WPR: A Fault-tolerant Routing Scheme for Cayley Graphs

Daniela Aguirre-Guerrero, Lluís Fàbrega, and Pere Vilà and Enrique Rodriguez-Colina

***Abstract*—The abstract goes here.**

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

THE increasing demand of high-performance computer systems (HPCS) has imposed the challenge of designing routing schemes and large-scale network topologies that support the optimal operation of such systems. Recent trends in the design of routing schemes for large-scale networks have led to the design routing schemes together with families of network topologies. This approach ensures that routing schemes take advantage of the topological network properties [1]–[5]. However, the well-known tradeoff between space and efficiency is still one of the main challenges in the design of routing schemes for large-scale networks. In addition, the design of network topologies that are scalable and robust plays a pivotal role in the optimal operation of HPCS.

This paper proposes the Word-processing based Routing (WPR), a new routing scheme for large-scale networks whose underlying topology is given by a Cayley graph (CG). Cayley graphs are a geometric representation of algebraic groups. These graphs have been used as topologies of a wide variety of communication networks [**?**], [6], [7], e.g. processor interconnection networks, wireless sensor networks, DCNs, etc. The reason is that their properties of node-transitivity, link-connectivity and low average distance between nodes enable high performance and robustness in large-scale networks [**?**].

In spite of these advantages, the use of CGs as underlying topologies of HPCS is limited. We argue that it is a consequence of the fact that most of communication networks, whose topologies are defined by CGs, apply traditional routing schemes based on topology-agnostic algorithms for path computation, such as the Bellman-Ford and Dijsktra algorithms. These algorithms receive the whole graph as input, which results in high space and time complexity for the routing schemes using them. On the other hand, routing schemes dedicated to CGs have been designed to achieve low space and time complexity taking advantage of the aforementioned properties of CGs. However, the challenge of achieving low time and space complexity is major for generic proposals as they must work on CGs with different topological structures.

The state of the art on routing schemes for CGs includes deterministic proposals [**?**], [**?**] and just only one fault-tolerant proposal [**?**], which does not provide minimal routing and does not guarantee packet delivery. In contrast, we propose the WPR that is a generic routing scheme for CGs that guarantees packet delivery and provides: minimal routing, path diversity and fault-tolerance. As far as this author known, the WPR if the first generic routing scheme for CGs with these features. The WPR applies the path computation algorithms presented in [**?**] and a novel mechanism of fault-tolerance, presented in this work. The fault-tolerant mechanism supports multiple failures and provides minimal routing in spite of nodes do not keep a global record of failures. This mechanism allows that the WPR can be deployed in topologies defined by subgraphs of CGs, which enables the design of scalable and robut topologies based on CGs. Through a space and time complexity analysis, it is shown that the WPR stays competitive with respect to the state of the art on generic routing in CGs.

The remainder of the paper is as follows: Section II introduces the theoretical framework of the word-processing approach applied in this proposal. Section III gives an overview of the WPR, while sections IV and V details the operation of the WPR for static and fault-tolerant routing, respectively. Section VI presents a comparison between the WPR and the state of art on routing schemes for CGs. Finally, Section VII provides the conclusions.

II. The Word-Processing Approach

This section explains how paths and nodes in a CG can be represented as words of a regular language; and how this representation was applied for developing path computation algorithms in CGs [**?**]. These algorithms are a key component of the WPR. Before presenting the word-processing approach, it is necessary to give the formal definition of CGs.

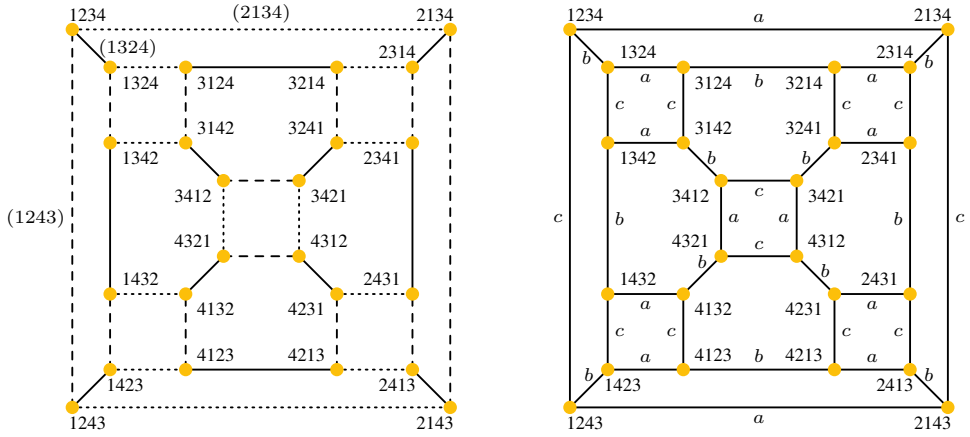
Let $\mathcal{G} =< S|R >$ be an algebraic group, where S and R are the set of generators and relators, respectively [**?**, Section 2.2]. Then $\mathcal{G}$ has an associated CG, denoted by $\Gamma(\mathcal{G}, S)$, where the set of vertices $V(\Gamma)$ is given by the set of group elements, and there is an edge $e \in E(\Gamma)$ from g to h if and only if $g \cdot s = h$ for $g, h \in \mathcal{G}$ and $s \in S$.

Hereafter vertices in $V(\Gamma)$ and elements of $\mathcal{G}$ are used interchangeably, and likewise links in $E(\Gamma)$ and generators in S. For instance, consider the symmetric group Sym_p, where the group elements are given by the permutations of

D. Aguirre-Guerrero is with the Universidad Autónoma Metropolitana, Unidad Cuajimalpa, Cuajimalpa 52005, Mexico (e-mail: d.aguirre@correo.ler.uam.mx).

Lluís Fàbrega, and Pere Vilà are with the Universitat de Girona, Girona 17071, Spain (e-mail: luis.fabrega@udg.edu; pere.vila@udg.edu).

E. Rodriguez-Colina is with the Universidad Autónoma Metropolitana, Unidad Iztapalapa, Iztapalapa, Mexico City 09340, Mexico (e-mail: erod@xanum.uam.mx).



(a) Nodes represent group elements in Sym_4 and edges represent generators in S. Each generator (2134), (1324) and (1243) is represented by dotted, solid and dashed links respectively.

(b) Edges labelled with letters representing their corresponding generators according with Eq. (2).

Fig. 1: Cayley graph of the symmetric group Sym_4 with generators $S = \{(2134), (1324), (1243)\}$. This graph is called Bubble-sort graph and denoted by $BS(4)$.

the set $\{1, \ldots, p\}$, and the group operation is the composition of permutations. Figure 1a presents the CG of Sym_4 with the group generators are given by the set of permutations $S = \{(2134), (1324), (1243)\}$, which is known as Bubble-sort graph and denoted by $BS(4)$. Note that nodes 1234 and 2134 are connected through the link (2134) due to 2134 results of apply the permutation (2134) to 1234 and vice versa.

A. Letters as links and words as paths

Assume that edges of a CG are labelled according to a bijective map

$$\phi : \{S \cup S^{-1}\} \to \mathcal{A}, \tag{1}$$

where S^{-1} is the set of inverses of S and $\mathcal{A}$ is an alphabet. Then ϕ assigns each generator and its inverse to lowercase and uppercase variants of the same letter. Thus every path in the CG can be represented by a unique word w over $\mathcal{A}$. Returning to $BS(4)$, where $S = S^{-1}$, let $\mathcal{A} = \{a, b, c\}$ be an alphabet and let ϕ denote the map:

$$\begin{array}{rccc} \phi: & S & \to & \mathcal{A} \\ & (2134) & \to & a \\ & (1324) & \to & b \\ & (1243) & \to & c \end{array} \tag{2}$$

Figure 1b shows $BS(4)$ with their edges labelled according to a map (1). Then, the word abc represents the sequence of edges $(2134)(1324)(1243)$ and thus a path, e.g. between 1234 to 2341.

Let v and w be words over $\mathcal{A}$, such that S and $\mathcal{A}$ satisfy map (2). It has that:

- The words v and w are called **equivalent**, i.e. $v =_{\mathcal{G}} w$, if they represent paths between the same pair of nodes. In Fig. 1b, $abc =_{\mathcal{G}} babca$ due to they represents paths between the same nodes, e.g. 1243 and 2431.
- The **inverse of** w, denoted by w^{-1}, is given by the reverse string of the inverse letters of w, e.g. if $a^{-1} = A$, $c^{-1} = C$ and $w = aC$, then $w^{-1} = cA$. Consider now that w represents a path from g to h then w^{-1} represents a path from h to g. In the case of $BS(4)$, abc represents a path from 1432 to 4312, meanwhile $(abc)^{-1} = cba$ represents a path from 4312 to 1432.
- The **reduced form of** w, denoted by w_{red}, results from removing the substrings of the form uu^{-1} from w. The words w and w_{red} are equivalent, i.e. $w =_{\mathcal{G}} w_{red}$, and thus represent paths between the same pair of nodes. Consider a path $w = abcaaca$ in Fig. 1b, then $w_{red} = aba$ due to $(ca)^{-1} = ac$. Since $abcaaca =_{\mathcal{G}} aba$, these words represent paths between the same pair of nodes, e.g. 1342 and 4321.

Definition 1: Let $\Gamma(\mathcal{G}, S)$ be a CG and $\mathcal{A}$ be an alphabet, such that S and $\mathcal{A}$ satisfy map (1). Then, the **free group** over $\mathcal{A}$, denoted by $\mathcal{F}(\mathcal{A})$, consists of all reduced words over $\mathcal{A}$ including the symbol $e_{\mathcal{A}}$ that denotes the null string.

From the above definition, the language $\mathcal{F}(\mathcal{A})$ defines all paths in $\Gamma(\mathcal{G}, S)$. In particular, $e_{\mathcal{A}}$ denotes the empty path. It is also possible to define a language for the shortest paths as follows.

