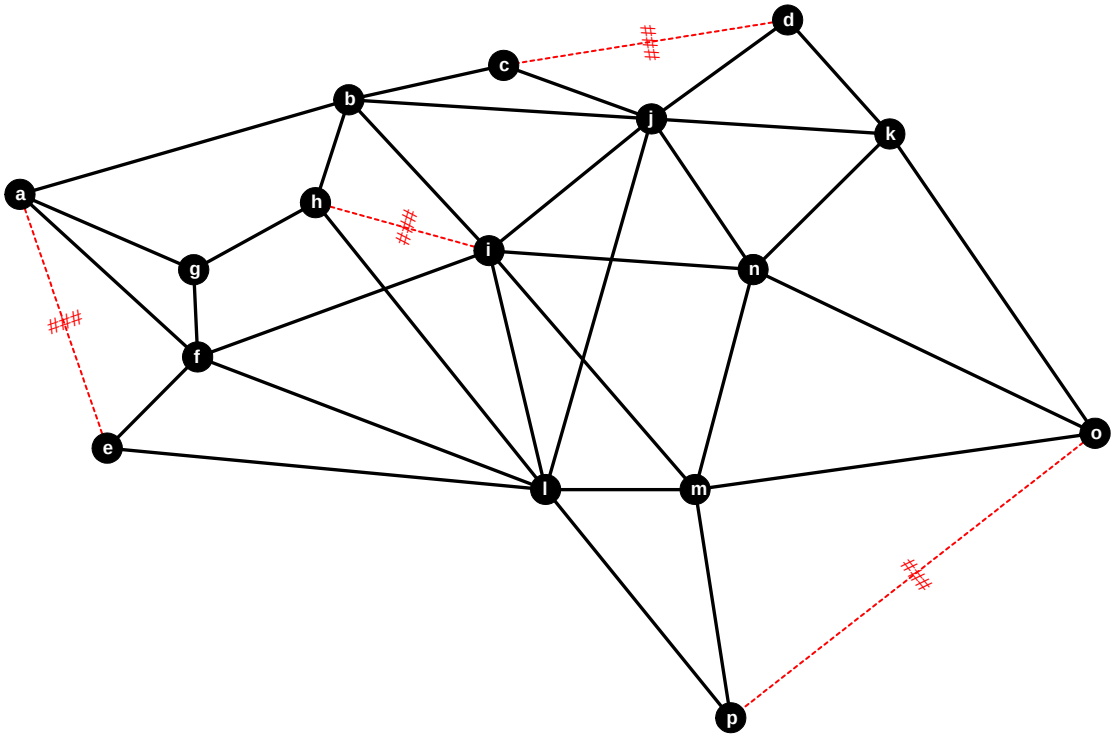

Figure 5.4: Topology with an Isolated Failure Set of four

The topology in Fig. 5.2 is exemplified to have an isolated failure set of three. However other sets exist, e. g. (a, j) that are maximal isolated, so that no other link has distance two. Even larger topologies do not potent increase the number of maximal isolated links. Fig. 5.4 has a maximum isolated link set of four, and also another maximum isolated set of one (j, l). We conclude, that recovery of two links in the same HammockSet is possible for all isolated failure sets, but not for other failure sets.

The isolation concept ensures link recovery without investigation of used HammockSets. Unfortunately, isolation is not strong enough to prove recovery of multiple links. Anyway, we have seen examples of link failure sets, that could be recovered. And beyond that, there might be several links that are not isolated, but can also be recovered in a given set of HammockSets.

5.3 Node Failures

With a power outage at a network node, namely a failing node, several links are detected to be unavailable. In the earlier chapters we have seen, that HammockSets are stronger than two-connectivity. However, HammockSets can usually not recover such failures.

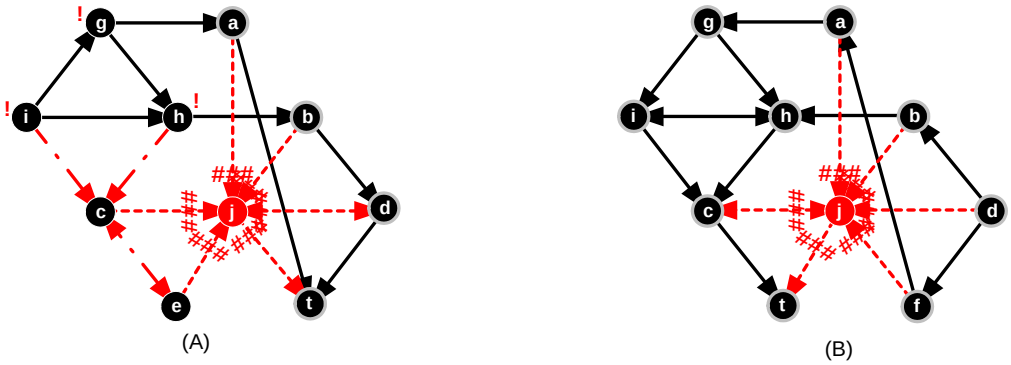

Figure 5.5: Two HammockSets after Node j Failure

Fig. 5.5 (A) shows HammockSet $H(t)$ with failing node j. Here three nodes (a,b,d) can continue operation, while c,e are disconnected and g,h,i have several paths, where some reach the destination and others do not. Since the failure is distant, these nodes cannot select the correct outgoing links. HammockSet routing does not propagate link failure messages. The next figure 5.5 (B) is an example, where by accident routing converges for all remaining nodes. This is because a,b,d,f recognize one of their direct neighbor j as destroyed and utilize the remaining routing paths, which gains here a full-coverage routing. This cannot be guaranteed and is rather unusual.

In the KING project [Sch01], topologies and extended routing graphs had been examined, where node failures can be recovered. However with this HammockSets there exist failures, which cannot be recovered. Therefore we summarize, that we cannot guarantee recovery of node failures.

5.4 Seamless HammockSet Restoration

After a failure detection, the routers at both ends recover all HammockSets, which were using this edge. The remaining HammockSets are not guaranteed to be resilient any more. To restore resilience, new HammockSets must be generated for the remaining topology. In our prototype, a central control system then notifies all routers of the new routing information, that re-establishes the HammockSet one-link-recovery guarantee.[1]

.

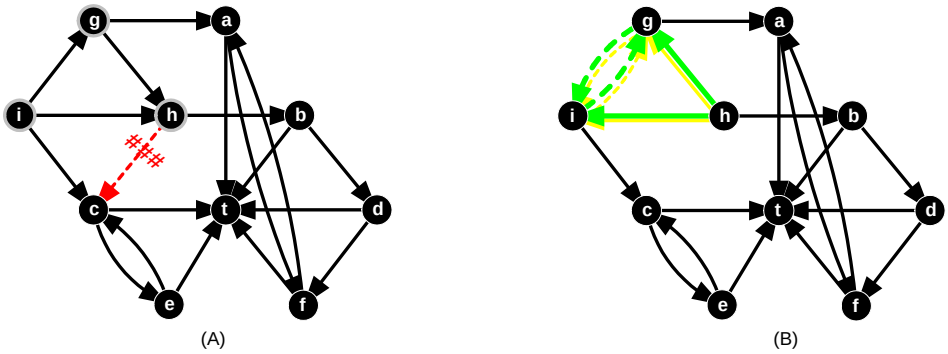

Figure 5.6: Newly Established HammockSet after Link Failure (h,c)

In Fig. 5.6 (A) link (h,c) fails. As soon as router h detects the failure, it removes the link from its routing tables and recovers the Extended Routing Graph $H(t)$ with its other active outgoing link (h,b). However, this graph can not guarantee to recover next link failures. To restore resilience, we process a new HammockSet for the shrunken topology (Fig. 5.6 (B)). Then we distribute the new routing information with the following method that enforces to have active spanning trees available over the whole restoration process. Besides, it guarantees providing cycle-free paths to destination during transform.

