Name Management Using IOTA in ICN

***Abstract*—Information-centric networking (ICN) has received a wide attention as a next-generation network. Unlike conventional IP networks, which forward packets based on IP addresses, ICNs packets are sent based on the name of the contents which are cached to the routers involved, then delivered to the consumer. Since any participating party can upload content to the ICN, the risk of content poisoning attack (CPA) is ever-present. In CPA, an attacker degrades the cache efficiency by uploading fake content under a real name posting as a legitimate publisher. As a countermeasure to CPA, many existing methods determine the legitimacy of content using digital signatures with public keys, and alerts routers of unjustified content upon detection. However, it is difficult for them to detect fake-CPA attacks which use public keys of fabricated content to generate digital signatures. Most methods also lack counter-measures to spoofed fake-CPA attacks, in which the certification authority (CA) that manages the public key or its staff member colludes with the attacker to rewrite the legitimate publisher's public key to the attacker's and inject fake contents that pretends to be authentic contents that are high in popularity into the cache. In this paper, we propose a method to prevent the spoofed fake-CPA by managing content names with IOTA, a distributed ledger technology that blocks tampering of contents registered on the system. We also numerically compare the search time and memory requirement of four search methods that search content names managed in the ledger in the proposed method. As a result, we confirm the trade-off between the search time and memory requirement.**

I. Introduction

In the conventional Internet, users request domain name resolution from a DNS (domain name system) server, and content such as video images and websites are delivered based on the IP address. Therefore, it is necessary to determine the content distributor at the start of communication. However, it is difficult to determine it efficiently considering dynamically changing network load conditions and network topology. Therefore, ICN (information-centric networking), which does not perform name resolution at the start of communication but forwards the request packet (interest) based on the name of the requested content and delivers the content from the server (publisher) to the content requester (consumer) while caching it in the router through which it passes, is attracting attention as a next-generation content delivery method.

In ICN, the consumer sends an interest with the name of the content that the consumer wants to acquire. If the requested content exists in the cache of the router that receives the interest, it is discarded, and the content is delivered from there. If it does not exist, the interest is forwarded to the router or publisher where the requested content exists, and delivery is performed in the same way. If the requested content does not exist, it forwards the interest to the router or the publisher where the requested content exists and distributes it in the same way. Since the content is forwarded while caching it from the source to the router through which it is forwarded, it is possible to deliver the content from a nearby router where the content is cached the next time the same request is received from a consumer. As a result, the delivery cost is expected to be reduced. Some ICNs, such as IPFS, use hash values as IDs, but the common part of prefixes is small, making FIB aggregation difficult. In this study, we assume an ICN that uses readable prefixes.

Although anyone can upload content as a publisher on ICN, a CPA (content poisoning attack) [1], which an attacker pretending to be a legitimate publisher uploads fake content under a real content name, thereby degrading the cache functionality, has been pointed out. As a countermeasure against the CPA, a method has been proposed in which consumers determine the legitimacy of contents by digital signatures using public-key cryptography and notify routers of unauthorized contents [2].

There are two types of the CPA: a fake type that matches the digital signature generated by the public key associated with the content, and a corrupted type that does not match [3], so it is difficult for the method described in [2] to detect the fake-CPA. Especially, it is also difficult to take measures against the spoofed fake-CPA [4], in which the staff of the certification authority (CA) that manages the public key colludes with the attacker to rewrite the legitimate publisher's public key to the attacker's and inject fake contents that pretends to be real and popular contents into the cache. Although it is not so easy that a malicious attacker has a partner in the CA office, we cannot perfectly avoid the collusion between them. Also, it is assumed that the impact of this attack will be significant if popular content with many accesses is spoofed. In addition, the attacker's content is more legitimate because the public key of the legitimate publisher is taken over and rewritten.

A spoofed fake-CPA occurs when a public key is managed by only one authority, such as a CA. In this paper, we propose a method to prevent it by managing content names using IOTA [5], a DAG (directed acyclic graph)-based distributed ledger technology. Although blockchain is representative, it has issues with scalability, but IOTA is highly scalable. In the proposed method, it is necessary to search for the corresponding transaction in the ledger at the time of content registration by the publisher and at the time of delivery request by the consumer. We compare the search time and the amount of memory required for each of the following search methods for transactions: hash-chain method, binary search tree, breadth-first search, and depth-first search.