Definition 2: Let $<_{\mathcal{A}}$ be a **lexicographical order** over $\mathcal{A}$. Let w and v be words over $\mathcal{A}$, it is said that w is *shortLex* than v, if w is shorter than v, i.e. $|w| < |v|$, or w and v have the same length but w comes before v in the order $<_{\mathcal{A}}$. The **language of the *shortLex* words** in $\mathcal{F}(\mathcal{A})$ representing a unique group element in $\mathcal{G}$ is given by

$$L = \{w \in \mathcal{F}(\mathcal{A}) : w <_{\mathcal{A}} v, \forall v \in \mathcal{F}(\mathcal{A}) \text{ s.t. } w =_{\mathcal{G}} v\}. \tag{3}$$

Language L gives a unique representation for the shortest paths in $\Gamma(\mathcal{G}, S)$. Continuing with $BS(4)$, let $a < b < c$ be a lexicographic order over the alphabet $\mathcal{A} = \{a, b, c\}$, then the language of the *shortLex* words of $BS(4)$ is

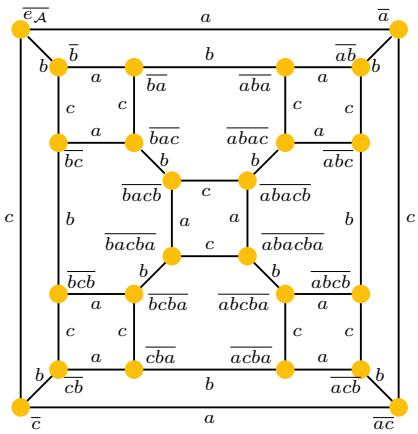


Fig. 2: $BS(4)$ with links and node labelled. Each Link is labelled with a letter in Eq. (2) representing its corresponding generator. Meanwhile, each node is labelled with a word in Eq. (4) representing the *shortLex* path from $\overline{e_{\mathcal{A}}}$ to it.

$$L = \{e_{\mathcal{A}}, a, b, c, ab, ac, ba, bc, cb, aba, abc, acb, bac, bcb, cba, abac, abcb, acba, bacb, bcba, abacb, abcba, bacba, abacba\}. \quad (4)$$

Equation (4) defines the shortest paths between each pair of nodes in $BS(4)$. These paths are called ***shortLex* paths** due to they are represented by the *shortLex* words. For instance, ac and ca represents paths between the same pair of nodes in $BS(4)$, e.g. 4213 and 2431. Although both paths are the shortest ones, the *shortLex* path is $ac \in L$ due to $ac <_{\mathcal{A}} ca$. Hereafter, a path represented by a word w will be denoted as $\widehat{w}$.

B. Words as nodes

In addition to paths, words in L also give a unique representation for nodes of its related CG[1]. This representation proceeds as follows. First, the symbol $e_{\mathcal{A}}$ is assigned to an arbitrary node in $V(\Gamma)$. Then, each of the remaining nodes is assigned to the word in L representing the *shortLex* path from $e_{\mathcal{A}}$ to it. From now on, a node represented by a word w will be denoted as $\overline{w}$. Figure 2 shows $BS(4)$, where nodes are labelled with its corresponding word in L, see Eq. (4).

C. Path computation algorithms

The word-processing approach presented in this section allows to compute paths in CGs as follows. Note that there is a path between every two nodes $\overline{u}$ and $\overline{v}$ given by $\widehat{u^{-1}v}$ [2]. From the definition of equivalent words, every word $w =_{\mathcal{G}} u^{-1}v$ represents a path between $\overline{u}$ and $\overline{v}$, see Section II-A. Therefore the problem of compute the paths between $\overline{u}$ and $\overline{v}$ is equivalent to compute words equivalent to $u^{-1}v$. Authors in [**?**] present a set of path computation algorithms that take as input the word $u^{-1}v$, and then compute the words equivalent to $u^{-1}v$ as in shown in Table I. The paths are computed in an ordered manner from the shortest to the largest one by using an automaton called the *Word-Difference Automaton* (WDA). The WDA encodes the topological structure of the CG and recognises words representing the same pair of nodes [**?**, Section 13.2.2].

TABLE I: Path computation algorithms, its computed words and paths [**?**].

	Computed words	**Paths represented by the computed words**
Alg. A	The *shortLex* word.	The shortest path.
Alg. B	The K-*shortLex* words.	The K-shortest paths.
Alg. C	The *shortLex* words without equivalent substrings to each other.	The shortest disjoint paths.
Alg. D	The *shortLex* word without substrings equivalent to a set of words.	The shortest path avoiding a set of nodes and links.

As is explained in the section below, the path computation algorithms presented in [**?**] are used in the WPR to compute the routing paths. The following lemma states the time complexity of these algorithms.

Lemma 1: The algorithms presented in [**?**] are able to compute the K-shortest paths, the shortest disjoint paths and the set of paths avoiding nodes and edges in time $O(K\ell|Diff|)$, where K denotes the number of computed paths, ℓ denotes the length of the largest computed path and $|Diff|$ denotes the size of the WDA. [**?**, Colloraries 5-7].

III. Overview of the WPR

The WPR is a routing scheme designed to work on networks whose topology is defined by Cayley Graphs (CGs). Table II summarises the basic notation used in the definition of the WPR. This section describes the initial configuration, general operation and properties of the WPR. The detailed operation and evaluation of the WPR is presented in the following sections.

A. Initial configuration

The information required in each node for maintaining the WPR consists of:

1) A routing table: It only consists of port labels. A port label is defined by a letter in the alphabet $\mathcal{A}$ representing its corresponding generator as stated Eq. (1).

2) A node label: It is given by the word in the language of the *shortLex* words, i.e. L, representing a path from $\overline{e_{\mathcal{A}}}$ to each node as stated Eq. (3).

3) A record of node and link failures: Each node keeps a record of failures close to it. Initially this record is given by an empty set that is updated after a near failure occurs.

4) The Word-difference Automaton (WDA): This automaton encodes the topological structure of its related CG and is used by the path computation algorithms.

[1]For every CG, $|V(\Gamma)| = |L|$.

[2]The path $\widehat{u^{-1}v}$ goes from $\overline{u}$ to $\overline{e_{\mathcal{A}}}$ and from $\overline{e_{\mathcal{A}}}$ to $\overline{v}$. Taking Fig. 2 as example, the path $\widehat{(bc)^{-1}ab} = \widehat{cbab}$ goes from $\overline{bc}$ to $\overline{e_{\mathcal{A}}}$ and from $\overline{e_{\mathcal{A}}}$ to $\overline{ab}$.

TABLE II: Notation of the Word-Processing-based Routing.

Parameter	Definition
$\Gamma(\mathcal{G}, S)$	Cayley graph of a group $\mathcal{G}$ with generating set S.
$V(\Gamma)$	Vertices of $\Gamma(\mathcal{G}, S)$.
$E(\Gamma)$	Links of $\Gamma(\mathcal{G}, S)$.
n	Number of nodes in $\Gamma(\mathcal{G}, S)$.
Δ	Regular degree of $\Gamma(\mathcal{G}, S)$.
D	Diameter of $\Gamma(\mathcal{G}, S)$.
$\mathcal{A}$, where $\lvert\mathcal{A}\rvert = \Delta$	A lexicographically ordered alphabet defining the set of port labels.
$\mathcal{F}(\mathcal{A})$	A language over $\mathcal{A}$ defining all the paths in $\Gamma(\mathcal{G}, S)$.
$L \subset \mathcal{F}(\mathcal{A})$	A language over $\mathcal{A}$ defining the node labels and the *shortLex* paths in $\Gamma(\mathcal{G}, S)$.
$\widehat{w}$, where $w \in \mathcal{F}(\mathcal{A})$	A path in $\Gamma(\mathcal{G}, S)$, where w denotes a sequence of output ports.
$\overline{w}$, where $w \in L$	A node in $V(\Gamma)$, where w denotes the node label and $\widehat{w}$ denotes the *shortLex* path from $\overline{e_{\mathcal{A}}}$ to $\overline{w}$.
$\widehat{w}(\overline{u})$	A path in $\Gamma(\mathcal{G}, S)$, where $\overline{u}$ denotes the initial node.
$(u, v) \in L \times L$	A link in $E(\Gamma)$, where $\overline{u}$ and $\overline{v}$ denote its incident links.
$\mathcal{V}_u \subset L$	Faulty nodes recorded in $\overline{u}$.
$\mathcal{E}_u \subset L \times L$	Faulty edges recorded in $\overline{u}$.

In the initial configuration of the WPR, the record of failures is initialized as an empty set and the WDA is copied in each node. In addition, ports and nodes must be labelled as follows.

- **Port labelling**. Ports of each node must be labelled with their corresponding letter in $\mathcal{A}$, which represents a generator in S, see Fig. 1b. If it is unknown which generator is associated to each link, then it is necessary to perform a process to match links to generators. Appendix **??** presents a process of port label assignment for CG whose links are not identifying with relators.
- **Node labelling**. Nodes must be labelled with words in L, where the label of a node represents the *shortLex* path from the node $\overline{e_{\mathcal{A}}}$ to said node, see Fig. 2. Algorithm 1 in [**?**] presents the steps for performing the node label assignment.

Figure 2 shows the graph $BS(4)$ after the process of port and node labelling. The following lemma states the space complexity of ports and node labels.

Lemma 2: A port label is given by a letter in $\mathcal{A}$, where $|\mathcal{A}| = \Delta$, thereby it can be represented by $\log(\Delta)$ bits. If every node is labelled with a word over $\mathcal{A}$ representing the *shortLex* path from the node $\overline{e_{\mathcal{A}}}$ to it. Then a node label consists of at most D letters in $\mathcal{A}$ and it can be represented by $O(D \log(\Delta))$ bits [**?**, Theorem 4].