Fig. 5.7 (A) displays the HammockSet component list before recovery. To regain resilience to single link failures, we activate step-by-step a new HammockSet. Fig. 5.7 (B) shows the resulting state after activating the new routing table at routers g and i, but before changing router h. Note, that all processed routers have redundant connections, but node h has only a single connection. The transformation is finished in Fig. 5.7 (C), where the routing tables have been downloaded and activated at router h. Now the HammockSet property is restored.

We show, that with this partially transformation the routing graph is valid at any time. Let $N_{renewed}$ be the set of nodes, that have already activated updated routing tables. Then we have

[1] Another approach could be to provide routing information to all routers once and then use timed events for step-by-step activation. This can reduce communication overhead. However, if one node does not correctly receive its instructions, the routing may fail.

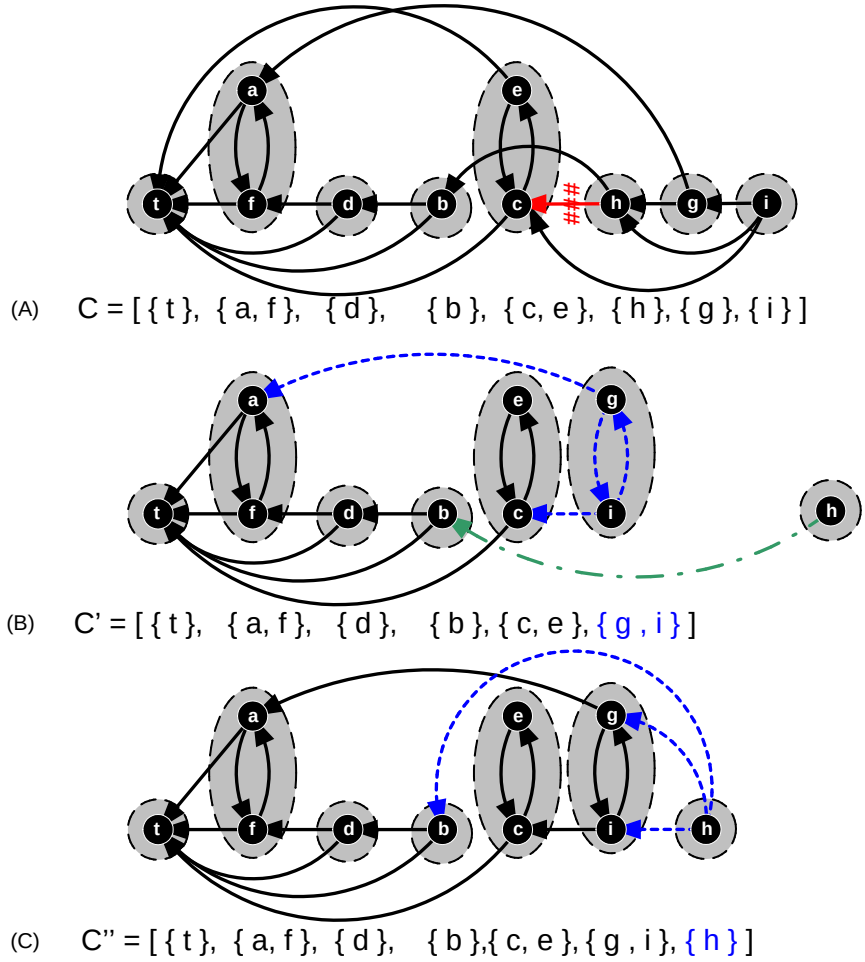

Figure 5.7: Restoration changes HammockSet Component List

Algorithm 1
```
forall HammockSet components C_i do
    download new routing table fragments to routers in C_i
    wait for activation
od
```

Table 5.1: Activation Procedure for new HammockSets avoiding Flow Disruption

three kinds of edges: edges that have been renewed, which forward messages to other already renewed nodes. Secondly, edges that still use the previously established routing. Third, one reserve link, activated to recover routing. This only appears, if the disconnected node had no second active outgoing link.

Theorem 27 *During any stage i of rerouting from HammockSet $H(t)$ to $H'(t)$ after failure of link (z,z'), the following edges $D(i)$ are actively used by routers.*

$$\begin{aligned} N_{renewed}(i) &:= \bigcup_{j\le i} C_j \\ D(i) &:= \{(x,y) \in H'(t) \mid (y,x) \notin H'(t), x \in N_{renewed}(i)\} \\ &\cup \{(x,y) \in H(t) \mid (y,x) \notin H(t), x \notin N_{renewed}(i), (x,y) \neq (z,z')\} \\ &\cup \{(z,i) \in H(t) \mid (i,z) \in H(t)\} \end{aligned}$$

Proof. The set of nodes $N_{renewed}$ contains exactly that nodes where the routing information has been updated for HammockSet $H'(t)$, while $V \setminus N_{renewed}$ has the nodes with the previous HammockSet $H(t)$ routing information. Then the first set of the union consists of all outgoing edges for renewed nodes, while the other set contains unchanged edges for the remaining nodes but no Shared-Reserve-Links. The last line includes a reserve link from z, that has been activated to recover the failing link (z,z'). If node z has several outgoing links no Shared-Reserve-Link is adjacent but the second set contains an outgoing link for z. In both cases, with or without SRL at least one edge is included granting active paths towards destination from all nodes. □

Theorem 28 *Invariant: At any rerouting stage, $D(i)$ is an edge set for a cycle-free routing graph containing all nodes.*

Proof. Let be a HammockSet $H(t)$ with destroyed link (x,y), a new HammockSet $H'(t)$ for the topology $N \setminus \{(x,y)\}$ and a routing graph $T(i) = (V,E,D(i),t)$. $T(i)$ morphs from $D(0)$, the remaining edge set with some recovery for link (z,z') into the active edge set of the new HammockSet $H'(t)$. We are going to show, that $D(i)$ has at any transformation step a spanning tree and does not contain cycles.

The nodes in $N_{renewed}(i)$ have a spanning tree to destination, since all their outgoing edges already provide HammockSet properties. Further there exist two active edge types from nodes outside $N_{renewed}(i)$: either such an edge leads to node in $N_{renewed}(i)$, for which the spanning tree has been shown above. Otherwise the edge points to a node not in $N_{renewed}(i)$, then this has a path going to destination within $H(t)$ or leading to some node in $N_{renewed}(i)$. In any case, active paths are available from any node and therefore a spanning tree exists.

The sub topology of nodes $N_{renewed}(i)$ is cycle free, since it contains only non-reserve-links from a HammockSet. Also the edges within $V \setminus N_{renewed}(i)$ have been active edges of a HammockSet. Therefore these are acyclic, too. Renewed edges target already renewed nodes and there is no edge to a node $\notin N_{renewed}(i)$. Therefore $D(i)$ must be cycle free. □

With the above algorithm the availability of routing can be guaranteed, unless a second failures happens to the same node destroying its reserve. This procedure reestablishes the HammockSet properties and afterwards resilience is restored.

