Almost Optimal Algorithms for Token Collision in Anonymous Networks

Sirui Bai
State Key Laboratory for Novel Software Technology, Nanjing University, China

Xinyu Fu
State Key Laboratory for Novel Software Technology, Nanjing University, China

Xudong Wu
State Key Laboratory for Novel Software Technology, Nanjing University, China

Penghui Yao
State Key Laboratory for Novel Software Technology, Nanjing University, China
Hefei National Laboratory, China

Chaodong Zheng
State Key Laboratory for Novel Software Technology, Nanjing University, China

Abstract

In distributed systems, situations often arise where some nodes each holds a collection of *tokens*, and all nodes collectively need to determine whether all tokens are distinct. For example, if each token represents a logged-in user, the problem corresponds to checking whether there are duplicate logins. Similarly, if each token represents a data object or a timestamp, the problem corresponds to checking whether there are conflicting operations in distributed databases. In distributed computing theory, unique identifiers generation is also related to this problem: each node generates one token, which is its identifier, then a verification phase is needed to ensure that all identifiers are unique.

In this paper, we formalize and initiate the study of *token collision*. In this problem, a collection of k tokens, each represented by some length-L bit string, are distributed to n nodes of an *anonymous* CONGEST network in an arbitrary manner. The nodes need to determine whether there are tokens with an identical value. We present near optimal deterministic algorithms for the token collision problem with $\tilde{O}(D + k \cdot L/\log n)$ round complexity, where D denotes the network diameter. Besides high efficiency, the prior knowledge required by our algorithms is also limited. For completeness, we further present a near optimal randomized algorithm for token collision.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Token collision, anonymous networks, deterministic algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.4

Related Version *Full Version*: `https://arxiv.org/abs/2408.10519`

Funding This work is supported by the National Natural Science Foundation of China (Grant No. 62332009, 62172207, 12347104), the Department of Science and Technology of Jiangsu Province of China (Grant No. BK20211148) and Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302901).

1 Introduction

Imagine the following scenario: a group of servers is hosting an online-banking, online-gaming, or online-exam service; for security reasons, users are not allowed to log into multiple servers simultaneously. If we interpret each logged-in user as a *token*, the servers need to check whether all active tokens are distinct. Similar problems could also arise in distributed database systems. For example, in some distributed databases, optimistic concurrency control schemes are employed to increase concurrency and performance [17]. Motivation for such schemes is the observation that system clients are unlikely to access the same object concurrently. Nonetheless, before the system commit clients' transactions, verification must be performed to ensure that all read and write operations are disjoint or that no operations occur at the same time, otherwise a rollback is necessary. In this setting, tokens represent data objects or timestamps [19, 20].

Apart from above practical scenarios, detecting colliding tokens is also important from a theoretical perspective as we can interpret identifiers as tokens. Specifically, it is well-known that a number of fundamental distributed computing tasks, such as coloring, leader election, bipartiteness testing, and planarity testing, are impossible to resolve deterministically in anonymous networks [3, 15, 21]. Hence, unique identifiers generation becomes an important primitive for anonymous networks, as it breaks the symmetry within the network, thus making the aforementioned tasks possible. Moreover, the lengths of these identifiers could affect the performance of corresponding algorithms; examples include renaming algorithms, Linial's classical n-to-Δ^2 coloring algorithm [21], recent deterministic network decomposition algorithms by Ghaffari et al. [11], and recent MST algorithm focusing on energy complexity [5]. A plausible approach for generating unique identifiers in anonymous networks is to employ randomness: for instance, each of the n nodes generates an identifier by sampling $\Theta(\log n)$ uniform random bits; by the birthday paradox, all identifiers are distinct with probability at least $1 - 1/n$. However, this is a Monte Carlo algorithm that is subject to error. If we seek a Las Vegas (i.e., zero-error) algorithm for generating unique identifiers in anonymous networks, a deterministic algorithm for detecting colliding identifiers (i.e., tokens) is necessary: nodes repeatedly run a (Monte Carlo) randomized identifiers generation algorithm and use a deterministic algorithm to check whether the generated identifiers are unique.

Despite the various applications for *token collision*, somewhat surprisingly, this problem has not been explicitly studied in the context of distributed computing to the best of our knowledge. In this paper, we initiate the study of this generic distributed computing task, and we begin by giving a definition for it:

▶ **Definition 1 (Token Collision).** *Assume that there are k tokens, each having a value represented by a length-L binary string. Consider a distributed system consisting of n nodes. The k tokens are divided into n collections, some of which may be empty. Each of the n nodes is assigned one collection as input. In the* token collision *problem, the nodes need to determine whether there are tokens with identical value.*

We focus on understanding the time complexity of token collision in anonymous networks. The reason for considering anonymous networks is two-fold. First, if nodes in a distributed system already have unique identifiers, then token collision (or almost any distributed computing task) could be resolved by first electing the node with the smallest identifier as the leader, then aggregate necessary information to the leader, and finally let the leader locally compute the result and disseminate the result to the rest of the network.[1] Second, anonymous networks also arise in real-world scenarios. For example, the nodes in a distributed system (e.g., sensor networks) may be indistinguishable since they are fabricated in a large-scale industrial process, in which equipping every node with a unique identifier is not economically feasible (e.g., MAC addresses are not necessarily unique nowadays). In other cases nodes may not wish to reveal their identities out of privacy or security concerns.

[1] Nonetheless, our lower bounds for token collision hold even for named networks, and our algorithms nearly match these lower bounds. Thus, anonymity does not make token collision harder.

We consider standard CONGEST model [21] in distributed computing. A CONGEST network is described by a graph $G = (V, E)$ with $|V| = n$ nodes being processors with unlimited computational power (we do not exploit this ability in this paper) and edges being communication channels with bounded bandwidth. Specifically, we assume that any message sent through a channel cannot exceed $\Theta(\log n)$ bits. To simplify presentation, we often use $B = \Theta(\log n)$ to denote this bandwidth limitation. Processors exchange messages synchronously, round-by-round along the channels. When proving impossibility results for the token collision problem, we also consider an alternate model known as the LOCAL model [18], where communication channels have unbounded bandwidth.

1.1 Results and contribution

Deterministic scenario. We first consider the case where the tokens are not too large – particularly, every token can fit into one message. In this scenario, we offer a deterministic algorithm that works so long as every node knows the exact value of n or k.

▶ **Theorem 2** (**Deterministic Upper Bound, Part 1**). *In an n-node anonymous CONGEST network with diameter D, for any instance of the token collision problem in which k tokens are encoded by length-L bit strings, if every node knows the exact value of n or k, then there exists an $O(D + k \cdot L/\log n)$-round deterministic algorithm when $L = O(\log n)$.*

The above theorem implies when $L = \Theta(\log n)$, token collision can be resolved within $O(D+k)$ rounds. As a result, one could easily derive a Las Vegas unique identifiers generation algorithm with $O(n)$ expected runtime. On the other hand, for problem instances where tokens are small – $L = o(\log n)$ in particular, the runtime of the algorithm is $O(D) + o(k)$.

At a high level, our algorithm tries to find the token(s) that have the global minimum value and selects the node(s) that own(s) such token(s) as leader(s). Critically, our algorithm may elect multiple nodes as leaders, so it does not solve the leader election problem. (In fact, leader election cannot be solved deterministically in our setting.) Nevertheless, alongside this election process, BFS-trees will be built with these leaders being the roots, thus the network graph becomes a forest logically. Then, by convergecasting [4] tokens within each tree and computing the size of each tree, root nodes can correctly determine the result. Similar ideas have been used in the design of Las Vegas leader election algorithms, but the analysis of our deterministic algorithm is more challenging, see Section 1.2 for more discussion.

We then extend our algorithm to the scenario where each token cannot fit into one message, this could occur in applications like plagiarism checking in which each token is a text segment. In this case, a simple solution is to divide each token into $\Theta(L/\log n)$ parts, and use multiple rounds to simulate one round of our above algorithm. However, the resulting algorithm would have a round complexity of $O((D+k) \cdot L/\log n)$, which is too large. Instead, we devise a variant that uses pipelining techniques and extend the analysis accordingly. The following theorem states the time complexity of this variant.

▶ **Theorem 3** (**Deterministic Upper Bound, Part 2**). *In an n-node anonymous CONGEST network with diameter D, for any instance of the token collision problem in which k tokens are encoded by length-L bit strings, if every node knows the exact value of n or k, then there exists an $O(D \cdot \max\{(\log(L/\log n))/\log n, 1\} + k \cdot L/\log n)$-round deterministic algorithm when $L = \omega(\log n)$.*

To complement the algorithmic results, we have also established a lower bound on the round complexity of the token collision problem.

▶ **Theorem 4** **(Deterministic Lower Bound).** *In an n-node anonymous CONGEST network with diameter D, there are instances of the token collision problem in which k tokens are encoded by length-L bit strings such that, any deterministic algorithm requires $\Omega(D + k \cdot (L - \log k + 1)/\log n)$ rounds to solve it when $2^L \geq k$.*

It is easy to verify, if $L \geq (1+\delta)\log k$ for some constant $\delta > 0$, then the round complexity of our algorithm is tight when $L = O(\log n)$, and near-optimal (within multiplicative $\log(L/\log n)/\log n$ factor) when $L = \omega(\log n)$. We also note that the assumption of $2^L \geq k$ is without loss of generality, as otherwise collision must occur. To obtain the lower bound, we reduce a variant of the set-disjointness problem in the study of two-party communication complexity to the token collision problem.