B. General operation and properties

The WPR supports static routing in modes single-path and multi-path, and fault-tolerant routing in mode single-path. In static routing, the routing paths are computed by source nodes only once for each message. In fault-tolerant routing, nodes keep a partial record of failures, and then computes routing paths avoiding nearby failures.

The WPR allows the following properties:

1) Minimal routing: In spite of link and/or node failures, messages are routed always through minimal paths.

2) Fault-tolerance: In spite of link and/or node failures, the packet delivery is guaranteed.

3) Path diversity: The WPR provides multi-path routing exploiting the path diversity of CGs.

4) Complexity efficient: The forwarding algorithms have low time and space complexity.

IV. Static Routing

Static routing applies source routing in single and multi-path modes. The routing paths are computed by source nodes just once for each message. These paths are denoted by words in $\mathcal{F}(\mathcal{A})$ representing the sequence of output ports from the source node to the destination node. This section presents the configuration of message headers. Then the forwarding processes is explained. Finally, static routing examples for single and multi-path modes are given.

A. Message headers

Let $\overline{u}$ and $\overline{v}$ be two nodes, such that $\overline{u}$ wants to send a message `MSG` to $\overline{v}$ through the routing path $\widehat{w} = x_1 x_2, \ldots x_\ell$. The header of `MSG` is given by a word h representing the routing path from the next node in the path to $\overline{v}$, i.e. $h = x_2 \ldots x_\ell$. If the next node in the path is $\overline{v}$, i.e. $\widehat{w} = x_1$, then $h = e_{\mathcal{A}}$, where $e_{\mathcal{A}}$ denotes the empty path. The process of prepare a message header is presented in Algorithm 1.

Algorithm 1 Prepare a message header.

Input: A word $w = x_1 x_2 \ldots x_\ell \in \mathcal{F}(\mathcal{A})$ representing the routing path.
Output: A word $h \in \mathcal{F}(\mathcal{A})$ representing the header for a message that must be routed through the path $\widehat{w}$.

```
if ℓ = 1 then
    h ← e_A.
else
    h ← x_2 ... x_ℓ.
return h.
```

The following lemma states the space complexity of message headers.

Lemma 3: A message header defined by the WPR, in static routing, can be represented with $O(\ell \log(\Delta))$ bits, where ℓ is the length of the routing path. If the routing mode is single-path, then $\ell \leq D$ and a message is represented by $O(D \log(\Delta))$ bits. This lemma is analogous to Lemma 2

B. Forwarding

The forwarding process in source nodes proceeds as follows. In single-path forwarding, the source node computes the *shortLex* path to the destination node. In multi-path forwarding, the source node decides which paths will be computed depending on the routing requirements. These paths can be the K-shortest paths, the shortest link-disjoint paths or the shortest node-disjoint paths, see Table I.

In both modes, single and multiple-path forwarding, each computed path will be a routing path, which denotes a

sequence of output ports from the source to the destination node. As is explained in the above subsection, a message is associated to its routing path, where the first letter in the path denotes the output port and the remaining string denotes the message header.

Algorithms 2 and 3 present the steps for single-path and multi-path forwarding in source nodes.

Algorithm 2 Single-path forwarding in source nodes

Input: A message MSG.
Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
1: Compute the shortest path $\widehat{w}$ from $\overline{u}$ to $\overline{v}$, where $w = x_1x_2 \dots x_\ell$. {Table I - Alg. A}
2: Prepare a message header h. {Algorithm 1}
3: Attach h to MSG.
4: Set the output $p \leftarrow x_1$.
5: Forward MSG through p.

Lemma 4: The forwarding decision of static routing in single-path mode is taken by source nodes using Algorithm 2, which runs in time $O(D|Diff|)$ due to Lemma 1.

Algorithm 3 Multi-path forwarding in source nodes

Input: A set of messages MSG_1, MSG_2, ..., MSG_K.
Input: Label of the source node $u \in L$.
Input: Label of the destination node $v \in L$.
1: Compute a set of K paths $\widehat{w}_1, \widehat{w}_2, \dots \widehat{w}_K$ from $\overline{u}$ to $\overline{v}$, where $w_i = x_{i,1}x_{i,2} \dots x_{i,\ell_i}$. {Table I - Alg. B, C}
2: **for each** MSG_i **do**
3: Prepare a message header h_i. {Algorithm 1}
4: Attach each h_i to MSG_i.
5: Set the output port $p_i \leftarrow x_{i,1}$.
6: Forward MSG_i through p_i.

Lemma 5: The forwarding decision for static routing in multi-path mode is taken by source nodes using Algorithm 3, which runs in time $O(K\ell|Diff|)$, where the K denotes the number of computed paths and ℓ denotes the length of the largest of them. If the routing paths are disjoint, then $\ell \leq D$ and $K \approx D$, thereby the forwarding decision in source nodes is takes in time $O(\Delta D|Diff|)$. It follows from Lemma 1.

The forwarding process in intermediate nodes is the same for both modes, single-path and multi-path forwarding. This process consists in reading the header of the incoming message, where first letter denotes the output port and the remaining string denotes the new message header. Algorithm 4 presents the steps for forwarding in intermediate nodes.

Lemma 6: The forwarding decision for static routing in intermediate nodes is taken using Algorithm 4, which runs in time $O(1)$ due to the next node in the path is encoded in the message header and thus no path is computed.

C. Routing example: Static routing in single-path mode

Consider a network whose topology is given by the graph $BS(4)$, which is defined in Section II. To support the WPR, ports and nodes in $BS(4)$ must be labelled as is explained in Section III-A, Fig. 2 shows $\overline{BS(4)}$ after the labelling processes. Suppose now that node $\overline{bacba}$ wants to send a single message to node $\overline{a}$. The routing process is illustrated in Fig. 3 and proceeds as follows.

Algorithm 4 Forwarding in intermediate nodes

Input: A message MSG.
1: Read the header $h' = y_1y_2 \dots y_j$ from MSG.
2: **if** $h' = e_{\mathcal{A}}$ **then**
3: Finish.
4: **if** $j > 1$ **then**
5: Prepare a new header $h \leftarrow y_2 \dots y_j$.
6: **else**
7: Prepare a new header $h \leftarrow e_{\mathcal{A}}$.
8: Replace the old header h' by the new header h in MSG.
9: Set the output $p \leftarrow y_1$.
10: Forward MSG through p.

The source node $\overline{bacba}$ executes Algorithm 2. First, the routing path $\widehat{abacba}$ is computed, and then Algorithm 1 is executed to compute the upcoming message header, i.e. $bacba$. Finally, the message is forwarded through the port given by the first letter in the routing path, i.e. a. Thereby the message arrives to node $\overline{bacb}$, which executes Algorithm 4. The algorithm reads the incoming message header, i.e $bacba$, and selects the new upcoming message header $acba$ and output port b. Finally, message is forwarded to $\overline{bac}$. This process is repeated in all the intermediate nodes until the message reaches the destination node $\overline{a}$.

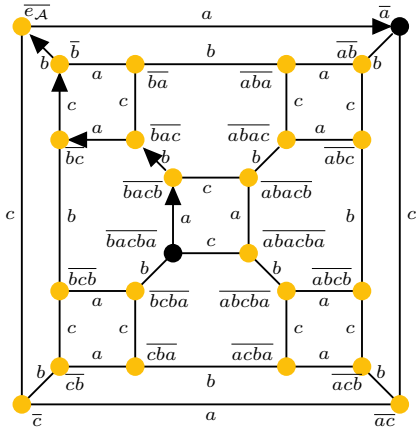
D. Routing example: Static routing in multi-path mode

Consider now the same network as in the above example and suppose that node $\overline{bacba}$ wants to send messages to node $\overline{a}$ through the shortest node-disjoint paths. The routing process is illustrated in Fig. 4 and proceeds as follows.

The source node $\overline{bacba}$ executes Algorithm 3. First, the routing paths $\widehat{abacba}$, $\widehat{bacbac}$ and $\widehat{cabacb}$ are computed. Then, for each message, Algorithm 1 is executed to compute the upcoming message header, i.e. $bacba$, $acbac$ and $abacb$ respectively. Finally, each message is forwarded through the port given by the first letter in its routing path, i.e. a, b and c, repectively. Thereby the messages arrives to nodes $\overline{bacb}$, $\overline{bcba}$ and $\overline{abacba}$. Each of these nodes executes Algorithm 4 which reads the incoming message header and selects the new upcoming message header and output port. Finally, message is forwarded to the output port. This process is repeated in all the intermediate nodes until the messages reach the destination node $\overline{a}$.

V. Fault-tolerant Routing

Fault-tolerant routing operates in single-path mode. To provide fault-tolerance, every node in the network keeps a record of nearby faulty nodes and links. The routing paths are computed by the source node and intermediate nodes having recorded failures. To update their failure records, nodes must be notified about nearby nodes and links that failed or recovered. This section presents the processes of failure records



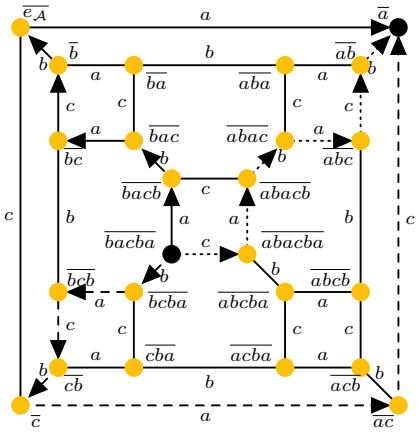
(a) The routing path computed between nodes $\overline{bacba}$ and $\overline{a}$ is $\widehat{abacba}$, which corresponds to the shortest path.

Routing path: $\widehat{abacba}$

Forwarding node	Incoming message header	Upcoming message header	Output port	Next node
$\overline{bacba}$	–	$bacba$	a	$\overline{bacb}$
$\overline{bacb}$	$bacba$	$acba$	b	$\overline{bac}$
$\overline{bac}$	$acba$	cba	a	$\overline{bc}$
$\overline{bc}$	cba	ba	c	$\overline{b}$
$\overline{b}$	ba	a	b	$\overline{e_{\mathcal{A}}}$
$\overline{e_{\mathcal{A}}}$	a	$e_{\mathcal{A}}$	a	$\overline{a}$

(b) Message headers and output ports selected by the forwarding nodes. These headers and ports are determined by the routing path $\widehat{abacba}$ that is computed in the source node $\overline{bacba}$.

