Dino: A Block Transmission Protocol with Low Bandwidth Consumption and Propagation Latency

Zhenxing Hu and Zhen Xiao
School of Computer Science, Peking University
{hzx, xiaozhen}@pku.edu.cn

***Abstract*—Block capacity plays a critical role in maintaining blockchain security and improving transactions per second (TPS). Increasing block capacity can help attain higher TPS, but it also prolongs block propagation latency and degrades system security. In the present paper, we argue that existing work compressing the block size to shorten block propagation latency introduces an undesired side effect, which is that the size of compressed blocks will increase with transaction volume. Instead, we propose Dino, a new block transmission protocol between two peers. Once a node receives a Dino block, it can recover the original block with that Dino block and transactions in its transaction pool. Since Dino transmits block construction rules instead of compressed block content, it has good scalability to transmit blocks with larger transaction volume. We deploy Dino into Bitcoin and Bitcoin Cash to compare it with the state-of-art protocols: Compact, XThin, and Graphene. For a block with 3,000 transactions, its corresponding Dino block is no more than 1 KB in size, which is only 4% of a XThin block, 5% of a Compact block, and 20% of a Graphene block. The size of Dino blocks stays constant when the transaction volume reaches Bitcoin and Bitcoin Cash's protocol limit. Simulation experiments show that Dino scales well with higher transaction generation rates and can reduce block propagation latency.**

***Index Terms*—blockchain, P2P, block transmission**

I. INTRODUCTION

TPS is a pain point in the public blockchain. A general method to improve TPS is adding more transactions in one block. However, this leads to the increase of block size as well as the block propagation latency, incurring security issues like forking [1], double spending [2] [3], and other mining attacks [4] [5]. A best-of-both-world solution is to compress the block size while increasing the number of transactions in it. Existing mainstream public blockchains like Bitcoin (BTC) and Bitcoin Cash (BCH) have already deployed block compression approaches, namely the Compact [6], Graphene [7], Xthinner [8], and XThin [9]. Although they achieve good compression results, they still cannot avoid the increased block size due to increased transaction volume since their mechanisms depend on compressing the original block content. Furthermore, previous research [10] [1] show that when the block size exceeds 20 KB, the block propagation latency increases rapidly as its size increases. An efficient block transmission protocol will have the following benefits: (1) Each block can contain more transactions to improve TPS; (2) It takes less bandwidth to transmit blocks; (3) It helps reduce block propagation latency and improve system security. These features make such a protocol appealing. However, it is still uncertainty that if there exists such a protocol.

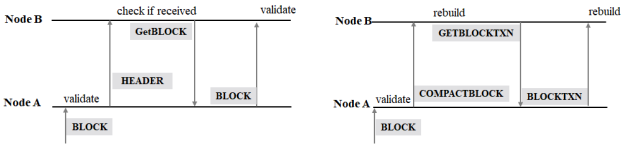
Due to the research gap, we propose a new block transmission protocol, Dino, which solves the drawback of the previous approaches by transmitting block reconstruction rules instead of compressed block content. When transmitting a new block in Dino protocol, the sender only needs to transmit missing transactions and a block reconstruction rule. The recipient can quickly recover the original block based on the reconstruction rule and transactions in its transaction pool. Because Dino no longer transmits block content, its bandwidth consumption is low and does not increase with the transaction volume in a block. In summary, this paper makes the following contributions:

- We propose a new block transmission protocol, Dino, that transmits block construction rules instead of compressing block content.
- We deploy Dino on Bitcoin and Bitcoin Cash to compare it with the state-of-the-art block transmission protocols. For a block with 1 MB, the size of a Dino block is 4% of a XThin block, 5% of a Compact block, and 20% of a Graphene block. Besides, the size of Dino blocks remains almost constant when the transaction volume increases from zero to its maximum volume limit.
- We estimate Dino's performance in simulation experiments with larger transaction volume and higher transaction generation rates. The results show that Dino's bandwidth consumption increases slowly when the transaction generation rate rises, which means Dino can help reduce block propagation latency and improve system security.

