algorithms

Article

BCCC Disjoint Path Construction Algorithm and Fault-Tolerant Routing Algorithm under Restricted Connectivity

Jialiang Lu [1], Xiaoyu Du [2,3,*], Huiping Li [1] and Zhijie Han [1]

1 Software College, Henan University, Kaifeng 475000, China
2 School of Computer and Information Engineering, Henan University, Kaifeng 475000, China
3 Henan Engineering Laboratory of Spatial Information Processing, Henan University, Kaifeng 475004, China
* Correspondence: dxy@henu.edu.cn

Abstract: Connectivity in large-scale data center networks is a critical indicator to evaluate network state. A feasible and performance-guaranteed algorithm enables us to find disjoint paths between network vertices to ensure effective data transfer and to maintain the normal operation of network in case of faulty nodes. As an important data center network, BCube Connected Crossbars (BCCC) has many excellent properties that have been widely studied. In this paper, we first propose a vertex disjoint path algorithm with the time complexity of $O(nk)$ in BCCC, where n denotes a switch connected to n servers and k denotes dimension. Then, we prove that the restricted connectivity of $BCCC(n,k)$. Finally, we present an $O(kn\kappa^1(G))$ fault-free algorithm in BCCC, where $\kappa^1(G)$ is the restricted connectivity. This algorithm can obtain a fault-free path between any two distinct fault-free vertices under the condition that each vertex has at least one fault-free neighbor in the BCCC and a set of faulty vertices F with $|F| < \kappa^1(G)$.

Keywords: data center network; BCCC; restricted connectivity; vertex disjoint path; fault-free algorithm

Citation: Lu, J.; Du, X.; Li, H.; Han, Z. BCCC Disjoint Path Construction Algorithm and Fault-Tolerant Routing Algorithm under Restricted Connectivity. *Algorithms* **2022**, *15*, 481. https://doi.org/10.3390/a15120481

Academic Editors: Andras Farago, Ionut Brandusoiu and Héctor Migallón

Received: 4 November 2022
Accepted: 6 December 2022
Published: 17 December 2022

1. Introduction

The data center network (DCN) is a critical infrastructure of cloud computing, which can provide important cloud services such as GFS [1], Bigtable [2], and Dryad [3]. A data center network is a bridge connecting large-scale servers in the data center for distributed computing. As a widely used server-centric DCN, BCCC [4] can provide good network performance using inexpensive off-the-shelf switches and commodity servers with only two network interface card (NIC) ports. BCCC has many desirable properties such as high scalability, a near-equal-length parallel path, and a small diameter. The Hamilton properties and routing algorithm of BCCC have been studied [5–8].

Server failures are normal, since there are a large number of servers in the actual network. When the network fails, how to restore normal communication should be considered first. Furthermore, whether there is a fault-free path between any two different fault-free vertices and how to construct a fault-free path using an algorithm should be also taken into account. In other words, this is a problem of fault-tolerant routing, which is also the major problem we will study in this paper.

When designing or selecting a network topology for a data center, the most fundamental consideration is the fault tolerance of the network. Fault tolerance refers to the ability to continue working in the case of a network failure. Fault tolerance is the maximum number of failed devices that the network will allow to operate normally. In this paper, fault tolerance is directly determined by the connectivity of the network. The larger the connectivity scale, the better fault-tolerant performance of the data center network. In order to overcome the shortcoming of traditional connectivity, Harary [9] introduced the concept of the restricted faulty nodes on the resulting graph. Following this trend, Esfahanian [10] proposed the concept of restricted connectivity. The restricted connectivity is the connectivity under the condition that each fault-free node has at least one fault-free neighbor.

There have been many studies on the restricted connectivity of networks. Li et al. [11] establish a universally h-restricted connectivity for a class of hypercube-based compound networks, such as hierarchical cubic network and its generalization complete cubic network. Cheng [12] studies the h-restricted connectivity of an n-dimensional balanced hypercube BH_n. Li et al. [13] investigate the h-restricted connectivity of the generalized hypercube. Liu et al. [14] prove the k-restricted edge connectivity of the data center network DCell and Lin et al. [15] establish the 3-restricted connectivity of an n-dimensional split-star network. Wang [16] shows the r-restricted connectivity of hyper Petersen graphs. Ma et al. [17] prove the restricted edge connectivity of Kronecker product graphs.

Finding disjoint paths between two vertex sets in a massively parallel system establishes secure communication paths that do not interfere with each other. Constructing multiple disjoint paths for two vertices increases the probability of constructing a fault-free path. In addition to being used to avoid congestion, speed up transmission rates, and provide alternative propagation paths, disjoint paths can also enhance the robustness of vertex failures and the ability to load balance. Wu et al. [18] propose an algorithm to give the disjoint path between any two distinct vertices in dragonfly networks. Kern [19] gives exact algorithms for disjoint paths and disjoint connected subgraphs on graphs with n vertices and m edges in H-free graphs. Hadid et al. [20] propose a self-stabilizing One-to-Many node disjoint path routing algorithm in star networks.

Fault-tolerant techniques are one of the pillar technologies of the network. At the routing algorithm level, routing fault tolerance is achieved by selecting different routing paths to bypass faulty links and nodes. Therefore, a fault-tolerant routing algorithm with good effects can be achieved by improving the traditional routing algorithm. Fault-tolerant routing algorithms for different networks have been studied. Pushparaj et al. [21] propose an algorithm for routing the packets in the case of a link fault in Mesh-of-Tree (MoT) topology. The algorithm ensures that the packets will reach their destination via the shortest possible path. Nehnouh et al. [22] introduce a new fault-tolerant routing algorithm, to tolerate permanent and transient faults in network-on-chips. Zhang et al. [23] propose a novel fault-tolerant routing algorithm for 3-ary n-cube networks based on the disjoint path with structure faults. Thuan et al. [24] propose a stochastic link-fault-tolerant routing algorithm in a folded hypercube by introducing a type of limited global information called routing probabilities. Ipek et al. [25] propose a highly adaptive fault-tolerant routing algorithm for two-dimensional network-on-chips.

In this paper, we study the vertex disjoint path construction algorithm and fault-tolerant routing algorithm under restricted connectivity in the BCCC. The major contributions are as follows:

(1) We first propose a vertex disjoint path construction algorithm in BCCC. Time complexity is o(nk). With this algorithm, n vertex disjoint paths can be constructed for any two vertices in the BCCC.
(2) We prove that under the condition that each vertex has at least one fault-free neighbor in BCCC, its restricted connectivity is $2\kappa(G) - k - 1$ and $2\kappa(G) - n - 2$, when $n < k + 1$ and $n \geq k + 1$, respectively, where $\kappa(G)$ is the connectivity.
(3) We give an $O(kn\kappa^1(G))$ fault-free algorithm, where $\kappa^1(G)$ denotes the restricted connectivity of BCCC. This algorithm can obtain a fault-free path between any two distinct fault-free vertices under the condition that each vertex has at least one fault-free neighbor.

The rest of this paper is organized as follows: Section 2 provides the preliminaries. Section 3 proposes a vertex disjoint path construction algorithm. Section 4 discusses the restricted connectivity of BCCC. Section 5 gives a fault-free algorithm to obtain a fault-free path and an analysis of the time complexity of the algorithm. Section 6 provides the discussion and conclusions.

2. Preliminaries

2.1. Graph-Theoretic Terms

Given a simple graph G, we use $V(G)$ and $E(G)$ to denote the vertex set and the edge set, respectively. An edge with end vertices u and v is denoted by (u, v). For each vertex $v \in V(G)$, if $(u, v) \in E(G)$, we say that u is a neighbor of v or u is adjacent to v. The set of all the neighbors of u is called the neighbor set of u in G, denoted by $N_G(u)$. The total number of vertices of graph G is denoted by N.