Fig. 3: Static routing in mode single-path for $BS(4)$ and between nodes $\overline{bacba}$ and $\overline{a}$. The routing path is computed just by the source node.



(a) The routing paths computed between nodes $\overline{bacba}$ and $\overline{a}$ are $\widehat{abacba}$, $\widehat{bacbac}$ and $\widehat{cabacb}$, which correspond to the shortest node-disjoint paths.

Routing path: $\widehat{bacbac}$

Forwarding node	Output port	Next node
$\overline{bacba}$	b	$\overline{bcba}$
$\overline{bcba}$	a	$\overline{bcb}$
$\overline{bcb}$	c	$\overline{cb}$
$\overline{cb}$	b	$\overline{c}$
$\overline{c}$	a	$\overline{ac}$
$\overline{ac}$	c	$\overline{a}$

Routing path: $\widehat{cabacb}$

Forwarding node	Output port	Next node
$\overline{bacba}$	c	$\overline{abacba}$
$\overline{abacba}$	a	$\overline{abacb}$
$\overline{abacb}$	b	$\overline{abac}$
$\overline{abac}$	a	$\overline{abc}$
$\overline{abc}$	c	$\overline{ab}$
$\overline{ab}$	b	$\overline{a}$

(b) Output ports selected by the forwarding nodes. These ports are determined by the routing paths $\widehat{bacba}$ and $\widehat{cabacb}$ that are computed in the source node $\overline{bacba}$. The output ports for $\widehat{abacba}$ are presented in Fig. 3b

Fig. 4: Static routing in mode multi-path for $BS(4)$ and between nodes $\overline{bacba}$ and $\overline{a}$. The routing paths are computed just by the source node.

update, failure notification and recovery notification. Then the forwarding processes is explained. In addition, examples of failure notification and fault-tolerant routing are given.

A. Failure record update

This section focuses on the criterion to determine whether a node u (notified of a failure) must update its record of node or link failures, i.e. $\mathcal{V}_u$ and $\mathcal{E}_u$ respectively. The processes of failure and recovery notification are presented in the following subsections.

Algorithm 5 presents the steps to determine whether a node $\overline{u}$ notified about the failure of a node $\overline{f}$ must update their failure records. If so, the algorithm returns *True*, otherwise, it returns *False*. First, it is checked if $\overline{f}$ is already registered. If so, the algorithm finishes (lines 1-2). Otherwise, it searched for links in $\mathcal{E}_u$ that are adjacent to $\overline{f}$. If some links are founded, they are removed from $\mathcal{E}_u$ and $\overline{f}$ is recorded in $\mathcal{V}_u$, then the algorithm finishes (lines 3-7). Otherwise, the algorithm proceeds as follows. For each node $\overline{v}$ adjacent to $\overline{f}$ that is not recorded (for loop at line 9), the following paths from $\overline{u}$ to $\overline{v}$ are computed:

- The *shortLex* path avoiding both the failed node $\overline{f}$ and the failures recorded in $\mathcal{V}_u$ and $\mathcal{E}_u$, said path is denoted by $\widehat{w}_1$ (line 9).
- The *shortLex* path avoiding just the failures recorded in $\mathcal{V}_u$ and $\mathcal{E}_u$, said path is denoted by $\overline{u}$ (line 10).

The failed node $\overline{f}$ is recorded in $\mathcal{V}_u$ if some of the following conditions is *True* (lines 11-13):

1) There is no path between $\overline{u}$ to $\overline{v}$, i.e. $w_1 = Null$.
2) The next nodes in the *shortLex* paths that avoid and do not avoid $\overline{f}$ are different, i.e. $w_1(1) \neq w_2(1)$.

Algorithm 5 Update the faulty nodes record $\mathcal{V}_u$ with the faulty node $\overline{f}$.

Input: Label of node $u \in L$.
Input: Label of the faulty node $f \in L$.
Input: Faulty nodes record $\mathcal{V}_u \subset L$.
Input: Faulty links record $\mathcal{E}_u \subset L \times L$.
1: **if** $f \in \mathcal{V}_u$ **then**
2: **return** False.
3: $\mathcal{E}'_u \leftarrow \{(d,e) \in \mathcal{E}_u : f \in (d,e)\}$.
4: **if** $\mathcal{E}'_u \neq \emptyset$ **then**
5: $\mathcal{E}_u \leftarrow \mathcal{E}_u \setminus \mathcal{E}'_u$.
6: Add f to $\mathcal{V}_u$.
7: **return** True.
8: **for** $v \in \{f \cdot x : x \in \mathcal{A}\} \setminus \mathcal{V}_u$ **do**
9: $w_1 \leftarrow$ the *shortLex* path from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u \cup \{f\}$ and links in $\mathcal{E}_u$. {Table I - Alg. D}
10: $w_2 \leftarrow$ the *shortLex* path from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u \setminus \{f\}$ and links in $\mathcal{E}_u$. {Table I - Alg. D}
11: **if** $(w_1 = \textit{Null})$ OR $(w_1(1) \neq w_2(1))$ **then**
12: Add f to $\mathcal{V}_u$.
13: **return** *True*.
14: **return** *False*.

Algorithm 6 Update the faulty links record $\mathcal{E}_u$ with the faulty link (e,f).

Input: Label of node $u \in L$.
Input: Label of the faulty link $(e,f) \in L \times L$.
Input: Faulty nodes record $\mathcal{V}_u \subset L$.
Input: Faulty links record $\mathcal{E}_u \subset L \times L$.
1: **if** $(e,f) \in \mathcal{E}_u$ **then**
2: **return** False.
3: **for** $v \in (e,f)$ **do**
4: $\mathcal{E}'_u \leftarrow \{(d,e) \in \mathcal{E}_u : d = v$ OR $e = v\}$.
5: **if** $|\mathcal{E}'_u| = \Delta - 1$ **then**
6: $\mathcal{E}_u \leftarrow \mathcal{E}_u \setminus \mathcal{E}'_u$.
7: Add v to $\mathcal{V}_u$.
8: **return** True.
9: **for** $v \in \{e,f\} \setminus \mathcal{V}_u$ **do**
10: $w_1 \leftarrow$ the *shortLex* path from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links in $\mathcal{E}_u \cup \{(e,f)\}$. {Table I - Alg. D}
11: $w_2 \leftarrow$ the *shortLex* path from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links in $\mathcal{E}_u \setminus \{(e,f)\}$. {Table I - Alg. D}
12: **if** $(w_1 = \textit{Null})$ OR $(w_1(1) \neq w_2(1)$ **then**
13: Add (e,f) to $\mathcal{E}_u$.
14: **return** *True*.

Similarly to Algorithm 7, Algorithm 8 determines whether a node $\overline{u}$ notified about the failure of a link (e,f) must update their failure records. Notice that in the event of node or link failure, every node $\overline{u}$ will update their failure records if after failure: 1) the network is disconnected, or 2) there are destinations $\overline{v}$, such that the next node in the path from $\overline{u}$ to $\overline{v}$ have changed.

Lemma 7: When a node $\overline{u}$ is notified about the failure of a node $\overline{f}$ or link (e,f), it determines whether to update or not their failure records using algorithms 5 or 6, which run in time $O(K\ell|Diff|)$, where K denotes the number of computed paths and ℓ denotes then length of the largest of them. From Lemma 5, $\ell \leq D$ and $K \approx D$, thereby Algorithms 5 and 6 run in time $O(\Delta D|Diff|)$. It follows of Lemma 1.

B. Failure notification

The notification processes of faulty nodes and links are presented in algorithms 7 and 8 respectively. In both cases, the failure notification begins at each node $\overline{z}$ adjacent to the faulty node or incident to the faulty link. First, the label of the faulty node or link is computed by concatenating the label of node $\overline{z}$ and the port connected to the faulty element. Then, the resulting label is added to the corresponding record of failures. Finally, whether the failure corresponds to a node or link, a message `FAULTY_NODE` or `FAULTY_LINK` is sent through the active ports of $\overline{z}$ except for the one connected to the faulty element. This message contains the label of the faulty element and the failure records of $\overline{z}$.

Algorithm 7 Notification of a faulty node.

Upon a node $\overline{z}$ realises that the node connected through its port x has failed, node $\overline{z}$ **does**:
1: $f \leftarrow$ the canonical form of $z \cdot x$.
2: Add f to $\mathcal{V}_z$.
3: Send out the message `FAULTY_NODE`$(f, \mathcal{V}_z, \mathcal{E}_z)$ through its active ports.

Every node $\overline{u}$ receiving a message `FAULTY_NODE`$(f, \mathcal{V}_v, \mathcal{E}_v)$ through its port x **does**:
1: $update \leftarrow$ Update $\mathcal{V}_u$ with f. {Algorithm 5}
2: **if** $update$ **then**
3: **for** $f \in \mathcal{V}_v \setminus \mathcal{V}_u$ **do**
4: Update $\mathcal{V}_u$ with f. {Algorithm 5}
5: **for** $(e,f) \in \mathcal{E}_v \setminus \mathcal{E}_u$ **do**
6: Update $\mathcal{E}_u$ with (e,f). {Algorithm 6}
7: Send out a message `FAULTY_NODE`$(f, \mathcal{V}_u, \mathcal{E}_u)$ through its active ports except x.

Algorithm 8 Notification of a faulty link.

Upon a node $\overline{z}$ realises that that the link associated to its port x has failed, node $\overline{z}$ **does**:
1: $f \leftarrow$ the canonical form of $z \cdot x$.
2: Add (z,f) to $\mathcal{E}_z$.
3: Send out the message `FAULTY_LINK`$((z,f), \mathcal{V}_z, \mathcal{E}_z)$ through its active ports.