II. BACKGROUND

Blockchain is a hash list of blocks that contain transactions. The blocks, which contain many transactions generated by participants, are stored in the peer-to-peer network which manages the blockchain. Before being packaged into a block, every transaction must be validated by every node in the network. In general, there are two types of nodes in the blockchain network, light nodes and full nodes. The light nodes only store the block headers of blockchain, and the full nodes hold the complete blockchain data and can mine new blocks to extend the blockchain. In the following, we take Bitcoin as an example to explain activities in the blockchain network.

Zhen Xiao is corresponding author.



(a) The Full protocol. (b) The Compact protocol.

Fig. 1. The Full and Compact protocol.

A. Transaction Relay

In the blockchain network, most transaction relay protocols are variations of Gossip [11] [12]. For instance, Bitcoin, Bitcoin Cash, and Ethereum use Diffusion [13] to relay transactions. Since a node usually has multiple connections and a typical transaction is about several hundred bytes, a node usually does not directly relay a newly received transaction to its peers. For example, when node A receives a new transaction, A checks the transaction's validity first. If it is valid, A will broadcast an inventory (INV) message that contains the transaction hash to other neighbors. When A's neighbor B receives that INV message, it will send a GetData message to A to ask for that transaction if B does not have it. As we can see from the above process, it usually takes three messages to transmit a new transaction between two nodes.

If a newly received transaction is valid, a node will put it into its transaction pool. The transaction pool (also called mempool) is used to store information on unconfirmed transactions.

B. Mining Process

Miners are nodes that compete to extend the blockchain. They collect transactions generated in the network and package them into blocks continuously.

To motivate nodes to participate in the mining process, nodes who mined a valid new block will get some rewards. Miners generate a block according to Equation 1 in which B denotes a block, C denotes the block capacity, P denotes its mempool and F denotes the transaction package algorithm. As miners prefer to package transactions with higher fee rates into blocks to obtain higher profits, the transaction package algorithms F in BTC and BCH are designed to package transactions with higher fee rates first. When a miner finds a proper nonce for the mining block, it will broadcast the new block to its peers in a snap to guarantee its leading position. Besides, miners also need to receive newly generated blocks as early as possible to guarantee that they always mine on the legal chain.

$$B = F(C, P) \tag{1}$$

C. Block Transmission

There are two kinds of block relay protocols in the Bitcoin core client. One is the Full block relay protocol which transmits the whole block data, and the other is the Compact block relay protocol [6], which only relays the first 6 bytes of a transaction hash of all transactions. When a node receives a Compact block, it needs to reconstruct the block with transactions in its mempool first. A node can reconstruct the block if all transactions in the new block exist in its mempool and there is no hash collision. Otherwise, it needs to send a message to request missing transactions. Fig. 1 demonstrates those two protocols. There are also some other block transmission protocols such as XThin [14] and Graphene [7], which are deployed on Bitcoin Cash.

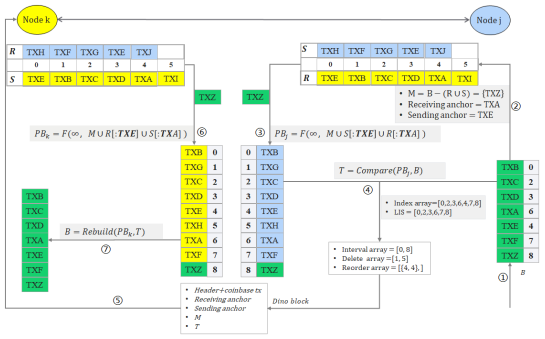


Fig. 2. An example of Dino protocol.

III. DINO PROTOCOL

The bandwidth consumption of the current block compression protocols increases linearly with their transaction volume because they always compress the whole block content, such as the Compact [6], XThin [14], and Graphene [7]. Henceforth, if the transaction volume in a block doubles, their bandwidth consumption will double too. The aim of this paper is to explore a block transmission protocol with good scalability to a larger transaction volume among a decentralized blockchain network. The next section is focused on the protocol conditions of Dino.