Pathway P is a sequence consisting of $(x_1, x_2, \ldots, x_n)$ with a length of n, and there is $(x_m, x_m + 1) \in E(G)$ for any $0 \leq m \leq n$. For any two different paths, P_1 and P_2, the starting and ending points are the same, named x and y, respectively. If $V(P_1) \cap V(P_2) = \{x, y\}$, then P_1 and P_2 will be vertex disjoint paths. Furthermore, if all vertices in P are fault-free, then P will be a fault-free path. If P_1 is a path between u and x, and P_2 is a path between x and v in G, then let (P_1,P_2) be a path between u and v in G. The reverse of P is $(x_n, x_{n-1}, \ldots, x_1)$, denoted by P^{-1}.

Let $F \subset V(G)$ and F be a set of faulty vertices. For any vertex x in G, if $x \in F$, then x is faulty; otherwise, x is fault-free. The connectivity of graph G is defined as $\kappa(G)$. This is the minimum number of vertices needed to make the graph disconnected or trivial. If we specify that every vertex has at least one fault-free neighbor, we call the connectivity in this case the limited connectivity, which is defined as $\kappa^1(G)$.

2.2. Structure and Properties of BCCC

The BCCC structure connects servers by using recursively defined squares. In $BCCC(n, k)$, n denotes the n-port server and k denotes the dimension. First, we let n servers connect to an n-port switch as an element, denoted by BCCC(n,0). BCCC(n,k) is composed of n BCCC(n,k−1)s connecting n^k elements. Two types of switch are used in BCCC, which are the type A switch and type B switch. A type A switch has n ports for forming an element and a type B switch has $(k + 1)$ ports for connecting different elements. Any server x in BCCC(n,k) can be represented by $x_k x_{k-1} \ldots x_i \ldots x_0, x_i \in \langle n - 1\rangle$ and $i \in \langle k\rangle$. Any switch s can be represented by $s_k s_{k-1} \ldots s_i \ldots s_0$ and $s_i \in \langle n - 1\rangle, 1 \leq i \leq k + 1, s_0 \in \langle n + k\rangle$. Overall, to build BCCC(n,k), we need $(k + 1)n^{k+1}$ dual-port servers, $(k + 1)n^k$ type A switches, and n^{k+1} type B switches. An example of $BCCC(3, 2)$ is shown in Figure 1.

We can see that a pair of servers $a_{k+1}a_k a_{k-1} \ldots a_0$ and $a'_{k+1}a'_k a'_{k-1} \ldots a'_0$ are neighbors, meaning that they are connected by the same switch, if and only if $\exists i$, $i = a_0 + 1$ or $i = 0$, such that $a_i \neq a'_i$, and $\forall j$, $0 \leq j \leq k + 1$, $i \neq j$, such that $a_j = a'_j$.

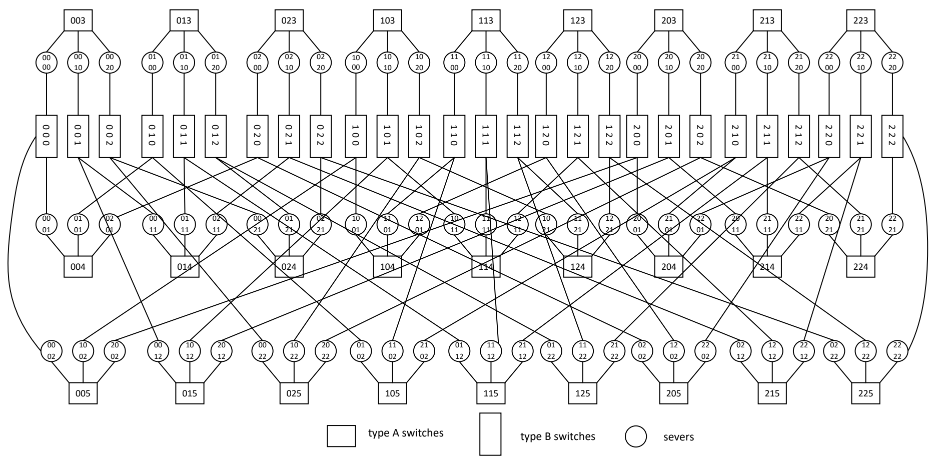

Figure 1. Structure of $BCCC(3, 2)$.

According to the numbering rules of the $BCCC$ server, x_0 connects different layers. $x_i \in [0, n - 1], 1 \leq i \leq k + 1$. Therefore, BCCC can be divided into n subgraphs. We divide BCCC into n subgraphs according to the second digit number of the server from right to left. Servers with the same second digit number are located in the same subgraph. These subgraphs are named $BCC^1_{n,k}$, $BCC^2_{n,k}$, …, $BCC^n_{n,k}$. The second digit number is 0 and the subgraph is $BCC^1_{n,k}$. We named the server connected to other subgraphs u_1, and named those not connected to other subgraphs u_2. An example of $BCCC^1_{3,2}$ is shown in Figure 2.

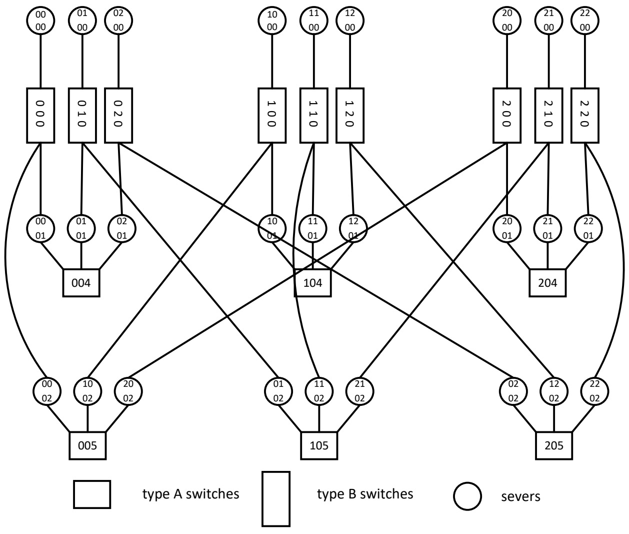

Figure 2. Structure of $BCCC^1_{3,2}$.

Lemma 1. *BCCC (n,k) has the following properties [4]:*

(1) *$BCCC(n,k)$ is (n+k−1)-regular graph. $N = (k+1)n^{k+1}$.*
(2) *The connectivity of $BCCC(n,k)$ is $\kappa(G) = n + k - 1$.*
(3) *There are n^k vertex disjoint paths connecting different $BCCC(n,k-1)$ in $BCCC(n,k)$.*

3. Vertex Disjoint Path Construction Algorithm

In this section, we propose a vertex disjoint path construction algorithm for BCCC. The algorithm constructs n vertex disjoint paths between any two nodes u and v in the BCCC.

3.1. Algorithm Description and Implementation

There are two vertex disjoint path algorithms, including BuildPathSet and Convert. BuildPathSet is the master algorithm. The Convert algorithm can change one bit of the u node number to the corresponding bit of the v node. The flow of the vertex disjoint path algorithm is as follows (refer to Algorithm 1). First, a series of v nodes located at the vertices of the same type A switch $v_2, v_1, \ldots, v_{n-1}$ are selected. Then, a vertex v_n located on the same type B switch as v is determined. The mode of the first path is as follows. First, we reach node x. The value of the u_0 bit of x is 0. In addition, x is on the same type A switch as the source node. The path is selected from node x to node v_1. The second path pattern is as follows. First, some data arrive at the y node. The value of the u_0 bit of y is 1. In addition, node y and the source node reside on the same type A switch. The path is from node y to node v_2. The other paths follow the same rules. The Convert algorithm is invoked $n-1$ times during the process from the source node to the target node. One digit is changed from the highest digit of the number to the lowest digit each time.

