

TDK-dolgozat

Becsó Gergely



Eötvös Loránd
University

Eötvös Loránd University
Faculty of Science
Department of Operations Research

Pearl: Perfectly 2-resilient Skipping Routing with Few Header Bits via Cut Decomposition

Scientific Students' Association Report

Author:

Becsó Gergely
Applied Mathematics MSc year 3

Advisors:

Erika Bérczi-Kovács
assistant professor

Balázs Vass
assistant professor

2024

Contents

1	Introduction	4
2	Problem formulation and our contribution	6
2.1	Preliminaries	6
2.2	Our contributions	7
3	A 2-resilient algorithm with 1 header bit for subdivision graphs	8
3.1	Running time analysis	12
4	Perfectly 2-resilient skipping routing for 2-connected graphs	13
4.1	A constructive characterization by inserting pearls	13
4.1.1	Level of a pearl	14
4.2	2-resilient routing via pearl-construction	15
4.3	Running time analysis	15
5	Evaluation	18
6	Related works	21
7	Conclusion	22
	Appendix	26

Abstract

To meet their stringent dependability requirements, modern communication networks support local fast rerouting to quickly react to link failures. However, designing rerouting mechanisms that are resilient to multiple failures is algorithmically challenging, as routing decisions can only rely on local information. In particular, researchers have shown that it is generally impossible to achieve perfect 2-resilience: without packet header rewriting, it is impossible to ensure connectivity on the routing layer if there are two simultaneous link failures, even if the underlying network graph is still connected.

This thesis shows that perfectly 2-resilient rerouting can be achieved if just a small number of packet header fields can be rewritten by the routers. Our approach, *Pearl*, is based on a cut decomposition of the network. In particular, we define a new metric, L , that captures the structure of 2-edge cuts in the graph, and describe an efficient forwarding mechanism that gives a perfectly 2-resilient routing with only $\lceil \log_2(L) \rceil + 1$ extra header bits. Our approach is also efficient and based on skipping routing: routers only need to store a small number of routing rules. Our extensive simulations on real-world networks reveal that parameter L can be bounded by 5 in practice, yielding a perfectly 2-resilient skipping routing mechanism with 4 header bits for real-world instances. We also give an algorithm that uses only 1 header bit for a special class of graphs, solving about 37% of tested networks.

Kivonat

A modern távközlési hálózatok a megbízhatósági követelmények teljesítésére gyakran gyors helyi újrairányítást alkalmaznak, hogy az esetleges kapcsolati hibákra reagáljanak. Ugyanakkor az újrairányítási mechanizmusok tervezése algoritmikusan nehéz feladat, hiszen ezek csak a lokális információkra építhetnek. Korábbi munkákban kiderült, hogy még a tökéletes 2-rugalmasság is elérhetetlen: fejléc átírása nélkül lehetetlen két hiba esetén csak helyi információk révén biztosítani minden üzenet sikeres célba érését, még akkor is, ha az épen maradt élek részgrájában elérhető a cél.

Az alábbi munka megmutatja, hogy a tökéletes 2-rugalmasság megoldható kevés számú fejlécbit használatával. Bemutatásra kerül egy algoritmus, ami biztosít egy tökéletesen 2-rugalmas útírányt, kevés számú fejlécbittel. Az átírandó fejlécbitek számára felső becslést ad $\lceil \log_2(L(d)) \rceil + 1$, ahol $L(d)$ a 'gyöngymélység', a gráfok általunk definiált mérőszáma, ami a gráf 2-vágásai alapján méri a gráf összetettségét. A megoldásunk hatékony, és ugró útírányításon alapszik, ami lehetővé teszi, hogy az csomagtovábbító céleszközöknek csak kevés útírányítás szabályt kell eltárolniuk. Részletes szimulációink rámutatnak, hogy a való életbeli hálózatokon az L mérőszám 5-tel felülről becsülhető, ami egy olyan tökéletesen 2-rugalmas megoldást eredményez, ami a fejlécben 4 bit átírásával operál. Emellett adunk egy algoritmust egy speciális gráfosztályra, ami egyetlen bit használatával a teszhálózat 37%-át sikeresen megoldja.

Chapter 1

Introduction

Highly dependable communication networks rely on fast re-routing mechanisms [1]: based on local link availability information, routers forward network traffic to alternative neighbors in case of failures, without any information about potential additional failures downstream. Technically, such fast rerouting mechanisms are implemented in the routers' data plane, which is orders of magnitudes faster than reactions in the control plane.

Intuitively, one may expect that as long as the network remains connected after a set of k link failures, it should still be possible to route traffic between any source-destination pair. A network with this desirable property would be called *perfectly k -resilient*. However, due to only local information being available, this is not the case: Feigenbaum et al. [2] and Foerster et al. [3] proved that while 1-perfect resilience is always achievable, already 2-perfect resilience is not always realizable. This work is motivated by the open question whether perfect resilience can at least be achieved if we allow for some limited form of packet header rewriting [4]: by modifying a small number of bits in the packet header, essential information may be carried in the packet to allow routers to improve their routing decisions and re-establish connectivity. Indeed, heuristic approaches suggest that high-resilient routing can be solved effectively with a low number of header bits [5], and also for 3-edge connected graphs, 2-resilient routings have been shown to exist [4]. The enumerative search for 2-resilient routing using the SyPer tool [6] found that most real-world topologies admit a 2-resilient routing strategy without any header bit modifications. However, the best known upper bound on the number of header bits needed uses source matching, thus giving a bound of $\log(n)$, where n denotes the number of nodes [7].

Our work was motivated by this contradicting phenomenon, and our goal is to give a theoretical explanation why a very low number of header bits suffice for real-world instances.

Previous works gave theoretical guarantees for only highly connected networks, and little is known about topologies with low connectivity. This work is a first step towards understanding this area of problems.

Our main contribution in this thesis is a perfectly 2-resilient routing with just few header bit modifications, which we call *Pearl*¹. Our approach is based on a cut decomposition of the graph and applies to any network topology. The number of header bits used by our approach depends on a new graph-parameter called 'pearl-depth' that

¹The name is due to the cut decomposition we use.

captures the structural properties of 2-edge-cuts of the graph. We show that if the pearl-depth of the graph for a given destination is L , then a skipping routing with $\lceil \log_2(L) \rceil + 1$ extra header bits can be given. Our approach is also attractive, as it is based on skipping [8]: routing tables are small and essentially just skip failed links according to a fixed order.

Our experiments show that real-world networks have pearl-depth of at most 5, thus all real-world instances we consider can be solved by at most 4 header bits (see Table 1.1).

In addition, we also give a perfectly 2-resilient skipping algorithm using only 1 header bit for a special class of graphs called *subdivision graphs*. Our experiments show that about 37% of tested networks fall into this class.

The thesis is organized as follows. In Chapter 2 we define the problem and summarize our theoretical results. Chapter 3 presents the algorithm for subdivision graphs, which uses 1 header bit. The general algorithm is described in Chapter 4. Our evaluations are presented in 5.

Number of bits used	0	1	2	3	4	> 4
Number of topologies	56	89	63	32	3	0
Share of topologies	23%	36.6%	25.9%	13.1%	1.23%	0%

Table 1.1: Number of header bits used by our approach on Topology Zoo instances

Chapter 2

Problem formulation and our contribution

Let $G = (V, E)$ be a graph and let $D_G = (V, A)$ denote the directed graph arising from G by replacing each undirected edge uv by two arcs uv, vu of opposite orientation. For each node v we add a loop arc vv to A symbolizing packets originated at v . A vector $h \in \{0, 1\}^m$ is called an **m -length header**. A **skipping routing with m header bits** on graph G is a function $R : A \times \{0, 1\}^m \rightarrow \{A \times \{0, 1\}^m, STOP\}^p$ for some positive integer length p , where for each arc $uv \in A$ and $h \in \{0, 1\}^m$, if $R(uv, h)_i = (e, h')$, then $e \in A$ is an arc leaving v ($1 \leq i \leq p$).

For a given skipping routing function $R()$, a packet originated at node r (symbolized by loop rr) is forwarded to the first non-failed arc in $R(rr, 0)$ with its corresponding header, then so on, when arriving at a node v on an arc uv with header h , the packet is forwarded on the first non-failed arc in $R(uv, h)$ with the corresponding header defined by $R(uv, h)$. Function value 'STOP' indicates that the failures cut the packet from d and the packet cannot be delivered to its destination, thus the routing stops. A routing function is **perfectly k -resilient** (or shortly k -resilient) with respect to a destination node d if, in case of at most k edge failures, it delivers packets from every node to d that is not separated from d by the failures.

We consider the case of perfectly 2-resilient skipping routing functions and investigate the minimum number of header bits needed to have a 2-resilient routing (see Problem 1). Note that in case of 2-resiliency we can assume that length $p = 3$ in the routing function.

Problem 1: Perfectly 2-resilient skipping routing to d with m header bits

Input: $G = (V, E)$ undirected graph with a destination $d \in V$.