A. Protocol Conditions

The first condition is that almost all transactions in a new block have already existed in nodes' mempools. As there is no way to let a node obtain missing transactions in a new block other than directly sending them, the current block compression protocols such as the Compact [6], XThin [14], and Graphene [7] are based on this assumption. Based on this condition, a node can reconstruct new blocks through its mempool. We will measure this condition in section IV.

The second condition is that miners are profit-oriented and prefer to package transactions with high fee rates into blocks. It is nearly the truth since the nature of rational miners is profit-reapers. As the transaction package algorithms in BTC and BCH are designed to gain the maximum profit, miners usually use them to generate blocks. Based on this condition, it is possible for a node to predict which transactions in its mempool will occur in the next new block. In the next section, we will describe the details of Dino.

B. An example of the Dino Protocol

The node in this paper refers to a full node that participates in the transaction and the block relaying process. For ease of

TABLE I
DEFINITION OF SYMBOLS IN DINO.

Symbol	meaning
i, j, k	Three full nodes.
C	The block capacity.
B	A valid block mined by a miner.
P	The mempool of a node.
F	The transaction package algorithm.
R	The transaction hash receiving list.
S	The transaction hash sending list.
M	The missing transaction set get from $B - (R \cup S)$.
PB	The prediction block.
PB_x	The prediction block built by node x.
T	The transformation message that transforms PB to B.
L_r	The local received transactions for R.
R_r	The remote received transactions for S.

exposition, when we say a node sends or receives a transaction hash, it means a node sends or receives an INV message, and vice versa. Suppose a node j receives a new block B and tries to send it to its peer k. Fig. 2 is an example that shows how node j transmits B to node k through the Dino protocol. In this instance, we ignore the block header and the coinbase transaction of B. Table I lists all symbols we used in the Dino protocol.

In the example, node j uses a receiving list R to store transaction hashes it receives from node k, and it also uses a sending list S to store transaction hashes it sends to node k, and so does node k. During the transaction relay process, when node j sends a transaction hash to node k, node j puts the transaction hash into its S, and node k puts that hash into its R when it receives that transaction hash.

In step 1, node j receives a new block B. In step 2, node j iterates all transactions in B and compares them with transactions in its R and S. After that, node j finds that transaction TXZ does not exist in its R and S, then puts TXZ into the missing transaction set M, which contains transactions that both node k and node j do not have. Further, when iterating over all transactions in B, node j finds transactions in S that locate behind TXE and do not exist in block B. It also finds transactions in R that locate behind TXA and do not exist in B. We call TXE and TXA the sending anchor and the receiving anchor, respectively.

In step 3, node j inputs transactions in M, $S[: TXE]$ (include TXE), and $R[: TXA]$ (include TXA) into Equation 1 and gets a prediction block PB_j. When creating the PB_j, node j sets the block capacity to infinite to ensure all transactions in B will occur in PB_j.

In step 4, node j compares PB_j with B and gets a transformation message T that contains information about how to transform PB_j to B.

In step 5, node j packs M, sending anchor, receiving anchor, and T into a Dino block and sends it to node k. We will explain the detail of T in section III-F.

In step 6, node k receives that Dino block and uses transactions in M, $S[: TXA]$, and $R[: TXE]$ to generate a prediction block PB_k which is exactly the same as PB_j.

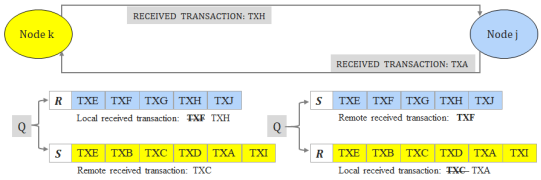


Fig. 3. Two peers sending received transaction messages.

In step 7, node k transforms PB_k to block B with T in that Dino block.