Another advantage of our algorithm is that it requires little prior knowledge. Beside input tokens, each node only needs to know the value of n *or* the value of k. In particular, nodes do not need to know the network diameter D. In fact, we can prove via an indistinguishability argument that without any global knowledge, deterministic token collision detection is impossible. (Nonetheless, what is the minimal prior knowledge required remains to be an interesting open question.)

▶ **Theorem 5** **(Impossibility Result).** *In the anonymous LOCAL model, if every node has no knowledge regarding the network graph except being able to count and communicate over adjacent links locally, and if every node also has no knowledge regarding the tokens except the ones given as local input, then there is no deterministic algorithm that solves the token collision problem.*

Randomized scenario. We also investigate the randomized round complexity of token collision. At the upper bound side, selecting a unique leader can be easily achieved with desirable probability when randomness is allowed. Then a random hash function is employed to reduce the bit-length of tokens if they are too large, without increasing the probability of collision significantly in case tokens are distinct. Finally, with a convergecast process similar to the deterministic algorithm, the unique leader can collect all tokens and determine whether there are collisions. At the lower bound side, we again employ the strategy of reduction, and utilize existing results on the hardness of randomized set-disjointness to obtain the desired result. Our findings for the randomized scenario are summarized below; see Appendix B for more details. Note that in contrast with the deterministic setting, the length of the token L no longer appears in the lower bound, and L's impact on the upper bound is also limited.

▶ **Theorem 6** **(Randomized Upper and Lower Bound).** *Consider an n-node anonymous CONGEST network with diameter D. For any instance of the token collision problem with k tokens, if every node knows the exact value of n or k, then there exists a randomized algorithm that solves it in $O(D \cdot \max\{(\log(\log k/\log n))/\log n, 1\} + k \cdot \log k/\log n + L/\log n)$ rounds with probability at least $1 - 1/k$. On the other hand, assuming $2^L \geq k$, there are instances of the token collision problem with k tokens such that any randomized algorithm that succeeds with probability at least $2/3$ requires $\Omega(D + k/\log n)$ rounds.*

1.2 Related work and discussion

Though token collision has not been explicitly studied in distributed computing, similar problems have been investigated elsewhere. For example, element distinctness, which decides whether a given set of elements are distinct, has been extensively studied in the context of query complexity. Specifically, linear lower bounds were proved for deterministic and randomized algorithms [7, 12]. A sublinear quantum algorithm was proposed by Buhrman et al. [9], which applies $O(n^{3/4})$ quantum queries. The upper bound was later improved to $O(n^{2/3})$ by Ambainis using quantum walk [2], and matched the lower bound given by Aaronson and Shi [1]. To the best of our knowledge, this paper is the first one that studies token collision in classical distributed computing models, focusing on round complexity.

Token collision is also related to leader election, a classical and fundamental distributed computing primitive, in several aspects.

On the one hand, the design of Las Vegas leader election algorithms [14, 23, 10] share similar ideas with ours. In those algorithms, usually nodes first randomly generate identifiers, then the node with the smallest identifier is elected as the leader if that identifier is owned by a single node, otherwise the process restarts. As can be seen, the problem of checking whether the smallest identifier is unique is a variant of the token collision problem. Indeed, the routine developed by Tel in [23] for this checking procedure is very similar to our algorithm. Nonetheless, the analysis in our setting is more involved: in the context of Las Vegas leader election, restart when the smallest identifier is unique is fine (i.e., false negative is fine), yet in our context this is unacceptable. In fact, proving such false negative will not occur is highly non-trivial (see Lemma 12 in Section 4). Moreover, our algorithm can handle the scenario that tokens are of arbitrary size, making it more generic.

On the other hand, as mentioned earlier, with a unique leader almost any distributed computing problem can be solved. This observation raises the question that whether defining and studying token collision is necessary. We believe the answer is positive. First, the leader election approach is not necessarily better. Taking the unique identifiers generation problem as an example, the approach of "first elect a leader and then let the leader aggregate and check whether the generated identifiers are unique" share same round complexity with our algorithm. Second, and more importantly, in situations where leader election is infeasible (e.g., deterministic leader election in anonymous networks, randomized leader election that always terminates in anonymous rings of unknown size) [24], our algorithm still works with deterministic correctness and time complexity guarantees.

2 Preliminary

In this section, we briefly introduce some known results on the communication complexity of the set-disjointness problem as it is used in our lower bound proof.

Communication complexity was introduced by Yao [25], which is nowadays a versatile method to prove lower bounds in distributed computing. In the two-party communication complexity model, two players Alice and Bob, respectively, receive $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ as input and need to compute $f(x, y)$, where $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is a two-argument function. The communication complexity of f is the minimum number of bits Alice and Bob need to exchange to compute $f(x, y)$ for any input x and y.

The problem of set-disjointness (denoted as DISJ) is one of the most well-studied problems in communication complexity [16, 22, 6], where Alice and Bob are given a set, respectively, and they need to decide whether their sets are disjoint. In this paper, we are interested in a variant of set-disjointness: $\mathrm{DISJ}^p_q : \binom{[p]}{q} \times \binom{[p]}{q} \to \{0, 1\}$. Alice is given a set $S \subseteq [p]$ and Bob is given a set $T \subseteq [p]$, where $|S| = |T| = q$. They aim to determine whether the two sets are disjoint, that is, $\mathrm{DISJ}^p_q(S, T) = 1$ iff $S \cap T = \emptyset$. The communication complexity of DISJ^p_q is established by Håstad and Widgerson [13].

▶ **Fact 7** ([13]). *For every $q \le p/2$, $D(\mathrm{DISJ}^p_q) = \Omega(\log \binom{p}{q})$ and $R_{1/3}(\mathrm{DISJ}^p_q) = \Omega(q)$. Here, $D(\mathrm{DISJ}^p_q)$ denotes the deterministic communication complexity of problem DISJ^p_q, and $R_{1/3}(\mathrm{DISJ}^p_q)$ denotes the randomized communication complexity of problem DISJ^p_q with the probability of error being at most $1/3$.*

3 The Deterministic Algorithm

In this section, we focus on the most common scenario where each token can be fitted into one message (that is, $L = O(\log n)$). We will extend our algorithm to other settings later.

Broadly speaking, our algorithm can be divided into two parts: the first part concerns with building rooted BFS-trees, while the second part concerns with calculating the size of the BFS-trees and aggregating tokens at the roots for decision-making. Although the high-level idea of our algorithm is not complicated, implementing it correctly and efficiently is non-trivial, especially in the setting where nodes only have limited global knowledge.

We now describe the algorithm in detail. (Complete pseudocode of the algorithm is provided in Appendix A.)

Build BFS-tree(s). Initially, each node v sets its identifier to be the smallest token it received as input, or a special symbol if v received no token. Then, it attempts to construct a BFS-tree rooted at itself by broadcasting its identifier to its neighbors in each round. Whenever v receives a smaller identifier from some neighbor u, it updates its identifier to match that of neighbor u. Moreover, it designates u as its parent and sends a notification to its parent in all subsequent rounds. As a result, whenever v changes its identifier to that of some neighbor u's, node v is appending the BFS-tree rooted at itself to the BFS-tree that includes u. We note that each node v uses a variable $\texttt{rid}_v$ to store its identifier. Intuitively, $\texttt{rid}_v$ stores the root's identifier of the BFS-tree that v belongs to.[2] We also note that each node v uses an integer $\texttt{p}_v \in [\Delta_v]$ to store its parent, where Δ_v is the degree of v. That is, each node v locally labels each incident edge with a unique integer in $[\Delta_v]$, and uses the edge label as its local identity for the node at the other endpoint of the edge.

When node v discovers that all of its neighbors share the same identifier as itself, it attempts to ascertain whether the BFS-tree rooted at itself is fully constructed. To this end, note that the BFS-tree rooted at node v consists of node v and the BFS-trees rooted at its children. Therefore, our algorithm's criterion for node v to confirm that the BFS-tree rooted at itself is fully constructed is: all v's neighbors share v's identifier and the BFS-trees rooted at its children are fully constructed. To implement this idea, each node v stores a boolean variable $\texttt{f}_v$ to indicate whether the BFS-tree construction process is completed, and $\texttt{f}_v$ is sent to v's parent in each round. Initially $\texttt{f}_v$ is $false$, and $\texttt{f}_v$ becomes $true$ if: (1) all v's neighbors share identical identifier as v; and (2) each child u of v has $\texttt{f}_u = true$ or v has no children (that is, v is a leaf node).

Lastly, if node v determines that the BFS-tree rooted at itself is fully constructed and it does not have a parent, then it broadcasts a termination signal to its neighbors once and stops the BFS-tree building procedure. The node v will then proceed to the second stage of the algorithm. On the other hand, whenever a node receives a termination signal, it also stops its BFS-tree building procedure, broadcasts this signal to all neighbors once, and then proceed to the second stage of the algorithm. During algorithm execution, each node v uses a boolean variable $\texttt{build}_v$ to maintain this signal: $\texttt{build}_v$ is initially $true$, and will be set to $false$ when v's BFS-tree building procedure is done. Notice that if there are multiple BFS-trees being constructed simultaneously, after the first one completes, the flooding mechanism of the termination signal may stop the remaining ones from being completed. Nonetheless, such disruption is fine: the existence of multiple BFS-trees implies there are token collisions, and our algorithm can correctly detect this later.