Chapter 6

Evaluation

In this chapter we evaluate HammockSets. We have three applications in mind: first, we want to choose the optimal HammockSet if the construction process has ambiguous transitions (steps with several candidates for the TR4 or TR5). Second, we compare the results of our construction algorithm with other HammockSet construction methods. Third, we review how topologies can be evaluated to gain better HammockSets and increase network performance.

In Chapter 3, we introduced enhancement of routing graphs with additional edges resulting in additional paths and an increased bandwidth. Our construction algorithm implicitly utilizes the maximum amount of possible network edges (Theorem 14). Maximized HammockSets are always superior to HammockSets with fewer edges, since these can be emulated using zero for distribution parameters. Without loss of generality, we assume all HammockSets to be maximized.

6.1 Link Significance

The HammockSet is represented as a subset of edges for a network topology. It enforces the properties listed in Chapter 3: first-hop redundancy, an active spanning tree, and cycle avoidance. The overall HammockSet objective is to provide adequate bandwidth from each node to the HammockSet destination. Orthogonal to this several HammockSets share link capacities. In HammockSet routing we can approximate the number of HammockSets using one direction of a link. This is because maximized HammockSets with x SRLs use $|E| - x$ directed active links plus x Shared-Reserve-links. As reserve edges are not actively used, $|E|$ directed edges from $2 \cdot |E|$ available are used. Therefore roughly $\frac{1}{2} \cdot |V|$ HammockSets should use one edge into one direction in average.

Next we must understand what traffic amount is propagated through a HammockSet. Therefore we assume roughly the same traffic characteristics from each node to destination. Then we can just sum up the number of nodes that possibly use paths running along that link (see Fig. 6.1 (A)).

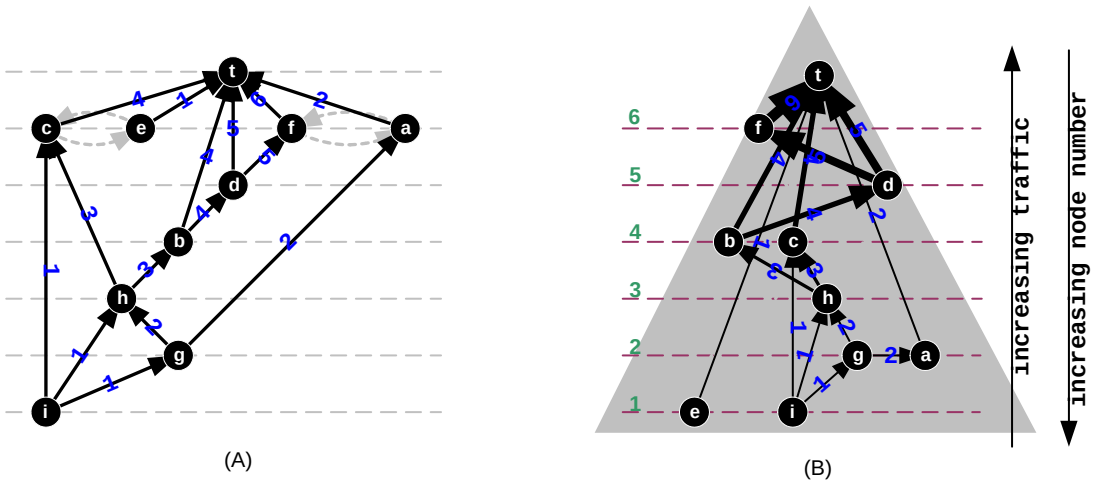

Figure 6.1: Link Usage and Significance

Definition 42 *The* **significance of a link** *in a HammockSet is defined through the number of nodes, whose paths utilize that link.*

The significance of a link indicates several aspects:

Prediction of Traffic Amount
If any node introduces the same amount of traffic then a link carrying many node paths will receive higher amounts of traffic.

Impact to the HammockSet in Case of Failure
A link carrying traffic from one node is less important than a link participating in many paths from different nodes. A link failure with high significance causes many flows to be rerouted and therefore heavily influences the link load on the remaining network.

Measure of Strong Relation to Destination
A high number reflects the link's closeness to the top of the HammockSet pyramid. While each level in the pyramid (Fig. 6.1 (B)) introduces additional traffic and the node number per level decreases upward, a link with strong relation to the destination node is important to the HammockSet. This relationship is indicated by a high number of paths through this link.

We believe that this reflects a better measure than counting of outgoing edges and other simple measurements which do not reflect the forwarding function of inner HammockSet nodes. Fig. 6.1 (B) provides a HammockSet ordered by link significance. However, link significance can be a bad criterion in networks with various link capacities or odd traffic characteristics.

From two HammockSets with equal destination and equal topology, the one with the higher significance values provides better possibilities for traffic distribution. However, a network with high load requires low significance values with a maximum number of links to avoid exceeding

link capacity.

6.2 Traffic Distribution

Computing the distribution parameters is beyond the scope of this thesis. However, a low average of effective link load is the main objective for optimal HammockSet construction. Therefore we cannot ignore these parameters completely. HammockSets make use of multiple paths and these influence the effective link load.

Input:	HammockSets $H(t_i)$ Traffic Matrix $M[s,t]$
Processing:	Multi-Commodity Flow with $n = \|V\| \times \|V\|$ Information Flows
Output:	Distribution Parameters q

Table 6.1: Problem Class: Traffic Distribution Parameters

The problem class in Tab. 6.2 is related to multi-commodity flows but has additional constraints: flow splitting is realized by destination-address only, while multi commodity flows use a bunch of paths for each pair of source and destination. Approximation algorithms increase linearly with the amount of different commodities. To parameterize HammockSets, we have $|V| * |V| - 1$ commodities. The approach of Leighton, et. al.[LMP+91] provides an approximation with time complexity $O(k^{2.5}n^2m^{.5}log(n\varepsilon^{-1}DU))$ where D the largest demand, U the largest edge capacity, and for our application $k = |V|^2$.

In this section we show, how the effective link load can be calculated for a given set of distribution parameters. By processing the formulas, the link load can be tested in the model before being implemented to a real-world system. Besides, the robustness to traffic fluctuation can be tested for rivalling HammockSets.

$$N^t(o_{\text{calc}}) = \left(M[x,t] + \sum N^t(i_n)\right) * q_{\text{calc}} \quad (6.1)$$

$$N(u) = \sum_t N^t(u) \quad (6.2)$$

Equation (6.1) calculates the link load produced from one HammockSet during normal operation. It consists of the load deriving at node x itself and all incoming data streams. The total amount is divided by q, the distribution parameter for the processed outgoing link. Equation (6.2) summarizes the load of all HammockSets for one link. For valid distribution parameters, the link traffic during normal operation $N(u)$ must be within the link capacity.

$$\Delta q_{\text{calc}} = \begin{cases} \left(\frac{1}{1-q_{\text{jam}}} - 1\right) * q_{\text{calc}}, & (q_{\text{jam}} < 1) \\ 1, & (q_{\text{jam}} = 1) \end{cases} \tag{6.3}$$

$$R^t(o_{\text{calc}})_{\text{direct}} = \left(M[x,t] + \sum N^t(i_n)\right) * \Delta q_{\text{calc}} \tag{6.4}$$

$$\begin{aligned} R^t(o_{\text{calc}}) &= \max\left\{R^t(o_{\text{calc}})_{\text{direct}}, R^t(o_{\text{calc}})\right\} \\ &= \max\left\{R^t(o_{\text{calc}})_{\text{direct}}\,, R^t(i_n)_{\text{direct}} * q_{\text{calc}}\right\} \end{aligned} \tag{6.5}$$

$$R(u) = \max_t\left\{R^t(u)\right\} \tag{6.6}$$

Formulas (6.3)-(6.6) process the peak load for any possible single link failure. First, we process which link has the maximum distribution quota and therefore causes maximum traffic to other links in case of failure. This yields the additional amount for that node and allows us to deal with distant augmented traffic.