Output: Perfectly 2-resilient skipping routing $R()$ with m header bits

2.1 Preliminaries

Before presenting our results we explain why the problem can be reduced to 2-node-connected graphs: suppose that node x separates v from destination d . Then a 2-resilient routing task from v to d can be partitioned into two routing problems: first

<i>Number of header bits</i>	<i>Solvable graph class</i>	
no header rewriting	3-edge-connected graphs	[10]
$\log V $ bits (source matching)	every graph	[11]
1 bit	subdivisions of 3-edge-connected graphs	Thm 1
$\log L(d)$ bits	every graph	Thm 4

Table 2.1: Summary of former and current results on perfectly 2-resilient skipping routing. $L(d)$ denotes the pearl-depth of G with respect to the destination d .

routing from v to x , and then from x to d . More generally, for a given graph G we can consider its 2-node-connected components, and solve the resilient skipping routing task for each component, where the destination node of a component is the cut node separating the component from d . Then the number of extra header bits the routing mechanism uses is the maximum number of bits that the 2-node-connected components of the graph use. For example, the graph in Fig. 3.1 has three 2-node-connected components, and the brown and red cut nodes are separators. As we will see, the pink-orange and green-blue components admit a perfectly 2-resilient skipping routing with 1 header bit, and the purple cycle component needs no header bits, thus the whole graphs admits a routing with 1 bit, too.

Moreover, if all 2-node connected components admit a perfectly 2-resilient skipping routing without header bits, then so does the whole graph. Note that this is the case when each component is either 3-edge-connected or a cycle (see the case of 0 bits in Table 1.1).

Thus in our algorithms we will assume that the graph is 2-node-connected.

2.2 Our contributions

Our main contributions are the following (see Table 2.1 for a comparison with previous results):

- We show that if the graph is the subdivision of a 3-edge-connected graph, then $m = 1$ header bit is enough to give a perfectly 2-resilient skipping routing. The mechanism is presented in Chapter 3, Theorem 1. Since it was shown earlier that some instances of this graph class cannot be solved without extra header information [4], our routing is optimal on this class.

Our experiments show that about 37% percent of Topology Zoo [9] have this property.

- We construct a perfectly 2-resilient skipping routing with $m = \lceil \log_2(L(d)) \rceil + 1$ header bits, where $L(d)$ denotes the 'pearl-depth' parameter of the graph, related to the structure of its 2-edge cuts (see 4.1). The related theorem and algorithm are described in Chapter 4 and in Theorem 4.

Our evaluations give a pearl-depth of 5 in the worst case, thus all our real-world instance can be solved by at most 4 header bits.

More specifically, the contributions of the author of this thesis are the following: he proposed the pearl-depth as the good metric for graph complexity, created the high level algorithm idea and implemented the simulation codebase, ran the measurements and collected results.

Chapter 3

A 2-resilient algorithm with 1 header bit for subdivision graphs

In this Chapter we give a perfectly 2-resilient skipping routing algorithm using only 1 extra header bit information for a graph class called subdivision graphs.

A **subdivision** of an edge uv is its replacement by a new node x and two new edges ux, xv . We say that node x is the **subdivision node** of edge uv . In this chapter we consider a special case of 2-connected graphs that arise from 3-edge-connected graphs by subdivision steps, and show how to add one header bit of information to a circular-arborescence routing in order to remain 2-resilient for this graph class. This forwarding mechanism will be the starting point for the general case of 2-node-connected graphs.

This graph class played an important role also in [4], where it was shown that 2-resiliency cannot be achieved without header rewriting if the source is not known. The counterexample was a subdivision of a 3-edge connected graph. This counterexample shows that our construction is sharp in the sense that this graph class cannot be solved with less number of header bits.

In a directed graph $D = (V, A)$, a d -(**in**)-**arborescence** $T \subset A$ is a tree in the undirected sense such that all of its arcs are oriented towards d . Note that there is a unique directed path in T to d from every node in $V(T)$. An arborescence is spanning if $V(T) = V$. By the theorem of Edmonds [12], if there are k arc-disjoint vd -paths in D from every node v in V , then there are k arc-disjoint spanning d -arborescences in D .

Theorem 1. *If G is a graph that can be constructed from a 3-edge connected graph G_0 by applying subdivision steps only, then there is a perfectly 2-resilient skipping routing mechanism with one header bit. Moreover, the packet arrives at the destination d or the routing stops in $O(|V|)$ steps.*

Proof. The main idea of our algorithm is to start from a circular-arborescence routing $R_0()$ of the underlying 3-edge-connected graph, and extend it to another routing $R_b()$ on G using one header bit b .

First let us assume that G is the subdivision of the 3-edge connected graph G_0 and consider arc-disjoint spanning d -arborescences T_1, T_2, T_3 in D_{G_0} . By Edmonds' Theorem [12], such arborescences exist. We use the circular-arborescence approach of [4] as a subroutine. The following forwarding $R_0()$ gives a 2-resilient skipping routing

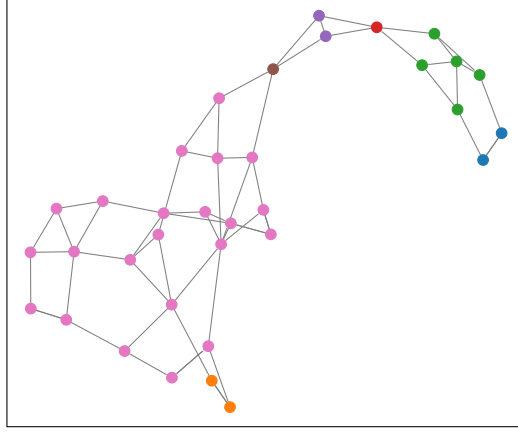


Figure 3.1: An example of a graph where each 2-node-connected component is a subdivision of a 3-edge connected graph, thus 2-resiliency can be solved by one header bit.

to d on G_0 (without header): routing towards d on an arbitrary starting arborescence T_p as long as possible, and if an arc uv is a failed arc on T_p , switching at u to $T_{p+1} \pmod 3$ and so on.

Now consider the natural extension of $R_0()$ on G : forwarding rules at $V(G_0)$ remain the same, and subdivision nodes perform the straightforward forwarding rule, that is forwarding if possible, and returning the packet if the other incident edge is a failed edge. This extended forwarding is denoted by $R()$.

Then we come across the following problem with $R()$: when a failure on a subdivided edge routes a packet backwards on the edge, it may happen that the packet is forwarded not on the right arborescence according to the circular-arborescence routing, that is from T_i not to T_{i+1} but to T_{i+2} or to an arc that is not part of any arborescence. We call such an event a 'bouncing back', which is indicated by the header bit b (see Figure 3.3). So we define a new routing mechanism $R_b()$ on G that is based on $R()$ but uses one header bit b .

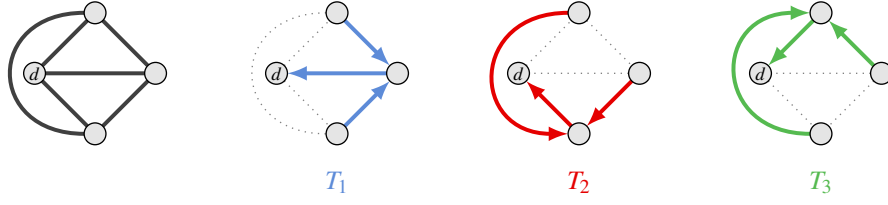
We assume that an edge uv is subdivided by new nodes $x_1, x_2, \dots, x_q$ in this order. We replace uv by $ux_1, x_1x_2, \dots, x_qv$ and vu by $vx_q, \dots, x_2x_1, x_1u$ if uv or vu is part of an arborescence T_p , respectively (see Fig. 3.2).

We extend $R_b()$ the following way at node x_i ($1 \leq i \leq q$). We describe the forwarding rules for arc $x_{i-1}x_i$, the case of arc x_ix_{i+1} is analogous. Assume that $u = x_0$ and $v = x_{q+1}$.

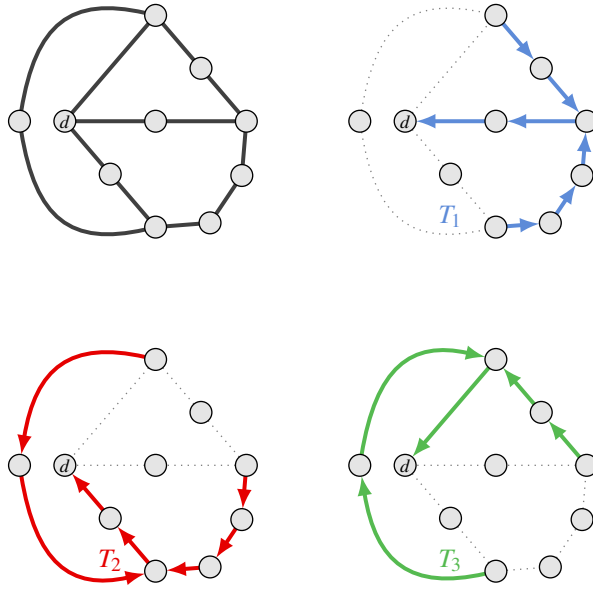
1. If arc x_ix_{i+1} is working, $R_b()$ is a simple forwarding (and no header change is applied).
2. If x_ix_{i+1} is a failed edge, and the arriving packet has header bit $b = 0$:
 - (a) If $x_{i-1}x_i \in T_p$ and $x_ix_{i-1} \in T_{p+1}$, the package returns on x_ix_{i-1} and the header b remains 0:

$$R_b(x_{i-1}x_i, 0) = (x_ix_{i+1}, 0); (x_ix_{i-1}, 0); (-, -).$$
 - (b) In every other case the package returns on x_ix_{i-1} but the header b is switched to 1:

$$R_b(x_{i-1}x_i, 0) = (x_ix_{i+1}, 0); (x_ix_{i-1}, 1); (-, -)$$



(a) K_4 (clique of size 4) without subdivisions



(b) K_4 with some subdivisions

Figure 3.2: Illustrations for the 1 extra bit setting

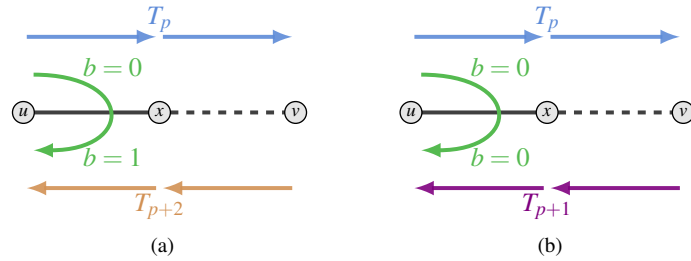


Figure 3.3: First bouncing back scenarios

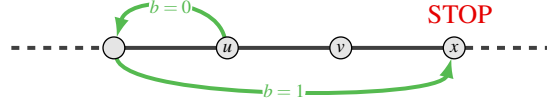


Figure 3.4: Stopping the forwarding

Algorithm 1: 2-resilient routing algorithm with 1 header bit for subdivision graphs

Input: $G = (V, E)$ subdivision of a 3-edge connected graph, $d \in V$

Output: perfectly 2-resilient skipping routing $R_b()$ to d with one header bit b

- 1 determine subdivision nodes and underlying graph G_0
 - 2 find arc-disjoint d -arborescences T_1^0, T_2^0, T_3^0 in G_0
 - 3 calculate circular arborescence routing $R^0()$ on $V(G_0)$ without header
 - 4 modify $R^0()$ to $R_b^0()$ on $V(G_0)$ with 1 header bit
 - 5 extend $R_b()$ on subdivision nodes
 - 6 **return** $R_b()$
-

3. If a packet with header bit $b = 1$ arrives at a failed edge, the routing stops:

$$R_b(x_{i-1}x_i, 1) = (x_ix_{i+1}, 1); STOP$$

For packets originated at x_i we forward it on x_ix_{i+1} or x_ix_{i-1} arbitrarily.

Finally we modify the forwarding rules at nodes of G_0 : If a packet with header bit $b = 1$ arrives on arc uv of G , then it is forwarded on arborescence T_i , where arc $vu \in T_{i-1}$. If arc vu is not part of any arborescence T_i , the packet is forwarded on an arbitrary arborescence not containing arc uv .

Claim 2. Routing $R_b()$ is perfectly 2-resilient.

Proof. First we show that if a packet reaches a node in $V(G_0)$, then it reaches d . To see this, observe that when a packet arrives at a node in $V(G_0)$, its routing becomes identical to $R_0()$ in the sense that the sequence of reached nodes in $V(G_0)$ is the same. Thus it is enough to show that if a packet is not cut from the destination, then it reaches a node in $V(G_0)$.

Since G_0 is 3-edge-connected, a node x is cut from d if and only if it is a subdivision node on an edge uv of G_0 and both failed edges are also on the subdivision of uv such that x is between them. In this case the routing starts towards one of the failures, and turns back, then it reaches the other failure and turns back. Note that at least one of these turnings is a bounce back, so the header bit b is set to 1 after at most two turning backs, and stops after at most three. If a packet with header bit $b = 1$ arrives at a subdivision node x on arc vx and arc xu is failed, then the packet is cut from d , and the routing can be stopped.

If a packet originated at a subdivision node is not cut from d by the failures, then it reaches a node in $V(G_0)$ and thus d , as we discussed above. $\square$

There are at most 4 arborescence-switches during a circular-arborescence routing (Theorem 1, [4]), and a packet turns back on a subdivided edge at most three times, so routing $R_b()$ terminates in $O(|V|)$ steps indeed. $\square$

3.1 Running time analysis

Subdivision nodes can be contracted in $O(|V|)$, and 3 arc-disjoint arborescences can be computed in $O(|E| \log |V| + |V| \log^2 |V|)$ [13], giving the bottleneck of the algorithm. Finally routing $R_b()$ can be determined in $O(|E|)$. Thus the overall running time of the algorithm is $O(|E| \log |V| + |V| \log^2 |V|)$.

Chapter 4

Perfectly 2-resilient skipping routing for 2-connected graphs

4.1 A constructive characterization by inserting pearls

Let $G = (V, E), d$ be an instance of Problem 1. Recall that we may assume that G is 2-node-connected. In order to create a 2-resilient routing for 2-connected graphs, we use a special construction of this graph class. We call a graph $G' = (V', E')$ and nodes $s, t \in V'$ a **pearl** between s and t if $G' + \{st\}$ is a 3-edge-connected graph. Nodes s, t are called the **gates** of the pearl.

Lemma 3. *Every 2-node-connected graph G can be built up from a 3-edge connected graph G_0 by the repetition of the following steps:*

- A) Subdividing an edge uv by a new node x .
- B) Replacing an edge uw by a pearl G' between nodes s, t and adding edges us, tw .

Proof. We prove the lemma by induction on the number of nodes. If G is not 3-edge connected, there is a 2-edge-cut in G (note that G is 2-node-connected). We consider a node set X such that i) $d \notin X$, ii) edges connecting X and $V \setminus X$ form a 2-edge-cut, and iii) X is minimal with respect to these properties.

If $X = \{x\}$, then we can apply the reverse of step A) on x and its incident edges xu, xw by replacing them with a single edge uw .

If $|X| > 1$, let su and tw denote the cut edges where $s, t \in X$ (and $u, w \notin X$). Then, it is easy to see that X is a pearl between s and t , and we can apply the reverse of step B).

In both cases, we can apply the induction on the resulting graph. $\square$

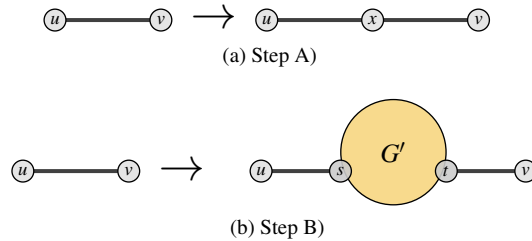


Figure 4.1: Illustration for Lemma 3

Algorithm 2: Pearl-construction of a 3-edge-connected graph

Input: $G = (V, E)$ 2-node-connected, $d \in V$
Output: pearl-construction $\mathcal{G} = \{G_1, \dots, G_k\}; G_0$
1 determine pearl construction $\mathcal{G} = \{G_1, \dots, G_k\}; G_0$
2 **while** exists 2-degree node **do**
3 find x 2-degree node with incident edges ux, vx
4 replace x, ux, vx by uv
5 **while** G' not 3-edge-connected **do**
6 find 2-edge-cut su, vt with $d \notin X$ and $|X|$ minimal
7 replace X by uv
8 add X to $\mathcal{G}$
9 **return** $\mathcal{G} = \{G_1, \dots, G_k\}; G_0$

We call such a construction starting from a 3-edge connected graph G_0 and applying steps A) and B) the **pearl-construction** of graph G . Observe that the set of pearls is the same for every pearl-construction of G .

We will use these building steps A) and B) to construct a proper routing on G , starting from a circular arborescence routing of the 3-edge-connected graph G_0 . We have seen in Theorem 1 that when only step A) is applied, there is 2-resilient forwarding function with only 1 header bit. We need to deal with the case when step B) is applied, too.

Suppose that $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$ denotes the set of pearls in the pearl-construction of G starting from G_0 . We need some further structural properties of the construction.

4.1.1 Level of a pearl

A packet crossing a pearl requires a different routing strategy than those originated at that pearl. In order to differentiate between these two type of packets, we will apply a counter in the header, called the **level**, which is a function $l : \{\mathcal{G}\} \cup E \cup V \rightarrow \mathbb{N}^+$. The pearl construction of G gives the level values:

- Starting graph G_0 and all its nodes and edges have level 1.
- New edges in step A) inherit the level of the subdivided edge.
- When step B) is applied on edge uw , we can associate a level to the inserted pearl G' (its nodes and edges inherit the same level):
 $l(G') := l(uw) + 1$, and $l(uv) := l(G')$ for every edge $uv \in E(G')$, and $l(v) = l(G')$ for every node $v \in V(G')$.

Note that applying step B) on an edge uw naturally defines a parent-child relation between the inserted pearl G' and the pearl (or G_0) that contains edge uw . The goal of the level function is to capture this relation locally. If a packet is originated at a node $v \in G'$, its starting header is $l(v)$ until it reaches a gate of G' . Then its header is set to $l(G') - 1$.

The maximum value of l is called the **pearl-depth** of G with respect to the destination d , denoted by $L(d)$. As we will see in the next Section, $\lceil \log_2(L(d)) \rceil + 1$ number of header bits are enough to define a 2-resilient forwarding. Although the number of

pearls and pearl-depth can be $\Omega(|V|)$ (see Fig. 4.2), pearl-depth is very small in real-world instances (see Chapter 5).