[2] This is merely an "intuition" and not always true during BFS-tree construction, as identifiers are propagating gradually and tree shape may change frequently.

Detect token collision. The token-collision detection procedure has two main tasks: compute BFS-tree's size and aggregate tokens. A node only starts this procedure if itself and all its neighbors have terminated the BFS-tree construction procedure.

To determine the size of the BFS-tree rooted at itself, node v first identifies its children. It filters out the neighbors that have the same identifier as itself and have designated node v as their parent. Node v uses $\texttt{chi}_v$ to store this set of children. If node v finds that all of its children in $\texttt{chi}_v$ have already computed the size of their respective BFS-trees, then node v can calculate the size of the BFS-tree rooted at itself. This is done by summing up the sizes of the BFS-trees rooted at its children and adding one to account for node v itself. During algorithm execution, each node v uses $\texttt{cnt}_v$ to track the size of the BFS-tree rooted at itself. Initially $\texttt{cnt}_v$ is set to a special symbol $\perp$. Later when v has finished counting, $\texttt{cnt}_v$ becomes an integer.

It remains to aggregate the tokens. In each round, after node v receives all messages (which may include tokens from its children), if v has a parent u and the token list of v is not empty, then v ejects one token from its token list and sends that token to u in the next round. Node v also needs to tell its parent u whether all tokens in the BFS-tree rooted at v has already been transferred to u. To this end, in each round, after v has received all messages, if the token list of v is empty and every child of v indicates all tokens have already been transferred to v, then v concludes that all tokens in the BFS-tree rooted at itself has already been transferred to its parent. It will inform its parent u about this in the next round. During algorithm execution, each node v uses $\boldsymbol{x}^v$ to store its token list and uses $\texttt{ele}_v$ to denote the token that v intends to send to its parent. We note that $\texttt{ele}_v$ is set to $\perp$ when v's token list is empty (that is, $|\boldsymbol{x}^v| = 0$) and every child w of v indicates all tokens in the subtree rooted at w has been transferred to v (that is, w tells v $\texttt{ele}_w = \perp$); and $\texttt{ele}_v$ is set to $\top$ when v's token list is empty but some child w of v indicates there still are tokens pending to be transferred to v (that is, w tells v $\texttt{ele}_w = \top$).

If node v does not have a parent, it must be the root of some entire BFS-tree and is responsible for deciding the result of token collision. To this end, once v has obtained the size of the BFS-tree rooted at itself and all its children signal that the tokens have been transferred to v, it determines the result of token collision as follows. In the case that all nodes know the exact value of n, if the size of the BFS-tree rooted at v equals n and no token collision is found in the token list of v, then v can confirm the non-existence of token collision. Otherwise, a token collision must exist. In the case that all nodes know the exact value of k, if the size of the token list of v equals k and no token collision is found, then v can confirm the non-existence of token collision. Otherwise, a token collision must exist. Node v uses a boolean variable $\texttt{res}_v$ to store the result. It will broadcast the result to all its neighbors once in the next round and then halt. Upon receiving the result, every node also broadcasts the result to its neighbors once in the next round and then halts.

4 Analysis of the Deterministic Algorithm

In this section, we show the correctness of our algorithm and analyze its running time. Omitted proofs are provided in the full version of the paper.

4.1 Correctness

We begin with the correctness guarantees: if all nodes know n or k, then all nodes return an identical and correct result on whether there are collisions among the k tokens.

To prove the above claim, we first argue the correctness of our BFS-tree construction procedure. Specifically, we intend to show that our algorithm always maintains a directed forest $G' = (V, E')$ where a directed edge $(v, u) \in E'$ if node v assigns node u as its parent. To this end, we introduce the notion of *identifier-induced graph.*

▶ **Definition 8 (Identifier-induced Graph).** *At the end of any round, define directed graph* $G' = (V, E')$ *as the* identifier-induced graph *in the following way:* V *is the node set of the network graph, and a directed edge* $(v, u) \in E'$ *if* v *assigns* u *as its parent.*

To show that the identifier-induced graph is a forest, we begin with the following observation. Recall that each node v uses variable `rid` to store its identifier. Intuitively, this lemma holds since node v only updates the value of $\texttt{rid}_v$ to the value of $\texttt{rid}_u$ and sets its parent pointer to u when v receives $\texttt{rid}_u$ from some neighbor u with $\texttt{rid}_u < \texttt{rid}_v$.

▶ **Lemma 9.** *At the end of any round, for any directed path in the identifier-induced graph, the identifiers of the nodes along the directed path are non-increasing.*

Then, we can show the identifier-induced graph is a directed forest containing one or more rooted trees. To prove the lemma, the key is to show there are no directed cycles in the identifier-induced graph, which can be done by induction on round number.

▶ **Lemma 10.** *At the end of any round, the identifier-induced graph is a directed forest in which every weakly connected component is a rooted tree. In particular, in each tree, the unique node with no parent is the root of that tree.*[3]

Next, we show an important property regarding the rooted trees in the identifier-induced graph. Intuitively, it states that within each such tree, nodes may have different identifiers, but for any subtree within the tree, the nodes that have identical identifiers with the root of the subtree are connected and are at the "top" of the subtree.

▶ **Lemma 11.** *At the end of any round, for any node* r*, within the subtree rooted at node* r *in the identifier-induced graph, the subgraph induced by the nodes having identical identifier with node* r *is also a tree rooted at node* r*.*

The following key lemma shows that when there are no token collisions, the BFS-tree building procedure constructs a single rooted tree containing all nodes. Though the claim seems straightforward, proving it rigorously turns out to be highly non-trivial.

▶ **Lemma 12.** *If there are no token collisions, then after all nodes quit the* BFS-Tree-Building *procedure – that is, after each node* v *sets* $\texttt{build}_v = false$*, the identifier-induced graph contains a single tree rooted at the node having the minimum token as input, and all nodes in that tree have identical identifier.*

Proof sketch. Throughout the proof, assume there are no token collisions. For each node v, let $\texttt{id}_v$ denote the minimum token that v received as input. Let $v_{\min}$ denote the unique node having the smallest input token. For any two nodes $u, v \in V$, let $\texttt{dist}(u, v)$ denote the distance between u and v in the network graph G. Define $d = \max_{v \in V} \texttt{dist}(v, v_{\min})$. For any node v, let d_v denote the distance between node v and the nearest node u with $\texttt{id}_u$ smaller than v. That is, $d_v = \min_{u \in V, \texttt{id}_u < \texttt{id}_v} \texttt{dist}(u, v)$. We set $d_{v_{\min}} = +\infty$.

We make the following three claims and prove their correctness via induction on rounds. These claims highlight the key properties our BFS-tree building procedure can enforce.

[3] A weakly connected component of a directed graph is a connected component of the graph when ignoring edge directions.

1. No node quits the BFS-tree building procedure within d rounds. Formally, for any $0 \le i \le d$, each node v has $\texttt{build}_v = \mathit{true}$ by the end of round i.
2. For any node v, any $0 \le i \le d$, let node $\hat{u}$ be the unique node that has minimum $\texttt{id}$ among all nodes u with $\texttt{dist}(u,v) \le i$. At the end of round i, we have $\texttt{rid}_v = \texttt{id}_{\hat{u}}$. Formally, at the end of round i, it holds that $\texttt{rid}_v = \min_{u \in V, \texttt{dist}(u,v) \le i} \texttt{id}_u$.
3. For any node v, any $0 \le i \le d$, at the end of round i, one of the following cases holds.
   - Case I: $d_v > 2i$. The nodes with distance at most i to v form a height-i tree rooted at v in the identifier-induced graph. Moreover, for any node u with distance i to v, for any node w on the directed path from u to v in the rooted tree, it holds that $\texttt{f}_w = \mathit{false}$.
   - Case II: $i < d_v \le 2i$. There is a tree rooted at v in the identifier-induced graph. Let node w denote the unique node with distance d_v to node v that has the smallest $\texttt{id}$. There exists a node u with $\texttt{dist}(u,v) = d_v - i - 1$ and $\texttt{dist}(u,w) = i+1$ such that $\texttt{f}$ is false for all nodes along the length-$(d_v - i - 1)$ directed path from u to v in the identifier-induced graph.
   - Case III: $d_v \le i$. Node v has a parent in the identifier-induced graph.

The claims above easily lead to the lemma. By Item 1, no node will quite the BFS-Tree-Building procedure within d rounds. By Item 2, at the end of round d, all nodes have identical $\texttt{rid}$, which is $\texttt{id}_{v_{min}}$. By Case I of Item 3, at the end of round d, all nodes form a single tree rooted at v_{min} in the identifier-induced graph. Moreover, no node will ever change parent or $\texttt{rid}$ later. ◀

We now proceed to argue the correctness of the token aggregation process, which will lead to the correctness of our entire algorithm. Recalling Lemma 11, we begin by defining *identifier-induced subtree* to facilitate presentation.

▶ **Definition 13** (**Identifier-induced Subtree**)**.** *At the end of any round, for any node r, within the subtree rooted at r in the identifier-induced graph, call the subtree induced by the nodes that have identical identifier with r as the* identifier-induced subtree rooted at r.