Algorithm 1 Vertex Disjoint Path Algorithm of BCCC.

```
Input: Two different vertices $u = u_{k+1}u_k...u_0$ and $v = v_{k+1}v_k...v_0$ in $BCCC(n,k)$. Let the
    neighbor node of the same type $A$ switch with $v$ be $v_1, v_2, ..., v_{n-1}$, and select any a
    neighbor node in the same type $B$ switch as $v_n$.
Output: $n$ vertex disjoint paths from $u$ to $v$.
function BUILDPATHSET($BCCC(n,k), u, v$)
    $z = v$;
    for $i = 0; i = n-1; i++$ do
        $v = v_{i+1}$, $Path = \emptyset$;
        add $u$ to $Path$;
        if $u_{u0} \neq$ i then
            $u = u_{k+1}u_k...u_{u0}...u_0$, $u_{u0} = i$, $i \in \{1,2,...,k+1\}$;
            add $u$ to $Path$;
        for $j = k+1; j > 0; j--$ do
            if $u == z$ then
                break;
            if $u_j \neq v_j$ then
                $u$, $path$ = CONVERT($u, v, u_j, v_j, Path$);
        if $u_0 \neq v_0$ then
            $u = u_{k+1}u_k...v_0$;
            add $u$ to $Path$;
        if $u \neq z$ then
            add $v$, $z$ to $Path$;
        print($Path$);
function CONVERT($u$, $v$, $u_x$, $v_y$, $Path$)
    $u = u_{k+1}u_k...u_x...u_0$, $x \in \{1,2,...,k+1\}$;
    $v = v_{k+1}v_k...v_y...v_0$, $y \in \{1,2,...,k+1\}$;
    if $u_0 == v_0$ then
        $w = u$;
    else
        $u_0 = v_y$;
        $w = u_{k+1}u_k...v_y$;
        add $w$ to $Path$;
    $u_x = u_y$;
    $u = u_{k+1}u_k...v_y...u_0$;
    add $u$ to $Path$;
    return($u$, $Path$);
```

3.2. Analysis of Vertex Disjoint Path Construction Algorithm

First, the time complexity of the Convert algorithm is analyzed. The node number of u is changed from one bit to v. This node is added to the path set and the time complexity is $O(1)$. The second 'for' loop in the BuildPathSet algorithm runs a total of k times from high to low with $O(k)$ concomitant time complexity. The first 'for' loop causes the algorithm to form a total of n paths with $O(n)$ adjoint time complexity. In summary, the time complexity of the BuildPathSet algorithm is $O(nk)$.

3.3. Application Examples

Example 1. *Given the network $BCCC(3,2)$ (see Figure 1), three vertex disjoint paths are constructed by the algorithm. The source node is selected as* 0000 *and the target node as* 2011. *It can be obtained that the nodes located in the same switch with the target node are* 2111, 2211, *and* 2010. *The first path is from* 0000 *to* 2111. *The Convert algorithm is used to gradually change from the* *highest bit to the lowest bit. Therefore, Path*$1 = 0000 \rightarrow 0002 \rightarrow 2002 \rightarrow 2001 \rightarrow 2101 \rightarrow 2100 \rightarrow 2110 \rightarrow 2111$. *This eventually reaches 2011 through the type A switch.*

The second path first selects node 0010*, which is located in the same type A switch as node* 0000*, and reaches* 2211 *from* 0010*, using the Convert algorithm. Therefore, Path*$2 = 0000 \rightarrow 0010 \rightarrow 0012 \rightarrow 2012 \rightarrow 2011$. *Because the* 2011 *node is encountered on the path to the* 2211 *node and* 2011 *is the target node, the path is obtained by jumping out directly according to lines* 10 *and* 17 *of the algorithm.*

The third path selects node 0020*, which is located in the same type A switch as node* 0000*. The Convert algorithm is used to reach* 2010 *from* 0020*. Therefore, Path*$3 = 0000 \rightarrow 0020 \rightarrow 0022 \rightarrow 2022 \rightarrow 2020 \rightarrow 2010 \rightarrow 2011$. *Finally, three vertex disjoint paths from* 0000 *to* 2011 *are obtained.*

4. Restricted Connectivity of BCCC

In this section, we prove the restricted connectivity of BCCC. We refer to the entire graph of $BCCC$ as G.

Theorem 1. *The restricted connectivity of BCCC is*
$\kappa^1(G) \leq 2n + k - 3 = 2\kappa(G) - k - 1$, *if* $n < k + 1$,
$\kappa^1(G) \leq n + 2k - 2 = 2\kappa(G) - n$, *if* $n \geq k + 1$, *when* $n \geq 3$ *and* $k \geq 2$.

Proof. Let u and v be two vertices of the same type A switch ($v \in N_G(u)$). Therefore, $|N_G(u) \cap N_G(v)| = n - 2$.

If $|N_G(u)| = |N_G(v)| = \kappa(G)$, according to the definition,

$$\begin{aligned} |N_G(\{u,v\})| &= |N_G(u) \cup N_G(v) \backslash (N_G(u) \cap N_G(v))| \\ &= |N_G(u) - 1| + |N_G(v) - 1| - (n-2) \\ &= 2(\kappa(G) - 1) - (n-2) \\ &= 2\kappa(G) - n \end{aligned} \tag{1}$$

Let $F = N_G(\{u,v\})$, so $G - F$ is divided into two disconnected subgraphs: one is $G(u,v)$ and the other is $G - (N_G(\{u,v\}) \cap \{u,v\})$. Furthermore, u has a fault-free neighbor v. Therefore, each vertex of subgraph $G(u,v)$ has at least one fault-free neighbor. Let $H = N_G(\{u,v\}) \backslash \{u,v\}$, for $BCCC(n,k)$, when u and v are located in the same type A switch, each node in $N_G(H)$ has only one neighbor that is a faulty node. According to Lemma 1 (2), $\kappa(G) = n + k - 1$. Therefore, $\kappa(G)$ is constantly greater than 2 when $n \geq 3$ and $k \geq 2$. It can be ensured that each node in $G - (N_G(\{u,v\}) \cap \{u,v\})$ has at least one fault-free neighbor.

In summary, $G - F$ is disconnected and each vertex in both subgraphs contains at least one fault-free neighbor. Hence, $\kappa^1(G) \leq 2n + k - 2 = 2\kappa(G) - n$. Therefore, the theorem holds.

Let x and y be two vertices of the same type B switch ($v \in N_G(u)$) ($y \in N_G(x)$). Therefore, $|N_G(x) \cap N_G(y)| = k - 1$.

If $|N_G(x)| = |N_G(y)| = \kappa(G)$, according to the definition,

$$\begin{aligned} |N_G(\{x,y\})| &= |N_G(x) \cup N_G(y) \backslash (N_G(x) \cap N_G(y))| \\ &= |N_G(x) - 1| + |N_G(y) - 1| - (k-1) \\ &= 2(\kappa(G) - 1) - (k-1) \\ &= 2\kappa(G) - k - 1 \end{aligned} \tag{2}$$

Proving that the two vertices are in the same type B switch is the same as proving the two vertices are in the same type A switch. Let $F = N_G(\{x,y\})$, so $G - F$ is disconnected and each vertex in both subgraphs contains at least one fault-free neighbor. Above all, $\kappa^1(G) \leq 2k + n - 3 = 2\kappa(G) - k - 1$. Therefore, the theorem holds. □

Theorem 2. *The restricted connectivity of BCCC is*

$\kappa^1(G) \geq 2n + k - 3 = 2\kappa(G) - k - 1$, if $n < k + 1$,
$\kappa^1(G) \geq n + 2k - 2 = 2\kappa(G) - n$, if $n \geq k + 1$, when $n \geq 3$ and $k \geq 2$.

Proof. Let F be the set of faulty vertices in G. Proving this theorem is equivalent to proving, for any two vertices u and v in $G - F$, there exists a fault-free path from u to v. Furthermore, any vertex in $G - F$ has at least one fault-free neighbor.