As Dino block does not compress block content and only contains some missing transactions and a transformation message, it costs only several hundred bytes to transmit a large block. Before further discussing Dino's overall performance, we will focus on its components first.

C. Sending and Receiving Lists

As Dino is a protocol that sends block reconstruction rules instead of block content, the key here is that two peers can generate two identical prediction blocks. To this end, there should be a common transaction set for two peers, which functions as an object of reference. The problem is how to maintain that common transaction set and keep its consistency.

In Dino, each node uses a sending list S and a receiving list R to function as a mempool of reference. However, it will cost too much for every two peers to maintain a common mempool. Thus, Dino allows S and R only store transaction hashes to save memory, and the complete transactions are stored in a node's own mempool. For two nodes, once a peer receives a new transaction that does not exist in its S and R, it will send that transaction hash to the other peer and put it into its S. Moreover, once a peer receives a transaction hash from the other peer, it will put it into its R. Hence, the transaction order in a node's R is the same as that in the other node's S. The design of the two lists is perfectly compatible with the transaction broadcast process in the current public blockchain, such as Bitcoin and Bitcoin Cash.

D. Received Transaction Message

In the transaction broadcast process, when node k sends an INV message to node j, node j may not ask node k for that transaction because node j may already have received it or node j is waiting for another node to send that transaction. It means node k can not know which transactions in its S node j has received. That prevents node j and node k from generating two identical prediction blocks. Thus, Dino allows each node periodically to send a message to the other peer to tell which transactions it has received so far. That message is called the *received transaction message*. Each node uses two variables: local received transaction L_r and remote received transactions R_r. L_r is maintained by the node itself and R_r is maintained by the other peer.

Fig. 3 shows how node j and node k send *received transaction messages* to each other. Node k sends a *received transaction message* that contains a transaction hash TXH to

node j, which means node k has received all transactions in its receiving list R before, including TXH. After sending that message, node k updates its L_r from TXF to TXH. Node j will update its R_r from TXF to TXH when it receives that message. Similarly, node j sends a *received transaction message* that contains a transaction hash TXA to node k, which means node j has received all transactions in its receiving list R before, including TXA. After sending that message, node j updates its L_r from TXC to TXA. Node k will also update its R_r from TXC to TXA when it receives that message.

By periodically sending *received transaction messages*, both node j and k can know which transaction messages the other one has received in their receiving and sending lists. For the nodes that significantly care about privacy leakage by periodically sending received transaction messages, we provide an alternative approach in section VI-B.

E. Prediction Block

The recipient can successfully transform PB to B on the condition that PB contains all transactions in B. Therefore, upon receiving a new block, the sender needs to find those transactions in the new block who do not exist in S and R. We call those transactions *missing transactions*, and they will be put into the missing transaction set M and sent to the recipient. Having found the *missing transactions*, the sender will put transactions in M, R, and S into the transaction package algorithm F to generate a PB whose transaction order is very similar to the transaction order in B. It should be noted that not all transactions in R and S are used to generate PB because some transactions in R and S that do not exist in the new block can be eliminated. We will discuss that process in section III-G.

In Dino's implementation, each node uses transactions in its mempool to generate PB, and the transaction hashes in M, R, and S function like filters to filter out transactions absent in them. Since two peers maintain S and R, conflicting transactions may occur in M, R, and S. Dino gives transactions in M higher priority than those in R and S. If there are conflicting transactions in R and S, the transactions with higher fee rates are considered valid.

F. Transformation Message

After building PB, the sender needs to compute the transformation message T. As PB contains all transactions in B, the transactions in B exist in an interval of PB. Moreover, since PB and B are generated by the same greedy algorithm F, their transaction orders are very similar. By deleting and reordering a few transactions in PB, the transaction order in PB can be the same as that in B. The sender gets T as follows:

(1) For each transaction in B, the sender finds their corresponding indexes in PB and put them into an array, which is called the *index array*.