In Section II, we discuss the CPA, and discuss IOTA in Section III. In Section IV, we describe the proposed method, and describe performance evaluation in Section V. Finally, we conclude this paper in Section VI.

II. CPA

While it is easy to detect corrupted-CPA by checking signatures, it is difficult to detect the fake-CPA. We previously classified the fake-CPA into the original fake-CPA and the spoofed fake-CPA [4]. Figure 1 shows the schematic of the

operation of the fake-CPA.

In the original fake-CPA, the attacker injects fictitious content created by the attacker into the router by requesting it from the colluder. In the example shown in Figure 1, an attacker (A_a) publishes his own fake content y on the network. In the original fake-CPA, there are no requests for fake content from legitimate users, but only from a few colluders, so the amount of fake content flowing through the network is small. Therefore, the impact of the attack is small because fake content is cached in the router only temporarily. On the other hand, the spoofed fake-CPA injects fake content into the router by requesting it from legitimate users. In the example shown in Figure 1, an attacker (A_b) publishes fake content x_{fake} on the network that imitates real content x, taking over the name of legitimate content and distributing the fake content. In the spoofed fake-CPA, since many legitimate users request fake content, it is highly likely that the fake content will remain in the cache for a long time, and the impact of the attack is estimated to be high. This paper focuses on preventing the spoofed fake-CPA, which has high impact of attacks.

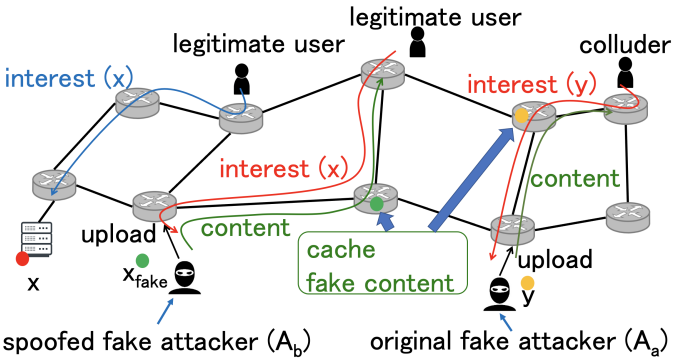

Fig. 1. fake-CPA

III. IOTA

In the IOTA, new transactions select two of the unselected transactions, i.e., the tips, and the transactions form a DAG, called tangle [5]. In the tangle structure, there is a genesis transaction, which is the first transaction that receives approval directly or indirectly from all transactions. Figure 2 shows an example of a DAG. A, B, C, D, E, and F represent transactions, each of which refers to two existing transactions. The number in the lower right corner indicates the weight of the transaction, and in the upper left corner indicates the cumulative weight. The cumulative weight is defined as the sum of the weights of other transactions that directly or indirectly refer to the transaction and its own weight [6]. For example, F has a cumulative weight of 12, which is the sum of the weights of A, B, C, D, and E plus its own weight. The unselected transactions like A or B are tips, and the following three tip-selection algorithms are typical.

- **uniform random selection (URS)**: Selecting two transactions randomly from the existing tips.
- **unweighted random walk (URW)**: Starting from the genesis transaction and selecting which transaction to move to with equal probability. By doing this twice, two tips are selected.
- **weighted random walk (WRW)**: Starting from the genesis transaction and selecting which transaction to move to considering the cumulative weights. In WRW, transactions with large cumulative weights are preferentially selected.

The transition probability P_{xy} from transaction y to x in WRW is obtained by

$$P_{xy} = \frac{e^{-\alpha(H_x - H_y)}}{\sum_{z:z\to x} e^{-\alpha(H_x - H_z)}} \tag{1}$$

where H_x and H_y is the cumulative weights of transaction x and y, and α (≥ 0) is the parameter of the cumulative weight. When $\alpha = 0$, it is an unweighted random walk. As the value of α increases, the influence of the cumulative weight increases, resulting in a bias in the tips selected. In IOTA, double-payment attacks, in which an attacker fraudulently obtains funds by using the same cryptographic asset multiple times, have become a problem. Examples include a splitting attack [7], in which an attacker splits a tangle into two and increases the cumulative weight of double-payment transactions to get them approved, and a parasite chain attack [8], in which a double payment transaction is added to a different tangle than the main tangle, making it easier to get approval by increasing the number of links in a short period of time. These attacks are likely to occur in URS and URW that do not consider cumulative weight in tip selection. Therefore, it is effective to employ WRW with a large value of α in (1) so that even if a tangle branches or splits, it will only transition to the one with the larger cumulative weight.