4.2 2-resilient routing via pearl-construction

Theorem 4. *Let G be a 2-connected graph and $d \in V$ a destination node. There is a perfectly 2-resilient skipping routing with $\lceil \log_2(L(d)) \rceil + 1$ header bits, where $L(d)$ denotes the pearl-depth of G with respect to d .*

Proof. We define a 2-resilient routing $R()$ on G according to the pearl-construction of G .

Let $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$ denote the ordered set of pearls in the pearl-construction of G , starting from G_0 .

First we consider G_0 and its subdivision nodes, and define $R()$ as $R_b()$ described in Chapter 3.

Next we define $R()$ on pearls $G_1, \dots, G_k$ in this order.

Let G_i between nodes s, t be the next pearl of the construction. For packets originated at G_j , circular-arborescence-type routing is applied until the packet reaches one of the gates s or t . The arborescences are found the following way: there are three arc-disjoint spanning s -arborescences in D_{G_j+st} . Deleting the extra arc st we get two spanning s -arborescences and one t -arborescence that spans $V(G_j) - s$. These arborescences are denoted by T_1^j, T_2^j, T_3^j , respectively.

Applying a circular-arborescence routing in G_j delivers a packet to either s or t up to 2 edge failures. Adapting the routing algorithm of Chapter 3 with one header bit b , $R_b^j()$ delivers a packet to either s or t up to 2 edge failures even if we add the subdivision nodes of G_j .

Routing $R()$ is 'merged' with $R_b^j()$ on G_j the following way (see Figure 4.3):

Case 1: For packets coming from a node in G_j , the header is set to $(l(G_j), 0)$ and circular arborescence routing with one header bit $R_b^j()$ is applied with arborescences T_1^j, T_2^j, T_3^j . If the packet reaches s or t , its header is set to $(l(G_j) - 1, 0)$ and forwarded on su or tv , respectively.

Case 2: For crossing packets that are coming from either s or t we apply a different routing. Note that these packets are exactly the ones arriving at s or t with level $l < l(G_j)$ in the header. We fix two edge-disjoint paths P_1^j and P_2^j in G_j , and apply the following routing (see Figure).

A packet from arc us is forwarded on path P_1^j towards t . If there is a failed edge, the packet turns back and continues on path P_2^j . Analogously, a packet on arc vt continues on path P_2^j towards s . If there is a failed edge on P_2^j , it turns back and continues on P_1^j .

In both cases, if both paths have a failed edge, the packets return to u or v , respectively.

If a packet reaches a gate, say s , and su is a failed edge, the packet returns on P_1^j with level $l(G_j) - 1$, and the bouncing back bit is set analogously to the subdivision case, as if s were a subdivision node on edge uv .

4.3 Running time analysis

The pearl construction can be determined in $O(|E||V|)$. Since the pearls are node-disjoint, all arc-disjoint arborescence triplets T_1^j, T_2^j, T_3^j and paths pairs P_1^j, P_2^j can be

Algorithm 3: Perfectly 2-resilient skipping routing algorithm

Input: $G = (V, E)$ 2-node-connected, $d \in V$

Output: 2-resilient skipping routing $R()$ to d

- 1 determine pearl construction $\mathcal{G} = \{G_1, \dots, G_k\}; G_0$ (see Alg. 2)
 - 2 calculate levels $l(G_i)$
 - 3 find arc-disjoint d -arborescences T_1^0, T_2^0, T_3^0 in G_0
 - 4 calculate $R_b^0()$ on $V(G_0)$ and its subdivision nodes (see Alg. 1)
 - 5 set $R()$ as $R_b^0()$
 - 6 **for** $i=1..k$ **do**
 - 7 find arc-disjoint arborescences T_1^i, T_2^i, T_3^i in G_i
 - 8 calculate $R_b^j()$ on $V(G_j)$ and subdivision nodes of $E(G_j)$ (see Alg. 1)
 - 9 find edge-disjoint paths P_1^j, P_2^j
 - 10 merge $R_b^j()$ with $R()$ for messages with header $l(G_j)$
 - 11 extend $R()$ on P_1^j, P_2^j for messages with header $< l(G_j)$
 - 12 **return** $R()$
-

computed in $O(|E| \log |V| + |V| \log^2 |V|)$ total time. Finally routing $R()$ can be determined in $O(|E|)$. Thus the overall running time of the algorithm is $O(|E||V|)$. $\square$

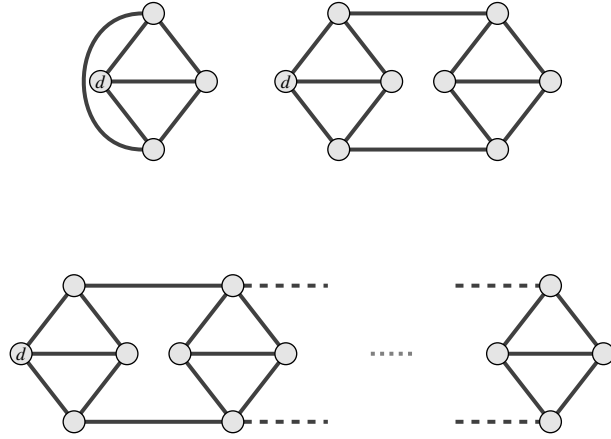


Figure 4.2: There can be as much as $\Theta(|V|)$ pearls. While the upper bound is imminent, the figure above hints a construction for $\Omega(|V|)$ pearls and pearl-depth.

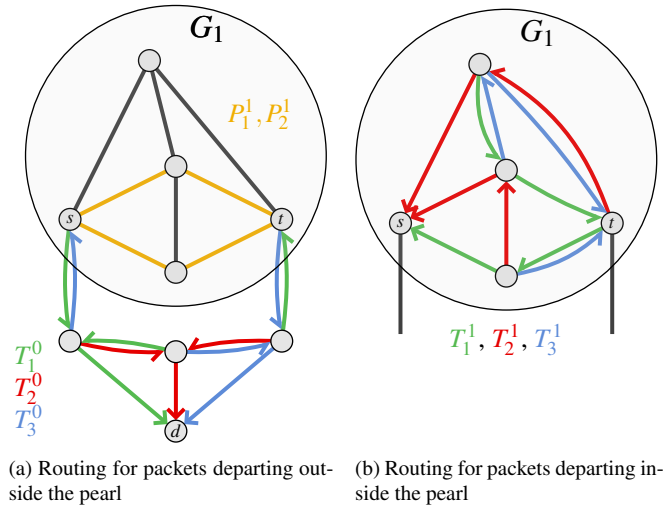


Figure 4.3: Example on routing with a pearl

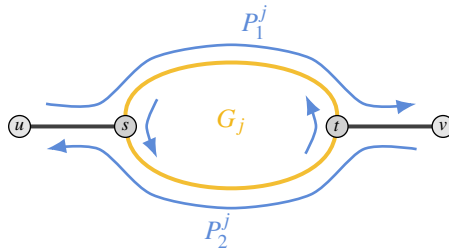


Figure 4.4: Forwarding rules for crossing packets, i.e. those arriving on us or vt .

Chapter 5

Evaluation

We run our experiments on the Topology Zoo network collection, which is a publicly available benchmark of ISP topologies [9]. For the experiments we used a commodity laptop with 16 GB RAM and an 2.2 GHz processor. The implementation is done in Python and can be found at <https://github.com/begeri/fast-failover>.

On Fig. 5.1, we have plotted the number of pearls in each graph in the Topology Zoo. For reference, we have also added the number of nodes and number of edges of the graphs to the plot. We can see that the number of pearls compared to the graph size $|V|$ is low in practice, typically being at least an order of magnitude smaller than the number of nodes.

Fig. 5.2 investigates the pearl-depth of the topologies, where pearl-depths were measured with randomly chosen destination nodes. It is clear that in spite of the $O(|V|)$ theoretical upper bound on the pearl-depth (see Fig. 4.2), it is typically small in practice. At its highest value, the pearl-depth reached 5, in only 3 topologies in our simulations, that is 1.23% of the cases. Also, the pearl-depth is typically much lower, see Table 5.1. As we can see, roughly 3 out of 5 topologies have a minimal pearl-depth, that is of 1. Further, a quarter, and a tenth of the graphs have pearl-depths of 2 and 3, respectively. In the remaining $< 4\%$ of the cases, the pearl-depth equaled 4, or 5. In conclusion, the higher pearl-depth the lower the probability that a random Topology Zoo instance has that depth. Further, this decrease in probability is dramatic.

Pearl-depth	1	2	3	4	5	> 5
Number of topologies	145	63	26	6	3	0
Share of topologies	59.7%	25.9%	10.7%	2.47%	1.23%	0%

Table 5.1: Number and share of pearl-depths of Topology Zoo graph instances

The fact that the pearl-depths typically do not exceed 4 means that in most cases, not more than 3 header bits are required to our algorithm to secure a perfectly 2-resilient routing. This is due to the $\lceil \log_2(\text{pearl-depth}) \rceil + 1$ bound on the extra bits. Displayed at the beginning of the thesis, Table 1.1 explored more in-depth the share of the number of header bits used. Note that for special graph classes like 3-edge-connected graphs or cactus graphs, no header bit rewriting is needed. In this case, the above-mentioned expression on the number of header bits is only an upper bound.