As the HammockSet components can be totally ordered and inside a component no active edges exist, we can sequentially process these equations resolving all dependencies on the fly. We gain two result arrays: $N[u]$ the link load in normal operation and $R[u]$ the additional peak load after a link failure.

To guarantee message delivery under normal operation and any single failure, all edge capacities must hold $N[u] + R[u] \leq (1-\varepsilon) \cdot C[u]$. Choosing $\varepsilon \approx 0.3$ would restrict link load to 70%, the suggested rate to avoid buffer overruns. The peak link load grouped by link capacity can be output as part of the result. With this, we can test sets of traffic distribution parameters for various HammockSets and find bottlenecks.

6.3 HammockSet Cohesion

As we pointed out in the last section, evaluation of a single HammockSet does not give us the big picture of network load. Several HammockSets all using one link in one direction, must share link capacities in the real-world system among each other. Many HammockSets using one link shrinks the available bandwidth for these HammockSets, while other links are sparely used and carry few load.

We suggest the following solution (tab. 6.3) originating from the min-cut/max-flow algorithms. We use a deterministic candidate selection as described in Appendix B.3. Then we reduce the remainder link capacity by the amount utilized from this HammockSet. The next HammockSet will then be computed using the remaining capacities and therefore choosing a path with the maximum remaining bandwidth. We should start with the HammockSet that transports the biggest amount of traffic. This can be done by sorting destination nodes descending by the size of total traffic.

Input:	traffic matrix $M[s,t]$
Processing:	1. sort nodes descending by total traffic per destination 2. create a HammockSet for a destination node adjust remaining capacity not yet used repeat from 2
Output:	HammockSets $H(i)$

Table 6.2: Construct HammockSets with Cohesion

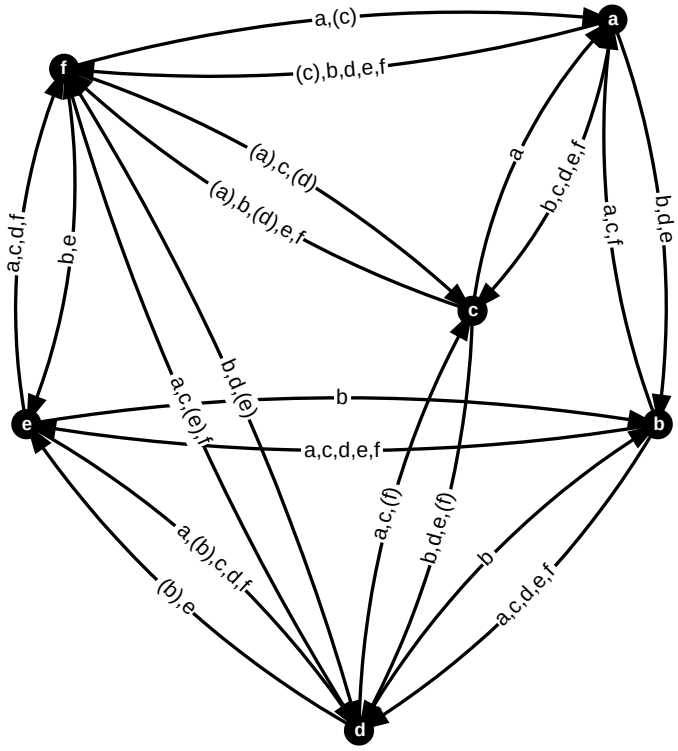

Figure 6.2: Topology with annotated HammockSet edges

Fig. 6.2 shows a small topology of six nodes with the HammockSet link utilization. Let us review the HammockSet on some prominent links. Link (a,b) is even-weighted, meaning that both of its directed links carry the same amount of HammockSets: three on (a,b) and three on (b,a). At other links one direction is stressed: (b,e) carries five HammockSets, while (e,b) covers a single HammockSet. For the recovery case, overloading must be avoided. This can be prevented if any link carries at most one reserve. At least this should be restricted through the number of SRLs divided by edge count. Fig. 6.2 contains such a link with two reserves (c,f), but these will not be activated at the same time through a single failure.

6.4 Rules of Thumb

We now give a compressed overview of the evaluation. A network is operated with the optimum set of HammockSets, if

1. the HammockSets provide many paths between any source and destination
2. it allows for an even distribution of traffic
3. it avoids link overloading during normal operation and recovery mode
4. the deviation of HammockSet count using one edge is low
5. the number of participating edges is maximized

Summarizing, we can compare maximized HammockSets by the significance of their links. However, we must consider all HammockSets together to judge the network performance.

Chapter 7

Related Work

In the network community, two trends towards resilience can be recognized. One approach improves routing protocols to avoid lengthy propagation delays and enables fast convergence of network operation. Today these methods succeed with a converging time below one second even for complex networks. The second popular approach, is to utilize multiple paths and provide zero-time recovery for a restricted amount of failing links and nodes. Our current work fits into the latter group.

7.1 Routing Protocol Convergence Analysis

A variety of measurement and evaluation schemes can be found in the literature. The boundary of the HELLO-messaging frequency had been the router processing speed under worst-case conditions. The default detection frequency had been set to three seconds. Regarding to [BR01], today's hardware is capable of processing a higher frequency, thus gaining a better resolution in link failure detection and better convergence at the same time. But it requires more reliable time until receiving, to speed up HELLO-messages. Besides, [ICM+02] report an extremely high rate of false detections. This demands a better understanding of failure detection.

Analysis of Link Failures in an IP Backbone [ICM+02]
: Iannaccone, et. al. examine the American SPRINT wide area network for IS-IS link change messages. They catch all failure messages and extract statistics about downtime of links. They announce to provide a model of link failures in IP networks. They discover three of eighty-five links that cause 25% of all link failures. For other links the mean time between failure (MTBF) ranges between five and fifteen days. Many of the logged failure events can be resolved within one minute, indicating a false detection, possibly caused by high CPU load so that uplink messages are not received. One important objective of their further research is modelling follow-up failures emerging after a first link failure.

Stability issues in OSPF routing [BR01]
: Basu and Riecke examine convergence time in OSPF networks. They test various HELLO intervals from the default 10 seconds to 500 ms and even 250 ms. Often messaging threatens to introduce high load on router CPUs. In their experiments they discover an increase of processor utilization from 10% to 15% if interval is changed from 10 seconds to 250 ms. This is not a significant change and so they suggest to shorten test intervals.