(2) The sender puts the minimum and the maximum elements of the *index array* into another array. We call it the *interval array* and use letter I to denote it. Thus, all transactions in B exists in $PB[I[0] : I[1]]$.

(3) The sender finds the longest increasing subsequence (LIS) in the *index array*, and we call it a *LIS array*.

(4) The sender iterates each transaction in $PB[I[0] : I[1]]$. If that transaction does not exist in B , the sender puts the index of that transaction into a *delete array*; if the transaction exists in B but does not exist in the *LIS array*, the sender puts its index in PB as well its index in B into a *reorder array*. If that transaction exists in the *LIS array*, the sender does nothing.

A transformation message T contains three elements, the *interval array*, the *delete array*, and the *reorder array*. The *interval array* in Fig. 2 is [0, 8], which means transactions in B exist in $PB[0 : 8]$. The *delete array* in Fig. 2 is [1, 5], which means node k should delete TXG and TXH. Each element in the *reorder array* contains two indexes, and the *reorder array* in Fig. 2 means node k should put the transaction at $PB[4]$ in $B[4]$.

When node k receives a Dino block, it can construct PB_k which is identical to PB_j. Then, node k can transform PB_k to B with T in that Dino block.

G. Optimize Dino's Bandwidth

Since all transactions in B exist in an interval of PB, "transaction in PB" in this section means transactions in $PB[I[0] : I[1]]$. The transaction set that generates PB is different from the one generating B, or, to be more precise, the former contains some transactions while the latter does not. We call those transactions the *redundant transactions*. The more *redundant transactions* there are, the more different the transaction order of PB from that of B is, and the more transactions there are to be deleted and reordered in T. Hence, we should try to eliminate those *redundant transactions* before building PB. We still take Fig. 2 as an example to explain our viewpoint. What is more, we suppose miner i mines B at time $t1$ and node j receives it at time $t2$.

1) Block Propagation Latency: Block propagation latency has a significant impact on the size of T. $t2 - t1$ is the time for B to propagate to node j. Although the transactions generated during block propagation do not exist in B, they may enter node j and node k's sending and receiving lists during the transaction broadcast process. If node j directly uses all transactions in its S and R to generate PB_j, it will lead to a significant difference between the transaction orders of PB_j and B. Take Fig. 2 as an example, suppose node j's L_r for R is TXI, and R_r for S is TXJ. If node j uses transactions in $R[: L_r]$ and transactions in $S[: R_r]$ to generate PB_j, PB_j will contain TXJ, TXH and TXI. However, those three transactions do not exist in B because they had not entered the miner's mempool when the miner generated B. As the blockchain network generates transactions all the time, if it takes much more time for B to propagate to node j, more *redundant transactions* will enter into its sending and receiving lists. The consequence is that the transformation message contains many *redundant transactions* that need to be deleted and reordered, making the Dino block very large.

We use two anchors to eliminate the impact of block propagation latency. One is the sending anchor for S, and the other is the receiving anchor for R. Anchor is a transaction hash, for a transaction hash list, transactions who are located behind an anchor do not exist in B. For node j, at step 2 in Fig. 2, the receiving anchor is TXA, which means transactions who are located after TXA do not exist in B, and the sending anchor is TXE, which means transactions who are located after TXE do not exist in B. Finally, The node j uses transactions in M, $S[:TXE]$, and $R[:TXA]$ to build PB_j. By using sending and receiving anchors, *redundant transactions* generated during the block propagation process can be excluded.