If u and v are adjacent, the theorem holds. Hence, we discuss the case that u and v are not adjacent. According to the structure of BCCC, BCCC can be divided into n subgraphs. Let $BCC^i_{n,k}$ for $i \in \langle n \rangle \backslash \{0\}$ be the subgraph of G. Furthermore, $u \in BCC^\alpha_{n,k}$, $v \in BCC^\beta_{n,k}$ for $\alpha, \beta \in \langle n \rangle \backslash \{0\}$. Let $F_\alpha = V(BCC^\alpha_{n,k}) \cap F$ and $F_\beta = V(BCC^\beta_{n,k}) \cap F$. Let $R = \langle n \rangle \backslash \{0, \alpha\}$, $BCC^R_{n,k}$ be a subgraph other than $BCC^\alpha_{n,k}$ in G and $F_R = V(BCC^R_{n,k}) \cap F$. If $N_G(u) \cap V(BCC^R_{n,k}) \neq \emptyset$, u is named as u_1. Otherwise, u is named as u_2.

Lemma 2. *If F is a set of vertices in $BCCC(n,k)$, and for any subgraph i in $BCCC(n,k)$, $BCC^R_{n,k}$ is connected if $|F \cap V(G - V(BCC^i_{n,k}))| \leq \kappa(G) - 2$.*

Proof. According to the construction rules of BCCC, U is set as the point adjacent to $BCC^i_{n,k}$ in $BCC^R_{n,k}$ to reach the conclusion that the point set of U is located in the same type A switch of subgraph $BCC^i_{n,k}$ and has no neighbors. Therefore, the point in the set of U points is the one with the fewest neighbors in $BCC^i_{n,k}$. These vertices have k vertex disjoint paths in the $BCC^i_{n,k}$ subgraph. This achieves $\kappa(BCC^i_{n,k}) = k (k \geq 2)$. Moreover, for any vertex x in $BCC^R_{n,k}$ to connect to $BCC^i_{n,k}$, one of x's neighbors in the same type A switch is not in the subgraph. Therefore, $\kappa(BCC^R_{n,k}) = \kappa(G) - 1$.

Because $|F \cap V(G - V(BCC^i_{n,k}))| \leq \kappa(G) - 2$ and $\kappa(BCC^R_{n,k}) = \kappa(G) - 1$, $BCC^R_{n,k}$ are connected, and the Lemma holds. □

According to the position of u and v vertices in BCCC, we have the following cases.

Case 1. $\alpha \neq \beta$.

Subcase 1.1. $|F_\alpha| \geq \kappa(BCC^\alpha_{n,k}) or |F_\beta| \geq \kappa(BCC^\beta_{n,k})$.

Without losing generality, suppose $|F_\alpha| \geq \kappa(BCC^\alpha_{n,k})$. Apparently, $|F_R| = |F \backslash F_\alpha| \leq 2\kappa(G) - n - 1 - (\kappa(G) - n + 1) \leq \kappa(G) - 2$. According to Lemma 2, $BCC^R_{n,k} - F_R$ is connected. Next, we discuss the following cases.

Subcase 1.1.1. u is u_1.

Subcase 1.1.1.1. $N_G(u_1) \cap V(BCC^R_{n,k}) \not\subseteq F$.

Selecting a vertex x, $x \subseteq (N_G(u_1) \cap V(BCC^R_{n,k}))$, there is a path P_1 from u_1 to x in $BCC^\alpha_{n,k} - F_\alpha$. Because $BCC^R_{n,k} - F_R$ is connected, there exists a fault-free path P_2 from x to v in $BCC^R_{n,k}$.

Subcase 1.1.1.2. $N_G(u_1) \cap V(BCC^R_{n,k}) \subseteq F$.

Because the neighbors of u_1 are all fault vertices in subgraph $BCC^R_{n,k}$, there must be a fault-free neighbor in subgraph $BCC^\alpha_{n,k}$. Now, we consider how many fault-free paths from u_1 to $BCC^R_{n,k}$ exist in $BCC^\alpha_{n,k}$. Furthermore, $N_G(u_1) \cap V(BCC^R_{n,k}) = n - 1$.

Let $x_1, x_2, \ldots, x_{n-1}$ be all neighbors of u_1 in $BCC^R_{n,k}$.

Let $y_1, y_2, \ldots, y_k$ be all neighbors of u_1 in $BCC^\alpha_{n,k}$; $y_{1'}, y_{2'}, \ldots, y_{k'}$ be the same type A switch neighbors of $y_1, y_2, \ldots, y_k$; $y_{1''}, y_{2''}, \ldots, y_{k''}$ be all neighbors of $y_{1'}, y_{2'}, .., y_{k'}$ in $BCC^\alpha_{n,k}$ and $N_G(y_{i''}) \cap v(BCC^R_{n,k}) \neq \emptyset, i \in \{1, 2, \ldots, k\}$. Thus, there are $(n-1)(k+1)$ paths from u_1 to $BCC^R_{n,k}$ as follows (refer to Figure 3). The specific paths are listed below:

$$\begin{cases} X_i = (u_1, x_i), 1 \le i \le n-1 \\ Y_i = (u_1, y_i, y_{i'}, y_{i''}), 1 \le i \le k, i' = k(n-1) \end{cases}$$

Because $N_G(u_1) \cap V(BCC^R_{n,k}) \subseteq F$, $F \cap V(BCC^R_{n,k}) \ge n-1$, $F \cap V(BCC^\alpha_{n,k}) \le \kappa^1(G) - 1 - n + 1 = \kappa^1(G) - n$. Whether $\kappa^1(G) = 2\kappa(G) - n$ or $2\kappa(G) - k - 1$, $(n-1)(k+1) - k^1(G) - n \ge 1$. Therefore, there is at least one fault-free path P_1 from u_1 to $BCC^R_{n,k}$. Assume that t is the last vertex in the path P_1. If $t = v$, then P_1 is a path from u_1 to v in $G - F$. Otherwise, since $BCC^R_{n,k} - F_R$ is connected, there is a fault-free path P_2 from t to v in $BCC^R_{n,k}$. Then, the path (P_1, P_2) is a fault-free path from u_1 to v in $G - F$.

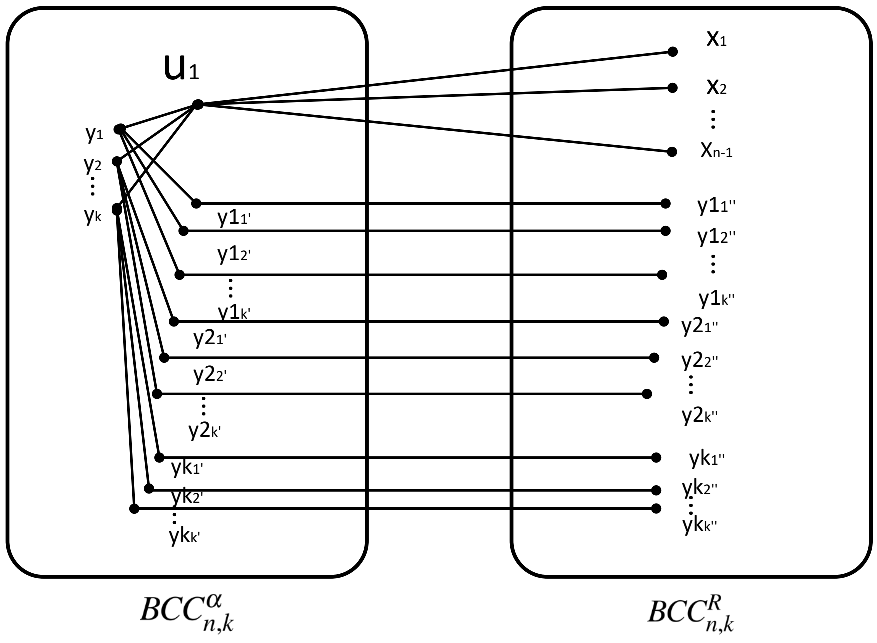

Figure 3. $(n-1)(k+1)$ paths from u_1 to $BCC^R_{n,k}$.

Subcase 1.1.2. u is u_2.

It can be obtained that u_2 is a vertex in $BCC^\alpha_{n,k}$ and its neighbors are all in this subgraph. According to the definition of restricted connectivity, there is a fault-free neighbor in $BCC^\alpha_{n,k}$. Now, we consider how many fault-free paths there are from u_2 to $BCC^R_{n,k}$ in $BCC^\alpha_{n,k}$.