Every node $\overline{u}$ receiving a message `FAULTY_LINK`$((e,f), \mathcal{V}_v, \mathcal{E}_v)$ through its port x **does**:
1: $update \leftarrow$ update $\mathcal{E}_u$ with (v,f) {Algorithm 6}.
2: **if** $update$ **then**
3: **for** $f \in \mathcal{V}_v \setminus \mathcal{V}_u$ **do**
4: Update $\mathcal{V}_u$ with f. {Algorithm 5}
5: **for** $(v,f) \in \mathcal{E}_v \setminus \mathcal{E}_u$ **do**
6: Update $\mathcal{E}_u$ with (e,f). {Algorithm 6}
7: Send out a message `FAULTY_LINK`$((e,f), \mathcal{V}_u, \mathcal{E}_u)$ through its active ports except x.

TABLE III: Paths computed during the failure notification of $\overline{bac}$ in $BS(4)$. If $\widehat{w_1}$ and $\widehat{w_2}$ begin at different letters (see red letters), then $\overline{bac}$ is recorded by the notified node.

Notified node	**Nodes adjacent to $\overline{bac}$**					
	$\overline{ba}$		$\overline{bc}$		$\overline{bacb}$	
$\overline{u}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$
$\overline{aba}$	$\widehat{b}$	$\widehat{b}$	$\widehat{bac}$	$\widehat{bac}$	$\widehat{bcb}$	$\widehat{cbc}$
$\overline{b}$	$\widehat{a}$	$\widehat{a}$	$\widehat{c}$	$\widehat{c}$	$\widehat{acb}$	$\widehat{abcbc}$
$\overline{bcb}$	$\widehat{bac}$	$\widehat{bca}$	$\widehat{b}$	$\widehat{b}$	$\widehat{aba}$	$\widehat{aba}$
$\overline{abacb}$	$\widehat{bcb}$	$\widehat{bcb}$	$\widehat{cba}$	$\widehat{bcbac}$	$\widehat{c}$	$\widehat{c}$
$\overline{bacba}$	$\widehat{abc}$	$\widehat{babca}$	$\widehat{aba}$	$\widehat{bab}$	$\widehat{a}$	$\widehat{a}$
$\overline{ab}$	$\widehat{ab}$	$\widehat{ab}$	$\widehat{abac}$	$\widehat{abac}$	$\widehat{abcb}$	$\widehat{acbc}$
$\overline{abac}$	$\widehat{cb}$	$\widehat{cb}$	$\widehat{bcba}$	$\widehat{cbac}$	$\widehat{bc}$	$\widehat{bc}$
$\overline{abacba}$	$\widehat{abcb}$	$\widehat{abcb}$	$\widehat{acba}$	$\widehat{cbab}$	$\widehat{ac}$	$\widehat{ac}$
$\overline{bcba}$	$\widehat{abac}$	$\widehat{abca}$	$\widehat{ab}$	$\widehat{ab}$	$\widehat{bc}$	$\widehat{bc}$
$\overline{abc}$	$\widehat{acb}$	$\widehat{acb}$	$\widehat{abcba}$	$\widehat{acbac}$	$\widehat{abc}$	$\widehat{abc}$
$\overline{abcba}$	$\widehat{abacb}$	$\widehat{abacb}$	$\widehat{bacba}$	$\widehat{bcbab}$	$\widehat{bac}$	$\widehat{bac}$

Each node $\overline{u}$ receiving a failure notification message (from a node $\overline{v}$) decides whether to update their failure records using Algorithm 5 or Algorithm 6 depending on the kind of message. If the failure records are not updated, node $\overline{u}$ ends its process of failure notification. Otherwise, for each failure recorded in $\overline{v}$ but not in $\overline{u}$, the corresponding faulty record is updated, and a new message of failure notification is sent through the active ports of $\overline{u}$ except for the one connected to $\overline{v}$.

C. Failure notification example

This section presents an example of failure notifications in $BS(4)$. It is assumed that node $\overline{bac}$ fails first and then link $(aba, abac)$ fails. Figure 5 illustrates the notification process in $BS(4)$ when node $\overline{bac}$ fails. This process ends at 3 hops from $\overline{bac}$. Figures 5a-c illustrate the notification process at 1, 2, and 3 hops from $\overline{bac}$. It is important to note that nodes should record the last notification received to ignore new notifications about the same failure. Figure 5d shows the final state of the network, where the blue nodes have updated their failures record. Figure 5e summarises the information of each failure notification. Recall that notified nodes decide whether to update their failure records by comparing different computed paths (Algorithm 5). Table III shows the paths that are computed by each notified node and indicates whether the faulty node must be recorded.

Similarly to Fig. 5, Fig. 6 illustrates the notification process when link $(aba, abac)$ fails in $BS(4)$, where node $\overline{bac}$ has already failed. Thereby the initial state of the network is shown in Fig. 5d, where nodes $\overline{ba}$. $\overline{bc}$, $\overline{aba}$, $\overline{abac}$, $\overline{bacb}$, $\overline{abacb}$, $\overline{bacba}$ and $\overline{abacba}$ have added $\overline{bac}$ to their failure records. The notification process of $(aba, abac)$ ends at 4 hops from it. Figures 6a-d illustrate the notification process at 1, 2, 3 and 4 hops from $\overline{bac}$. For each failure notification, Figure 6e summarises the information of failure notifications and Table IV shows the paths that are computed by each notified node. Finally, Figure 7 shows the final state of the network and the updated failure records after node $\overline{bac}$ and link $(aba, abac)$ had failed.

D. Recovery notification

Algorithm 9 Notification of a recovered node.

Upon a node $\overline{z}$ realises that the node connected through its port x has recovered, node $\overline{z}$ **does**:

1: $r \leftarrow$ the canonical form of $z \cdot x$.
2: Remove r from $\mathcal{V}_z$.
3: Send out the message RECOVERED_NODE(r) through its active ports except x.

Every node $\overline{u}$ receiving a message RECOVERED_NODE(r) through its port x **does**:

1: **if** $r \in \mathcal{V}_u$ **then**
2: Remove r from $\mathcal{V}_u$.
3: Send out the message RECOVERED_NODE(r) through its active ports except x.

Algorithm 10 Notification of a recovered link.

Upon a node $\overline{z}$ realises that the link associated to its port x has failed, node $\overline{z}$ **does**:

1: $t \leftarrow$ the canonical form of $z \cdot x$.
2: Remove (z, t) from $\mathcal{E}_z$.
3: Send out the message RECOVERED_LINK(z, t) through its active ports except x.

Every node $\overline{u}$ receiving a message RECOVERED_LINK(r, t) through its port x **does**:

1: **if** $(r, t) \in \mathcal{E}_u$ **then**
2: Remove (r, t) from $\mathcal{E}_u$.
3: Send out the message RECOVERED_LINK(r, t) through its active ports except x.

The notification processes of nodes and links recovered from a failure are presented in Algorithm 9 and Algorithm 10. Similarly to a failure notification, a recovery notification is initiated in each node $\overline{z}$ adjacent to the recovered node or incident to the recovered link. First, the label of the recovered node or link is computed and removed from the corresponding record of failures. Then, whether the recovered element is a node or link, a message RECOVERED_NODE or RECOVERED_LINK is sent through the active ports of $\overline{z}$ except for the one connected to the recovered element. This message contains the label of the recovered element. Each node $\overline{u}$ receiving a recovery notification message (from a node $\overline{v}$) checks if the recovered element is in its record of failures. If not, node $\overline{u}$ ends the process of recovery notification. Otherwise, the label of the recovered element is removed from the corresponding record of failures, and a new recovery notification is sent through the active ports of $\overline{u}$ except for the one connected to $\overline{v}$.

E. Forwarding

The forwarding process in source nodes is presented in Algorithm 11. First, it is checked if the source node has no recorded failures. If so, it computes the *shortLex* path to

TABLE IV: Paths computed during the failure notification of $(aba, abac)$ in $BS(4)$ after $\widehat{bac}$ had failed. If $\widehat{w_1}$ and $\widehat{w_2}$ begin at different letters (see red letters), then $(aba, abac)$ and/or $\overline{bac}$ are recorded by the notified node.

Notified node	**Nodes incident to** $(aba, abac)$				**Nodes adjacent to** $\overline{bac}$					
	$\overline{aba}$		$\overline{abac}$		$\overline{ba}$		$\overline{bc}$		$\overline{bacb}$	
$\overline{u}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$	$\widehat{w_1}$	$\widehat{w_2}$
$\overline{ba}$	$\widehat{b}$	$\widehat{b}$	$\widehat{bc}$	$\widehat{baca}$	N / A	N / A	N / A	N / A	N / A	N / A
$\overline{ab}$	$\widehat{a}$	$\widehat{a}$	$\widehat{ac}$	$\widehat{ca}$	$\widehat{ab}$	$\widehat{ab}$	$\widehat{abac}$	$\widehat{abac}$	$\widehat{abcb}$	$\widehat{cabc}$
$\overline{abacb}$	$\widehat{bc}$	$\widehat{baca}$	$\widehat{b}$	$\widehat{b}$	N / A	N / A	N / A	N / A	N / A	N / A
$\overline{abc}$	$\widehat{ac}$	$\widehat{ca}$	$\widehat{a}$	$\widehat{a}$	$\widehat{cab}$	$\widehat{cab}$	$\widehat{abcba}$	$\widehat{cabac}$	$\widehat{abc}$	$\widehat{abc}$
$\overline{a}$	$\widehat{ba}$	$\widehat{ba}$	$\widehat{bac}$	$\widehat{bca}$	$\widehat{aba}$	$\widehat{aba}$	$\widehat{abc}$	$\widehat{abc}$	$\widehat{abacb}$	$\widehat{bacbc}$
$\overline{abcb}$	$\widehat{bac}$	$\widehat{bca}$	$\widehat{ba}$	$\widehat{ba}$	$\widehat{bacb}$	$\widehat{bcab}$	$\widehat{abacba}$	$\widehat{abcbab}$	$\widehat{abac}$	$\widehat{abac}$
$\overline{e_{\mathcal{A}}}$	N / A	N / A	N / A	N / A	$\widehat{ba}$	$\widehat{ba}$	$\widehat{bc}$	$\widehat{bc}$	$\widehat{bacb}$	$\widehat{abacbc}$
$\overline{ac}$	N / A	N / A	N / A	N / A	$\widehat{acba}$	$\widehat{acba}$	$\widehat{abcb}$	$\widehat{abcb}$	$\widehat{abacba}$	$\widehat{abacba}$
$\overline{b}$	N / A	N / A	N / A	N / A	$\widehat{a}$	$\widehat{a}$	$\widehat{c}$	$\widehat{c}$	$\widehat{acb}$	$\widehat{abcbc}$
$\overline{c}$	N / A	N / A	N / A	N / A	$\widehat{cba}$	$\widehat{cba}$	$\widehat{bcb}$	$\widehat{bcb}$	$\widehat{bacba}$	$\widehat{bacba}$

the destination node (lines 1-2). Otherwise, it computes the *shortLex* path to the destination node avoiding the recorded failures (lines 3-4). If there is no path avoiding failures, the forwarding process finishes (lines 5-6). Otherwise, the computed path will be the routing path, where the first letter in the path denotes the output port (lines 7-10).