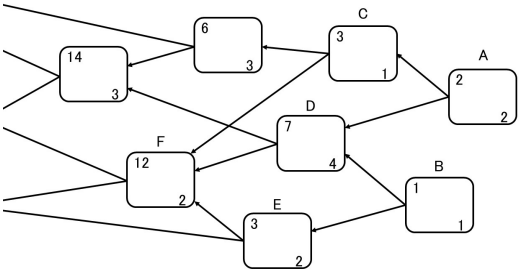

Fig. 2. DAG in IOTA

As a related study using IOTA, a P2P electricity trading scheme [6] has been proposed in which arbitrary households buy and sell electricity directly to each other in the energy market without any commission through the IOTA ledger, rather than through an electric power company or other parties. In [9], micropayments using IOTA have been proposed to allow users to pay network providers for guaranteed network resources such as low latency and bandwidth. This guarantees that users can use the service without paying a fixed fee regardless of network conditions. In [10], IOTA manages certificates attached to messages sent by vehicles in a cooperative intelligent transportation system. This guarantees the transparency of certificate issuance.

IV. PROPOSED METHOD

In the proposed method, each transaction contains pairs of prefix and corresponding content name which composes of the prefix, the public key, and the digital signature. Due to the

nature of distributed ledgers, it is difficult to tamper with the registered data, thus ensuring its legitimacy.

A. Overview

The definitions of the content prefix and the content name are shown with examples. The content prefix is a domain name such as ”www.ritsumei.ac.jp”. On the other hand, the content name consists of the content prefix, the public key, and the digital signature, as in ”www.ritsumei.ac.jp/514720/1307f51b14b5”, and is distinguished by ”/”. The flow of the proposed method is shown in figure 3.

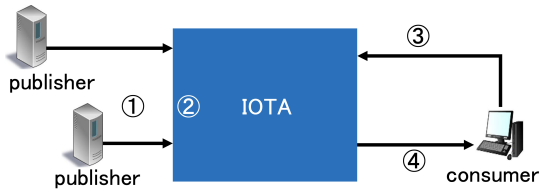

Fig. 3. Proposed Method

When the publisher uploads the content, the content prefix as well as the content name are registered in the transaction. To prevent duplicate content names from being managed, a search is performed in the IOTA ledger by content prefix to see if the same content name is already managed, and if it already exists, reject the registration; otherwise, register it. After the upload by the publisher is complete, the consumer requests the prefix of the content. After that, it searches the IOTA ledger by the requested prefix and replies to the consumer with the name of the contents that have hit. Finally, consumer sends an interest with that content name and requests the content.

Thus, in the proposed method, the consumer obtains the content name in a flow like DNS name resolution. This guarantees that only the publisher who first registers his/her name on IOTA for a given prefix is a legitimate publisher, thus preventing the spoofed fake-CPA in which an attacker publishes the fake content on the network for a real prefix. Since it is the content name, not the content, that is registered in IOTA, the content itself can be freely updated by the publisher. In addition, updating the name means registering a new content name.

B. Content Name Search Method in DAG

In the proposed method, it is necessary to search for the corresponding transaction in the DAG at two different times: when the content name is registered by the publisher and when the name is resolved by the consumer. The latency and amount of memory required for search are greatly affected by the search method of transaction. In this paper, we evaluate the following four search methods under the following assumptions.