Let x be the neighbor of u_2 and $N_G(x) \cap V(BCC^R_{n,k}) \ne \emptyset$.

Let $y_1, y_2, \ldots, y_{n-1}$ be the same type A switch neighbors of u_2; $y_{1'}, y_{2'}, \ldots, y_{n-1'}$ be all neighbors of $y_1, y_2, \ldots, y_{n-1}$ in $BCC^\alpha_{n,k}$, and $N_G(y_{i'}) \cap V(BCC^R_{n,k}) \ne \emptyset, i \in \{1, 2, \ldots, n-1\}$.

Let $z_1, z_2, \ldots, z_{k-1}$ be neighbors of u_2 in the same type B switch, except x; $z_{1'}, z_{2'}, \ldots, z_{k-1'}$ be the same type A switch neighbors of $z_1, z_2, \ldots, z_{k-1}$; $z_{1''}, z_{2''}, \ldots, z_{k-1''}$ be all neighbors of $z_{1'}, z_{2'}, \ldots, z_{k-1'}$ in $BCC^\alpha_{n,k}$ and $N_G(z_{i''}) \cap V(BCC^R_{n,k}) \ne \emptyset, i \in \{1, 2, \ldots, k-1\}$. Above all, there are $(k-1)(n-1) + n$ paths from u_2 to $BCC^R_{n,k}$ as follows (refer to Figure 4). The specific paths are listed below:

$$\begin{cases} X_i = (u_2, x) \\ Y_i = (u_2, y_i, y_{i'}), 1 \le i \le n-1 \\ Z_i = (u_2, z_i, z_{i'}, z_{i''}), 1 \le i \le k-1, i' = (k-1)(n-1) \end{cases}$$

Since $|F(G)| = 2k(G) - n - 1 = 2(n+k-1) - n - 1 = n + 2k - 3$, we have $(k-1)(n-1) + n - |F(G)| = 1$ when $n = k+1$ and $n = 3$. Furthermore, when $n > 3$, whether $\kappa^1(G) = 2\kappa(G) - n$ or $2\kappa(G) - k - 1$, $(k-1)(n-1) + n - \kappa^1(G) > 1$. Therefore, there is at least one fault-free path P in the above $(k-1)(n-1) + n$ paths from u_2 to $BCC^R_{n,k}$. Assume that w is the last vertex in the path P_1. If $w = v$, then P_1 is a path from u to v in $G - F$.

Otherwise, since $BCC^R_{n,k} - F_R$ is connected, there is path P_2 between w and v in $BCC^R_{n,k}$. Then, the path (P_1, P_2) is a path from u_2 to v in $G - F$.

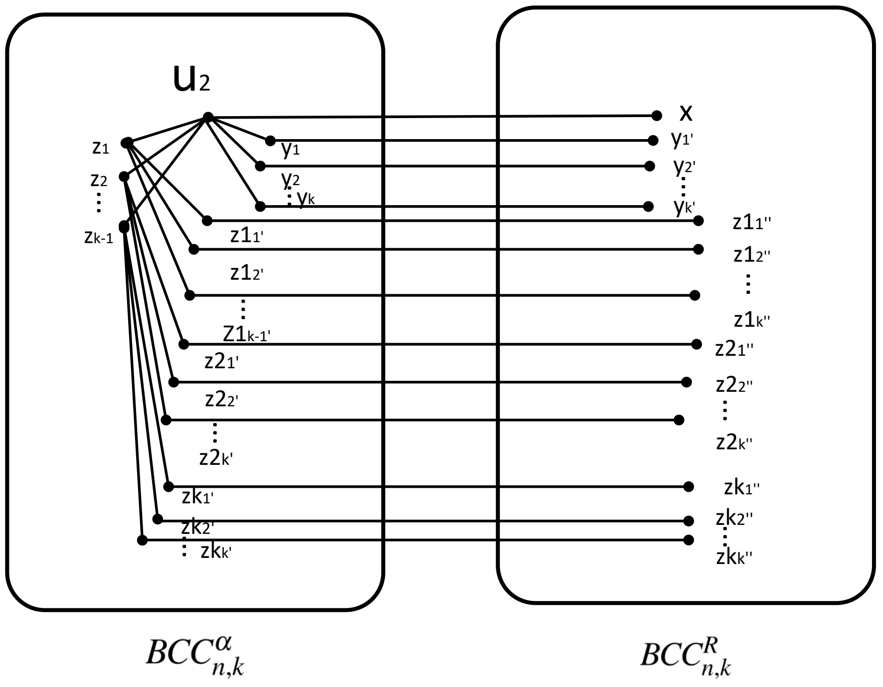

Figure 4. $(k-1)(n-1)+n$ paths from u_2 to $BCC^R_{n,k}$.

Subcase 1.2. $|F_\alpha| < \kappa(BCC^\alpha_{n,k})$ and $|F_\beta| < \kappa(BCC^\beta_{n,k})$.

Because $BCC^\alpha_{n,k} - F_\alpha$ and $BCC^\beta_{n,k} - F_\beta$ are all connected, let $\{x_i|x_1, x_2, \ldots, x_{n^2}\}$ be a set of vertices connected to other subgraphs in $BCC^\alpha_{n,k}$. Let $\{y_i|y_1, y_2, \ldots, y_{n^2}\}$ be a set of vertices connected to other subgraphs in $BCC^\beta_{n,k}$. It can be obtained that $|N_G(x_i) \cap y_i| = n^2$. Whether $\kappa^1(G) = 2\kappa(G) - n$ or $2\kappa(G) - k - 1, n^2 > \kappa^1(G) - 1$. Therefore, there must exist a fault-free vertex $x(x \in x_i)$ and $y(y \in y_i)$ in the same type A switch. If $u = x$, then it is directly connected to $BCC^\beta_{n,k}$. If $u \neq x$, since $BCC^\alpha_{n,k} - F_\alpha$ is connected, there exists a fault-free path P_1 from u to x in $BCC^\alpha_{n,k} - F_\alpha$. Since x and y are located in the same switch, there is a fault-free path P_2 from x to y. If $v = y$, it is directly connected to $BCC^\alpha_{n,k}$. If $v \neq y$, since $BCC^\beta_{n,k} - F_\beta$ is connected, there exists a fault-free path p_3 from y to v in $BCC^\beta_{n,k} - F_\beta$. Then, the path (P_1, P_2, P_3) is a path from u to v in $G - F$ (refer to Figure 5b).

Subcase 2. $\alpha = \beta$.

If $|F_\alpha| < \kappa(BCC^\alpha_{n,k})$, $BCC^\alpha_{n,k} - F_\alpha$ is connected. There exists a fault-free path from u to v in $BCC^\alpha_{n,k}$. Thus, we only need to consider that $|F_\alpha| \geq \kappa(BCC^\alpha_{n,k})$. According to subcase 1.1, if $BCC^\alpha_{n,k} - F_\alpha$ is not connected, $BCC^R_{n,k} - F_R$ is connected. According to subcase 1.1.1 and subcase 1.1.2, whether $u = u_1$ or $u = u_2$, there must exist a fault-free path from $u(v)$ to $BCC^R_{n,k}$. Thus, there exists a fault-free path P_1 from u to a vertex x in $BCC^\alpha_{n,k}$ and a fault-free path P_2 from v to a vertex y in $BCC^\alpha_{n,k}$. If $x = y$, the path (P_1, P_2^{-1}) is a path from u to v. If $x \neq y$, since $BCC^R_{n,k} - F_R$ is connected, there exists a fault-free path P_3 from x to y. Then, the path (P_1, P_3, P_2^{-1}) is a path from u to v in $G - F$ (refer to Figure 5c).