Our first lemma regarding the correctness of the token aggregation process states that, informally, every root r in the identifier-induced graph correctly computes the size of the identifier-induced subtree rooted at r if it outputs a decision for the token collision problem.

▶ **Lemma 14.** *Assume that in some round i, node v runs procedure Token-Collision-Detection and within that procedure updates $\texttt{res}_v$ for the first time (so that after the update $\texttt{res}_v \neq \perp$), then by the end of round i, the value of $\texttt{cnt}_v$ equals the size of the identifier-induced subtree rooted at v.*

Proof sketch. Notice that in our algorithm, only a root node in the identifier-induced graph can update its $\texttt{res}$ within procedure Token-Collision-Detection, so v must be a root node in the identifier-induced graph by the end of round i. Let $T_{v,i}$ be the identifier-induced subtree rooted at v by the end of round i. We need to show that $\texttt{cnt}_v$ equals the size of $T_{v,i}$ by the end of round i.

To prove the above result, we make the following claim and prove it via an induction on round number: for any node u, if i_u is the first round in which u updates $\texttt{cnt}_u$ to some non-$\perp$ value, then $\texttt{cnt}_u$ equals the size of T_{u,i_u} by the end of round i_u, where T_{u,i_u} is the identifier-induced subtree rooted at u by the end of round i_u. Moreover, by the end of any round $i' > i_u$, $\texttt{cnt}_u$ remains unchanged and $T_{u,i'}$ is identical to T_{u,i_u}.

With the above claim, the lemma is easy to obtain. Assume i_v is the first round in which v updates $\texttt{cnt}_v$ to some non-$\perp$ value, then $\texttt{cnt}_v$ equals the size of T_{v,i_v} by the end of round i_v. Later, at the end of round i, when v updates $\texttt{res}$ to some non-$\perp$ value in procedure Token-Collision-Detection, $\texttt{cnt}_v$'s value remains unchanged and is $|T_{v,i_v}| = |T_{v,i}|$. ◀

Our second lemma regarding the correctness of the token aggregation process states that, informally, every root r in the identifier-induced graph correctly collects the tokens within the identifier-induced subtree rooted at r if it outputs a decision for the token collision problem. The high-level strategy for proving this lemma is similar to the proof of Lemma 14, but the details are more involved as the convergecast process is more complicated.

▶ **Lemma 15.** *Assume that in some round i, node v runs procedure* Token-Collision-Detection *and within that procedure updates $\texttt{res}_v$ for the first time (so that after the update $\texttt{res}_v \neq \perp$), then by the end of round i, node v collects each token owned by the nodes within the identifier-induced subtree rooted at v exactly once.*

At this point, we are ready to show the correctness of our algorithm.

▶ **Lemma 16.** *After all nodes halt (that is, $\texttt{res} \neq \perp$), they return the correct result.*

Proof. Notice that by algorithm description and Lemma 10, only root nodes in the identifier-induced graph can generate $\texttt{res} \neq \perp$, other nodes can only passively adopt $\texttt{res} \neq \perp$ from neighbors. So, let v be an arbitrary node that generates $\texttt{res} \neq \perp$ and assume this happens in round i_v, then v must be a root in the identifier-induced graph by the end of round i_v. Let T_{v,i_v} denote the identifier-induced subtree rooted at v by the end of round i_v.

First consider the case v sets $\texttt{res} = true$ (that is, there are no token collisions). Then by Lemma 14 and Lemma 15, v has correctly collected the tokens in T_{v,i_v} and correctly counted the size of T_{v,i_v} by the end of round i_v, implying that T_{v,i_v} contains all input tokens and there are no collisions among input tokens; that is, the result v generated is correct. Moreover, since T_{v,i_v} is of size n, it must be the only tree in the identifier-induced graph. As a result, all nodes other than v will only passively adopt the result generated by v, implying that all nodes return identical result.

Next, consider the case v sets $\texttt{res} = false$ (that is, there are token collisions). Then, again, by Lemma 14 and Lemma 15, v has correctly collected the tokens in T_{v,i_v} and correctly counted the size of T_{v,i_v} by the end of round i_v. Since v sets $\texttt{res}$ to $false$, there are three possible reasons:

- All nodes know n but $|T_{v,i_v}| \neq n$. Recall Lemma 12, which states that if there are no token collisions, then there is only one tree in the identifier-induced graph that contains all nodes and all tokens. Hence, if $|T_{v,i_v}| \neq n$, then there are indeed token collisions. Furthermore, for any other root node u in the identifier-induced graph that also generates an $\texttt{res} \neq \perp$ by the end of some round i_u, node u must have also found $|T_{u,i_u}| \neq n$ and set $\texttt{res} = false$. Therefore, in this case, all nodes output the correct result.
- All nodes know k but the number of tokens v has collected is not k. By a similar argument as in the first case, we can conclude that all nodes output the correct result.
- Node v finds collisions among the tokens it has collected. In this case, v's decision to generate $\texttt{res} = false$ is obviously correct. Moreover, for any other root node u in the identifier-induced graph that also generates an $\texttt{res} \neq \perp$, that $\texttt{res}$ must be $false$, as the identifier-induced subtree rooted at u will not contain all n nodes or all k tokens.

This completes the proof of the lemma. ◀

4.2 Complexity

We now proceed to analyze the time complexity of the algorithm. The first lemma states that any identifier-induced subtree rooted at some node that has a global minimum token as input has limited height – particularly, $O(D)$. Moreover, each such node is a root in identifier-induced graph.

▶ **Lemma 17.** *Let v be a node having a minimum token as input. Then at the end of any round, v is a root in the identifier-induced graph, and the identifier-induced subtree rooted at v has height $O(D)$.*

With Lemma 17, we argue that all nodes finish BFS-Tree-Building in $O(D)$ rounds.

▶ **Lemma 18.** *After $O(D)$ rounds, every node v quits BFS-Tree-Building (that is, $\texttt{build}_v = false$).*

Proof. To prove the lemma, we only need to show that some node will quit BFS-Tree-Building within $O(D)$ rounds. This is because the flooding mechanism of a $false$-valued `build` variable ensures, once a node v sets $\texttt{build}_v = false$, all other nodes will set `build` to $false$ within (at most) another D rounds.

If some node quits BFS-Tree-Building within D rounds then we are done, so assume that this is not true. Then, by the end of round D, global minimum token's value is known by every node. Particularly, by the end of round D, each node has an `rid` with a value equals to some global minimum token. In other words, by the end of round D, each node is in some identifier-induced subtree rooted at some node that has a global minimum token as input. Moreover, no node will change its `rid` or parent ever since. By Lemma 17, any tree rooted at some node that has a global minimum token as input has $O(D)$ height.

Now, by our algorithm, starting from round $D + 1$, nodes within any such tree will start setting `f` to *true* from leaves to root. Since the height of any such tree is $O(D)$, after $O(D)$ rounds, either some node already sets `build` to $false$ and quits BFS-Tree-Building, or some root of such tree sets `build` to $false$ and quits BFS-Tree-Building. In both cases, some node quits BFS-Tree-Building within $O(D)$ rounds since the start of execution. ◀

The next lemma states the time complexity of our algorithm.

▶ **Lemma 19.** *After $O(D + k)$ rounds, every node v halts (that is, v returns $\texttt{res} \neq \perp$).*

Proof sketch. Recall that our algorithm guarantees that if one node generates an $\texttt{res} \neq \perp$ and then halts, then this `res` is broadcast to all other nodes. Hence, all other nodes will halt within another D rounds. As a result, to prove the lemma, we show that some node will halt within $O(D + k)$ rounds. To this end, we show that after all nodes quit BFS-Tree-Building which happens within $O(D)$ rounds (by Lemma 18), there exists a tree in the identifier-induced graph of height $O(D)$ (by Lemma 17), and the convergecast process inside this tree take $O(D + k)$ rounds. ◀

4.3 Proof of the main theorem

We now prove Theorem 2. When $L = \Theta(\log n)$ – meaning that each message can fit at most a constant number of tokens, by Lemma 16 and Lemma 19, the theorem is immediate.

When $L = o(\log n)$, to prove the theorem, we make a small modification to our algorithm: in the convergecast process, whenever a node forwards tokens to its parent, it packs as many tokens in a message as possible (particularly, $\Theta((\log n)/L)$ tokens in a message). Intuitively, this means that our algorithm is convergecasting $\Theta(kL/\log n)$ "packed tokens" each of size $\Theta(\log n)$, and each of these "packed tokens" contains $\Theta((\log n)/L)$ real tokens. Hence, the total runtime of our algorithm is still $O(D + kL/\log n)$ rounds.

