Bitwise Division Searching Algorithm for VANETs

Xin Ye 1,a, Gencheng Xu 2,b*, Zhiguang 1,c

1School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan Province, 610054, P.R. China
2School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan Province, 610054, P.R. China
aemail: yexin@uestc.edu.cn  cemail: qinzg@uestc.edu.cn
bemail: xugencheng@std.uestc.edu.cn

Abstract: Vehicular ad-hoc networks (VANETs) are the key technology supporting the intelligent transportation system (ITS), which is composed of vehicle nodes with wireless communication capability and roadside infrastructures. One of the obstacles is how to use batch verification to verify signatures simultaneously. To solve the problem of low verification efficiency in VANETs. This paper defines the problem and proposes an early-stopping factorial bitwise divisions (EFBD) algorithm. The proposed parallel-friendly algorithm achieves better performance in both theory and practice at low invalid signatures rate. Especially, in the parallel condition, the proposed algorithm costs only one aggregation-verification delay when the number of invalid signatures is 1.

1. Introduction
With the development of the automobile industry and the rapid growth of car ownership, the combination of vehicles and the rapidly developing network has given rise to Vehicular ad-hoc networks (VANETs). VANETs is the application of mobile ad-hoc networks (MANETs) in road traffic scene, which gradually evolves from MANETs. VANETs is an open mobile AD hoc network that utilizes wireless network communication technology to realize self-organizing wireless multi-hop communication between vehicle-to-vehicle (V2V) and vehicle-to-roadside facility (V2I). The goal of VANETs is to establish a self-organizing, convenient, inexpensive and open architecture of the vehicle communication network, so as to realize the automation of traffic warning, vehicle driving, enhance driving comfort and road traffic information query applications [1].

One of the obstacles is that VANETs requires strong timeliness. Existing single verification methods can verify a single signature. However, in a heavy traffic area, there are more vehicles which will lead to untimely verification and traffic chaos. Batch verifications emerged at the right moment. In batch verification, signatures are verified simultaneously and the process costs much less time than single verification. In this way, signatures can all pass the verifications when there exists no illegitimate signature. However, all the signatures will be rejected when there exists even just an illegitimate signature, which is a huge waste of time and can not take full advantage of batch verifications [2] [3].

In recent years, some researchers propose a few solutions to the aforementioned problem. Some just discard all signatures or verify them individually. Liu et al. [4] propose an efficient identity-based batch verification scheme for VANETs based on the ring signature to reduce the computation and communication cost. Huang et al. [5] use binary divisions detection (BDD) to solve the problem.

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd
However, it is not parallel-friendly. Matrix-detection algorithm (MDA) in [6] tries to reduce the escape probability of bad signatures, with which when a batch of signatures has less than four bad signatures or odd number of bad signatures, all bad signatures can be identified. But it only applies to a batch of signatures has less than four or odd number of bad signatures. Guan et al. [7] compare three methods in detecting invalid signatures in batch verification including random choosing method, small exponent test and randomly numbering test which is a simplified method of the MDA. The results show randomly numbering test is more efficient. Guan et al. [8] use cover-free families to certainly detect them when the number of invalid signatures is small enough. The probability of being detected will become higher when the number of invalid signatures is larger.

To solve the problem, this paper proposes a factorial bitwise divisions algorithm (FBD) and an early-stopping factorial bitwise divisions algorithm (EFBD) algorithm. Both adopt a classical bitwise method and are recursive. According to the index of each signature, they can be divided into a few groups. Then, each group will be verified aggregately. If the aggregate validation passes, all signatures in this group are supposed to be legitimate. Therefore, in the later aggregate verification process, we can exclude all signatures in this group. The size of each group is half the size of the input. Thus, half of the signatures are verified true at least. Eventually, we will get a set including all illegitimate signatures and \( n \) \((n \geq 0)\) good signatures. The aforementioned algorithm is recursively executed until the input and output are completely same. Later, two algorithms execute differently. FBD will reorder each group and make them as input for the algorithm, whereas EFBD singly verifies all signatures in the set.