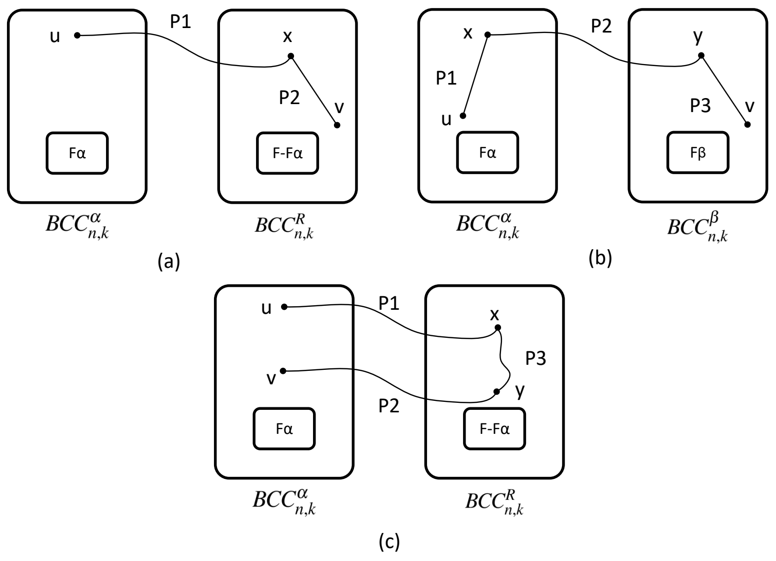

Figure 5. Figure (**a**) is subcase 1.1.1.1; Figure (**b**) is subcase 1.2; Figure (**c**) is subcase 2.

In summary, if there exists a fault-free path between any two vertices u and v in BCCC, the theorem holds. □

Theorem 3. *The restricted connectivity of BCCC is*
$\kappa^1(G) = n + 2k - 3 = 2\kappa(G) - k - 1$, *if* $n < k + 1$,
$\kappa^1(G) = 2n + k - 2 = 2\kappa(G) - n$, *if* $n \geq k + 1$, *when* $n \geq 3$ *and* $k \geq 2$.

Proof. According to Theorems 1 and 2, the theorem holds. □

5. Fault-Free Routing Algorithm of BCCC under the Restricted Connectivity

In this section, we first give a fault-free routing Algorithm *BCCFTP*. Furthermore, we analyze the time complexity of this algorithm.

5.1. Algorithm Description and Implementation

Algorithm *BCCFTP* (refer to Algorithm 2) can give a fault-free path between any two fault-free vertices u and v in $G - F$, where F is the set of faulty vertices in G, $|F| < \kappa^1(G)$, and any vertex in $G - F$ has at least one neighbor.

Algorithm 2 Fault-free routing algorithm of *BCCC*.

Input: Given the graph G to represent $BCCC(n,k)$, a set of fault vertices $F \subset V(G)$, two different vertices u and v. The subgraph α in which u is located, subgraph β in which v is located.

Output: A fault-free path from vertex u to v in $G - F$.

```
1:  function BCCFTP($u$,$v$,$\alpha$,$\beta$,$F$,$BCCC(n,k)$)
2:      if $F = \emptyset$ then
3:          Return BCTP($u$,$v$,$BCCC$,$F$);
4:      else if $(u,v) \in E(G)$ then
5:          Return($u$,$v$);
6:      $\kappa \leftarrow \kappa(BCC^{\alpha}_{n,k})$, $\alpha \leftarrow u$, $\beta \leftarrow v$;
7:      $F_\alpha \leftarrow F \cap (\kappa(BCC^{\alpha}_{n,k})$, $F_\beta \leftarrow F \cap (\kappa(BCC^{\beta}_{n,k})$
8:      if $\alpha \neq \beta$ then
9:          if $|F_\alpha| \geq \kappa$ then
10:             $P_1$ ←PATHSEQ($u$, $F$, $BCC^{\alpha}_{n,k}$, $G - BCC^{\alpha}_{n,k}$);
11:             let $s$ be the last vertex of $Path\ P_1$;
12:             if $s = v$ then
```

```
13:             return $P_1$;
14:         return($P_1$, BCTP($s$,$v$,$G - BCC^{\alpha}_{n,k}$, $F \backslash F_{\beta}$);
15:     **else if** $|F_{\beta}| \geq \kappa$ **then**
16:         $P_1 \leftarrow$ PATHSEQ($v$, $F$, $BCC^{\beta}_{n,k}$, $G - BCC^{\beta}_{n,k}$);
17:         let $s$ be the last vertex of $Path$ $P_1$;
18:         **if** $s = u$ **then**
19:             return $P_1$;
20:         return(BCTP($u$, $s$, $G - BCC^{\beta}_{n,k}$, F$\backslash F_{\alpha}$), $p_1^{-1}$);
21:     **else**$|F_{\alpha}| < \kappa$ && $|F_{\beta}| < \kappa$
22:         $P_1 \leftarrow$PATHSEQ($u$, $F$, $BCC^{\alpha}_{n,k}$, $BCC^{\beta}_{n,k}$);
23:         let $s$ be the last vertex of $Path$ $P_1$;
24:         **if** $s = v$ **then**
25:             return $P_1$;
26:         $P_1$+=BCTP($s$, $v$, $BCC^{\beta}_{n,k}$, $F_{\beta}$);
27:         return $P_1$;
28: **else if** $|F_{\alpha}| < \kappa$ **then**
29:     return(BCTP($u$, $v$, $BCC^{\beta}_{n,k}$, $F_{\alpha}$).
30: **else**
31:     $P_1 \leftarrow$PATHSEQ($u$, $F$, $BCC^{\alpha}_{n,k}$, $G - BCC^{\alpha}_{n,k}$);
32:     let $s$ be the last vertex of $Path$ $P_1$;
33:     $P_2 \leftarrow$PATHSEQ($v$, $F$, $BCC^{\alpha}_{n,k}$, $G - BCC^{\alpha}_{n,k}$);
34:     let $t$ be the last vertex of $Path$ $P_2$;
35:     **if** $w$ be the first common vertex of $P_1$ and $P_2$ **then**
36:         return ($Path$($P_1$, $u$, $w$),$Path$($P_2^{-1}$, $w$, $v$));
37:     $P_1$+=BCTP($s$, $t$, $G - BCC^{\alpha}_{n,k}$, F$\backslash F_{\alpha}$);
38:     return $P_1$+ $P_2^{-1}$;
39:
40: **function** PATHSEQ($u$, $F$, $H_1$, $H_2$)
41:     **if** $N_G(u) \cap V(H_2) \neq \emptyset$ **then**
42:         **if** $N_G(u) \cap V(H_2) \backslash F \neq \emptyset$ **then**
43:             select one fault-free vertex $x$ from $N_G(u) \cap V(H_2)$;
44:             return($u$, $x$);
45:         **else**
46:             **for all** $y_1 \in N_G(u) \cap V(H_1) \backslash F$ **do**
47:                 **if** $N_G(y_1) \cap V(H_1) \backslash F \neq \emptyset$ **then**
48:                     **for all** $y_2 \in N_G(u) \cap V(H_1) \backslash F$ **do**
49:                         **if** $N_G(y_2) \cap V(H_1) \backslash F \neq \emptyset$ **then**
50:                             **for all** $y_3 \in N_G(u) \cap V(H_2) \backslash F$ **do**
51:                                 **if** $N_G(y_3) \cap V(H_2) \backslash F \neq \emptyset$ **then**
52:                                     select one fault-free vertex $y_4$ from $N_G(y_3) \cap V(H_2) \backslash F$;
53:                                     return($u$, $y_1$, $y_2$, $y_3$, $y_4$)
54:     **else**
55:         Select a vertex $x_1$ connected to other subgraphs, select a vertex $y_1$ in the same
    type $A$ switch, and a vertex $z_1$ in the same type $B$ switch from $N_G(u) \cap V(H_1) \backslash F$;
56:         **if** $N_G(x_1) \cap V(H_2) \backslash F \neq \emptyset$ **then**
57:             select one fault-free vertex $x_2$ from $N_G(x_1) \cap V(H_2) \backslash F$;
58:             return($u$, $x_1$, $x_2$);
59:         **if** $N_G(y_1) \cap V(H_1) \backslash F \neq \emptyset$ **then**
60:             **for all** $y_2 \in N_G(y_1) \cap V(H_1) \backslash F$ **do**
61:                 **if** $N_G(y_2) \cap V(H_2) \backslash F \neq \emptyset$ **then**
62:                     **for all** $y_3 \in N_G(y_2) \cap V(H_2) \backslash F$ **do**
63:                         select one fault-free vertex $y_3$ from $N_G(y_3) \cap V(H_2) \backslash F$;
```

```
64:                     return($u, y_1, y_2, y_3$);
65:         for all $z_2 \in N_G(z_1) \cap V(H_1) \backslash F$ do
66:             if $N_G(z_2) \cap V(H_1) \backslash F \neq \emptyset$ then
67:                 for all $z_3 \in N_G(z_2) \cap V(H_1) \backslash F$ do
68:                     if $N_G(z_3) \cap V(H_1) \backslash F \neq \emptyset$ then
69:                         for all $z_4 \in N_G(z_2) \cap V(H_2) \backslash F$ do
70:                             select one fault-free vertex $z_4$ from $N_G(z_3) \cap V(H_1) \backslash F$;
71:                             return($u$, $z_1$, $z_2$, $z_3$, $z_4$);
72:
73: function BCTP($u$, $v$, $BCCC$, $F$)
74:     Select a fault free path from $BuildPathSet(BCCC, u, v)$ in $G - F$;
```