- **hash-chain method (hash)**: The data is stored in the element of the hash table at the address corresponding to the hash value obtained by applying a hash function to the data search key. Although different data can have collisions that result in the same hash value, the hash-chain method allows multiple data to be managed in the same bucket by connecting them with a concatenated list. In the proposed method, the prefix converted into a numerical value is used as a search key, and the address of the DAG transaction that stores the name of the corresponding content is managed in a hash table. In other words, the content name is obtained by accessing the transaction on the DAG based on the address obtained from the hash chain.
- **binary search tree (bst)**: It is a kind of tree structure in which each node key has the property *left node ¡ parent ¡ right node*. It begins its search at the root, the uppermost node, and moves to the left if the search key is less than the value of the node, and to the right if the key is greater than that. This process is repeated until the content name is found. Like the hash-chain method, the content prefix is used as a search key in the bst, and each node in it maintains the address of a DAG transaction.
- **breadth-first search (bfs)**: It searches nodes (transactions) on the DAG directly, breadth-first, starting with the transaction with the smallest number of hops from the genesis transaction on the DAG. If all the transactions with the same number of hops have been searched but none have been found, the search for a transaction with a higher number of hops is started and repeated until it is found or until all the transitions are searched.
- **depth-first search (dfs)**: It also directly searches for nodes (transactions) on the DAG in depth-first order. As with bfs, the search begins at the genesis transaction, but searches deeply until it reaches a transaction without children, i.e., tip. If a tip is reached but not yet discovered, it returns to the last branch and searches for unexplored transactions.

In the hash-chain method and bst, the address of the corresponding transaction in the DAG is managed in a hash table or binary tree, so it is necessary to access the DAG after obtaining the address to obtain the content name. In bfs and dfs, on the other hand, the content name is recorded in the DAG transaction, so the search is completed when the corresponding transaction is found.

V. Performance Evaluation

We compare the search time and memory requirements among the four search methods described in the previous section in the proposed method by computer simulation. It was performed using the IOTA simulator DAGsim [11] with modifications. Search time is evaluated by the mean, median, and the 95th percentile. In terms of memory requirements, bfs and dfs, which manage data directly on the DAG, are grouped together as a DAG and compared with hash and bst.

A. Evaluation Condition

The number of 7,131 domains for which a web page was displayed without error when browsing the top 8,000 accessed web pages published on Alexa’s web pages [12] in November 2017 was used as the set of the content names for evaluation. When generating the DAG, for the three tip selection algorithms URS, URW, and WRW, the number of transactions N_t is set to 100, 1,000, or 7,131 for URS and URW, and to 100 or 1,000 for WRW[1].

[1]In WRW, the process was not completed when $N_t = 7,131$, so it was excluded from the evaluation.