Algorithm 11 Fault-tolerant forwarding in source nodes

Input: A message `MSG`.
Input: A word $u \in L$ representing the source node label.
Input: A word $v \in L$ representing the destination node label.
Input: A set $\mathcal{V}_u \subset L$ representing the faulty nodes record of node $\overline{u}$.
Input: A set $\mathcal{E}_u \subset L \times L$ representing the faulty links record of node $\overline{u}$.
1: **if** $\mathcal{V}_u = \emptyset$ and $\mathcal{E}_u = \emptyset$ **then**
2: Compute the shortest path $\widehat{w}$ from $\overline{u}$ to $\overline{v}$, where $w = x_1 x_2 \dots x_\ell$. {Table I - Alg. 1}
3: **else**
4: Compute the shortest path $\widehat{w}$ from $\overline{u}$ to $\overline{v}$ avoiding nodes in $\mathcal{V}_u$ and links $\mathcal{E}_u$, where $w = x_1 x_2 \dots x_l$ {Table I - Alg. 4}.
5: **if** $w = Null$ **then**
6: Finish.
7: Prepare a message header h. {Algorithm 1}
8: Attach h to `MSG`.
9: Set the output $p \leftarrow x_1$.
10: Forward `MSG` through p.

The forwarding process in intermediate nodes is presented in Algorithm 12. As in source nodes, it is checked if the intermediate node has no recorded failures. If so, it proceeds as forwarding in intermediate nodes for static routing (lines 1-2). Otherwise, the incoming message's header is read (line 4). If the message has reached the destination node, i.e. $h' = e_{\mathcal{A}}$, then the forwarding process finishes (lines 5-6). Otherwise, it proceeds as forwarding in source nodes for fault-tolerant routing (lines 7-8).

Algorithm 12 Fault-tolerant forwarding in intermediate nodes

Input: A message `MSG`.
Input: A word $u \in L$ representing the intermediate node label.
Input: A set $\mathcal{V}_u \subset L$ representing the faulty nodes record of node $\overline{u}$.
Input: A set $\mathcal{E}_u \subset L \times L$ representing the faulty links record of node $\overline{u}$.
1: **if** $\mathcal{V}_u = \emptyset$ or $\mathcal{E}_u = \emptyset$ **then**
2: Forward `MSG` as static routing in intermediate nodes. {Algorithm 4}
3: **else**
4: Read the header $h' = y_1 y_2 \dots y_j$ from `MSG`.
5: **if** $h' = e_{\mathcal{A}}$ **then**
6: Finish.
7: **else**
8: Forward `MSG` as fault-tolerant routing in source nodes. {Algorithm 11}

Lemma 8: The forwarding decision of fault-tolerant routing is taken by source and intermediate nodes using algorithms 11 and 12 respectively, which run in time $O(K\ell|Diff|)$, where the K denotes the number of computed paths and ℓ denotes the length of the largest of them. From Lemma 5, $\ell \leq D$ and $K \approx D$, thereby the forwarding decision in fault-tolerant routing is taken in time $O(\Delta D|Diff|)$. It follows from Lemma 1.

F. Routing example

Consider a network whose topology is given by the graph $BS(4)$, where node $\overline{bac}$ and link $(aba, abac)$ have failed. To support the WPR, nodes must execute the failure notification processes as is explained in sections V-B and V-C. Figure 7 shows the final state of the network $BS(4)$ and the updated failure records after node $\overline{bac}$ and link $(aba, abac)$ have failed. Suppose now that node $\overline{bacba}$ wants to send a single message to node $\overline{a}$. The routing process is illustrated in Fig. 8 and proceeds as follows.

The source node $\overline{bacba}$ executes Algorithm 11. First, the routing path avoiding recorded failures is computed, i.e. $\widehat{abacba}$. Therefore, the upcoming message is forwarded through port a and its header is $bacba$. Thereby the message arrives to node $\overline{bacb}$, which executes Algorithm 12. Since node $\overline{bacb}$ has recorded failures and it is not the destination node, the forwarding process proceeds as fault-tolerant routing in source nodes. Finally, message is forwarded to $\overline{abacb}$. This process is repeated in all the intermediate nodes until the message reaches the destination node $\overline{a}$. Notice that the routing path is computed by each intermediate node due to these nodes have recorded failures.

VI. Comparison with related work

This section summarizes and compares the main proposals of generic routing for CGs. The routing schemes analyzed are the Routing based on Permutation Sort (RPS) [**?**], the Routing based on Chordal Ring Representation (RCRR) [**?**] and the Geometric Routing with Word-Metric Spaces (GRWMS) [**?**], which is extended and enhanced in this work. The comparison between the routing schemes was made according to their complexity of memory space requirements and forwarding decision time.

A. Generic routing schemes for Cayley graphs

1) Routing based on permutation sort (RPS): This scheme is supported on the Cayley's theorem, which states that every group is isomorphic to a subgroup of the symmetric group Sym_p. Then nodes in a CG of Sym_p, are labeled with arrays of the set $\{1, \dots p\}$ representing elements of Sym_p (see Section II). Therefore, a node label can be represented by $O(p \log(p))$ bits. The connection rules between nodes are given by the set of permutations representing the generating set, thus the shortest path problem being equivalent to finding an optimal set of permutations that lead from a source node to a destination node.

This scheme follows a greedy routing strategy, where message headers are given by the label of the destination node, and the routing table of a node consists of the labels of its adjacent nodes. Therefore, the size of message headers and routing tables is $O(p \log(p))$ and $O(\Delta p \log(p))$ respectively. To forward packets, nodes compute the distance (in terms of the number of permutations) from each of its adjacent nodes to the destination node. Then, packets are forwarded to the nearest node to the destination node. Assuming that comparing two node labels takes a total amount of time proportional to the label size, the forwarding decision takes $O(\Delta plog(p))$ time units.

The RPS provides shortest-path routing, however, it does not provide multi-path routing and thus also does not provide fault-tolerance. As examples of this scheme, it can mention the schemes designed to work on Pancake, Star [**?**], Hypercube and Butterfly graphs [**?**] respectively.

2) Routing based on chordal ring representations (RCRR): The RCRR takes as key idea that CG of Borel subgroups, also called Borel CG [**?**], [**?**], [**?**], [**?**], can be represented by Generalized Chordal Rings (GCR) [**?**]. In a GCR, nodes can be labeled with integers from 0 to $n-1$, and there is a divisor q of n such that every two nodes i and j are connected if node $i+q$ module n is connected to node $j+q$ module n. In the RCRR, nodes of a Borel CG are mapped to nodes of a GCR, then node labels are presented by $O(\log(n))$ bits. The static routing tables are constructed following the connection rules of the GCR as follows. The routing table of a node i consists of an entry of Δ bits for each destination node and it is represented by $O(n\Delta)$. The j-th entry indicates the links that lead to the minimal paths from i to j. The forwarding decision for single-path routing is taken identifying outgoing links for any incoming message, whose header consists of the destination label. It takes $O(1)$ time units.

This routing scheme was extended in [**?**] to support link failures. In the new version, nodes incorporate a dynamic routing table that indicates the distances to destination nodes taking into account link failures. Therefore, a routing table is represented by $O(n \log(D))$. The routing process consists of two stages, the first one proceeds as explained above. The second stage begins when a node can no longer forward a packet due to failures. Then the packet is returned to the source node, which forwards it to links that lead to paths without link failures. Hence the forwarding decision takes $O(D)$ time units. In this case, message headers consists of the source and destination labels and the path history of nodes traversed by the packet being routed, thus it can be represented by $O(D \log(n))$ bits.

The scheme was evaluated through computer simulations on a Borel CG of 1081 nodes in a random failure scenario, where the percentage of link failures increases from 5% to 35% [**?**]. Results show that the scheme is not shortest path and packet delivery is not guaranteed in the range of 1% to 32% of all pairs of source-destination nodes.

3) Geometric routing with word-metric spaces (GRWMS): The GRWMS applies techniques of word processing in groups presented in Section II. Thus node labels are given by words represented by $O(D \log(\Delta))$ bits. The GRWMS follows a greedy routing strategy similar to the RPS, where message headers consist of the label of the destination node and the routing table of a node consists of the labels of its adjacent nodes. To forward packets, nodes computes the shortest paths from each of its adjacent nodes to the destination node. Then, packets are forwarded to the nearest node to the destination. The shortest paths are computed as it is explained in Section II-C, thereby the forwarding decisions are taken in time $O(\Delta D|Diff|)$ and the space complexity is $O(|Diff|)$. As the RPS, the GRWMS provides shortest-path routing but it does not provide multi-path routing and fault-tolerance.