2. The proposed bitwise divisions algorithm

2.1 Problem Definition

This paper focuses on the the strategy of grouping, that is, how to divide all signatures into one or more batches to determine all invalid signatures as soon as possible. Here, the problem can be converted into a logical problem: How to find all \( x \) \((0 \leq x \leq n)\) invalid nodes from the n nodes efficiently. Then, the problem can be described in mathematics as follows: Given an input set whose size is \( n \) \((n \geq 1, \mathbb{Z}^+\)). In the set, all n elements are either true or false, and assume that they are distributed randomly and testing an element is true or false costs much more time than other operations, such as AND. The number of false elements is \( x \), and the number of true elements is \( n - x \).

| Algorithm 1 Binary Divisions Detection Algorithm |
| :------------- | :------------- |
| **Input**: Non-Empty Set \( A \), start index \( i \), end index \( j \) |
| **Output**: Set \( F \) |
| 1: function BDD(\( A \), \( i \), \( j \)) |
| 2: if Verify(\( A \), \( i \), \( j \)) then |
| 3: \hspace{1em} return \( \emptyset \) |
| 4: end if |
| 5: if \( i == j \) then |
| 6: \hspace{1em} return \( A[i] \) |
| 7: end if |
| 8: \( F = BDD(A, i, (i + j)/2) \) |
| 9: \( F = F + BDD(A, (i + j)/2 + 1, j) \) |
| 10: return \( F \) |
| 11: end function |

2.2 Binary Divisions Detection Algorithm

The algorithm is described as follows. Firstly, aggregate the input and verify the aggregated signature. If the verification passes, it reveals all are valid, and the function returns an empty set; if not, it will divide the input into two batches. Then treat each batch as the new input of the algorithm, and call the BDD recursively. Once the input contains only one signature, the algorithm should be terminated. Eventually, the algorithm will return a set which exactly includes all invalid signatures.
2.3 Description of FBD

FBD is the basis of EFBD, so FBD is introduced firstly. FBD is a recursive algorithm, and the maximal recursion time is \( \log_2 n \). Note that the output set \( F \) equals to the input set \( A \) in the beginning. Firstly, it will check whether all nodes are invalid or not. If so, it will return the full set \( F \), which means all nodes in the input set \( A \) are invalid. And if the size of the set \( A \) is 1, it means that the algorithm has gone into the final layer and the algorithm will return.

**Algorithm 2 Early-stopping bitwise divisions algorithm**

| Input: | Non-Empty Set \( A \) |
|---|---|
| Output: | Set \( F \) |

1. function EFBD(\( A \))
2. if \( A.size = 1 \) then
3. if Verify(\( A \)) then
4. return \( \varnothing \)
5. else
6. return \( A \)
7. end if
8. end if
9. \( G\{S_0^0, S_0^1, S_1^0, S_1^1, \ldots \} = \text{DivideByBits}(A) \)
10. \( X = \text{Verify}(G) \)
11. \( F = F \setminus X \)
12. if \( F.size = 1 \) then
13. return \( F \)
14. end if
15. if \( F = \varnothing \) then
16. return \( \varnothing \)
17. end if
18. if \( F.size \leq 2 \log_2 n \) then
19. return SingleVerify(\( F \))
20. end if
21. if \( F.size = A.size \) then
22. choose two groups \( S_1^0, S_0^1 \) on random bit
23. \( F = F \setminus \text{EFBD}(\text{Reorder}(S_1^0)) \)
24. \( F = F \setminus \text{EFBD}(\text{Reorder}(S_0^1)) \)
25. else
26. \( F = F \setminus \text{EFBD}(F) \)
27. end if
28. return \( F \)
29. end function

Function DivideByBits is utilized to divide the set \( A \) to \( 2 \cdot \log_2 n \) groups, it is a bitwise operation. For example, a signature whose index is 12 and the binary representation is 001010. It is expected to be divided into groups \( S_1^0, S_2^0, S_3^0, S_4^1, S_5^0, S_6^1 \). The superscript represents that the group is divided by 0 or 1, and the subscript represents which bit is used to divide the set. Then, all groups in \( G \) are verified and the set \( X \) includes all invalid groups. Then we can filter \( F \) and make it smaller. Recursive boundary is the size of \( F \) reaching 1 or 0. If no nodes are filtered, then the algorithm randomly selects a certain bit and gets two original groups \( S_1^0, S_1^1 \) and reorder them and make them as input to recursively call the algorithm. If some nodes are filtered, then use the output set \( F \) as an input to recursively call algorithm until the filter does not work.