Delayed Internet Routing Convergence [LABJ00]
: Labovitz, et. al. report that the convergence with the border gateway protocol can consume tenths of minutes until stabilizing. They find out this originates in differing implementation decisions between router vendors.

Towards Milli-Second IGP Convergence [AJY00]
: Current IP re-routing times are typically tens of seconds. This due to conservative timeout settings. These are chosen to guarantee correct detection of true link failures and not sense overloaded CPUs of neighbored routers. With suggested 30 ms HELLO messages, router CPUs gain 100% workload and then randomly dropp connections. Alaettinoglu, et. al. summarize that lower convergence time is possible with fast a algorithm and millisecond timer resolution.

End-to-end Routing Behavior in the Internet [Pax96]
: In 1997, Paxson found two recovery classes of internet route recovery: the fast one (hundreds of milliseconds to seconds), where the routing converges as soon as incoming routing information is processed, and a slower one that takes in the order of a minute and which requires propagating routing information through the network. Today the converging speed of carrier networks is below one second with propagation.

Still, we have an convergence time around one second. Summarizing, propagation delays will always exist, while we expect they will decrease further as new technologies are developed. However, the amount of lost data during convergence will probably remain constant as the throughput constantly grows.

7.2 Improving Routing Convergence

There are new routing protocol algorithms that converge faster, avoiding instable states. A new approach of dynamic routing tables adapts also to the current network load partitioning between exchanged routing information and an online-processed first-hop routing tables.

MDVA: A Distance-Vector Multipath Routing Protocol [VGLA01]
: Vutukury and Garcia-Luna-Aceves propose a new algorithm removing the count-to-infinity problem, which arose from Distributed Bellman-Ford algorithms. Is also provides multiple cycle-free paths. Their work includes the proof of instantaneous loop-freedom and correct termination of the protocol.

FIRE: Flexible Intra-AS Routing Environment [PSS+00]
: Partridge, et. al. describe a new approach for traffic distribution. They propose a run-time-configurable algorithm that can flexibly change the network routing behavior. They also introduce class-specific routing, which uses distinct routing tables for each traffic type.

A Path-Finding Algorithm for Loop-Free Routing [GLAM97]
: Garcia-Luna-Aceves and Murthy present an algorithm for hop-by-hop routing that converges faster. This is an replacement for shortest-path routing but does not utilize multiple paths. It provides recovery for single links with a worst-case performance of $O(x)$, x being the number of affected routers.

7.3 Redundant Routing

Shortest paths are used to guarantee acyclic routing. However, using shortest-paths only yields high load on few network links, while other carry very few traffic. [WC90] examine alternative paths to avoid oscillation between two paths. With the demand for controlled Quality-of-Service, the urge for traffic distribution emerges. Several approaches for the construction of multiple paths are listed next.

Improving the Resilience in IP Networks [SCK+03]
: The O2-algorithm introduced in this paper constructs multiple paths towards a destination also for the last hop using joker links, which remain unused until failure detection. However, the algorithm does not construct maximized HammockSets giving room for improvement.

Routing Reconfiguration in IP Networks [Nar00]
: Narváez examines routing reconfiguration on different time scales. The author works on the re-computation through dynamic shortest-path algorithms, link-state propagation and also multiple paths provided in routing tables. The motivation is also to reduce the delay until operation resumes after link failures.

  In this approach a set of cycle-free paths is computed and materialized in the routing table entries. This is ensured through partially ordered nodes based on edge weights and hop count. The next-hop nodes on all paths have a decreasing value, which can be used to guarantee path termination. His method applies to any topology, but it does not provide multiple outgoing links for all nodes. However, this algorithm is able to interoperate with ”naive” shortest-path-based routing devices. Narváez also implements a distributed solution that extends OSPF.

  This method is optimized for multiple paths between distant nodes. It cannot gain resilience between neighbor nodes.

A K-best Paths Algorithm for Highly Reliable Communication Networks [LW99]
: Lee and Wu search for the k paths, which skew maximum diversity and minimum total cost.

They apply this algorithm for planning high reliable communication networks. However their approach does not necessarily increase the resilience shortly after link failures.

Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs [MFB99]
This algorithm is dedicated to path-based routing. Medard, Finn and Barry construct two edge disjoint trees that provide alternative paths in case some edges of the active tree fail. An extension allows recovery of a node failure by merging two trees into one with all available routes.

Analysis of Multi-Path Routing [CRS99]
Cidon, et. al. compare the resource reservation in single- and multi-path algorithms. They conclude that a small number of parallel paths such as three or four performed best in the tested networks.

Loop-Free Multipath Routing Using Generalized Diffusing Computations [ZGLA98]
Zaumen and Garcia-Luna-Aceves introduce the Diffusing Algorithm for Shortest Multi-path. This allows an distributed implementation that constructs routing tables with multiple next-hop entries.

Finding the k Shortest Paths [Epp94]
The algorithm proposed by Eppstein can find k shortest paths from all nodes in $O(m + n \log n + k n)$ with n vertices and m edges.

Finding Disjoint Paths in Networks [SNA91]
Sidhu, Najr and Abdallah provide a distance-vector algorithm to find multiple network paths that have been optimized towards few communication. Their approach guarantees using the shortest path as one of the resulting paths making their approach superior to others.

Shortest-Path-First with Emergency Exits [WC90]
Wang and Crowcroft provide alternative paths if the shortest path is congested. This is an advantage for routing protocols that would oscillate otherwise between the shortest and an alternative path. The algorithm descends from Shortest-Path-First and gains better congestion control and fault tolerance.

The approach in [Nar00] can be applied to any topology, but it does not guarantee multiple outgoing links for any node. Then a link failure beyond this node cannot be resolved immediate.

Chapter 8

Conclusion

The main objective of network resilience is to provide alternative routing in case of link failure. The routing graph for *hop-by-hop* operation must be acyclic to avoid data loss. This work extends the O2-approach from the KING project [Sch01], which uses joker links. Our approach includes the maximum number of paths to enable traffic distribution.

Local recovery mechanisms are superior to convergence through routing protocols. For the latter recovery speed suffers from propagation delays that scale with the topology size. Note that local recovery requires a notably dense network to comply with the demands for a HammockSet topology. This is the price for immediate stand-by-recovery and loss-less networking. Remember that there are many applications urging for instant networking with guaranteed message delivery and Quality-of-Service. The next generation internet is going to satisfy these demands.

8.1 Contributions

We found out that a single joker link per HammockSet is not sufficient for many topologies. We kept the fundamental idea of non-actively-used links. So we analyzed the more general extended routing graph model and found topologies that demand for single reserve link usage not mentioned before and also topologies that require several Shared-Reserve-Links.

We then formalize the requirements of acyclic graphs with spanning trees combined with reserve links and transformed these into a cycle property. HammockSet Cycles must be single-fault-blocking to avoid activation by a single failure. A sub-class with cycle length two has been found to have advantageous features, that allow easy representation in strongly-connected components. Our new construction approach chooses candidates, either of single nodes or of connected node pairs. We also used candidates to formally define the topology class where all destinations have HammockSets covering all nodes.

We show the termination of our construction algorithm, its running time is $O(|E| + k \cdot |V| \; log|E|)$

and prove that it produces HammockSets with the maximum number of edges. These can recover isolated failure sets, but we also provide a method to seamlessly restore a HammockSet after the first failure event. We also suggest a heuristic to construct optimum HammockSets. Furthermore we suggest rules of thumb that can be used to judge HammockSet quality.