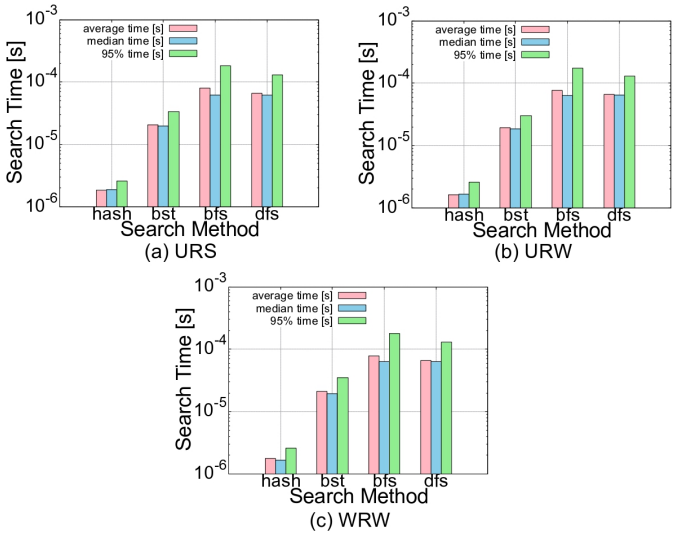

Fig. 4. Search time for content name registration in $N_t = 100$

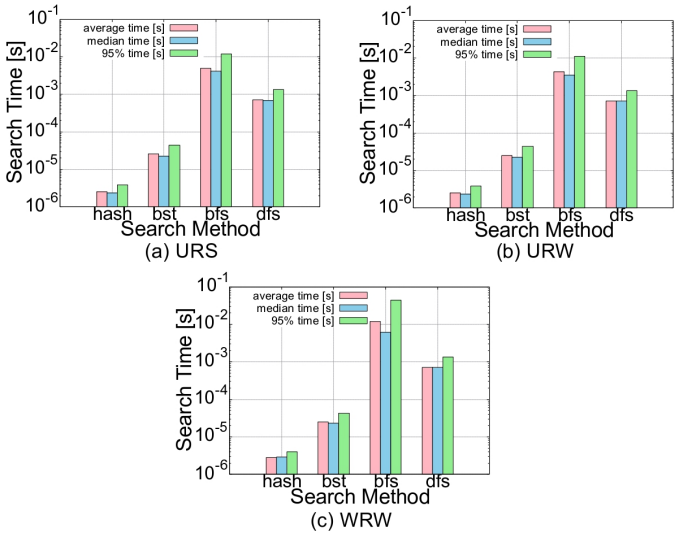

Fig. 5. Search time for content name registration in $N_t = 1,000$

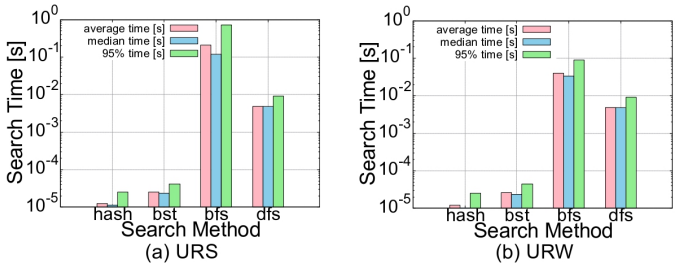

Fig. 6. Search time for content name registration in $N_t = 7,131$

Transactions are generated when a new content name is registered according to an exponential distribution with a rate of $\lambda_1 = 50$ (/second). The α in the transition probability equation (1) is set to 0.1. Requests by consumers are generated according to an exponential distribution with the total number of requests being 5,000 and the rate of generation being $\lambda_2 = 50$ (/second). The number of table buckets B in the hash-chain method is 100.

B. Search Time for Content Name Registration

Figure 4 to 6 show the mean, median, and the 95th percentile of the measured search time when registering content names for the hash-chain method, bst, bfs, and dfs for (a) URS, (b) URW, and (c) WRW at $N_t = 100, 1000, 7131$, respectively. In all cases in figure 4, the hash-chain method took the least amount of time, followed by bst, dfs, and bfs. In the hash-chain method, although the list managed in the bucket was searched based on the hash value, the number of data managed in the bucket was small at the beginning at the time of content name registration, so it was possible to search the entire contents in a shorter time than with other methods. In bst, the search time was also relatively short because the necessary parts of the data were searched instead of all the data. On the other hand, dfs and bfs took long time because they searched all DAG in an exhaustive manner. A comparison of the different tip selection algorithms confirmed that there was little difference in search time for any of the methods. The hash-chain method and bst used separate tables from the DAG to perform the search, so it was not affected by the shape of the DAG. In bfs and dfs, there was little difference in the shape of the DAG for a small number of transactions, and as a result, there was little difference in search time.

In figure 5, because of the larger size of the DAG compared to $N_t = 100$, the difference in search time between the hash-chain method or bst, which managed the data separately from the DAG, and bfs or dfs, which managed data directly in the DAG, was noticeable for any tip selection algorithms. Focusing on the bfs of WRW, it could be confirmed that the search time was larger than that of URS and URW. This was because WRW preferentially selects transactions with larger tip weights, and more transactions with larger hop counts from the genesis transaction were selected, so bfs spent more time on the full search. In dfs, there was no difference among different tip selection algorithms, but this was presumably since this method preferentially searched for transactions with many hops. The hash-chain method and bst did not differ much in the same way because the search time did not depend on the shape of the DAG.

As the number of transactions increased in figure 6, the difference between the hash-chain method, bst and bfs, dfs became even larger. It could be confirmed that URS spent more time in search than URW in bfs. This was because URS randomly selects two tips from among the existing tips, so even tips with large number of hops were selected, while URW selected a transaction with equal probability from among the genesis transaction, so there were many transactions with small number of hops on the DAG.

C. Search Time for Name Resolution

Figure 7 to 9 show the mean, median, and the 95th percentile of the measured search time for name resolution at $N_t = 100, 1000, 7131$. As a result of the comparison, the hash-chain method took the least amount of time, followed by bst, dfs, and bfs. Again, because the number of transactions was small, there was little difference in search time due to differences in tip selection algorithms. There was almost no difference in search time compared to Figure 4 when the content name was registered.