2) Transaction Generation Rate: Besides the block propagation latency, the transaction generation rate, namely the number of transactions generated every second, is another factor that causes the difference between the transaction orders of PB_j and B. We assume the time for a transaction to propagate to the whole network is no longer than Δt. For node j's receiving anchor TXA, we suppose node j receives TXA at time tn and miner i receives TXA at time tm, and then we get Equation 2.

$$\begin{aligned} & |tn - tm| \leq \Delta t \\ \Rightarrow \quad & tn - \Delta t \leq tm \leq tn + \Delta t \end{aligned} \tag{2}$$

That B contains transaction TXA means miner i firstly receives TXA and then mines B. Thus, we get Inequality 3.

$$tm \leq t1 \tag{3}$$

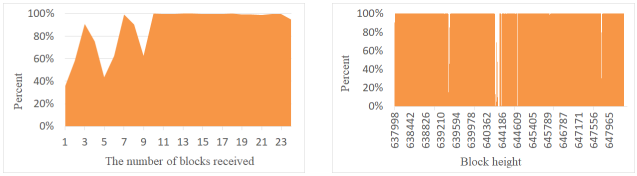
After $t1$, transactions miner i receives at $tx, (t1 \leq tx)$ may be received by node j at time t, then we get Inequality 4 according to Inequality 2.

$$tx - \Delta t \leq t \leq tx + \Delta t \tag{4}$$

Based on Inequality 2, Inequality 3, and Inequality 4, we get Inequality 5. Inequality 5 implies that transactions miners received after $t1$ do not exist in B but they may be received by node j between $tn - 2\Delta t$ and tn.

$$\begin{aligned} & tn - \Delta t \leq tm \\ \Rightarrow \quad & tn - \Delta t \leq t1 \leq tx \\ \Rightarrow \quad & tn - 2\Delta t \leq t1 - \Delta t \leq tx - \Delta t \\ \Rightarrow \quad & tn - 2\Delta t \leq t \end{aligned} \tag{5}$$

If the transaction generation rate is v, there are at most $2\Delta t \times v$ *redundant transactions* in $PB[I[0] : I[1]]$. Since those *redundant transactions* are located in the tails of S and R, Dino uses two bitsets [15] to eliminate those *redundant transactions*; one bitset is for the sending list, and the other is for the receiving list. For S, the sender traces $2\Delta t \times v$ transactions before the sending anchor, and for each transaction, if it exists in the new block, we set its bit to '1'; if it does not exist in the new block, we set its bit to '0', and so does R. Those transactions whose bit is '0' are *redundant transactions*, and they will not be used to generate the prediction block. The sender puts the two bitsets into that Dino block and sends them to the recipient. Upon receiving that Dino block, the recipient eliminates *redundant transactions* according to those two bitsets and anchors before building PB.



(a) First 24 blocks after restart. (b) Proportion of transactions.

Fig. 4. Transaction Proportion in mempool.

One bit can filter out one redundant transaction. It helps Dino eliminate those *redundant transactions* caused by the transaction generation rate with low bandwidth cost. Since the two anchors and two bitsets can help eliminate almost all *redundant transactions*, the transaction order in PB is almost identical to that in B.

The next section is focused on the evaluation of Dino.

IV. EVALUATION

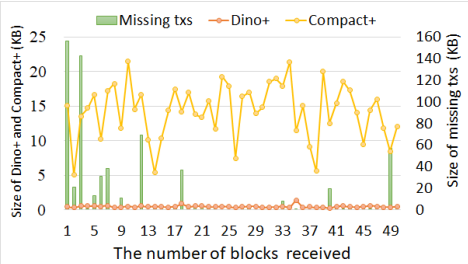
As far as we know, the Compact, XThin, Xthinner, and Graphene are the state-of-art protocols in the current production blockchain network. As the author of Xthinner states that Xthinner is not as compact as Graphene [16], it is not considered for comparison in this paper. The Compact is deployed on the Bitcoin and Bitcoin Cash, and the XThin and Graphene are deployed on the Bitcoin Cash. Hence, we deploy Dino on Bitcoin Core 0.19 and Bitcoin Cash's 0.19 client to compare with Compact, XThin, and Graphene.