Fig. 5.3 depicts the running time of the pearl decomposition algorithm in function of the network node number. Note that by the theoretical running time analysis, this

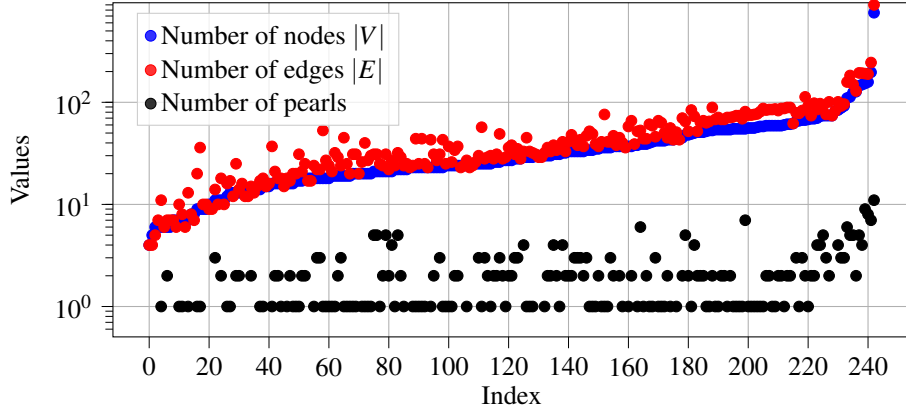


Figure 5.1: A scatter plot of the number of nodes, edges, and the number of pearls. At index i the data related to the i^{th} smallest graph is displayed.

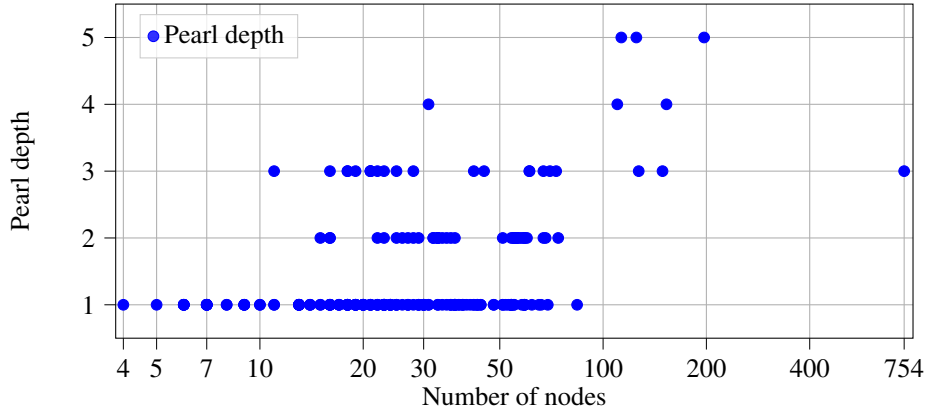


Figure 5.2: A scatter plot of the number of nodes and the pearl depth

step asymptotically dominates the running time. It is noticeable that while the algorithm was implemented in Python, and was run on a commodity laptop, apart from the 3 biggest graphs, the running time of the algorithm never exceeded 1 second. Also, for the biggest graphs studied with 754 nodes, the runtime was around 30 seconds. Judging from the data points that roughly align around a straight line on Fig. 5.3 with log-log scale, we can see that the empirical runtime complexity is also polynomial.

Fig. 5.4 gives us insight about the size distribution of the pearls. For each graph, the minimum pearl size, the maximum pearl size and the average pearl size is shown. As the first 49 graphs do not contain real pearls, they are not shown on the plot. We can see, that the minimal pearl size often is very small even in big graphs.

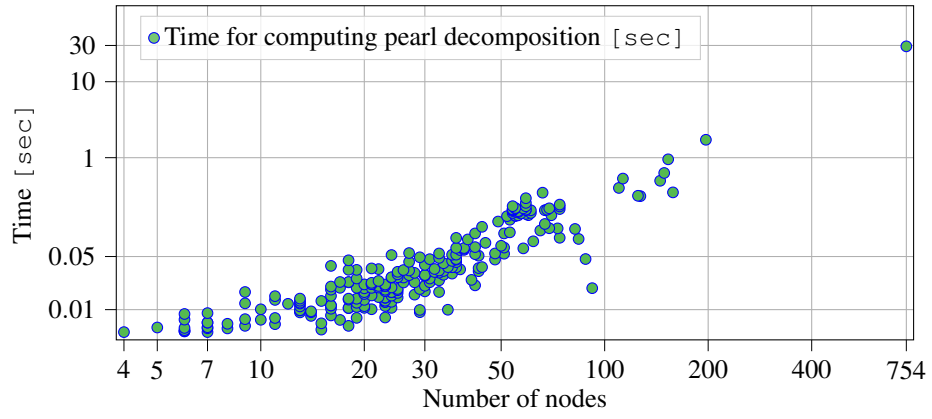


Figure 5.3: A scatter plot of the running times for constructing the pearl decomposition. Note that computing the pearl decomposition dominates the runtime of our algorithm.

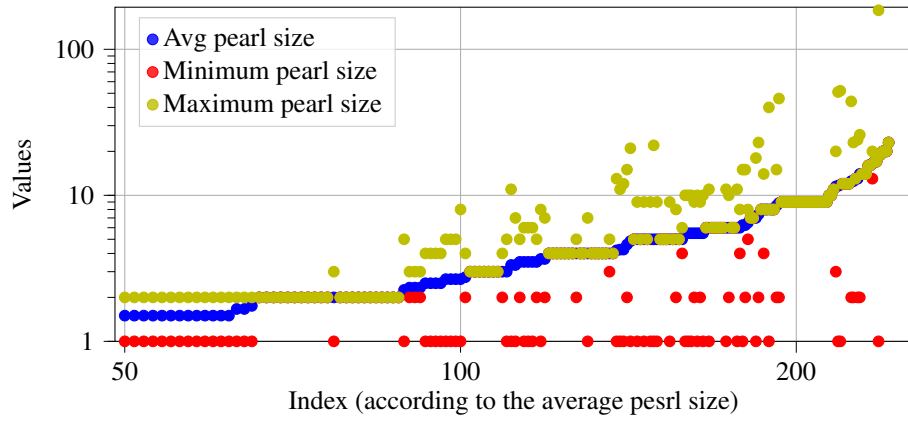


Figure 5.4: A scatter plot of pearl sizes, the average, the minimum and the maximum is shown. The indexes of the graphs here are according to their average pearl sizes. There are 49 topologies without a pearl, thus we plot data for graphs with having an index ≥ 50 .

Chapter 6

Related works

Györgyi et al. [6] gave a perfectly 1-resilient routing that uses a shortest path to the destination if there is no failure in the network. As for the directed case, 1-resilient routing is not always possible in such networks, shown by Akker and Foerster [14]. Foerster et al. proved the impossibility for an even broader problem class [15]. Chiesa et al. [4] showed that a 2-resilient routing can be given if the graph is 3-edge-connected, but such a routing does not always exist if the graph has lower connectivity. As for the general case, Dai et al. showed that 2-resilient routing is possible if both the source and the destination nodes are known [11], thus their solution requires $\log(n)$ extra header bits to encode the source node.

Former algorithms with theoretical guarantees mainly consider highly connected networks. A routing is ideal if it is $k - 1$ resilient and the graph is k -edge connected. Chiesa et al. [10] showed that 4-resilient or 5-resilient skipping routing can be given without header bits if the graph is 5- or 6-connected, respectively. It is an open question whether this property holds for higher connectivity as well. Foerster et al. showed that if both the source and destination nodes are known, such a routing exists for any k [16]. Dai et al. generalized the former setting for nonstatic failures and showed that $\log k$ header bits suffice for $k - 1$ -resilency if the graph is k -edge-connected [17]. Perfect resilience is impossible; Foerster et al. gave a counterexample of only eight nodes [18].

Several heuristic approaches were proposed for different variations of the problem [19]. Foerster et al. [20] combined multiple arborescences, while Karamoussanlis et al. adapted face routing with arborescence routing [21]. Foerster et al. proposed a post-processing approach [22]. Chiesa et al. implemented a framework addressing the problem on programmable switches [23]. Yang et al. offered a heuristic based on a special traversal on the graph edges [5]. Recently, in [24], Grobe et al. adapted the algorithm of [5] to accommodate directed graphs.

Chapter 7

Conclusion

We investigated perfectly 2-resilient skipping routing mechanisms. We gave an algorithm that solves the problem with only 1 header bit for subdivision graphs that covers a considerable fraction of real-world instances. For the general case we proposed an algorithm that uses $\lceil \log_2(L(d)) \rceil + 1$ header bits, where $L(d)$ denotes the pearl-depth of the underlying graph with respect to destination node d . Our experiments show that all real-world networks can be solved by at most 4 header bits.

It is still an open question whether perfectly 2-resilient routing can be solved by a constant number of header bits for 2-connected graphs. It is also an open problem to decide which subdivisions of 3-connected graphs have a 2-resilient routing without header.