As shown in Fig. 1, red grids represent false nodes. In grouping stage, the black and gray grids represent different groups. In this instance, three groups are verified true, then we can get the final set including 8 nodes.

A few proofs will be necessary to demonstrate to support FBD and EFBD algorithms. Theorem 1 and Theorem 2 prove the recursive boundary and Theorem 3 proves that the result is correct. With the following four proofs, FBD and EFBD can work normally and efficiently.

**Theorem 1** If the output set \( F \) is an empty set, then the algorithm is supposed to be terminated.

**Proof.** As is described in the algorithm, if the output set \( F \) is an empty set, then all nodes are filtered.
and there exist no invalid nodes in set $A$.

Theorem 2 If the output set $F$ has just one signature, then the algorithm is supposed to be terminated.

Proof. As the algorithm goes, if the output set $F$ has only one invalid signature, it means that the number of both invalid and valid groups definitely is $\log_2 n$. According to FBD, one valid group can halve the size of $F$ and $\log_2 n$ valid groups.

Theorem 3 The output set $F$ consists of all invalid nodes.

Proof. Assume that the output set $F$ does not consist of all invalid nodes, which means that the algorithm has omitted certain invalid nodes. According to the algorithm, the only way to filter from the set $F$ is that the group is verified as true. However, when there exist invalid nodes in a certain group, it will definitely not pass the verification, and the whole group will not be filtered which contradicts the assumption.

The filter will definitely be invalid and cannot filter any valid nodes. Then, the algorithm will reorder each group. The group can go back to FBD as the group $A$ recursively. Eventually, the algorithm will go to the recursive boundary which can be seemed as a single verification.

2.4 Description of EFBD

In FBD, it is usual that the majority of the output set $F$ are invalid nodes. As FBD goes, it will branch to the final layer which is a waste of time. As an optimized algorithm for FBD, EFBD adopts single verification when the size of the set $F$ is less than $2 \cdot \log_2 n$. The algorithm is designed especially for parallel calculation, and $2 \cdot \log_2 n$ verifications can be calculated in a time circle. The specific algorithm is shown in Algorithm 2.

Most part of EFBD is completely consistent with FBD. The difference is that when the loop terminates, it will singly check each signature in $F$ which will filter all valid nodes. Eventually, it will get a final set $F$ including exactly all invalid nodes.

However, there still exist some drawbacks in EFBD.

Definition 1 For two or more binary codes, if both 0 and 1 appear in every bit, then they are called repulsion numbers.

If there exist false repulsion numbers, as EFBD algorithm goes, they will be divided into different groups no matter which bit is used to group. As a result, all groups will be verified false and the filter will be invalid.
3. Performance analysis

Define \( N = 2 \cdot \log_2 n \) which equals to the number of groups in a bitwise partition. For BDD algorithm, it can calculate at most \( N \) nodes in the same layer at a time. However, it cannot execute cross-layer computing. For FBD and EFBD, they can also calculate \( N \) nodes at a time.

If there exist no invalid nodes, then both algorithms need only one verification. Then, one invalid signature circumstance can be idealized. Time complexity of binary search is reproduced below

\[
O(n) = \frac{1}{N} + \frac{2}{N} + \frac{4}{N} + \cdots + \frac{n}{N} = 1 + \log_2 n \times \frac{n}{N}
\]  

(1)

In general, \( N \) is always greater than or equal to 2. Therefore, Eq. 2 can be further simplified.

\[
O(n) = 1 + \log_2 n
\]  

(2)

According to the algorithm, the time complexity of FBD is

\[
O(n) = \frac{2\log_2 n}{N} = 1.
\]

Considering the worst case, that is, all nodes are invalid, BDD algorithm will branch on each leaf. Its time complexity is as follows.