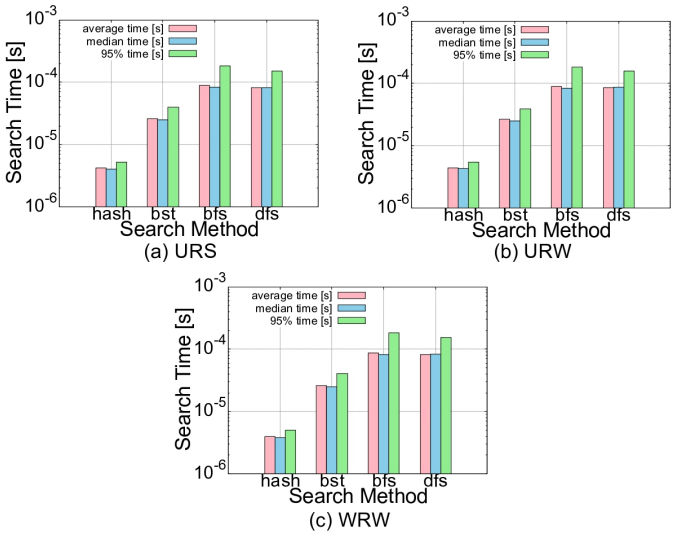

Fig. 7. Search time for name resolution in $N_t = 100$

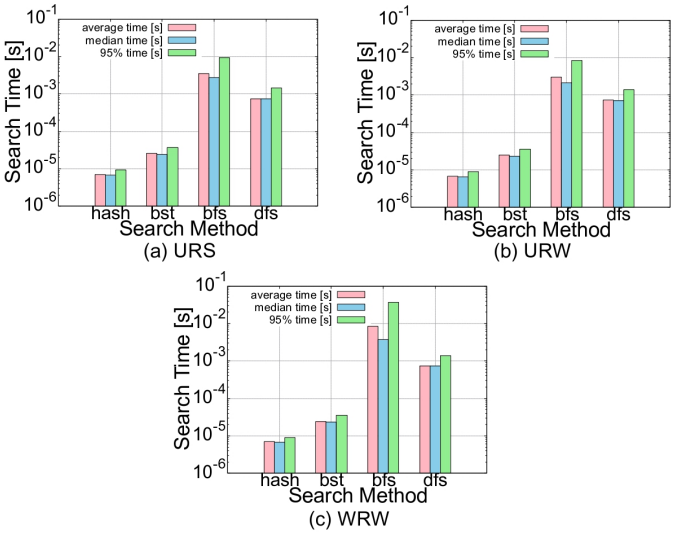

Fig. 8. Search time for name resolution in $N_t = 1,000$

In figure 8, due to the increase in the number of transactions, the difference between bfs and dfs, which were managed by the DAG, and the hash-chain method and bst, which were managed by tables, etc., is larger than in Figure 7. In addition, because there were many transactions with many hops from the genesis transaction, the bfs of WRW required more time to search than URS and URW. Compared to Figure 5 when the content name was registered, there was not much difference as well.

In both cases of URS and URW at figure 9, there was little difference in search time when comparing the hash-chain method and bst. The average computational complexity of the hash-chain method was $O(N_t/B)$, whereas bst was represented as $O(\log_2 N_t)$, so when the number of transactions was small, the hash-chain method was more efficient, and when the number of transactions was large, bst was less computationally expensive, resulting in a shorter search time difference.