Bibliography

- [1] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, “A survey of fast recovery mechanisms in the data plane,” *TechRxiv*, May 2020.
- [2] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, “Brief announcement: on the resilience of routing tables,” in *PODC*. ACM, 2012, pp. 237–238.
- [3] K.-T. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, “On the feasibility of perfect resilience with local fast failover,” in *Proc. SIAM APOCS*, 2021, pp. 55–69.
- [4] M. Chiesa, I. Nikolaevskiy, S. Mitrović, A. Panda, A. Gurtov, A. Maidry, M. Schapira, and S. Shenker, “The quest for resilient (static) forwarding tables,” in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.
- [5] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, “Keep forwarding: Towards k-link failure resilient routing,” in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 1617–1625.
- [6] C. Györgyi, K. G. Larsen, S. Schmid, and J. Srba, “Syper: Synthesis of perfectly resilient local fast re-routing rules for highly dependable networks,” in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024, pp. 2398–2407.
- [7] W. Dai, K.-T. Foerster, and S. Schmid, “A tight characterization of fast failover routing: Resiliency to two link failures is possible,” in *Proc. ACM SPAA*, 2023, p. 153–163.
- [8] K.-T. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, “On the feasibility of perfect resilience with local fast failover,” in *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.
- [9] “The internet topology zoo,” <http://www.topology-zoo.org/dataset.html>.
- [10] M. Chiesa, I. Nikolaevskiy, S. Mitrović, A. Gurtov, A. Madry, M. Schapira, and S. Shenker, “On the resiliency of static forwarding tables,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1133–1146, 2017.

- [11] W. Dai, K.-T. Foerster, and S. Schmid, “A tight characterization of fast failover routing: Resiliency to two link failures is possible,” in *35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2023.
- [12] J. Edmonds, “Edge-disjoint branchings,” *Combinatorial Algorithms*, pp. 91–96, 1973. [Online]. Available: <https://cir.nii.ac.jp/crid/1573387451139139584>
- [13] A. Bhargat, R. Hariharan, T. Kavitha, and D. Panigrahi, “Fast edge splitting and edmonds’ arborescence construction for unweighted graphs,” in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, 2008, pp. 455–464.
- [14] E. van den Akker and K.-T. Foerster, “Brief announcement: On the feasibility of local failover routing on directed graphs,” in *Stabilization, Safety, and Security of Distributed Systems*, T. Masuzawa, Y. Katayama, H. Kakugawa, J. Nakamura, and Y. Kim, Eds. Cham: Springer Nature Switzerland, 2025, pp. 375–380.
- [15] K.-T. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, “On the price of locality in static fast rerouting,” in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 215–226.
- [16] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, “Casa: Congestion and stretch aware static fast rerouting,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 469–477.
- [17] W. Dai, K.-T. Foerster, and S. Schmid, “On the resilience of fast failover routing against dynamic link failures,” in *Arxiv*, 2024.
- [18] K. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, “Brief announcement: What can(not) be perfectly rerouted locally,” in *34th International Symposium on Distributed Computing (DISC 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), H. Attiya, Ed. Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, international Symposium on Distributed Computing, DISC ; Conference date: 12-10-2020 Through 16-10-2020. [Online]. Available: <http://www.disc-conference.org/wp/disc2020/>
- [19] H.-K. Tan and T.-W. Kuo, “Optimistic fast rerouting,” in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1692–1697.
- [20] K.-T. Foerster, A. Kamisiński, Y.-A. Pignolet, S. Schmid, and G. Tredan, “Grafting arborescences for extra resilience of fast rerouting schemes,” in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.
- [21] G. Karamoussanlis, S. Althoff, E. van den Akker, and K. Foerster, “Analyzing network routing resilience: A hybrid approach of face and tree routing,” *RNDM. IEEE*, 2024.

- [22] K.-T. Foerster, A. Kamisiski, Y.-A. Pignolet, S. Schmid, and G. Tredan, “Improved fast rerouting using postprocessing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 537–550, 2022.
- [23] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, and S. Schmid, “Fast reroute on programmable switches,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 637–650, 2021.
- [24] J. Grobe, S. Althoff, and K.-T. Foerster, “Local fast failover routing on directed networks,” in *International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2024.

Appendix

Table 7.1: Data on pearl depths in the alphabetical order of graph names

$ V $	$ E $	Pearl depth	Graph name	# pearls	Avg. pearl size	Mim. pearl size	max. pearl size	Runtime [sec]
19	24	3	Aarnet	3	2	2.0	2.0	$1.94 \cdot 10^{-2}$
11	14	3	Abilene	3	2	2.0	2.0	$1.34 \cdot 10^{-2}$
23	31	2	Abvt	2	5	4.0	6.0	$2.14 \cdot 10^{-2}$
23	31	3	Aconet	3	2	2.0	2.0	$1.43 \cdot 10^{-2}$
25	30	1	Agis	1	6	6.0	6.0	$2.67 \cdot 10^{-2}$
10	9	1	Ai3	0	0	0.0	0.0	$7.35 \cdot 10^{-3}$
16	37	1	Airtel	1	8	8.0	8.0	$3.76 \cdot 10^{-2}$
25	24	1	Amres	0	0	0.0	0.0	$1.45 \cdot 10^{-2}$
18	25	2	Ans	2	5.5	1.0	10.0	$2.29 \cdot 10^{-2}$
30	29	1	Arn	0	0	0.0	0.0	$2.85 \cdot 10^{-2}$
34	47	2	Arnes	3	4	2.0	5.0	$2.87 \cdot 10^{-2}$
4	4	1	Arpanet196912	0	0	0.0	0.0	$5.06 \cdot 10^{-3}$
9	10	1	Arpanet19706	0	0	0.0	0.0	$6.12 \cdot 10^{-3}$
18	22	3	Arpanet19719	3	2	2.0	2.0	$4.46 \cdot 10^{-2}$
25	28	3	Arpanet19723	3	2	2.0	2.0	$1.66 \cdot 10^{-2}$
29	32	1	Arpanet19728	1	6	6.0	6.0	$9.26 \cdot 10^{-3}$
65	79	1	AsnetAm	3	3.67	2.0	7.0	0.11
21	22	2	Atmnet	2	1.5	1.0	2.0	$1.01 \cdot 10^{-2}$
25	57	1	AttMpls	1	20	20.0	20.0	$2.91 \cdot 10^{-2}$
22	25	1	Azrena	0	0	0.0	0.0	$1.46 \cdot 10^{-2}$
7	6	1	Basnet	0	0	0.0	0.0	$5.69 \cdot 10^{-3}$
27	28	2	Bbnplanet	2	1.5	1.0	2.0	$4.26 \cdot 10^{-2}$
48	65	2	Bellcanada	5	4.2	1.0	13.0	$5.48 \cdot 10^{-2}$
51	66	2	Bellsouth	2	2.5	1.0	4.0	0.1
23	43	1	Belnet2003	1	5	5.0	5.0	$1.26 \cdot 10^{-2}$
23	43	1	Belnet2004	1	5	5.0	5.0	$1.31 \cdot 10^{-2}$
23	44	1	Belnet2005	1	5	5.0	5.0	$1.11 \cdot 10^{-2}$
23	44	1	Belnet2006	1	5	5.0	5.0	$1.44 \cdot 10^{-2}$
21	31	3	Belnet2007	5	2	2.0	2.0	$2.25 \cdot 10^{-2}$
21	31	3	Belnet2008	5	2	2.0	2.0	$2.16 \cdot 10^{-2}$
21	31	3	Belnet2009	5	2	2.0	2.0	$2.16 \cdot 10^{-2}$
22	32	3	Belnet2010	5	2	2.0	2.0	$2.05 \cdot 10^{-2}$
53	65	1	BeyondTheNetwork	1	14	14.0	14.0	0.1
33	48	2	Bics	2	8.5	2.0	15.0	$5.49 \cdot 10^{-2}$
29	33	2	Biznet	4	1.75	1.0	2.0	$1.87 \cdot 10^{-2}$
37	38	1	Bren	0	0	0.0	0.0	$4.55 \cdot 10^{-2}$
18	24	1	BsonetEurope	1	6	6.0	6.0	$2.31 \cdot 10^{-2}$
20	31	1	BtAsiaPac	1	8	8.0	8.0	$1.91 \cdot 10^{-2}$
24	37	1	BtEurope	1	6	6.0	6.0	$1.7 \cdot 10^{-2}$
36	76	1	BtNorthAmerica	1	23	23.0	23.0	$3.61 \cdot 10^{-2}$
32	41	2	Canerie	4	2.5	1.0	4.0	$3.43 \cdot 10^{-2}$
44	43	1	Carnet	0	0	0.0	0.0	$3.63 \cdot 10^{-2}$
41	59	1	Cernet	1	8	8.0	8.0	$2.46 \cdot 10^{-2}$
10	9	1	Cesnet1993	0	0	0.0	0.0	$1.01 \cdot 10^{-2}$
13	12	1	Cesnet1997	0	0	0.0	0.0	$1.2 \cdot 10^{-2}$
13	12	1	Cesnet1999	0	0	0.0	0.0	$9.78 \cdot 10^{-3}$
23	23	1	Cesnet2001	0	0	0.0	0.0	$1.75 \cdot 10^{-2}$
29	33	1	Cesnet200304	1	4	4.0	4.0	$4.89 \cdot 10^{-2}$