The above argument is valid if, for every node that has some token(s) as input, that node receives at least $\Theta((\log n)/L)$ tokens. If some node only receives $o((\log n)/L)$ tokens as input (e.g., only one token), then a more careful analysis is required. Specifically, assume that there are x nodes that each receives $o((\log n)/L)$ tokens as input, call these nodes V_x, and the nodes in V_x in total have k_x tokens. So, there are $n-x$ nodes that each receives at least $\Theta((\log n)/L)$ tokens as input, call these nodes $V_{\overline{x}}$, and the nodes in $V_{\overline{x}}$ in total have $k-k_x$ tokens. Imagine a process in which we first aggregate the tokens owned by $V_{\overline{x}}$, and then aggregate the tokens owned by V_x. By the above analysis, aggregating the tokens owned by $V_{\overline{x}}$ takes $O(D+(k-k_x)L/\log n)$ rounds. On the other hand, for each token owned by some node in V_x, within $O(D)$ rounds, it either reaches the root, or arrives at a node that has at least $\Theta((\log n)/L)$ tokens pending to be sent. Effectively, this means that starting from the round we process the tokens owned by V_x, in $O(D)$ rounds, we again arrive at a scenario in which each node that has pending tokens to send has at least $\Theta((\log n)/L)$ tokens in its token list. As a result, these k_x tokens owned by V_x will all reach the root within $O(D+k_xL/\log n)$ rounds. Note that our modified algorithm cannot be slower than the imagined process, so the runtime of our modified algorithm when $L=o(\log n)$ is $O(D+kL/\log n)$.

5 Generalizing the Deterministic Algorithm when Tokens are Large

When tokens are large, $L=\omega(\log n)$ in particular, the time complexity of the BFS-tree building process and the token aggregation process are both affected. As mentioned in Section 1, we can apply the simple strategy of using $L/\log n$ rounds to simulate one round of our algorithm (as a token can be transferred in $L/\log n$ rounds), but the resulting algorithm would be too slow. Instead, in this section, we introduce and analyze a variant of our algorithm that costs only $O(D\cdot\max\{\frac{\log(L/\log n)}{\log n},1\}+k\cdot\frac{L}{\log n})$ rounds when $L=\omega(\log n)$.

The high level framework of this variant is the same as the algorithm introduced in Section 3: first build BFS-tree(s) and then detect token collisions within the tree(s). In this section, we focus on introducing the process of building BFS-tree(s) as the latter component is almost identical with the original algorithm. (Complete pseudocode of this variant is provided in Appendix A.) For the ease of presentation, we use $B=\Theta(\log n)$ to denote the bandwidth of CONGEST networks throughout this section.

5.1 Algorithm description

We first explain the key idea that allows this variant to be faster than the simulation strategy. Recall that in the original BFS-tree building process, each node v needs to record the minimum token it has seen in $\texttt{rid}_v$, and this is done by exchanging tokens in their *entirety* with neighbors. However, a key observation is, the relative order of two binary strings can be determined by a *prefix* of the strings that includes the most significant bit where they differ. As a result, we can employ the strategy that identifiers are sent successively starting from the most significant bit. Whenever a node v finds a prefix from some neighbor u is strictly smaller than the prefix of its current identifier, v updates its identifier to match the prefix and designates u as its parent. Moreover, when v sends its updated identifier, it does not need to restart from the first bit; instead, v starts from the bit where the updated identifier differs from the previous identifier. Effectively, we obtain an efficient "*pipeline*" approach on identifier broadcasting that can speed up the BFS-tree building process.

Build BFS-tree(s). We now detail how to implement the above idea. Similar to the original algorithm, each node v attempts to construct a BFS-tree rooted at itself by broadcasting its identifier $\texttt{rid}_v$. Due to bandwidth limitation, each identifier is divided into multiple *pieces* so that one piece can fit into one message. Denote these pieces as $\texttt{rid}_v[1],\cdots,\texttt{rid}_v[\lceil L/B\rceil]$, where $\texttt{rid}_v[1]$ contains the B most significant bits while $\texttt{rid}_v[\lceil L/B\rceil]$ contains the B least significant bits. The BFS-tree building procedure contains multiple *iterations*, each of which contains $\Theta(\frac{\log(L/B)}{B})$ rounds. In each iteration, $\Theta(\frac{\log(L/B)}{B})$ identifier pieces are sent, along with the position of the first sent piece in $\texttt{rid}_v$ – we use $\texttt{sent}_v$ to denote this position. (Notice that sending $\texttt{sent}_v$ may require $\frac{\log(L/B)}{B}$ rounds when L is large, this is why each iteration may contain multiple rounds.) Each node v locally maintains an identifier prefix for each neighbor based on received pieces. Whenever v finds a prefix of some neighbor u is strictly smaller than the prefix of its current identifier, v updates its identifier to match the prefix and designates u as its parent. At this point, v should send the updated identifier to neighbors. Particularly, v starts with the first piece where the updated identifier differs from v's previous identifier. This implies v may send non-successive piece position, in which case each neighbor of v should abandon the old prefix of v and record the new one.

Node v waits until all neighbors and itself have sent complete identifiers. Then, if v finds that all neighbors share the same identifier as itself, it attempts to ascertain whether the BFS-tree rooted at itself is fully constructed. Similar to the original algorithm, each node v uses a boolean variable $\texttt{f}_v$ to indicate whether BFS-tree construction is completed. Initially $\texttt{f}_v$ is $false$, and $\texttt{f}_v$ becomes *true* if: (1) v and all its neighbors have sent complete identifiers; (2) v and all its neighbors have identical identifier; and (3) each child u of v has $\texttt{f}_u = true$ or v has no children.

Lastly, if node v determines that the BFS-tree rooted at itself is fully constructed and it does not have a parent, then it terminates the BFS-tree building procedure and broadcasts a termination signal to all neighbors once. The node will then proceed to the second stage of the algorithm. Any node receiving such a signal will also forward it to neighbors once, stop the BFS-tree building procedure, and proceed to the second stage of the algorithm.

5.2 Analysis

The analysis for the above generalized algorithm is similar to the analysis for the original algorithm. Most claims and lemmas can carry over with little or no modifications, so are the proofs for these claims and lemmas. Others, however, require non-trivial extension or adjustments. To avoid redundancy, we only state these claims and lemmas here and provide proofs that require noticeable extension or adjustments in the full paper.

Correctness. The definition for identifier-induced graph remains unchanged in the generalized setting, except that such graph is defined at the end of each iteration.

▶ **Definition 20** (Analogue of Definition 8). *At the end of any iteration, define directed graph $G' = (V, E')$ as the* identifier-induced graph *in the following way: V is the node set of the network graph, and a directed edge $(v, u) \in E'$ if v assigns u as its parent.*

Following lemma is an analogue of Lemma 9, its proof is almost identical to that of Lemma 9, with small adjustments to account for the fact that identifiers are sent in pieces.

▶ **Lemma 21** (Analogue of Lemma 9). *At the end of any iteration, for any directed path in the identifier-induced graph, the identifiers of the nodes along the directed path are non-increasing.*

With Lemma 21, analogues of Lemma 10 and Lemma 11 hold automatically.

▶ **Lemma 22** (Analogue of Lemma 10). *At the end of any iteration, the identifier-induced graph is a directed forest in which every weakly connected component is a rooted tree. In particular, in each tree, the unique node with no parent is the root of that tree.*

▶ **Lemma 23** (Analogue of Lemma 11). *At the end of any iteration, for any node r that has sent its complete identifier to neighbors (that is, $\texttt{sent}_r = \lceil L/B \rceil$), within the subtree rooted at node r in the identifier-induced graph, the subgraph induced by the nodes having identical identifier with node r is also a tree rooted at node r.*

Lemma 12 is critical for the original algorithm, which states that a single BFS tree containing all nodes will be built when there are no token collisions. In the generalized setting, this claim still holds, but the proof needs to be extended in a non-trivial fashion to deal with the complication introduced by the pipeline approach for sending identifiers.

▶ **Lemma 24** (Analogue of Lemma 12). *If there are no token collisions, then after all nodes quit the BFS-Tree-Building procedure, the identifier-induced graph contains a single tree rooted at the node having the minimum token as input, and all nodes in that tree have identical identifier.*

Much like the case of Definition 20, the definition for identifier-induced subtree remains largely unchanged in the generalized setting.

▶ **Definition 25** (Analogue of Definition 13). *At the end of any iteration, for any node r that has sent its complete identifier to neighbors (that is, $\texttt{sent}_r = \lceil L/B \rceil$), within the subtree rooted at r in the identifier-induced graph, call the subtree induced by the nodes having identical identifier with r as the* identifier-induced subtree rooted at *r.*

Lemma 14 and Lemma 15 (and their proofs) still hold in the generalized setting, as we utilize the mechanism in the original algorithm for counting tree size and aggregating tokens.

▶ **Lemma 26** (Analogue of Lemma 14). *Assume that in some iteration i, node v runs procedure Token-Collision-Detection and within that procedure updates $\texttt{res}_v$ for the first time (so that after the update $\texttt{res}_v \neq \perp$), then by the end of iteration i, the value of $\texttt{cnt}_v$ equals the size of the identifier-induced subtree rooted at v.*

▶ **Lemma 27** (Analogue of Lemma 15). *Assume that in some iteration i, node v runs procedure Token-Collision-Detection and within that procedure updates $\texttt{res}_v$ for the first time (so that after the update $\texttt{res}_v \neq \perp$), then by the end of iteration i, node v collects each token owned by the nodes within the identifier-induced subtree rooted at v exactly once.*

We conclude this part with the following lemma which shows the correctness of our generalized algorithm, its proof is essentially identical to that of Lemma 16.

▶ **Lemma 28** (Analogue of Lemma 16). *After all nodes halt (that is, $\texttt{res} \neq \perp$), they return identical and correct result.*

Complexity. We now analyze the round complexity of the generalized algorithm, focusing on the BFS-tree construction process. Firstly, an analogue of Lemma 17 can be established.