Figure 10 plots the cumulative distribution of the number

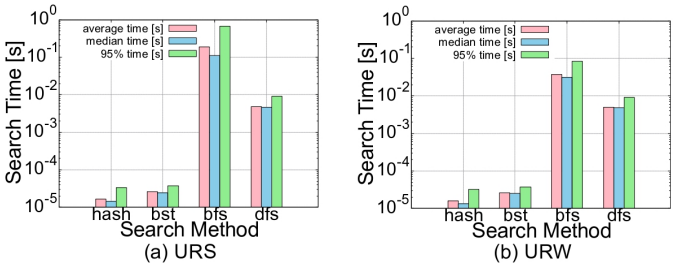

Fig. 9. Search time for name resolution in $N_t = 7,131$

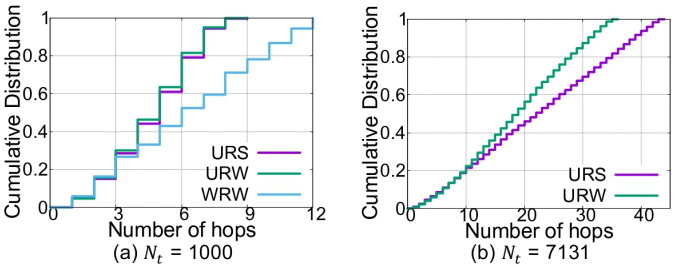

Fig. 10. Cumulative distribution of the number of hops from the genesis transaction on the DAG

of hops from the genesis transaction for each transaction on the DAG. For $N_t = 1,000$, URS and URW were similarly distributed, suggesting a similar DAG shape. In addition, WRW had more transactions with many hops than the other two, and as shown in Figures 5 and 8, it could be seen that WRW took the longest time for both bfs. On the other hand, for $N_t = 7,131$, we saw that URS had more transactions with more hops than URW. Therefore, in both Figures 6 and 9, it was supported that URS spent more time on bfs than URW. For $N_t = 100$, only transactions with hop counts of 1 and 2 existed, and since they were approximately distributed among the different tip selection algorithms, it could be inferred that there was little difference in the shape of the DAG. Therefore, as shown in Figure 4, there was no difference in search time for any of the search methods for URS, URW, and WRW.

The average computational complexity of each search method is expressed as $O(N_t/B)$ for the hash-chain method and $O(\log_2 N_t)$ for bst, as mentioned above, and $O(E)$ for DAG, assuming the number of edges $E = 2N_t$, since any transaction selects two existing transactions. These computational complexities are the same for both content name registration and name resolution. Figure 11 shows these average computations for the number of DAG transactions, N_t. Here, the hash-chain method is plotted with $B = 10, 100, 1,000$, and bfs and dfs as the DAG. In the hash-chain method, as the B increased, the number of data managed in a single bucket decreased, thus reducing the computational complexity. Comparing the hash-chain method with bst for $B = 100$ as done in this paper, the hash-chain method was computationally less expensive for $N_t < 1,000$, and bst was less expensive for $N_t \geq 1,000$. In fact, in Figures 4 and 7 for $N_t = 100$, the search time was smaller for the hash-chain method than for bst, and the difference between the two search times shrank as N_t increased. Also, the DAG was the most computationally expensive, independent of the number of transactions, and the simulations showed similar results.

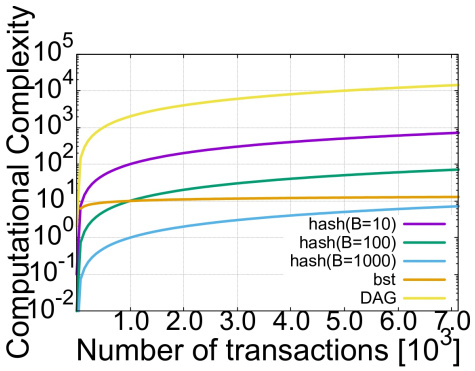

Fig. 11. Average computational complexity for the number of transactions in the DAG