39	44	2	Cesnet200511	2	2.5	1.0	4.0	$6.46 \cdot 10^{-2}$
39	44	1	Cesnet200603	1	4	4.0	4.0	$6.12 \cdot 10^{-2}$
44	51	1	Cesnet200706	1	6	6.0	6.0	0.12
52	63	1	Cesnet201006	1	12	12.0	12.0	0.17
42	66	1	Chinanet	1	9	9.0	9.0	0.1
15	18	1	Claranet	1	3	3.0	3.0	$1.3 \cdot 10^{-2}$
197	245	5	Cogentco	7	8.86	2.0	46.0	1.72
153	191	4	Colt	9	3.33	1.0	11.0	0.95
70	85	4	Columbus	4	6.25	1.0	15.0	0.17
14	17	1	Compuserve	1	5	5.0	5.0	$9.49 \cdot 10^{-3}$
33	38	3	CrlNetworkServices	3	2.67	1.0	5.0	$1.71 \cdot 10^{-2}$
51	52	2	Cudi	2	1.5	1.0	2.0	$6.55 \cdot 10^{-2}$
36	41	1	Cwix	1	7	7.0	7.0	$2.98 \cdot 10^{-2}$
30	29	1	Cynet	0	0	0.0	0.0	$2.85 \cdot 10^{-2}$
28	31	3	Darkstrand	3	2	2.0	2.0	$1.58 \cdot 10^{-2}$
6	11	1	Dataxchange	1	5	5.0	5.0	$8.76 \cdot 10^{-3}$
113	183	5	Deltacom	5	11.8	1.0	52.0	0.53
58	87	1	Dfn	1	20	20.0	20.0	$6.4 \cdot 10^{-2}$
31	38	4	Digex	4	2.75	2.0	4.0	$2.57 \cdot 10^{-2}$
19	29	1	Easynet	2	4	3.0	5.0	$2.57 \cdot 10^{-2}$
13	16	1	Eenet	0	0	0.0	0.0	$1.1 \cdot 10^{-2}$
20	30	1	EliBackbone	1	12	12.0	12.0	$1.34 \cdot 10^{-2}$
6	7	1	Epoch	0	0	0.0	0.0	$5.37 \cdot 10^{-3}$
30	32	1	Ernet	1	3	3.0	3.0	$3.87 \cdot 10^{-2}$
68	92	1	Esnet	1	12	12.0	12.0	0.21
37	45	2	Evolink	2	4	1.0	7.0	$4.63 \cdot 10^{-2}$
17	21	2	Fatman	2	1.5	1.0	2.0	$1.94 \cdot 10^{-2}$
23	27	1	Fccn	0	0	0.0	0.0	$1.26 \cdot 10^{-2}$
62	62	1	Forthnet	0	0	0.0	0.0	$7.93 \cdot 10^{-2}$
26	31	2	Funet	2	3.5	2.0	5.0	$2.28 \cdot 10^{-2}$
28	28	1	Gambia	0	0	0.0	0.0	$2.29 \cdot 10^{-2}$
16	18	1	Garr199901	1	4	4.0	4.0	$1.52 \cdot 10^{-2}$
23	25	1	Garr199904	1	4	4.0	4.0	$2.05 \cdot 10^{-2}$
23	25	1	Garr199905	1	4	4.0	4.0	$2.05 \cdot 10^{-2}$
22	24	1	Garr200109	1	4	4.0	4.0	$2.38 \cdot 10^{-2}$
24	26	1	Garr200112	1	4	4.0	4.0	$2.63 \cdot 10^{-2}$
27	29	1	Garr200212	1	3	3.0	3.0	$2.84 \cdot 10^{-2}$
22	24	1	Garr200404	1	4	4.0	4.0	$1.95 \cdot 10^{-2}$
54	71	1	Garr200902	1	9	9.0	9.0	0.2
54	68	1	Garr200908	1	9	9.0	9.0	0.19
55	69	1	Garr200909	1	9	9.0	9.0	0.19
54	68	2	Garr200912	2	5	1.0	9.0	0.2
54	68	2	Garr201001	2	5	1.0	9.0	0.21
54	71	1	Garr201003	1	9	9.0	9.0	0.23
54	71	2	Garr201004	2	5	1.0	9.0	0.21
55	72	1	Garr201005	1	9	9.0	9.0	0.22
55	74	1	Garr201007	1	9	9.0	9.0	0.17
55	74	1	Garr201008	1	9	9.0	9.0	0.17
56	75	1	Garr201010	1	9	9.0	9.0	0.19
56	75	1	Garr201012	1	9	9.0	9.0	0.17
56	76	1	Garr201101	1	9	9.0	9.0	0.2
57	77	1	Garr201102	1	9	9.0	9.0	0.18
58	81	1	Garr201103	1	9	9.0	9.0	0.22
59	83	2	Garr201104	2	6	1.0	11.0	0.29
59	84	2	Garr201105	2	5	1.0	9.0	0.21
59	85	2	Garr201107	2	5	1.0	9.0	0.25
59	85	1	Garr201108	1	9	9.0	9.0	0.23
59	86	1	Garr201109	1	9	9.0	9.0	0.18
59	87	2	Garr201110	2	5.5	1.0	10.0	0.2
60	87	1	Garr201111	1	10	10.0	10.0	0.18
61	89	2	Garr201112	2	5.5	1.0	10.0	0.19
61	89	2	Garr201201	2	5.5	1.0	10.0	0.2
8	7	1	Gblnet	0	0	0.0	0.0	$6.52 \cdot 10^{-3}$
27	38	2	Geant2001	2	6	1.0	11.0	$5.56 \cdot 10^{-2}$
34	52	1	Geant2009	1	16	16.0	16.0	$4.08 \cdot 10^{-2}$
37	58	1	Geant2010	1	16	16.0	16.0	$5.29 \cdot 10^{-2}$
40	61	1	Geant2012	1	17	17.0	17.0	$8.39 \cdot 10^{-2}$
7	8	1	Getnet	1	2	2.0	2.0	$6.74 \cdot 10^{-3}$