B. Comparison of routing schemes for Cayley graphs

Table V summarizes the features of the WPR and the routing schemes mentioned above. As is shown, the WPR is the only scheme that provides multi-path routing and thus fault-tolerance while packet delivery is guaranteed. Note that the RCRR was proposed as a fault-tolerant scheme [**?**], although it does not guarantee packet delivery in failures scenarios. Thus the WPR is the most robust routing scheme among the analyzed ones.

TABLE V: Features of generic routing schemes for CGs.

Routing scheme	Shortest-path	Multi-path	Fault-tolerance
WPR	Yes	Yes	Yes
GRWMS	Yes	No	No
RCRR	No	No	Yes
RPS	Yes	No	No

Table VI presents the complexity of memory space requirements and forwarding decision time for the analyzed schemes. The memory space requirements includes the size of message headers, node labels, routing tables and extra information that would be required in the routing process. The RCRR has the highest memory space requirements as it requires full routing tables for static routing. While for fault-tolerant routing, it uses dynamic routing tables and adds the path history in the message header.

The memory space requirements for the rest of the analysed schemes depends on the CG family. In the case of RPS, its requirements depend on the symmetric group parameter, i.e. p, whose value varies in many families of CG used as model of communication networks. For instance in Hypercubes and Bubble-sort graphs [**?**], the value of p keeps low, i.e. $p = O(\log(n))$. Meanwhile, the value p is high for grid topologies, such as 2D-Torus graphs, where $p = O(\sqrt{n})$. As result, the RPS has the lowest space complexity in Hypercubes and Bubble-sort graphs and the highest space complexity in 2D-Torus graphs. Note that this scheme not require extra routing information due to it does not provide fault-tolerance.

In the case of the GRWMS and the WPR, both schemes use the same message headers and routing information, except by the failure record used by the WPR to provide fault-tolerance. These schemes use the WDA, whose size, i.e. $|Diff|$, depends on the CG's family. In the worst case $|Diff| = O(\Delta n)$, however, $|Diff|$ keeps low for most families of CG used as model of communication networks. For instance, $|Diff| = O(\Delta)$ for Hypercubes and Bubble-sort graphs [**?**], while $|Diff| = O(1)$ for 2D-Torus graphs.

Turning now to the complexity of the forwarding decision time, Table VI shows the values for static routing (in modes single and multi-path) and fault-tolerant routing. Regarding to static routing in single-path mode, the RCRR takes forwarding decisions in constant time due to this scheme uses full routing tables. The RPS and GRWMS take forwarding decisions in time proportional to p and $|Diff|$ respectively. Therefore, the GRWMS takes forwarding decisions faster than the RPS in Hypercubes, Bubble-sort and 2D-Torus graphs. Although the GRWMS and the WPR uses the same information for static routing, the WPR is faster than the GRWMS due to the WPR computes the routing paths just in source nodes, hence the forwarding decision time is constant in intermediate nodes. Meanwhile, the GRWMS computes the routing paths in source and intermediate nodes. In addition, the WPR is the only scheme, among the analyzed ones, that is able to perform static routing in mode multi-path.

Regarding to the schemes that provides fault-tolerance, which are the WPR and the RCRR, the RCRR has the lowest forwarding decision time complexity that is proportional to D. However, it is important to recall that this scheme does not ensure the packet delivery in failure scenarios. In contrast, the forwarding decision of the WPR are taken in the same time for multi-path routing and fault-tolerant routing. In both cases, the packet delivery is guaranteed.

VII. Conclusions

This paper introduces the WPR, a generic routing scheme for CGs, which guarantees packet delivery and provides: minimal routing, path diversity and fault-tolerance. As far as these authors known, the WPR is the first routing scheme with these features. The core of the WPR is a novel mechanism of fault-tolerance, which support multiple failures and provides minimal routing in spite of nodes do not keep a global record of failures. Through a complexity analysis, it was shown that the WPR stays competitive with respect to the state of the art on generic routing in CGs.

There are several future research lines on which this work can be taken as a base. The future work can be gathered in three main aspects: 1) designing new network architectures based on CGs and the WPR; 2) deploying the WPR in wireless sensor networks and; 3) applying the *word-processing approach* to design distributed algorithms for CGs, which solve problems different from the routing problem, e.g. consensus, leader election etc.

Acknowledgment

The authors would like to thank...

References

[1] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 63–74. [Online]. Available: https://doi.org/10.1145/1592568.1592577

[2] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 75–86. [Online]. Available: https://doi.org/10.1145/1402958.1402968

[3] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, 1985.

[4] M. Besta, J. Domke, M. Schneider, M. Konieczny, S. D. Girolamo, T. Schneider, A. Singla, and T. Hoefler, "High-performance routing with multipathing and path diversity in ethernet and hpc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 943–959, 2021.

[5] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, Q. Dong, and Y. Liu, "Toward a scalable, fault-tolerant, high-performance optical data center architecture," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2281–2294, 2017.

[6] S. Zhou, "A class of arc-transitive cayley graphs as models for interconnection networks," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 2, pp. 694–714, 2009. [Online]. Available: https://doi.org/10.1137/06067434X

[7] M.-M. Gu, K.-J. Pai, and J.-M. Chang, "Subversion analyses of hierarchical networks based on (edge) neighbor connectivity," *Journal of Parallel and Distributed Computing*, vol. 171, p. 54 – 65, 2023, cited by: 0. [Online]. Available:

TABLE VI: Complexity of generic routing schemes for CG.

	Memory space requirements				Forwarding decision time		
Routing scheme	**Message header**	**Node label**	**Routing table**	**Other routing information**	**Static routing**		**Fault-tolerant routing**
					Single-path	**Multi-path**	
WPR[a]	$O(D\log(\Delta))$	$O(D\log(\Delta))$	$O(\Delta)$	1) WDA: $O(\lvert Diff\rvert)$ 2) Failure record: $O(D\log(\Delta))$	1) Source nodes: $O(D\lvert Diff\rvert)$ 2) Intermediate nodes: $O(1)$	1) Source nodes: $O(\Delta D\lvert Diff\rvert)$ 2) Intermediate nodes: $O(1)$	$O(\Delta D\lvert Diff\rvert)$
RCRR	$O(D\log(n))$	$O(\log(n))$	$O(n\Delta)$	1) Dynamic routing table: $O(n\log(D))$	$O(1)$	N/A	$O(D)$
GRWMS[a]	$O(D\log(\Delta))$	$O(D\log(\Delta))$	$O(\Delta)$	1) WDA: $O(\lvert Diff\rvert)$	$O(\Delta D\lvert Diff\rvert)$	N/A	N/A
RPS[b]	$O(p\log(p))$	$O(p\log(p))$	$O(\Delta p\log(p))$	N/A	$O(\Delta p\log(p))$	N/A	N/A

[a]The size of the WDA, i.e. $|Diff|$ depends on the CG's family.
[b]The value of the symmetric group parameter, i.e. p, depends on the CG's family.

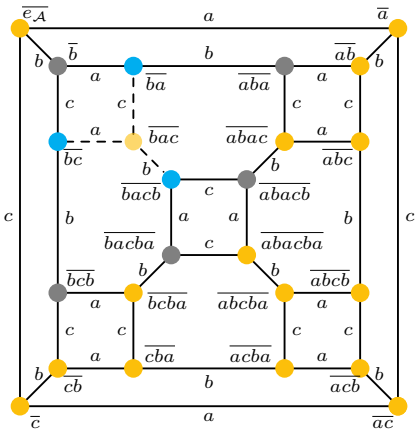
PLACE PHOTO HERE

Daniela Aguirre-Guerrero Biography text here.

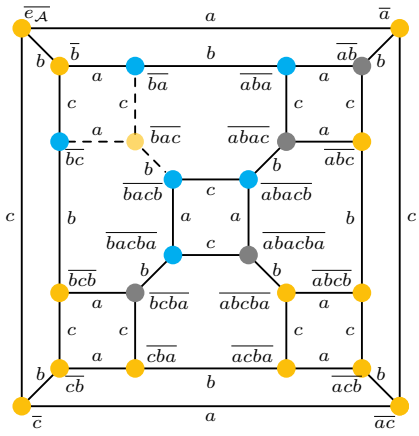
Lluís Fàbrega Biography text here.

Pere Vilà Biography text here.

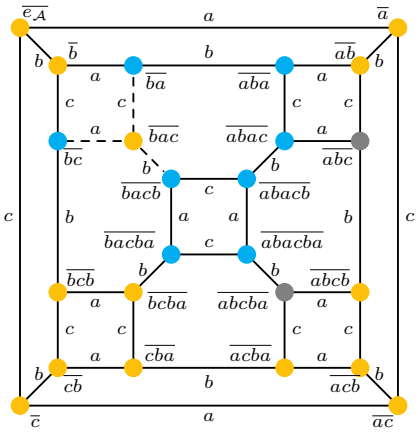
Enrique Rodriguez-Colina Biography text here.



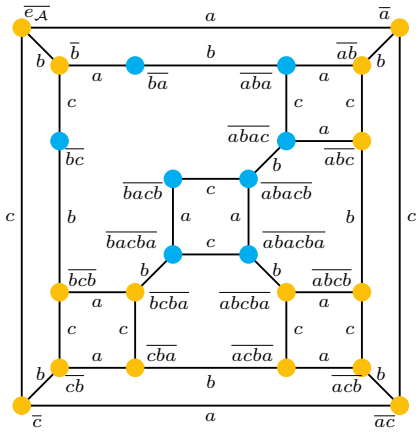
(a) Node $\overline{bac}$ fails and their adjacent nodes $\overline{ba}$, $\overline{bc}$ and $\overline{bacb}$ add bac to their failures records, and then notify their adjacent nodes (gray nodes).