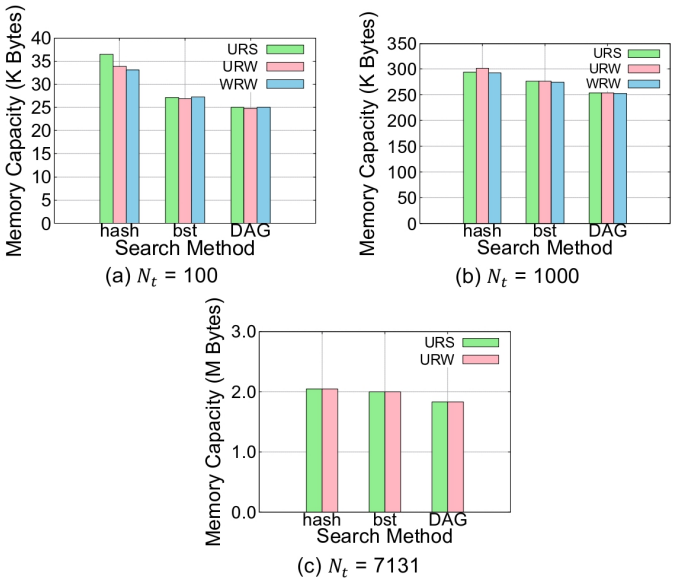

Fig. 12. Amount of memory required

D. Amount of Memory Required

Figure 12 shows the amount of memory required for each search method. In all cases (a), (b), and (c), the hash-chain method had the largest memory requirement, followed by bst and DAG. The hash-chain method required a large amount of memory because the bucket with the largest capacity in the hash table was allocated for all buckets, so memory efficiency was poor because some buckets had unused memory. On the other hand, bst required a node capacity equal to the number of transactions. Therefore, it required less memory than the hash-chain method because there was no extra memory. Whereas DAG required the least amount of memory because it did not need to be managed by other tables. It could also be confirmed that the difference between the hash-chain method and bst decreased as the number of transactions increased. This was because the hash-chain method required a fixed amount of memory for the hash table regardless of the number of generated data.

These results confirmed the trade-off between search time and memory requirements for each search method.

VI. Conclusion

In this paper, we proposed a method to prevent the spoofed fake-CPA by managing content names in IOTA. We also compared the search time and memory requirements of four search methods for searching content names in the ledger and confirmed the following through simulation evaluation in the proposed method.

In the proposed method, content names were searched at two times: when a content name was registered by a publisher and in name resolution by a consumer. In both cases, the hash-chain method took the shortest search time, followed by bst, dfs, and bfs. As the number of transactions increased, the search time for each search method increased, especially for dfs and bfs, which were affected by the shape of the DAG.

However, the hash-chain method required the most amount of memory, followed by bst and DAG. The DAG required the least amount of memory because it didn't need to be managed by other tables. On the other hand, the hash-chain method and bst required a lot of memory to manage data in hash tables and binary trees, in addition to the amount of memory in the DAG. Also, the hash-chain method had a large amount of unused memory because the bucket with the largest capacity in the hash table was allocated for all buckets. Meanwhile, bst required less memory than the hash-chain method because it allocated memory for data.

Based on the above results, it is desirable to use the hash-chain method or binary search tree which requires short search time when the search time is important, and to use depth-first search or breadth-first search which requires less memory when the amount of memory is important.

References

[1] T. Nguyen, et al., Content Poisoning in Named Data Networking: Comprehensive Characterization of real Deployment, IFIP/IEEE IM 2017.

[2] W. Cui, et al., Feedback-Based Content Poisoning Mitigation in Named Data Networking, IEEE ISCC 2018.

[3] P. Gasti, et al., DoS and DDoS in Named Data Networking, IEEE ICCCN 2013.

[4] T. Kudo and N. Noriaki, Investigating Impact of Fake Type Content Poisoning Attack on ICN, IEICE 2022, CQ2022-23, pp. 36-41, Jul. 2022.

[5] S. Popov, et al., Equilibria in the Tangle, Computers & Industrial Engineering, 136, pp.160-172, Oct. 2019.

[6] J. Park, et al., A Block-Free Distributed Ledger for P2P Energy Trading: Case with IOTA?, CAiSE 2019, LNCS 11483, pp. 111-125, 2019.

[7] G. Bu, et al., G-IOTA: Fair and confidence aware tangle, IEEE INFOCOM WKSHPS 2019.

[8] S. Ghaffaripour and A. Miri, Parasite Chain Attack Detection in the IOTA Network, IEEE IWCMC 2022.

[9] H. Masaki, et al., Guaranteed Network Resource Provision with Micropayment, IEICE 2021, NS2021-32, pp. 1-6, Jul. 2021.

[10] A. Tesei, et al., IOTA-VPKI: a DLT-based and Resource Efficient Vehicular Public Key Infrastructure, IEEE VTC 2018.

[11] M. Zander. 2018. Python IOTA Tangle simulation. https://github.com/manuelzander/iota_simulation

[12] Alexa webpage, https://www.alexa.com/siteinfo