5.2. Analysis of Fault-Free Unicast Algorithm

First, the time complexity of $PATHSEQ(u, F, H_1, H_2)$ and $BCTP(u, v, BCC, F)$ functions is analyzed. u in H_1 exists in the $PATHSEQ(u, F, H_1, H_2)$ function. This depends on whether u is connected to other subgraphs, including u_1 and u_2. The path from u_1 to the subgraph H_2 is (u, x), (u, y_1, y_2, y_3, y_4). The time complexity of $PATHSEQ$ in different situations is as follows. First, the path is (u, x), because $(N_G(u) \cap V(H_2)) \backslash F \neq \emptyset$ finds a vertex x in H_2, and returns a path (u, x). The time complexity is $O(1)$ in this case. Second, the path is (u, y_1, y_2, y_3, y_4). Each vertex in the set is y_1 and the time complexity is $O(1)$ after traversing $(N_G(u) \cap V(H_1)) \backslash F$. Each vertex in the set is y_2 and the time complexity is less than $O(\kappa(G))$ after traversing $(N_G(y_1) \cap V(H_1)) \backslash (F \cup u)$. Each vertex in the set is y_3 and the time complexity is less than $O(\kappa(G))$ after we go through $(N_G(y_2) \cap V(H_1)) \backslash (F \cup y_1)$. Each vertex in the set is y_4 and the time complexity is $O(1)$. After traversing $(N_G(y_3) \cap V(H_2)) \backslash F$, paths $(u, y_1, y_2, y_3, y_4))$ are obtained. The time complexity is less than $\mathrm{O}(\kappa(G)^2)$ in this case.

The path from u_2 to subgraph H_2 includes (u, x_1, x_2), (u, y_1, y_2, y_3), and (u, z_1, z_2, z_3, z_4). First, the path is (u, x_1, x_2). Each vertex in the set is x and the time complexity is $O(1)$ after the function traverses $(N_G(u) \cap V(H_1)) \backslash F$. Next, vertex x_2 is found from $(N_G(x_1) \cap V(H_2)) \backslash F$. The time complexity in this case is $\mathrm{O}(k) < O(k(G))$. Second, the path is (u, y_1, y_2, y_3). The function traverses $(N_G(u) \cap V(H_1)) \backslash F$. The node is set on the same type A switch as u to y_1 and the time complexity to $O(1)$. Then, traversing $(N_G(y1) \cap V(H_1)) \backslash (F \cup u)$ sets the point connected with other subgraphs as y_2, and the time complexity is less than $O(\kappa(G))$. Finally, $(N_G(y_2) \cap V(H_2)) \backslash F$ is traversed and node y_3 selected. In this case, the time complexity is less than $O(\kappa(G))$. Third, the path of u_2 (u, z_1, z_2, z_3, z_4) is the same as the path of u_1 (u, y_1, y_2, y_3, y_4). In summary, the time complexity of the $PATHSEQ(u, F, H_1, H_2)$ function is $O(\kappa(G)^2)$ combined with u_1 and u_2.

Next, we analyze $BCCFTP(u, v, \alpha, \beta, F, BCC(n, k))$. $BCCFTP$ is divided into two cases, including $\alpha \neq \beta$ and $\alpha = \beta$.

In the case of $\alpha \neq \beta$, if $|F_\alpha| \geq k$, Algorithm $BCCFTP$ will first obtain a fault-free path P_1 from u to a vertex s in $G - BCC^\alpha_{n,k}$ by $PATHSEQ(u, F, BCC^\alpha_{n,k}, G - BCC^\alpha_{n,k})$. The time complexity of this part is $O(\kappa(G)^2)$. Next, Algorithm $BCCFTP$ will return P_1 if $s = v$; then, the time to obtain the path from u to v in $G - F$ is $O(\kappa(G)^2)$. Otherwise, Algorithm $BCCFTP$ will return (P_1,$BCTP(s, v, G - BCC^\alpha_{n,k}, F \backslash F_\beta)$); then, the time to run $BCTP(s, v, G - BCC^\alpha_{n,k}, F \backslash F_\beta)$ is $\mathrm{O}(kn\kappa^1(G))$ and the time to obtain a path (P_1,$BCTP(s, v, G - BCC^\alpha_{n,k}, F \backslash F_\beta)$) by connecting path P_1 and $BCTP$ is $\mathrm{O}(1)$. Thus, the time to obtain the path from u to v in $G - F$ is $\mathrm{O}(\kappa(G)^2 + kn\kappa^1(G) + 1)$ = $\mathrm{O}(kn\kappa^1(G))$. Furthermore, for subcases $|F_\beta| \geq k$, $|F_\alpha| < k$, and $|F_\beta| < k$, the time to obtain the path from u to v in $G - F$ is $\mathrm{O}(kn\kappa^1(G))$, which is similar to that discussed in the case of $|F_\alpha| \geq k$.

In the case of $\alpha = \beta$, if $|F_\alpha| \leq k$, Algorithm $BCCFTP$ will firstly obtain a fault-free path from u to v in $BCC^\alpha_{n,k}$ by PATHSEQ $(u, F, BCC^\alpha_{n,k}, F_\alpha)$. The time complexity of this part is $\mathrm{O}(\kappa(G)^2)$. Then, Algorithm BCCFTP will first obtain a fault-free path P_1 from u to a vertex s and path P_2 from v to a vertex t in $BCC^\alpha_{n,k}$ in the time of $\mathrm{O}(\kappa(G)^2)$. If P_1 and P_2 have a common vertex w, Algorithm $BCCFTP$ will return (Path(P_1, u, w), Path$(P_2^{-1}, w, v))$. Since the time to obtain Path(P_1, u, w) is O(1), the time complexity is O$(2 * \kappa(G)^2 + O(1))$ = O$(\kappa(G)^2)$. If P_1 and P_2 have no common vertex, the function will run $BCTP(s, t, G - BCC_{n,k}^{\alpha}, F \backslash F_{\alpha})$ to construct a fault-free path from s to t. The time complexity of this part is O$(kn\kappa^1(G))$. Therefore, the time to obtain the path from u to v in G is O$(2 * \kappa(G)^2 + kn\kappa^1(G) + 2 * 1)$ = O$(kn\kappa^1(G))$.

In summary, the time complexity of Algorithm $BCCFTP$ is O$(kn\kappa^1(G))$.