(b) Nodes $\overline{aba}$, $\overline{abacb}$ and $\overline{bacba}$ add bac to their failures records, and then notify their adjacent nodes (gray nodes).



(c) Nodes $\overline{abac}$ and $\overline{abacba}$ add bac to their failures records, and then notify their adjacent nodes (gray nodes).

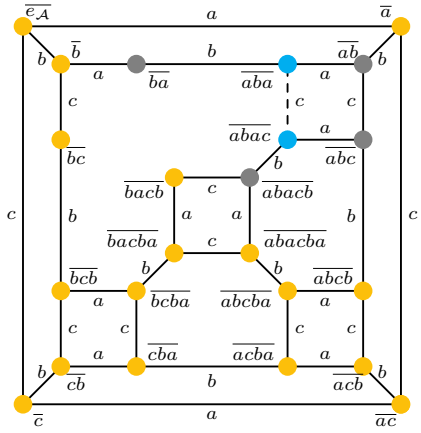


(d) None of the notified nodes updates its failures record, then the process of failure notification finishes. The blue nodes have updated their failures records through this process.

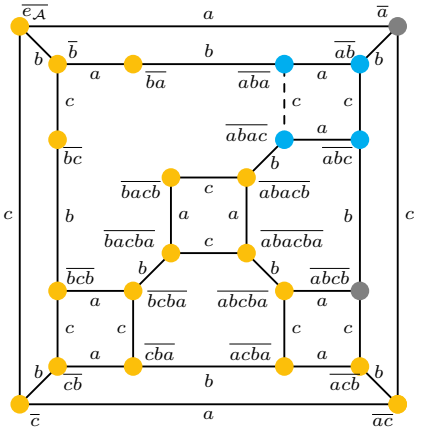
Node sending the notification	**Node receiving the notification**	**Notified failures**		**Failures records after notifying**		**Failures records before notifying**	
$\overline{v}$	$\overline{u}$	$\mathcal{V}_v$	$\mathcal{E}_v$	$\mathcal{V}'_u$	$\mathcal{E}'_u$	$\mathcal{V}_u$	$\mathcal{E}_u$
$\overline{ba}$	$\overline{aba}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{ba}$, $\overline{bc}$	$\overline{b}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{bc}$	$\overline{bcb}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{bacb}$	$\overline{abacb}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{bacb}$	$\overline{bacba}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{aba}$	$\overline{ab}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{aba}$, $\overline{abacb}$	$\overline{abac}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{abacb}$, $\overline{bacba}$	$\overline{abacba}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{bacba}$	$\overline{bcba}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{abac}$	$\overline{abc}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{abacba}$	$\overline{abcba}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

(e) Failures records after and before the notification of the faulty node $\overline{bac}$. Each notified node computes paths from it to the nodes adjacent to $\overline{bac}$, see Table III.

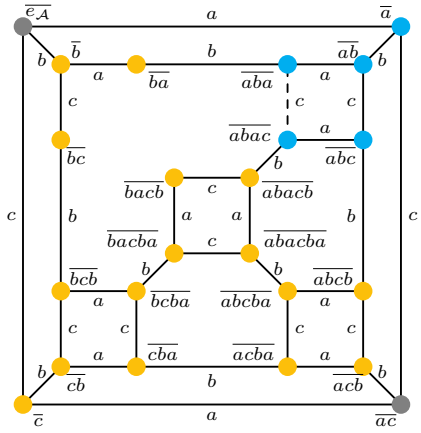
Fig. 5: Notification of the faulty node $\overline{bac}$ in $BS(4)$.



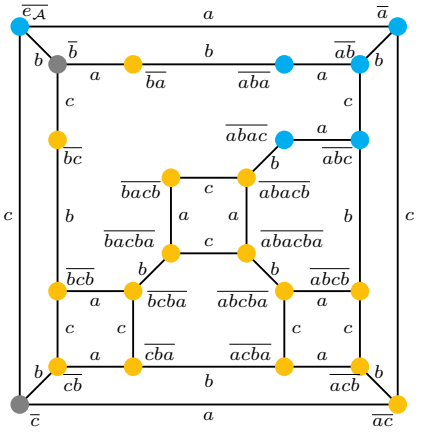
(a) Link $(\overline{aba}, abac)$ fails and their incident nodes $\overline{aba}$ and $\overline{abac}$ add $(aba, abac)$ to their failures records, and then notify their adjacent nodes (gray nodes).



(b) Nodes ab and abc add $(aba, abac)$ and bac to their failures records, and then notify their adjacent nodes (gray nodes).



(c) Node $\overline{a}$ adds bac to its failures records, and then notify their adjacent nodes (gray nodes).

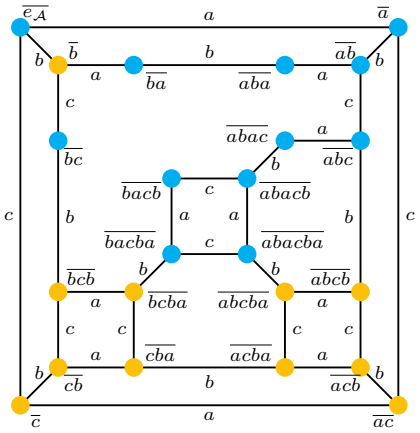


(d) Node $\overline{e_{\mathcal{A}}}$ adds abc to its failures records, and then notify their adjacent nodes (gray nodes).

Nodes sending the notification	**Nodes receiving the notification**	**Notified failures**		**Failures records after notifying**		**Failures records before notifying**	
$\overline{v}$	$\overline{u}$	$\mathcal{V}_v$	$\mathcal{E}_v$	$\mathcal{V}'_u$	$\mathcal{E}'_u$	$\mathcal{V}_u$	$\mathcal{E}_u$
$\overline{aba}$	$\overline{ba}$	$\{bac\}$	$\{(aba, abac)\}$	$\{bac\}$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{aba}$	$\overline{ab}$	$\{bac\}$	$\{(aba, abac)\}$	$\emptyset$	$\emptyset$	$\{bac\}$	$\{(aba, abac)\}$
$\overline{abac}$	$\overline{abacb}$	$\{bac\}$	$\{(aba, abac)\}$	$\{bac\}$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{abac}$	$\overline{abc}$	$\{bac\}$	$\{(aba, abac)\}$	$\emptyset$	$\emptyset$	$\{bac\}$	$\{(aba, abac)\}$
$\overline{ab}$	$\overline{a}$	$\{bac\}$	$\{(aba, abac)\}$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{abc}$	$\overline{abcb}$	$\{bac\}$	$\{(aba, abac)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{a}$	$\overline{e_{\mathcal{A}}}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{bac\}$	$\emptyset$
$\overline{a}$	$\overline{ac}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{e_{\mathcal{A}}}$	$\overline{b}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{e_{\mathcal{A}}}$	$\overline{c}$	$\{bac\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

(e) Failures records after and before the notification of the faulty link $(aba, abac)$. Each notified node computes paths from it to the nodes incident and adjacent to the notified failures, see Table IV.

Fig. 6: Process of link failure notification in $BS(4)$ with the failure of node $\overline{bac}$.

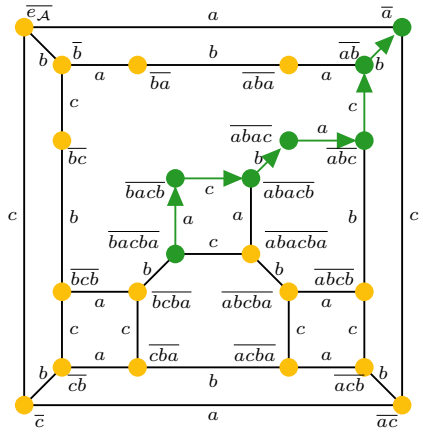


(a) $BS(4)$ after node $\overline{bac}$ and link $(aba, abac)$ have failed. The blue nodes have updated their failures records.

$\overline{u}$	$\mathcal{V}_u$	$\mathcal{E}_u$
$\overline{e_{\mathcal{A}}}$	$\{bac\}$	$\emptyset$
$\overline{a}$	$\{bac\}$	$\emptyset$
$\overline{ab}$	$\{bac\}$	$\{(aba, abac)\}$
$\overline{ba}$	$\{bac\}$	$\emptyset$
$\overline{bc}$	$\{bac\}$	$\emptyset$
$\overline{aba}$	$\{bac\}$	$\{(aba, abac)\}$
$\overline{abc}$	$\{bac\}$	$\{(aba, abac)\}$
$\overline{abac}$	$\{bac\}$	$\{(aba, abac)\}$
$\overline{bacb}$	$\{bac\}$	$\emptyset$
$\overline{abacb}$	$\{bac\}$	$\emptyset$
$\overline{bacba}$	$\{bac\}$	$\emptyset$
$\overline{abacba}$	$\{bac\}$	$\emptyset$

(b) Nodes with their updated failures records.

Fig. 7: Final state of $BS(4)$ after node $\overline{bac}$ and link $(aba, abac)$ have failed.



(a) Routing path computed between nodes $\overline{bacba}$ and $\overline{a}$, i.e. $\widehat{acbacb}$.

Forwarding node	**Computed path**	**Output port**	**Next node**
$\overline{bacba}$	$\widehat{abacba}$	a	$\overline{bacb}$
$\overline{bacb}$	$\widehat{cbcab}$	c	$\overline{abacb}$
$\overline{abacb}$	$\widehat{bacb}$	b	$\overline{abac}$
$\overline{abac}$	$\widehat{acb}$	a	$\overline{abc}$
$\overline{abc}$	$\widehat{cb}$	c	$\overline{ab}$
$\overline{ab}$	$\widehat{b}$	b	$\overline{a}$

(b) Output ports selected by the forwarding nodes. These ports are determined by the routing path computed in the each forwarding node.

Fig. 8: Routing in $BS(4)$ after node $\overline{bac}$ and link $(aba, abac)$ had failed. The routing path is computed by each node in the path taking into account the failure records.