A. Implementation Detail

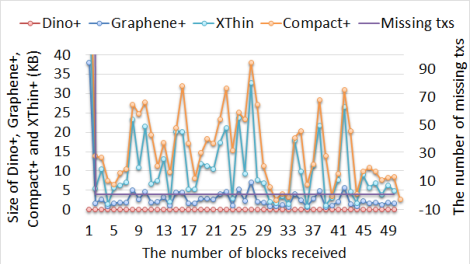
In implementing Dino, taking both the performance and privacy into account, we let a node send a *received transaction message* to its peer when it continually receives at least four transactions in its R, and let a *received transaction message* contain only the first 6 bytes of a transaction hash to save bandwidth. In current Bitcoin, the transaction generation rate v is about 7 tx/s, and the time for a new transaction to reach 90% nodes is about 16 seconds [17]. Thus, we set the bitset in Bitcoin and Bitcoin Cash to 28 bytes. After six-block confirmation for a block, a node can delete transaction hashes that belong to that block in the sending and the receiving lists to save memory.

B. Experiment Setup

We use 16 nodes to start the experiment and each of them has 4 cores and 16 GB memory, the bandwidth among our nodes is 1.5 Gb/s. Those nodes are distributed in Asia, America and Europe. Each node connects to at least 8 nodes and can have at most 125 connections. A node starts as follows: (1) Each node independently connects to the blockchain network to receive new blocks and transactions; (2) The node randomly connects to our own nodes with decreasing

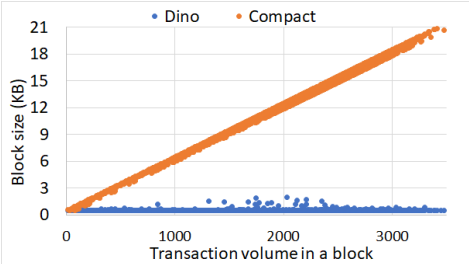


(a) Comparison in BTC.

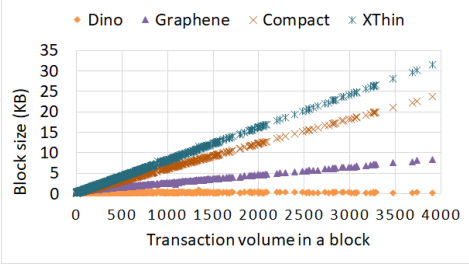


(b) Comparison in BCH.

Fig. 5. Bandwidth comparison with missing transactions.



(a) Comparison in BTC.



(b) Comparison in BCH.

Fig. 6. Bandwidth comparison with no missing transactions.

connection numbers. The first node connects to 15 other nodes, the second node randomly connects to 14 nodes, and the third node randomly connects to 13 nodes, etc..

C. Observation

Dino assumes that almost all transactions in a new block have already existed in nodes' mempools. To have a clear understanding of this assumption, we run a bitcoin node with the default configuration for one month to collect data. We stop and restart the node during the experiments to test the impact of missing transactions on block transmission. Fig. 4a shows that after receiving the first 12 blocks, the node's mempool contains the overwhelming majority of the transactions in new blocks. The proportion of missing transactions decreases with the increase of running time. Fig. 4b shows the proportion of transactions that already exist in the node's local mempool. We find that at least 97% of transactions in a new block exist in the local mempool. The white blank low ebbs in Fig. 4b is caused when the node finished block synchronization and started to receive new blocks; its mempool has a low transaction proportion because the node misses many transactions during its offline time. The transactions in mempool originate from transaction broadcast process, thus Dino's first condition is easy to satisfy.

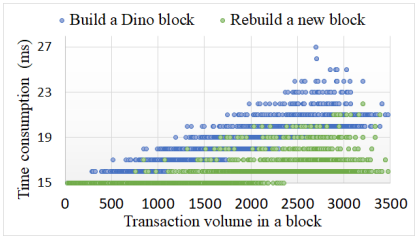
Fig. 4 also shows the inefficiency of Full block protocol and Compact block protocol. The Full block transmission protocol ensures that the recipient can ultimately receive all the new block transactions, but it wastes much bandwidth and may cause bandwidth spikes in the block relay process if the block size is too large. The Compact block just transmits transaction hashes whose size is much smaller than a full block. However, another round trip is needed to receive missing transactions if the recipient fails to reconstruct the new block.