6. Discussion and Conclusions

As the infrastructure of cloud computing, data center networking has become a hot topic in recent years. As an important DCN model, BCCC can support millions of servers with excellent communication characteristics and provide good fault tolerance. In this paper, a point disjoint path algorithm with time complexity $O(nk)$ is proposed. The algorithm can give n vertex disjoint paths between any two vertices. Multiple disjoint paths can enhance routing performance and improve network reliability. Then, we prove the condition that each vertex has at least one fault-free neighbor, and that the restricted connectivity of BCCC is $2\kappa(G) - k - 1$ and $2\kappa(G) - n$, when $n < k + 1$ and $n \geq k + 1$, respectively. Restricted connectivity can be obtained at almost twice the traditional connectivity. The larger the connectivity scale, the better the fault-tolerant performance of the data center network. Finally, we give the $O(kn\kappa^1(G))$ trouble-free algorithm with limited connectivity. This algorithm can obtain the fault-free path between any two different fault-free vertices. The premise is that each vertex has at least one fault-free neighbor in the BCCC.

Our algorithm is proposed based on the condition that each fault-free vertex has one fault-free neighbor. In the majority of cases, each vertex has more than one fault-free neighbor in large interconnected networks. In the future, the h-restricted connectivity $(h \geq 2)$ of BCCC will be studied, which is a guarantee for accommodating more failed servers when the network is large enough. Fault-free algorithms based on h-restricted connectivity can ensure that the network allows more fault nodes. In addition, many other network connectivity limitations in DCN have not been studied, such as Ficonn[26], HCN and BCN [27]. This, together with the design of related routing algorithms, may be worth further investigation. In the meantime, most networks only consider server failures. In fact, both switches and links could fail. There are currently few studies on switch and link failures. Therefore, the fault tolerance of switches and links in the network can be analyzed, and corresponding fault-tolerant routing algorithms can be designed to ensure the connectivity of the network.

Author Contributions: Conceptualization, J.L. and X.D.; methodology, J.L. and H.L.; software, J.L. and Z.H.; validation, J.L. and X.D.; formal analysis, J.L. and Z.H.; investigation, J.L. and X.D.; data curation, J.L. and H.L.; writing—original draft preparation, J.L. and X.D.; writing—review and editing, J.L., X.D., H.L. and Z.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Special Project for Key R&D and the Promotion of Science, Technology Department of Henan Province (202102210327, 222102210272, 222102210052, 222102210007, 222102210062 and 212102210094), the Key Technology Research and Development Project of Henan Province under Grant (222102210055), the Postgraduate Education Reform and Quality Improvement Project of Henan Province under Grant (YJS2022JD26), the Major Science and Technology Projects of Henan Province, the Major Public Welfare Projects of Henan Province (201300210400) and Kaifeng Science and Technology Development Plan (2201010).

Data Availability Statement: Not applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ghemawat, S.; Gobioff, H.; Leung, S.T. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003; pp. 29–43.
2. Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W.C.; Wallach, D.A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R.E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst. (TOCS)* **2008**, *26*, 1–26. [CrossRef]
3. Isard, M.; Budiu, M.; Yu, Y.; Birrell, A.; Fetterly, D. Dryad: Distributed data-parallel programs from sequential building blocks. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, Lisboa, Portugal, 21–23 March 2007; pp. 59–72.
4. Li, Z.; Guo, Z.; Yang, Y. BCCC: An expandable network for data centers. In Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Marina del Rey, CA, USA, 20–21 October 2014; pp. 77–88.
5. Castillo, A.C. BCube Connected Crossbar and GBC3 Network Architecture: An Overview. In Proceedings of the 2022 31st Conference of Open Innovations Association (FRUCT), Helsinki, Finland, 27–29 April 2022; pp. 37–44.
6. Li, X.Y.; Lin, W.; Liu, X.; Lin, C.K.; Pai, K.J.; Chang, J.M. Completely independent spanning trees on BCCC data center networks with an application to fault-tolerant routing. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 1939–1952. [CrossRef]
7. He, X.; Zhang, Q.; Han, Z. The Hamiltonian of Data Center Network BCCC. In Proceedings of the 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing,(HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), Omaha, NE, USA, 3–5 May 2018; pp. 147–150.
8. Han, Z.; Zhang, W. A summary of the BCCC data center network topology. In Proceedings of the 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), Omaha, NE, USA, 3–5 May 2018; pp. 270–272.
9. Harary, F. Conditional connectivity. *Networks* **1983**, *13*, 347–357. [CrossRef]
10. Esfahanian, A.H. Generalized measures of fault tolerance with application to n-cube networks. *IEEE Trans. Comput.* **1989**, *38*, 1586–1591. [CrossRef]
11. Li, X.; Zhou, S.; Ma, T.; Guo, X.; Ren, X. The h-Restricted Connectivity of a Class of Hypercube-Based Compound Networks. *Comput. J.* **2022**, *65*, 2528–2534. [CrossRef]
12. Cheng, D. The h-restricted connectivity of balanced hypercubes. *Discret. Appl. Math.* **2021**, *305*, 133–141. [CrossRef]
13. Li, X.; Zhou, S.; Guo, X.; Ma, T. The h-restricted connectivity of the generalized hypercubes. *Theor. Comput. Sci.* **2021**, *850*, 135–147. [CrossRef]
14. Liu, X.; Meng, J. The k-restricted edge-connectivity of the data center network DCell. *Appl. Math. Comput.* **2021**, *396*, 125941. [CrossRef]
15. Lin, L.; Huang, Y.; Wang, X.; Xu, L. Restricted connectivity and good-neighbor diagnosability of split-star networks. *Theor. Comput. Sci.* **2020**, *824*, 81–91. [CrossRef]
16. Wang, S. The r-Restricted Connectivity of Hyper Petersen Graphs. *IEEE Access* **2019**, *7*, 109539–109543. [CrossRef]
17. Ma, T.; Wang, J.; Zhang, M. The restricted edge-connectivity of Kronecker product graphs. *Parallel Process. Lett.* **2019**, *29*, 1950012. [CrossRef]
18. Wu, S.; Fan, J.; Cheng, B.; Yu, J.; Wang, Y. Connectivity and constructive algorithms of disjoint paths in dragonfly networks. *Theor. Comput. Sci.* **2022**, *922*, 257–270. [CrossRef]
19. Kern, W.; Martin, B.; Paulusma, D.; Smith, S.; van Leeuwen, E.J. Disjoint paths and connected subgraphs for H-free graphs. *Theor. Comput. Sci.* **2022**, *898*, 59–68. [CrossRef]
20. Hadid, R.; Villain, V. A Self-stabilizing One-To-Many Node Disjoint Paths Routing Algorithm in Star Networks. In *Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 186–203.
21. Pushparaj, J.; Soumya, J.; Veda Bhanu, P. A link fault tolerant routing algorithm for mesh of tree based network-on-chips. In Proceedings of the 2019 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS), Rourkela, India, 16–18 December 2019; pp. 181–184.
22. Nehnouh, C.; Senouci, M. A New Fault Tolerant Routing Algorithm for Networks on Chip. *Int. J. Embed.-Real-Time Commun. Syst. (IJERTCS)* **2019**, *10*, 68–85. [CrossRef]
23. Zhang, Y.; Fan, W.; Han, Z.; Song, Y.; Wang, R. Fault-tolerant routing algorithm based on disjoint paths in 3-ary n-cube networks with structure faults. *J. Supercomput.* **2021**, *77*, 13090–13114. [CrossRef]
24. Thuan, B.T.; Ngoc, L.B.; Kaneko, K. A stochastic link-fault-tolerant routing algorithm in folded hypercubes. *J. Supercomput.* **2018**, *74*, 5539–5557. [CrossRef]
25. Ipek, A.; Tosun, S.; Ozdemir, S. HAFTA: Highly adaptive fault-tolerant routing algorithm for two-dimensional network-on-chips. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e6378. [CrossRef]
26. Li, D.; Guo, C.; Wu, H.; Tan, K.; Zhang, Y.; Lu, S. FiConn: Using backup port for server interconnection in data centers. In Proceedings of the IEEE INFOCOM 2009, Rio de Janeiro, Brazil, 19–25 April 2009; pp. 2276–2285.
27. Guo, D.; Chen, T.; Li, D.; Li, M.; Liu, Y.; Chen, G. Expandable and cost-effective network structures for data centers using dual-port servers. *IEEE Trans. Comput.* **2012**, *62*, 1303–1317.