▶ **Lemma 29** (Analogue of Lemma 17). *Let v be a node having a minimum token as input. At the end of any iteration, if v has sent its complete identifier to neighbors (that is, $\texttt{sent}_v = \lceil L/B \rceil$), then v is a root in the identifier-induced graph, and the identifier-induced subtree rooted at v has height $O(D)$.*

The next lemma states the time consumption of the BFS-tree construction process, it highlights the advantage of using the pipelining approach over the simulation approach.

▶ **Lemma 30** (Analogue of Lemma 18). *After $O(D + \frac{L}{\log(L/\log n)})$ iterations, every node v quits BFS-Tree-Building (that is, $\texttt{build}_v = false$).*

The last lemma shows the total time complexity of the generalized algorithm.

▶ **Lemma 31** (Analogue of Lemma 19). *After $O(D \cdot \frac{\log(L/\log n)}{\log n} + k \cdot \frac{L}{\log n})$ rounds, every node v halts (that is, v returns $\texttt{res} \neq \perp$).*

Proof of the main theorem. Combine Lemma 28 and Lemma 31, Theorem 3 is immediate.

6 Impossibility Result and Lower Bound for Deterministic Algorithms

Impossibility result. Recall Theorem 5 which states that if each node has no knowledge about the network graph except being able to count and communicate over adjacent links, and if each node also has no knowledge regarding the tokens except the ones being given as input, then the token collision problem has no deterministic solution.

To obtain the above impossibility, the key intuition is: to solve the problem, nodes need to exchange their input tokens in some manner; but in the anonymous setting with no global knowledge regarding network graph or input tokens, whenever a node receives a token from some neighbor that collide with its own input, the node cannot reliably determine whether this token originates from itself or some other node, yet the correctness of any algorithm depends on being able to distinguish these two scenarios.

We now provide a complete proof. Note that our impossibility result is strong in that we can construct counterexamples for any network size $n \geq 3$.

Proof of Theorem 5. Assume that there is an algorithm $\mathcal{A}$ that solves the token collision problem in the considered setting. For any $n \geq 3$, we consider two problem instances. The first instance – henceforth called C_n – is a ring consisting of n nodes, denoted as $v_1, v_2, \cdots, v_n$. Each node in the network obtains one token as input. Particularly, for any $i \in [n]$, node v_i has a token with value i. The second instance – henceforth called C_{2n} – is a ring consisting of $2n$ nodes. To construct C_{2n}, we first build two paths. The first path contains n nodes, denoted as $v'_1, v'_2, \cdots, v'_n$; the second path also contains n nodes, denoted as $u_1, u_2, \cdots, u_n$. Then, we connect v'_n with u_1, and connect u_n with v'_1. At this point, we have a ring. Each node in C_{2n} obtains one token as input. Particularly, for any $i \in [n]$, node v'_i and node u_i each has a token with value i.

Clearly, token collisions exist in C_{2n} but not in C_n, yet we will prove that $\mathcal{A}$ outputs identical results in both instances, resulting in a contradiction. Specifically, call the execution of $\mathcal{A}$ on C_n as α and the execution of $\mathcal{A}$ on C_{2n} as β, we will prove by induction that by the end of every round, for any $i \in [n]$, the internal states of nodes v_i, v'_i and u_i are identical.

The base case, which is immediately after initialization (i.e., round 0), trivially holds.

Assume that the claim holds for all rounds up to the end of round $r \geq 0$, now consider round $r + 1$. Fix an arbitrary $i \in [n]$, by the induction hypothesis, nodes v_{i-1} and v'_{i-1} have identical states by the end of round r. Hence, in round $r + 1$, the message (if any) v_{i-1} sends to v_i and the message (if any) v'_{i-1} sends to v'_i will be identical. Similarly, in round $r + 1$, the message (if any) v_{i+1} sends to v_i and the message (if any) v'_{i+1} sends to v'_i will be identical. Also, notice that by the end of round r, by the induction hypothesis, v_i and v'_i have identical states. Hence, during round $r + 1$, the local views of v_i and v'_i are identical. In other words, for any $\hat{v} \in \{v_i, v'_i\}$, node $\hat{v}$ cannot distinguish whether it is in α or β. Therefore, by the end of round $r+1$, nodes v_i and v_i' have identical states. By a similar argument, we can show that by the end of round $r+1$, nodes v_i and u_i also have identical states. This completes the proof of the inductive step, hence proving the claim.

Since $\mathcal{A}$ solves the token collision problem, α and β both terminate. Moreover, due to the above claim, nodes in α and β output identical results, resulting in a contradiction. ◀

Deterministic lower bound. We reduce the set-disjointness problem to the token collision problem and obtain the following theorem. Theorem 4 is an immediate corollary of it (by setting $\text{mincut}(G) = 1$).

▶ **Theorem 32.** *Recall the parameters n, k, L introduced in the definition of the token collision problem (that is, Definition 1). Consider a size-n CONGEST network $G = (V, E)$ with diameter D. Assuming $2^L \geq k$, any deterministic algorithm that solves the token collision problem takes $\Omega(D + \frac{k(L-\log k+1)}{\text{mincut}(G)\cdot\log n})$ rounds. Here,* mincut(G) *denotes the mincut of G. That is,* $\text{mincut}(G) = \min_{U\subset V} |\{(u,v) \in E \mid u \in U, v \in V \setminus U\}|$.

Proof. Let $(U, V\setminus U)$ be a partition of V that attains $\text{mincut}(G)$. Let $u \in U$ and $v \in V \setminus U$ be a pair of farthest nodes between U and $V \setminus U$. Assume that u and v are assigned token sets S and T respectively, each containing $k/2$ tokens.

Notice that $\Omega(D)$ is a lower bound for the token collision problem, as the distance between u and v is $\Theta(D)$, and they need to communicate with each other to solve the problem.

On the other hand, recall the two-party communication model and the set-disjointness problem introduced in Section 2. Since $2^L \geq k$, by setting $p = 2^L$ and $q = k/2$, it holds $q \leq p/2$. Recall the bandwidth of the network is $B = \Theta(\log n)$. We claim, if there exists an r-round algorithm that deterministically solves token collision in the CONGEST model, then Alice and Bob can compute $\text{DISJ}_q^p(S,T)$ by communicating at most $2rB\cdot\text{mincut}(G)$ bits. Specifically, they can run the r-round algorithm by having Alice and Bob simulate nodes in U and $V \setminus U$ respectively. Communication between Alice and Bob is necessary only when messages (each of which is at most B bits) are exchanged between nodes in U and $V \setminus U$ in the simulation. Apply Fact 7, we have $2r \cdot \text{mincut}(G) \cdot B = \Omega(\log \binom{p}{q})$, implying $r = \Omega(\frac{k(L-\log k+1)}{\text{mincut}(G)\cdot B}) = \Omega(\frac{k(L-\log k+1)}{\text{mincut}(G)\cdot\log n})$. ◀

References

1 Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004. doi:10.1145/1008731.1008735.

2 Andris Ambainis. Quantum Walk Algorithm for Element Distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. doi:10.1137/S0097539705447311.

3 Dana Angluin. Local and Global Properties in Networks of Processors (Extended Abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, STOC, pages 82–93. ACM, 1980. doi:10.1145/800141.804655.

4 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley, 2004.

5 John Augustine, William K. Moses, and Gopal Pandurangan. Brief Announcement: Distributed MST Computation in the Sleeping Model: Awake-Optimal Algorithms and Lower Bounds. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC, pages 51–53. ACM, 2022. doi:10.1145/3519270.3538459.

6 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. doi:10.1016/J.JCSS.2003.11.006.

7 Michael Ben-Or. Lower Bounds for Algebraic Computation Trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, STOC, pages 80–86. ACM, 1983.

8 Joshua Brody, Amit Chakrabarti, Ranganath Kondapally, David P. Woodruff, and Grigory Yaroslavtsev. Beyond set disjointness: the communication complexity of finding the intersection. In *ACM Symposium on Principles of Distributed Computing*, PODC, pages 106–113. ACM, 2014. doi:10.1145/2611462.2611501.

9 Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum Algorithms for Element Distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005. doi:10.1137/S0097539702402780.

10 Wan Fokkink and Jun Pang. Simplifying Itai-Rodeh Leader Election for Anonymous Rings. *Electronic Notes in Theoretical Computer Science*, 128(6):53–68, 2005. doi:10.1016/J.ENTCS.2005.04.004.

11 Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. Improved deterministic network decomposition. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2904–2923. SIAM, 2021. doi:10.1137/1.9781611976465.173.

12 Dima Grigoriev, Marek Karpinski, Friedhelm Meyer auf der Heide, and Roman Smolensky. A Lower Bound for Randomized Algebraic Decision Trees. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 612–619. ACM, 1996. doi:10.1145/237814.238011.

13 Johan Håstad and Avi Wigderson. The Randomized Communication Complexity of Set Disjointness. *Theory of Computing*, 3(1):211–219, 2007. doi:10.4086/TOC.2007.V003A011.

14 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990. doi:10.1016/0890-5401(90)90004-2.

15 Ralph E. Johnson and Fred B. Schneider. Symmetry and Similarity in Distributed Systems. In *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing*, PODC, pages 13–22. ACM, 1985. doi:10.1145/323596.323598.

16 Bala Kalyanasundaram and Georg Schnitger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. doi:10.1137/0405044.

17 Hsiang-Tsung Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981. doi:10.1145/319566.319567.

18 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

19 Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 677–689. ACM, 2015. doi:10.1145/2723372.2749436.