8.2 Future Work

It is unknown whether our algorithm can always find the optimum HammockSet in terms of transmission. Coming up with a model of network failures, especially distant effects of overloaded links will give new hints for resilient network planning .

Extended routing graphs can also withstand node failures and seem to be more resistant, while having weaker topology demands at the same time. In this case an automatic construction of extended routing graphs with cycles longer than two is required.

Traffic distribution is another field of future research. We find it difficult to compute traffic-based distribution-parameters, as we cannot provide a low-polynomial solution. When traffic forecast is available, we suggest combination with the HammockSet construction which is able to utilize the remainder capacity.

Appendix A

Redundancy

Liebenau described examples where communication networks failed [Lie03]: in 1992 Hurricane Andrew hit Florida and destroyed the American PSTN, with the fire in Baltimore Tunnel 1999 which cut the main east coast cable and lately, the World Trade Center collapse, which disrupted business networks for days. *Forster*, *Proctor*, and *Smith* tell about an all-redundant network center that owned two air conditioning systems. Unfortunately, both were connected to one power source. As that failed the room overheated, all contained equipment failed, and shut down all operations [FPS02].

To ensure stable network communications after a failure, redundant systems are required. Proving redundancy is complex and must be analyzed through network operation layers (Fig. A.1).

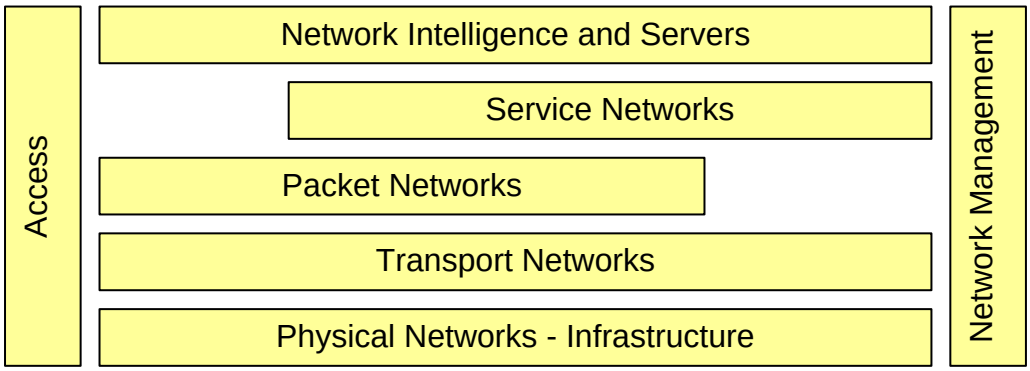

Figure A.1: Resilience Layers ([FSP02])

Forster, *Proctor*, and *Smith* describe many aspects of redundancy to consider and possible problems that could evolve [FPS02].

A.1 Physical Networks

Regarding to physical networks, two disjoint paths must cover each other. However, if there are two routes it must be ensured that these are really independent. Likely redundancy-breakers are: same interface card, same box, same wavelength, same fibre, same cable, same duct, same street, same building, same power, same air condition.

Token-Ring networks introduced redundant network structures. They consist out of two rings directed opposite to each other. Communcations use the primary ring, until one of that links fail. Then the reverse ring is used to bridge the destroyed connection. Attention must be given if remote stations are added (Fig. A.2). If both ring connections run through a common duct redundancy is threatened.

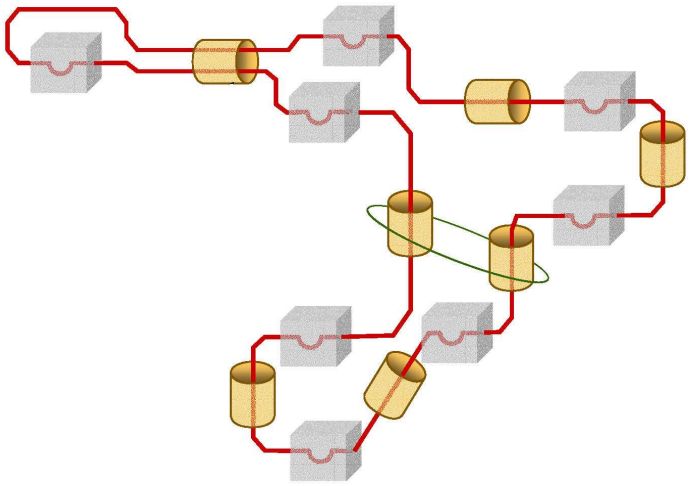

Figure A.2: Token Ring with Remote Stations ([FSP02])

In business operations, not all this facts are open information. Prooving redundancy requires proper research. There are examples in which two supposed-to-be physically disjoint lines had been rented from two different cable companies A and B. In a real case, they found out that B just rented some capacity from A, which offended the network with a single point of failure [FPS02]. This problem class is summarized under ownership.

It may happen that two companies lease you different wavelengths of a common fiber channel or two fibers going through one building in one common duct. If the receiver breaks, the building collapses, or the duct is destroyed, both fail together. Even if two discrete ducts run along the same street, they cannot provide redundancy.

A.2 Transport Networks

Some transport networks also provide redundancy features. Usually this is combined with Quality-of-Service, which requires capacity reservations. In ATM networks an alterative paths can only be setup, if the requests can be fulfilled with all existing connections.

The extra capacity must also be payed. *Forster*, *Proctor*, and *Smith* report that business cost-cuttings often cut down redundant networks to ”cheaper” single links [FPS02]. However, being totally disconnected is much more expensive than providing a reserve link.

A.3 Packet Networks

Packet-based networks can freely choose a route to destination. However the packet delivery is only successful, if all links on the path are working and the packet is not dropped from a waiting queue. These networks are optimized for ”best effort”, meaning that as much as possible traffic should be transferred but buffer overflows may occur and destroy data.

Best effort conflicts with resilience. Any link usage exceeding 70% results in probability that some packets will not be delivered. This is especially critical if the network must recover a total link failure and distributes that traffic on neighbored links. Then distant link overloading originated by a link failure occurs and influences other possibly-independent connections.

A.4 Service Networks

All above problem descriptions had been somewhat obviously. A more hidden dependency is the reliance on service networks. This means, that various networks could fail operation, if a common base service is destroyed.

The Domain Name Service (DNS) is one example. Some dynamic web services might use DNS addresses to refer to foreign data sources. With failing of DNS all such services will fail to acquire data and block operation.

Reliance to network service must be reviewed very critically, since it can cause several subsystems to fail at once.

A.5 Network Intelligence and Servers

The network recovery mechanisms that work distributed, depend on packet delivery. The simple network management protocol (SNMP) will not correctly detect failures if its own communica-

tion fails.

Another fact is that occurred failures handicap the normal operations. A routing device planning new route in case of a failure has fewer processing capacity to execute packet forwarding. And a router that performed well with three outgoing links and buffers may be overloaded with only two remaining links.

Redundancy also refers to the independence of router firmware. Two same-build routing devices could fail because of a common security hole or firmware bug. This can be avoided by using two distinct systems from independent vendors.

Appendix B

Implementation Details

In this second appendix, we give insight into the prototype implementation. We first introduce all tools that have been used and then sketch important function points.