9	36	1	Globalcenter	1	9	9.0	9.0	$1.71 \cdot 10^{-2}$
67	113	2	Globenet	2	14	2.0	26.0	0.2
17	31	1	Goodnet	1	9	9.0	9.0	$2.31 \cdot 10^{-2}$
16	15	1	Grena	0	0	0.0	0.0	$1.03 \cdot 10^{-2}$
9	20	1	Gridnet	1	9	9.0	9.0	$1.21 \cdot 10^{-2}$
37	47	2	Grnet	2	3.5	2.0	5.0	$6.48 \cdot 10^{-2}$
149	193	3	GtsCe	4	12.5	2.0	44.0	0.63
32	33	2	GtsCzechRepublic	2	1.5	1.0	2.0	$3.13 \cdot 10^{-2}$
30	31	1	GtsHungary	0	0	0.0	0.0	$1.74 \cdot 10^{-2}$
33	37	2	GtsPoland	2	3.5	1.0	6.0	$2.43 \cdot 10^{-2}$
21	24	2	GtsRomania	2	1.5	1.0	2.0	$1.54 \cdot 10^{-2}$
35	37	2	GtsSlovakia	2	2	2.0	2.0	$3.63 \cdot 10^{-2}$
21	25	1	Harnet	1	4	4.0	4.0	$3.49 \cdot 10^{-2}$
7	13	1	Heanet	1	4	4.0	4.0	$9 \cdot 10^{-3}$
13	14	2	HiberniaCanada	2	1.5	1.0	2.0	$1.23 \cdot 10^{-2}$
55	81	3	HiberniaGlobal	7	4.86	1.0	21.0	0.21
8	8	1	HiberniaIreland	0	0	0.0	0.0	$5.66 \cdot 10^{-3}$
18	22	3	HiberniaNireland	3	2	2.0	2.0	$1.4 \cdot 10^{-2}$
15	15	1	HiberniaUk	0	0	0.0	0.0	$6.6 \cdot 10^{-3}$
22	29	2	HiberniaUs	2	5.5	2.0	9.0	$3.42 \cdot 10^{-2}$
18	53	1	Highwinds	1	13	13.0	13.0	$3.35 \cdot 10^{-2}$
16	21	2	HostwayInternational	2	3	2.0	4.0	$1.83 \cdot 10^{-2}$
24	37	2	HurricaneElectric	2	5.5	1.0	10.0	$5.23 \cdot 10^{-2}$
18	24	1	Ibm	1	9	9.0	9.0	$1.35 \cdot 10^{-2}$
37	66	2	Iij	2	6	1.0	11.0	$8.8 \cdot 10^{-2}$
31	35	1	Iinet	1	4	4.0	4.0	$4.62 \cdot 10^{-2}$
14	15	1	Ilan	0	0	0.0	0.0	$8.27 \cdot 10^{-3}$
27	36	2	Integra	2	5.5	2.0	9.0	$3.28 \cdot 10^{-2}$
73	97	3	Intellifiber	4	6.25	2.0	15.0	0.12
19	45	1	Internetmci	1	17	17.0	17.0	$3.34 \cdot 10^{-2}$
66	78	1	Internode	1	10	10.0	10.0	0.35
110	158	4	Interoute	6	8	1.0	40.0	0.4
39	53	2	Intranetwork	6	2.67	1.0	8.0	$6.33 \cdot 10^{-2}$
125	150	4	Ion	5	7.4	2.0	23.0	0.32
33	41	2	IowaStatewideFiberMap	2	6	2.0	10.0	$4.79 \cdot 10^{-2}$
51	64	2	Iris	2	6.5	5.0	8.0	$5.45 \cdot 10^{-2}$
23	23	1	Istar	0	0	0.0	0.0	$1.83 \cdot 10^{-2}$
11	10	1	Itnet	0	0	0.0	0.0	$6.44 \cdot 10^{-3}$
29	45	1	Janetbackbone	1	7	7.0	7.0	$9.91 \cdot 10^{-3}$
20	40	1	Janetlense	1	8	8.0	8.0	$2.15 \cdot 10^{-2}$
18	17	1	Jgn2Plus	0	0	0.0	0.0	$1.05 \cdot 10^{-2}$
25	30	2	Karen	3	1.67	1.0	2.0	$3.15 \cdot 10^{-2}$
754	899	3	Kdl	11	18.64	1.0	185.0	29.2
26	28	2	KentmanFeb2008	2	1.5	1.0	2.0	$2.65 \cdot 10^{-2}$
38	39	1	KentmanJan2011	0	0	0.0	0.0	$3.39 \cdot 10^{-2}$
16	17	2	KentmanJul2005	2	1.5	1.0	2.0	$1.13 \cdot 10^{-2}$
13	12	1	Kreonet	0	0	0.0	0.0	$9.2 \cdot 10^{-3}$
42	46	3	LambdaNet	3	2.33	2.0	3.0	$2.09 \cdot 10^{-2}$
69	74	1	Latnet	2	2	2.0	2.0	0.12
6	7	2	Layer42	2	1.5	1.0	2.0	$7.39 \cdot 10^{-3}$
43	43	1	Litnet	0	0	0.0	0.0	$5.13 \cdot 10^{-2}$
20	27	1	Marnet	1	3	3.0	3.0	$1.11 \cdot 10^{-2}$
16	18	2	Marwan	2	2	2.0	2.0	$2.01 \cdot 10^{-2}$
67	83	3	Missouri	3	8	4.0	14.0	0.13
6	5	1	Mren	0	0	0.0	0.0	$5.74 \cdot 10^{-3}$
37	40	1	Myren	1	2	2.0	2.0	$3.6 \cdot 10^{-2}$
6	7	1	Napnet	0	0	0.0	0.0	$5.2 \cdot 10^{-3}$
13	17	1	Navigata	1	4	4.0	4.0	$9.8 \cdot 10^{-3}$
7	10	1	Netrail	1	3	3.0	3.0	$6.75 \cdot 10^{-3}$
35	39	1	NetworkUsa	1	4	4.0	4.0	$9.96 \cdot 10^{-3}$
17	20	1	Nextgen	1	2	2.0	2.0	$7.32 \cdot 10^{-3}$
36	41	1	Niif	1	5	5.0	5.0	$5.28 \cdot 10^{-2}$
19	25	1	Noel	1	6	6.0	6.0	$1.08 \cdot 10^{-2}$
7	6	1	Nordu1989	0	0	0.0	0.0	$5.87 \cdot 10^{-3}$
14	13	1	Nordu1997	0	0	0.0	0.0	$9.24 \cdot 10^{-3}$
9	10	1	Nordu2005	0	0	0.0	0.0	$7.49 \cdot 10^{-3}$
13	15	2	Nsfnet	2	2.5	1.0	4.0	$1.31 \cdot 10^{-2}$
20	26	1	Oxford	1	6	6.0	6.0	$1.06 \cdot 10^{-2}$

18	27	1	Pacificwave	1	3	3.0	3.0	$1.05 \cdot 10^{-2}$
21	27	2	Packetexchange	4	2.25	1.0	5.0	$1.93 \cdot 10^{-2}$
45	70	2	Palmetto	2	11.5	3.0	20.0	$7.57 \cdot 10^{-2}$
16	20	1	Peer1	1	5	5.0	5.0	$1.13 \cdot 10^{-2}$
127	129	2	Pern	2	2	2.0	2.0	0.31
36	41	3	PionierL1	3	3.33	1.0	7.0	$4.38 \cdot 10^{-2}$
38	52	1	PionierL3	1	14	14.0	14.0	$6.07 \cdot 10^{-2}$
24	25	1	Psinet	1	2	2.0	2.0	$1.53 \cdot 10^{-2}$
20	31	1	Quest	1	11	11.0	11.0	$1.7 \cdot 10^{-2}$
84	101	2	RedBestel	4	4.25	1.0	11.0	$8.52 \cdot 10^{-2}$
19	32	1	Rediris	1	9	9.0	9.0	$1.39 \cdot 10^{-2}$
5	4	1	Renam	0	0	0.0	0.0	$5.82 \cdot 10^{-3}$
24	23	1	Renater1999	0	0	0.0	0.0	$1.27 \cdot 10^{-2}$
24	27	1	Renater2001	1	4	4.0	4.0	$1.96 \cdot 10^{-2}$
30	36	2	Renater2004	2	3	1.0	5.0	$2.06 \cdot 10^{-2}$
33	43	3	Renater2006	3	2.67	1.0	5.0	$3.07 \cdot 10^{-2}$
33	43	3	Renater2008	3	2.67	1.0	5.0	$3.27 \cdot 10^{-2}$
43	56	1	Renater2010	1	12	12.0	12.0	$3.2 \cdot 10^{-2}$
19	21	1	Restena	1	3	3.0	3.0	$1.23 \cdot 10^{-2}$
37	36	1	Reuna	0	0	0.0	0.0	$4.22 \cdot 10^{-2}$
16	18	1	Rhnet	1	2	2.0	2.0	$8.39 \cdot 10^{-3}$
31	34	2	Rnp	2	1.5	1.0	2.0	$2.29 \cdot 10^{-2}$
42	50	1	Roedunet	1	6	6.0	6.0	$6.65 \cdot 10^{-2}$
48	52	2	RoedunetFibre	2	3.5	1.0	6.0	$4.56 \cdot 10^{-2}$
18	17	1	Sago	0	0	0.0	0.0	$6.12 \cdot 10^{-3}$
43	45	2	Sanet	2	2	1.0	3.0	$3.52 \cdot 10^{-2}$
7	7	1	Sanren	0	0	0.0	0.0	$5.06 \cdot 10^{-3}$
19	20	1	Savvis	1	2	2.0	2.0	$7.79 \cdot 10^{-3}$
28	35	3	Shentel	3	3.67	1.0	8.0	$3.05 \cdot 10^{-2}$
74	76	3	Sinet	3	1.67	1.0	2.0	0.21
11	10	1	Singaren	0	0	0.0	0.0	$7.77 \cdot 10^{-3}$
15	16	1	Spiralight	0	0	0.0	0.0	$5.46 \cdot 10^{-3}$
11	18	2	Sprint	2	3.5	1.0	6.0	$1.5 \cdot 10^{-2}$
26	49	2	Sunet	3	7	1.0	18.0	$3.32 \cdot 10^{-2}$
50	73	3	Surfnet	4	4.25	1.0	12.0	$6.92 \cdot 10^{-2}$
74	92	4	Switch	5	4.6	2.0	15.0	0.23
42	63	1	SwitchL3	1	14	14.0	14.0	$5.37 \cdot 10^{-2}$
74	74	1	Syringa	0	0	0.0	0.0	$8.83 \cdot 10^{-2}$
145	194	2	TataNld	5	11.6	1.0	51.0	0.5
6	6	1	Telecomserbia	0	0	0.0	0.0	$5.17 \cdot 10^{-3}$
53	89	2	Tinet	2	16.5	13.0	20.0	0.15
12	16	1	TLex	1	4	4.0	4.0	$1.19 \cdot 10^{-2}$
20	20	1	Twaren	0	0	0.0	0.0	$1.61 \cdot 10^{-2}$
82	82	1	Ulaknet	0	0	0.0	0.0	0.12
25	29	2	UniC	2	1.5	1.0	2.0	$1.87 \cdot 10^{-2}$
13	25	1	Uninet	2	6	4.0	8.0	$1.4 \cdot 10^{-2}$
74	101	2	Uninett2010	2	13	2.0	24.0	0.24
69	98	2	Uninett2011	2	12.5	2.0	23.0	0.22
24	24	1	Uran	0	0	0.0	0.0	$1.69 \cdot 10^{-2}$
158	189	5	UsCarrier	8	5	1.0	22.0	0.35
49	84	1	Uunet	1	20	20.0	20.0	0.14
25	26	1	Vinaren	0	0	0.0	0.0	$1.8 \cdot 10^{-2}$
24	23	1	VisionNet	0	0	0.0	0.0	$1.06 \cdot 10^{-2}$
88	92	3	VtlWavenet2008	3	2.33	2.0	3.0	$4.63 \cdot 10^{-2}$
92	96	3	VtlWavenet2011	3	2.33	2.0	3.0	$1.92 \cdot 10^{-2}$
30	33	2	WideJpn	2	1.5	1.0	2.0	$2.78 \cdot 10^{-2}$
24	34	2	Xeex	2	5	2.0	8.0	$2.67 \cdot 10^{-2}$
34	49	1	Xspedius	1	19	19.0	19.0	$3.22 \cdot 10^{-2}$
23	24	1	York	1	2	2.0	2.0	$7.88 \cdot 10^{-3}$