20 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 4 edition, 2020.

21 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

22 Alexander A. Razborov. On the Distributional Complexity of Disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992. doi:10.1016/0304-3975(92)90260-M.

23 Gerard Tel. Network orientation. *International Journal of Foundations of Computer Science*, 5(01):23–57, 1994. doi:10.1142/S0129054194000037.

24 Gerard Tel. *Introduction to distributed algorithms*. Cambridge university press, 2000.

25 Andrew Chi-Chih Yao. Some Complexity Questions Related to Distributive Computing (Preliminary Report). In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, STOC, pages 209–213. ACM, 1979. doi:10.1145/800135.804414.

Appendix

A Pseudocode of the Deterministic Algorithms

Main algorithm executed at each node v.

```
1: $\mathtt{build}_v \leftarrow true$, $\boldsymbol{x}^v \leftarrow v$'s input tokens, $\mathtt{rid}_v \leftarrow \min\{\boldsymbol{x}^v\}$, $\mathtt{p}_v \leftarrow \perp$, $\mathtt{chi}_v \leftarrow \perp$, $\mathtt{f}_v \leftarrow false$.
2: $\mathtt{cnt}_v \leftarrow \perp$, $\mathtt{ele}_v \leftarrow \top$, $\mathtt{res}_v \leftarrow \perp$.    ▷ End of initialization.
3: for (each round) do
4:     for (each incident edge with label $i \in [\Delta_v]$) do
5:         $\mathtt{ischild}_i \leftarrow \mathbb{I}[\mathtt{p}_v == i]$. ▷ $\mathtt{ischild}_i$ indicates whether edge $i$ connects to the parent of $v$.
6:         Send $\langle \mathtt{res}_v, \mathtt{build}_v, \mathtt{rid}_v, \mathtt{ischild}_i, \mathtt{f}_v, \mathtt{cnt}_v, \mathtt{ele}_v \rangle$ through edge $i$.
7:     if ($\mathtt{res}_v \neq \perp$) then Return $\mathtt{res}_v$ as final result.    ▷ Termination.
8:     For $i \in [\Delta_v]$, let $m_i = \langle \mathtt{res}_i, \mathtt{build}_i, \mathtt{rid}_i, \mathtt{ischild}_i, \mathtt{f}_i, \mathtt{cnt}_i, \mathtt{ele}_i \rangle$ be the message received via edge $i$.
9:     for (each edge $i \in [\Delta_v]$) do
10:        if ($\mathtt{res}_i \neq \perp$) then $\mathtt{res}_v \leftarrow \mathtt{res}_i$.
11:        $\mathtt{build}_v \leftarrow \mathtt{build}_v \wedge \mathtt{build}_i$.
12:    if ($\mathtt{build}_v == true$) then Execute Procedure BFS-TREE-BUILDING.
13:    else Execute Procedure TOKEN-COLLISION-DETECTION.
```

Procedure BFS-TREE-BUILDING executed at node v.

```
1: $ID_v \leftarrow \{\mathtt{rid}_i \mid i \in [\Delta_v]\}$.
2: if ($\min\{ID_v\} < \mathtt{rid}_v$) then
3:     Let $j \in [\Delta_v]$ be one edge label satisfying $\mathtt{rid}_j == \min\{ID_v\}$.
4:     $\mathtt{rid}_v \leftarrow \mathtt{rid}_j$, $\mathtt{p}_v \leftarrow j$, $\mathtt{f}_v \leftarrow false$.    ▷ Notice that $\mathtt{f}_v$ is reset to $false$.
5: else if ($\max\{ID_v\} == \mathtt{rid}_v$) then
6:     $\mathtt{chi}_v \leftarrow \{i \mid i \in [\Delta_v]$ and $\mathtt{ischild}_i == true\}$.
7:     if (($\forall i \in \mathtt{chi}_v$, $\mathtt{f}_i == true$) or $\mathtt{chi}_v == \emptyset$) then $\mathtt{f}_v \leftarrow true$.
8: if ($\mathtt{p}_v == \perp$ and $\mathtt{f}_v == true$) then $\mathtt{build}_v \leftarrow false$.
```

Procedure TOKEN-COLLISION-DETECTION executed at node v.

```
1: $\mathtt{chi}_v \leftarrow \{i \mid i \in [\Delta_v]$ and $\mathtt{build}_i == false$ and $\mathtt{ischild}_i == true$ and $\mathtt{rid}_i == \mathtt{rid}_v\}$.
2: Append $\{\mathtt{ele}_i \mid i \in \mathtt{chi}_v$ and $\mathtt{ele}_i \in \{0,1\}^L\}$ to $\boldsymbol{x}^v$.
3: if ($\forall i \in [\Delta_v]$, $\mathtt{build}_i == false$) then
4:     if (($\forall i \in \mathtt{chi}_v$, $\mathtt{cnt}_i \neq \perp$) or $\mathtt{chi}_v == \emptyset$) then $\mathtt{cnt}_v \leftarrow 1 + \sum_{i \in \mathtt{chi}_v} \mathtt{cnt}_i$.
5:     if ($\mathtt{p}_v \neq \perp$) then
6:         if ($|\boldsymbol{x}^v| > 0$) then Eject one token from $\boldsymbol{x}^v$ and let that token be $\mathtt{ele}_v$.
7:         else if (($\forall i \in \mathtt{chi}_v$, $\mathtt{ele}_i == \perp$) or $\mathtt{chi}_v == \emptyset$) then $\mathtt{ele}_v \leftarrow \perp$.
8:         else $\mathtt{ele}_v \leftarrow \top$.
9:     else if ($\mathtt{cnt}_v \neq \perp$ and (($\forall i \in \mathtt{chi}_v$, $\mathtt{ele}_i == \perp$) or $\mathtt{chi}_v == \emptyset$)) then
10:        if (know value of $n$ and $\mathtt{cnt}_v == n$ and no token collision in $\boldsymbol{x}^v$) then $\mathtt{res}_v \leftarrow true$.
11:        else if (know value of $k$ and $|\boldsymbol{x}^v| == k$ and no token collision in $\boldsymbol{x}^v$) then $\mathtt{res}_v \leftarrow true$.
12:        else $\mathtt{res}_v \leftarrow false$.
```

Figure 1 Pseudocode of the deterministic token collision algorithm.

The complete pseudocode of the algorithm in Section 3 is given in Figure 1. Below are the explanations of some key variables that are used in the pseudocode. For any node v,

- $\mathtt{build}_v$: a boolean variable indicating whether BFS-tree building is ongoing for v.
- $\mathtt{rid}_v$: the identifier of v, intuitively it stores the root of the BFS-tree that v belongs to.
- $\mathtt{p}_v$: the label of the edge connecting to the parent of v.
- $\mathtt{chi}_v$: the set of edge labels representing the children of v.

- $\texttt{f}_v$: a boolean variable indicating whether the BFS-tree rooted at v is fully constructed.
- $\texttt{cnt}_v$: the size of the BFS-tree rooted at v.
- $\texttt{ele}_v$: if $\texttt{ele}_v \notin \{\perp, \top\}$, then it is the token that v intends to send to its parent in the next round; if $\texttt{ele}_v = \top$, it indicates that there may exist a token in the BFS-tree rooted at v that has not been transferred to v's parent; if $\texttt{ele}_v = \perp$, it indicates that all tokens in the BFS-tree rooted at v have already been transferred to v's parent.
- $\texttt{res}_v$: the result of the token collision problem, that is, the algorithm's output at node v.

The pseudocode of the algorithm introduced in Section 5, which deals with the case that tokens are large, are given in Figure 2.

B Token Collision for the Randomized Scenario

For the sake of completeness, in this section, we briefly discuss the round complexity of the token collision problem when randomization is allowed.

Randomized upper bound. We first describe a randomized algorithm that solves token collision with probability at least $1 - 1/k$ within $O(D \cdot \frac{\log((\log k)/\log n)}{\log n} + k \cdot \frac{\log k}{\log n} + \frac{L}{\log n})$ rounds, hence proving the upper bound part of Theorem 6.

To begin with, we elect a leader among the nodes that have at least one token as input. Notice that there are at most k such nodes. Hence, by letting each such node v uniformly and independently sample $\texttt{id}_v \in \{0,1\}^{c \log k}$ for some sufficiently large constant c, there is a unique node $\hat{v}$ that obtains the global minimum $\texttt{id}$ with probability at least $1 - 1/k^{c-2}$. If we let each node continuously broadcast the minimum $\texttt{id}$ that it ever received, a size-n BFS tree rooted at $\hat{v}$ would be constructed with probability at least $1 - 1/k^{c-2}$. Moreover, by using the pipelining approach we introduced in Section 5, this process takes at most $O(D \cdot \frac{\log((\log k)/\log n)}{\log n} + \frac{\log k}{\log n})$ rounds (see Lemma 30).

Once a size-n BFS tree is built, the root – which is also the leader – will collect all tokens to determine whether collisions exist. Notice that with randomization, we do not have to transfer each token in its entirety. In particular, we can leverage the following fact on the collision probability of random hash function to reduce the length of each token.