B.1 Environment

Implementing the HammockSet construction algorithm had utilized STL, GTL, and Graphlet. As work bench the project used the Microsoft Development Environment 2003 (Version 7.1.3088) with Microsoft Visual C++ .NET 2003 in a campus license. Besides, the command line tools of WinCVS provided repository functions.

Standard Template Library
: The Standard Template Library (STL) is based on C++'s features on generic algorithms and classes. It consists of a variety of data structures which can be easily adapted to the user demands. STL can be downloaded from Hewlett Packard [Hew95], but many modern compilers include the STL libraries.

Graph Template Library
: The Graph Template Library (GTL) has been developed at the University of Passau, Germany. It provides a graph model, which supports efficient data structures to work with nodes and edges. GTL can be requested from the computer science department, chair of theoretical computer science, Passau [Him95]

Graphlet
: The Graphlet project has been developed from Michael Himsolt at the University of Passau, Germany [LfTI02]. It is based on the GTL library and TCL/TK. It includes GraphScript, a macro language that can operate on graph structures. Furthermore it provides multiple methods of laying out an abstract graph, so that very few edge crossings evolve. Graphlet imports GML format files.

B.2 HammockSet Construction

In this Section we present a solution that constructs a list of strongly-connected HammockSet components. This is a straight-forward implementation of the transition rules in Chapter 4.

Input to this algorithm is a network topology $N = (V,E,C)$ with routers V, physical links E and destination node t. It uses three internal data structures: the component list `numbering`, a list of nodes and node pairs. The border nodes are collected in a search tree `border` for easy access. Last but not least the queue `redundant` holds redundant connected nodes, which have been moved from the search tree as a second edge had been discovered.

In Chapter 3 we categorized to candidate types: single nodes with redundant edges (collected in `redundant`) and a pair of nodes with a common edge (both from `border`). We are going to examine the discovery phase corresponding to the transition rules 1,2,3.

```
func Align_Edges (edge) {
    x = edge.first();
    y = edge.second();
    if (numbering.edge_covered(x,y)) {
        if (numbering[x] < numbering[y]) {
            supergraph.new_edge (y,x);
        } else {
            supergraph.new_edge (x,y);
        }
    }
}

func Sequence (x) {
    numbering.append (x);
    for_each (x.adjacent_edges_begin(), x.adjacent_edges_end(), Align_Edges);
}

func Sequence ((x,y)) {
    numbering.append_nodepair (x,y);
    for_each (x.adjacent_edges_begin(), x.adjacent_edges_end(), Align_Edges);
    for_each (y.adjacent_edges_begin(), y.adjacent_edges_end(), Align_Edges);
}
```

Table B.1: Sequencing Code Fragment

The code fragment in fig B.1 lists the `sequence` functions. If called with a single node argument it appends this node into the node sequence `numbering` and then examines all adjacent edges. `Align_Edges` test first whether both end points of this edge are numbered yet. Then it can decide about the direction that is implied by the node numberings. The variant for sequencing

HammockSet components works similar. `append_nodepairs` assigns the same numbering to both nodes. It also examines adjacent edges of both nodes after inserting.

```
func Border (node) {
    if (redundant.find (node)) {
        // TR3: is already enqueued do nothing
    } else if (border.find (node)) {
        // TR2: detected an edge to node before, move into redundant
        border.erase (node);
        redundant.insert (node);
    } else {
        // TR1: this is the first detected edge
        border.insert (node);
    }
}

func Discover_Neighbors (x) {
    for_each (x.adjacent_nodes_begin(), x.adjacent_nodes_end(), Border);
}
```

Table B.2: Discover Code Fragment

The next code fragment B.2 shows the `discover_neighbors` procedure, which examines all neighbors of a newly sequenced node or node pair. There are three cases comparable to the transition rules: either the node has not been known before (TR1), then it is enlisted into the `border` search tree. Otherwise another edge to this node had been detected before (TR2), so this is now the second edge. Then that node is moved into the queue of `redundant` nodes. If there had been several edges detected before (TR3), then it is already in the `redundant` queue and nothing must be done.

Now we take a look how everything fits together. The main function (Fig. B.3) takes working elements from two data structures. First it test whether there are still redundant nodes that have not been processed (TR4). Such nodes are sequenced and then their neighbors are discovered. If that is empty, then there is no node left that has two edges into the destination HammockSet. Therefore, main constructs a candidate node pair (TR5), which has a Shared-Reserve-Link between two border nodes. Border nodes have been collected during discovery of neighbors.

B.3 Heuristic Optimum Construction

During the above construction process, some of the transition steps TR4 and TR5 might have been ambiguous and other candidates could have been elected, resulting in a different HammockSet. Construction of optimum HammockSets acquires topology information to select the best candidate that contributes to an outstanding result.

```
func Construct_HammockSet (destination) {
    assert (data structures empty);
    redundant.insert (destination);
    discover_neighbors (destination);
    while (!redundant.empty() || !border.empty()) {
        // here invariant of sequenced nodes always holds
        assert (invariant);

        if (!redundant.empty()) {
            // TR4: redundant consists of nodes with redundant edges into T
            x = redundant.top(); redundant.pop();
            sequence (x);
            discover_neighbors (x);
        }
        else if (!border.empty()) {
            // TR5: no single redundant node found,
            // search for an edge (Shared-Research-Link) in border
            // that has then two edges into T
            if (!(x,y) = border.find_nodepair ()) {
                abort ("no HammockSet in topology!");
            }
            border.erase (x);
            border.erase (y);
            sequence ((x,y));
            discover_neighbors (x);
            discover_neighbors (y);
        }
    }
}
```

Table B.3: Main Construction Loop

The main challenge for in-order selecting candidates is that the evaluation of partial HammockSets does not relate to evaluation of complete HammockSets, because then inner nodes act as forwarding stations. The criteria, required to find the optimal HammockSet, is not available during construction. We suggest other criteria that lead to better-than-average results. Chapter 4 tells us, that only for TR4 or TR5 ambiguous transition steps appear. We now suggest strategies and argument how we can gain better-than-average HammockSets through selecting such candidates.

For redundant node candidates to transition rule 4, we optimize for stable output bandwidth. This means beside actual bandwidth during normal operation also capacity after a link failure is relevant. Tab. B.4 sketches one possible rating function. Lets have some examples. We have four candidates: node u with $4 \times 10 Mbps$ of outgoing links, node v with $3\times$ 100 Mbps, node w with

1× 1 Gbps + 1× 100Mbps + 1× 10 Mbps and node *x* with 2× 100 Mbps + 2× 10 Mbps.

```
func rating (capacity[], count[]) {
    scale = 12;
    sum = 0;
    for all i do
        sum += (scaleˆcount[i]) * capacity[i];
    }
    return sum;
}
```

Table B.4: Rating Function

Rank	Node	1 Gbps	100 Mbps	10 Mbps	Rating(12)
1.	*v*	-	-	3	17.280
2.	*x*	-	2	2	15.840
3.	*u*	-	2	-	14.400
4.	*w*	1	1	1	1.110

Table B.5: Rating of four TR4 candidates

Tab. B.3 displays the rating results. Node *v* wins in this situation, since it has most stability. No single link failure can suddenly impact its outgoing capacity. The heuristic will place such nodes near to destination in the strongly-connected HammockSet component list. Because this, other nodes will use *v* as forwarding station. The traffic distribution will consider the low outgoing bandwidth of *v* which is reliable.