\[
O(n) = \left[ \frac{1}{N} \right] + \left[ \frac{2}{N} \right] + \left[ \frac{4}{N} \right] + \cdots + \left[ \frac{n}{N} \right] \geq \frac{2n}{N}
\]  

(3)

Time complexity of FBD is as follows.

\[
O(n) = \left[ \frac{2\log_2 n}{N} \right] + \left[ \frac{2^2(\log_2 n - 1)}{N} \right] + \left[ \frac{2^3(\log_2 n - 2)}{N} \right] + \cdots + \left[ \frac{2^{\log_2 n}}{N} \right] \leq \frac{2n - \log_2 n - 2}{N}
\]  

(4)

4. Simulation analysis

In the simulation experiment, BDD, FBD, EFBD are simulated to count \( T_a \) and \( T_v \) which represents the time of aggregation and verification, respectively, in different scales and invalid nodes. In addition, the parallel calculation is adopted and the scale \( N = 2 \cdot \log_2 n \).

For different invalid nodes and scales, the performance of the three algorithms is shown in Table 1 and Table 2. The number of nodes is from 4 (22) to 128 (27) and the invalid nodes are 1, 2, 3, 4, 5, respectively. Because the distribution of invalid nodes greatly influences the result, the simulation is repeated 1000 times and the result is equilibrated to avoid randomness. As showed in the Table 1, BS has a relatively constant number because of its regular hierarchic search. In comparison, FBD and EFBD have a high performance when the number of invalid nodes is less than 2 and 5, respectively. Because parallel calculation is adopted, BS has to go through \( \log_2 n \) layers to find the invalid nodes. However, as described in Section 2, FBD and EFBD can be seen as a filter, and it can efficiently filter most of the valid nodes and get a set containing invalid nodes when there exist a few invalid nodes. In Table 2, although the number of \( T_a \) of our algorithms is bigger than that of BBD, \( T_a \) is defined as a much smaller time cost compared to \( T_v \), which makes no difference in the practical circumstance.

Furthermore, Fig. 2 shows the total time cost comparison among the 3 algorithms. It should be noted that the total time cost includes both aggregation time and verification time. In our work, the \( T_a \) is 0.036 ms, and the \( T_v \) is 2.2899 ms. From the subfigures, when the number of elements and the number of false elements are small, the two proposed algorithms are better than the BDD significantly. Actually, in a real scene, the number of elements and the number of false elements is small.

![Figure 2. Total Time Cost of Different Number of Vehicles](image-url)
5. Conclusion
In this paper, a typical batch verification problem is defined. Furthermore, FBD and EFBD are proposed and they are featured by parallel calculation and filtering. They show their high efficiency after simulation experiments. There is still some work to do in the future. For example, simulate aggregators to send and receive signatures to simulate as much as possible, which can also test the parallel calculation more reasonably. In addition, FBD and EFBD can be optimized to attain higher performance in repulsion number problem.

Acknowledgment
This work is supported by Key international cooperation projects of the National Natural Science Foundation of China (No. 61520106007).

References
[1] R. Dewangan, F. Altaf, and S. Maity 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC) pp. 429–434
[2] X. Hu, W. Tan, C. Ma, F. Chen, and C. Yu 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS) pp. 343–346
[3] X. Hu, W. Tan, C. Yu, C. Ma, and H. Xu 2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI) pp. 1–6
[4] F. Liu and Q. Wang 2019 IEEE Vehicular Networking Conference (VNC) pp. 1–8
[5] J. L. Huang, L. Y. Yeh, and H. Y. Chien, “Abaka: An anonymous batch authenticated and key agreement scheme for value-added services in vehicular ad hoc networks” IEEE Transactions on Vehicular Technology vol. 60 no. 1 pp. 248–262
[6] Y. Huang, C. Lin, and F. Leu 2011 First International Conference on Data Compression, Communications and Processing pp. 299–306
[7] D. J. Guan, E. S. Zhuang, I. C. Chung, and Y. Lin 2017 12th Asia Joint Conference on Information Security (AsiaJCIS) pp. 10–14
[8] D. J. Guan, E. S. Zhuang, Y. Lin, and I. C. Chung 2017 IEEE Conference on Dependable and Secure Computing pp. 68–72