▶ **Fact 33** ([8]). *For any set $S \subseteq [2^L]$ of size $|S| = k$ and any $\beta \geq 0$, there exists a random hash function $h : [2^L] \to [q]$ with $q = O(k^{2+\beta})$ such that, with probability at least $1 - 1/k^\beta$, it holds that $h(x) \neq h(y)$ for all $x, y \in S$ with $x \neq y$. Moreover, h can be constructed using $O(L)$ random bits.*

Therefore, after BFS-tree construction, the leader can generate $O(L)$ random bits for constructing the random hash function, and broadcast these bits to all nodes in $O(D + \frac{L}{\log n})$ rounds. Then, each node uses the random hash function to reduce the length of its tokens to $O(\log k)$ bits. Finally, the k tokens each of length $O(\log k)$ is aggregated to the root in $O(D + k \cdot \frac{\log k}{\log n})$ rounds.

Clearly, the total runtime of the algorithm is

$$O\left(D \cdot \frac{\log((\log k)/\log n)}{\log n} + k \cdot \frac{\log k}{\log n} + \frac{L}{\log n}\right),$$

and it succeeds with probability at least $1 - 1/k$.

Randomized lower bound. The lower bound part of Theorem 6 can be obtained in the same manner as the proof of Theorem 32 by using the randomized lower bound of set-disjointness mentioned in Fact 7. We omit its proof to avoid redundancy.

Main algorithm executed at each node v for large tokens.

```
1: $\texttt{build}_v \leftarrow true$, $\boldsymbol{x}^v \leftarrow v$'s input tokens, $\texttt{rid}_v \leftarrow \min\{\boldsymbol{x}^v\}$, $\texttt{p}_v \leftarrow \perp$, $\texttt{chi}_v \leftarrow \perp$, $\texttt{f}_v \leftarrow false$.
2: Initialize four vectors $\{\texttt{rid}_i\}_{i\in[\Delta_v]}$, $\{\texttt{sent}_i\}_{i\in[\Delta_v]}$, $\{\texttt{ele}_i\}_{i\in\{0,\cdots,\Delta_v\}}$, and $\{\texttt{sente}_i\}_{i\in\{0,\cdots,\Delta_v\}}$.
3: Set $\texttt{ele}_0 \leftarrow \top$; for any $i \in [\Delta_v]$, set $\texttt{sent}_i \leftarrow 0$, $\texttt{rid}_i \leftarrow 2^L - 1$, $\texttt{sente}_i \leftarrow 0$.
4: $M \leftarrow \lceil L/B \rceil$, $P \leftarrow \lceil (\log \lceil L/B \rceil)/B \rceil$.                ▷ End of initialization.
5: for (each iteration containing $\Theta(\frac{\log(L/B)}{B})$ rounds) do
6:     $l \leftarrow \texttt{sent}_v + 1$, $r \leftarrow \min(\texttt{sent}_v + P, M)$.
7:     for (each incident edge with label $i \in [\Delta_v]$) do
8:         $\texttt{ischild}_i \leftarrow \mathbb{I}[\texttt{p}_v == i]$.
9:         Send $\langle \texttt{res}_v, \texttt{build}_v, \texttt{rid}_v[l,\cdots,r], \texttt{sent}_v, \texttt{ischild}_i, \texttt{f}_v, \texttt{cnt}_v, \texttt{ele}_v \rangle$ through edge $i$.
10:    $\texttt{sent}_v \leftarrow r$.
11:    if ($\texttt{res}_v \neq \perp$) then Return $\texttt{res}_v$ as final result.                ▷ Termination.
12:    Let $m_i = \langle \texttt{res}_i, \texttt{build}_i, \texttt{rid}'_i, \texttt{sent}'_i, \texttt{ischild}_i, \texttt{f}_i, \texttt{cnt}_i, \texttt{ele}'_i \rangle$ be the message received via edge $i \in [\Delta_v]$.
13:    for (each edge $i \in [\Delta_v]$) do
14:        $l \leftarrow \texttt{sent}'_i + 1$, $r \leftarrow \min(\texttt{sent}'_i + P, M)$, $\texttt{sent}_i \leftarrow r$, $\texttt{rid}_i[l,\cdots,r] \leftarrow \texttt{rid}'_i$, fill $\texttt{rid}_i[r+1,\cdots,M]$ with 1.
15:        if ($\texttt{res}_i \neq \perp$) then $\texttt{res}_v \leftarrow \texttt{res}_i$.
16:        $\texttt{build}_v \leftarrow \texttt{build}_v \wedge \texttt{build}_i$.
17:    if ($\texttt{build}_v == true$) then Execute Procedure BFS-TREE-BUILDING for large tokens.
18:    else Execute Procedure TOKEN-COLLISION-DETECTION for large tokens.
```

Procedure BFS-TREE-BUILDING executed at node v for large tokens.

```
1: $ID_v \leftarrow \{\texttt{rid}_i \mid i \in [\Delta_v]\}$.
2: if ($\min\{ID_v\} < \texttt{rid}_v$) then
3:     if ($\texttt{p}_v \neq \perp$ and $\texttt{rid}_{\texttt{p}_v} == \min\{ID_v\}$) then $j \leftarrow \texttt{p}_v$.
4:     else Let $j \in [\Delta_v]$ be one edge label satisfying $\texttt{rid}_j == \min\{ID_v\}$.
5:     $\texttt{rid}_v \leftarrow \texttt{rid}_j$, $\texttt{sent}_v \leftarrow \texttt{sent}'_j$, $\texttt{p}_v \leftarrow j$, $\texttt{f}_v \leftarrow false$.      ▷ Notice that $\texttt{f}_v$ is reset to $false$.
6: else if ($\max\{ID_v\} == \texttt{rid}_v$ and $\texttt{sent}_v == M$ and $(\forall i \in [\Delta_v]$, $\texttt{sent}_i == M)$) then
7:     $\texttt{chi}_v \leftarrow \{i \mid i \in [\Delta_v]$ and $\texttt{ischild}_i == true\}$.
8:     if ($(\forall i \in \texttt{chi}_v$, $\texttt{f}_i == true)$ or $\texttt{chi}_v == \emptyset$) then $\texttt{f}_v \leftarrow true$.
9: if ($\texttt{p}_v == \perp$ and $\texttt{f}_v == true$) then $\texttt{build}_v \leftarrow false$.
```

Procedure TOKEN-COLLISION-DETECTION executed at node v for large tokens.

```
1: $\texttt{chi}_v \leftarrow \{i \mid i \in [\Delta_v]$ and $\texttt{build}_i == false$ and $\texttt{ischild}_i == true$ and $\texttt{sent}_i ==$
   $M$ and $\texttt{rid}_i == \texttt{rid}_v\}$.
2: for ($i \in \texttt{chi}_v$ and $\texttt{ele}'_i \notin \{\top, \perp\}$) do
3:     $\texttt{ele}_i[\texttt{sente}_i + 1, \cdots, \min(\texttt{sente}_i + P, M)] \leftarrow \texttt{ele}'_i$, $\texttt{sente}_i \leftarrow \min(\texttt{sente}_i + P, M)$.
4:     if ($\texttt{sente}_i == M$) then Append $\texttt{ele}_i$ to $\boldsymbol{x}^v$ and set $\texttt{sente}_i \leftarrow 0$.
5: if ($(\forall i \in [\Delta_v]$, $\texttt{build}_i == false)$ and $\texttt{sent}_v == M$) then
6:     if ($(\forall i \in \texttt{chi}_v$, $\texttt{cnt}_i \neq \perp)$ or $\texttt{chi}_v == \emptyset$) then $\texttt{cnt}_v \leftarrow 1 + \sum_{i\in\texttt{chi}_v} \texttt{cnt}_i$.
7:     if ($\texttt{p}_v \neq \perp$) then
8:         if ($\texttt{sente}_0 \neq 0$) then
9:             $l \leftarrow \texttt{sente}_0 + 1$, $r \leftarrow \min(\texttt{sente}_0 + P, M)$, $\texttt{ele}_v \leftarrow \texttt{ele}_0[l,\cdots,r]$, $\texttt{sente}_0 \leftarrow r \bmod M$.
10:        else if ($|\boldsymbol{x}^v| > 0$) then
11:            Eject one token from $\boldsymbol{x}^v$ and let that token be $\texttt{ele}_0$.
12:            $\texttt{ele}_v \leftarrow \texttt{ele}_0[1,\cdots,P]$, $\texttt{sente}_0 \leftarrow P$.
13:        else if ($(\forall i \in \texttt{chi}_v$, $\texttt{ele}_i == \perp)$ or $\texttt{chi}_v == \emptyset$) then $\texttt{ele}_v \leftarrow \perp$.
14:        else $\texttt{ele}_v \leftarrow \top$.
15:    else if ($\texttt{cnt}_v \neq \perp$ and $((\forall i \in \texttt{chi}_v$, $\texttt{ele}_i == \perp)$ or $\texttt{chi}_v == \emptyset)$) then
16:        if (know value of $n$ and $\texttt{cnt}_v == n$ and no token collision in $\boldsymbol{x}^v$) then $\texttt{res}_v \leftarrow true$.
17:        else if (know value of $k$ and $|\boldsymbol{x}^v| == k$ and no token collision in $\boldsymbol{x}^v$) then $\texttt{res}_v \leftarrow true$.
18:        else $\texttt{res}_v \leftarrow false$.
```

Figure 2 Pseudocode of the deterministic token collision algorithm for large tokens.