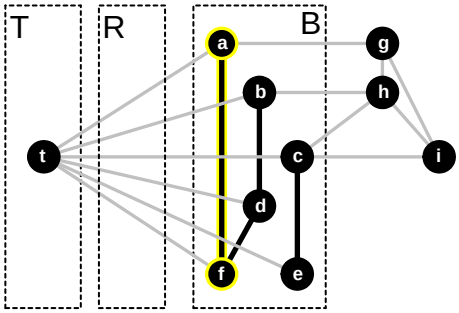

Figure B.1: A Normalized State before TR5

It is also possible, that an ambiguous candidate selection for TR5 occurs. Then candidates are all edges within the border node set. Fig. B.1 contains four candidate pairs where three are lightly-connected. This means, the edges have common nodes. If there are several lightly-connected components, it is sufficient to choose a component, since any SRL inside such a component redundantly connects all nodes of that component. For example choose candidate node pair (d, f) from fig. B.1. After that the remaining nodes of that component, namely *a* and *b* shift into

the redundant node set and can be processed next. We can even post-schedule the SRL-selection. After selection a lightly-connected component, we first assume all that nodes as connected and continue HammockSet construction. Afterwards we know the link significance and can then do a qualified selection.

Summarizing, heuristic construction can make some intelligent decisions, which are able to increase HammockSet quality. However, the cost criteria are orthogonal to construction decisions. Construction the optimal HammockSet has higher complexity than normal construction.

List of Figures

List of Tables

Bibliography

[AJY00] C. Alaettinoglu, V. Jacobson, and H. Yu. Toward millisecond igp convergence. http://www.nanog.org/mtg-0010/ppt/cengiz.pdf, 2000.

[Bel57] R. E. Bellmann. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[BR01] A. Basu and J. Riecke. Stability issues in ospf routing. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 225–236. ACM Press, 2001.

[Byr00] P. Byrnes. *Protocol management in computer networking*. Artech House telecommunications library, London, 2000.

[Cab02] Cabelingdb.com. Network redundancy. glossary page. http://cablingdb.com/GlossaryPages/GlossaryN/Network_Redundancy.asp, 2002.

[Cis02] Cisco_Systems. Routing basics. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/routing.htm, 2002.

[CRS99] I. Cidon, R. Rom, and Y. Shavitt. Analysis of multi-path routing. *IEEE/ACM Transactions on Networking (TON)*, 7(6):885–896, 1999.

[Epp94] D. Eppstein. Finding the k shortest paths. In *IEEE Symposium on Foundations of Computer Science*, pages 154–165, 1994.

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

[FPS02] A. Forster, R. J. Proctor, and P. A. Smith. Network resilience audit service. http://www.discoveryconsultancy.com/services/nras/NRAS.pdf, 2002.

[GLAM97] J. J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEETNWKG: IEEE/ACM Transactions on Networking, ACM Press*, 5, 1997.

[Gru02] T. Grubesic. Loss of major hub cities could cripple internet. http://www.acs.ohio-state.edu/researchnews/archive/intsurv.htm, 2002.

[Hed88] C. Hedrick. Routing information protocol. *RFC*, 1058, 1988.

[Hew95] Hewlett_Packard. Hp reference implementation of stl. `http://butler.hpl.hp.com/stl`, 1995.

[Him95] M. Himsolt. The graphlet homepage. `http://www.infosun.fmi.uni-passau.de/graphlet/`, 1995.

[Hop00] C. Hopps. Analysis of an equal-cost multi-path algorithm. *RFC*, 2992, 2000.

[Hui99] C. Huitema. *Routing in the internet*. Prentice Hall, Upper Saddle River, NJ 07458, second edition, 1999.

[ICM+02] G. Iannaccone, Chuah C., R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an ip backbone. In *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment workshop*, pages 237–242. ACM Press, 2002.

[LABJ00] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *SIGCOMM*, pages 175–187, 2000.

[LfTI02] Universität Passau Lehrstuhl für Theoretische Informatik. The gml homepage. `http://www.infosun.fmi.uni-passau.de/gml/`, 2002.

[Lie03] J. Liebenau. Resilient communication networks. `http://www2.cvn.columbia.edu/course/elene6901/General_Information.doc`, 2003.

[LMP+91] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Annual Symposium on Theory of Computing*, volume 23, pages 101–111, 1991.

[LW99] S.-W. Lee and C.-S. Wu. A *K*-best paths algorithm for highly reliable communication networks. *IEICE Trans. Communications*, E82-B(4):586–590, 1999.

[McQ74] J. McQuillan. Adaptive routing algorithms for distributed computer networks. In *BBN Report*, volume 2831, 1974.

[MFB99] M. Medard, S. G. Finn, and R. A. Barry. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking*, 7(5):641–652, 1999.

[Moy88] J. T. Moy. *OSPF : Anatomy of an Internet Routing Protocol*, page p. 276. Addison Wesley, Reading, MA, 1988.

[MRR78] J. M. McQuillan, I. Richer, and E. C. Rosen. Arpanet routing algorithm improvements. *BBN Technical Report*, 3803, 1978.

[Nar00] P. Narváez. Routing reconfiguration in IP networks, 2000.

[Pax96] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Proto-*

cols for Computer Communications, volume 26,4 of *ACM SIGCOMM Computer Communication Review*, pages 25–38, New York, August 1996. ACM Press.

[PSS+00] C. Partridge, A. C. Snoeren, W. T. Strayer, B. Schwartz, M. Condell, and I. Castineyra. FIRE: Flexible intra-AS routing environment. In *SIGCOMM*, pages 191–203, 2000.

[Sch01] T. Schmid. King (key components for the mobile internet of next generation). `http://www.zv.uni-wuerzburg.de/forschungsbericht/FOBE-akt/LS-00003031/KING-E.htm`, 2001.

[SCK+03] G. Schollmeier, J. Charznski, A. Kirstädter, C. Reichert, K. J. Schrodi, Y. Glickman, and C. Winkler. Improving the resilience in ip networks. `http://www.jcho.de/jc/Pubs/hpsr2003-ieee.pdf`, 2003.

[SNA91] D. Sidhu, R. Nair, and S. Abdallah. Finding disjoint paths in networks. In *Proceedings of the conference on Communications architecture and protocols*, pages 43–51. ACM Press, 1991.

[Tan03] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, Upper Saddle River, NJ 07458, 2003.

[TN94] M. To and P. Neusy. Unavailability analysis of long-haul networks. *IEEE J. Select. Areas Communications*, 12(1):pp. 100–109, 1994.

[VGLA01] S. Vutukury and J. J. Garcia-Luna-Aceves. MDVA: A distance-vector multipath routing protocol. In *INFOCOM*, pages 557–564, 2001.

[WC90] Z. Wang and J. Crowcroft. Shortest path first with emergency exits. In *Proceedings of the ACM symposium on Communications architectures and protocols*, pages 166–176. ACM Press, 1990.

[ZGLA98] W. T. Zaumen and J. J. Garcia-Luna-Aceves. Loop-free multipath routing using generalized diffusing computations. In *INFOCOM (3)*, pages 1408–1417, 1998.

Index