D. Bandwidth Consumption

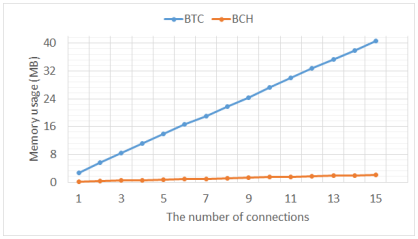
*1) **Bandwidth with Missing Transactions**:* Because many transactions are missing during the starting hours, Dino, Compact, XThin, and Graphene all need to transmit the missing transactions. To clearly show the impacts of missing transactions, we use "Dino+", "Compact+", "XThin+" and "Graphene+" to denote their bandwidth consumption excluding missing transactions. Fig. 5 shows the bandwidth consumption of those protocols when our nodes receive the first new 50 blocks after synchronizing history blocks. The left y-axes in Fig. 5a and Fig. 5b denote the size of "Dino+", "Compact", "XThin+" and "Graphene+". The right y-axis in Fig. 5a denotes the size of missing transactions. Because the transaction generation rate in Bitcoin Cash is lower than that in Bitcoin, our nodes can collect all transactions in BCH quickly, thus to show the result clearly, we set the right y-axis in Fig. 5b as the number of missing transactions.

As transactions in a block are sorted by their hashes in Bitcoin Cash, the transaction order in the new block is always a subsequence of the transaction order in Dino's prediction block. Thus, Dino's bandwidth consumption in Bitcoin Cash is better than that in Bitcoin. The results show that if there are missing transactions, the sizes of Dino, XThin, Compact, and Graphene largely depend on the number of missing transactions, while Dino's bandwidth is still the lowest.

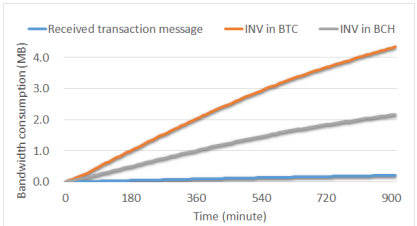
*2) **Dino's Scalability to Transaction Volume**:* When a node runs long enough, almost all transactions in a new block have already existed in nodes' mempools. Fig. 6 shows the bandwidth comparison when there are no missing transactions. When the transaction volume increases from zero to its upper limit, the sizes of the Compact block, the XThin block, and the Graphene block increase linearly, while the size of Dino blocks is about 600 bytes in BTC and 400 bytes in BCH. Furthermore,



(a) Time efficiency



(b) Memory usage



(c) Extra bandwidth usage.

Fig. 7. Dino's time, memory and extra bandwidth consumption.

TABLE II
BREAKDOWN OF BANDWIDTH COST IN A DINO BLOCK

Component	Bandwidth(BCH) %	Bandwidth(BTC) %
coinbase	61.13%	66.53%
header	32.59%	16.33%
deleted txs	6.00%	12.86%
reordered txs	0.00%	3.79%
bitset	0.28%	0.49%

the size of Dino blocks also remains constant. This is because transaction volume has no impact on the size of Dino blocks. For a block with 3,000 transactions, its corresponding Dino block is no larger than 1 KB, which is 4% of a XThin block, 5% of a Compact block, and 20% of a Graphene block. The result shows that Dino has good scalability to transaction volume for current Bitcoin and Bitcoin Cash.

*3) **Bandwidth Component Statistics**:* We break the component of Dino blocks that have no missing transaction. Table II shows the average proportion of every component in a Dino block in BTC and BCH, respectively. The highest proportions are the coinbase transaction and block header, which means that when there are no missing transactions, the size of a Dino block can be tiny. The low proportion of deleted and reordered transactions means the transaction orders between prediction blocks and the new block are very similar, proving our assumption that miners are greedy and prefer to package transactions with higher fee rates. As the transactions in a block are sorted by their hashes in BCH, the proportion of reordered transactions in BCH is always zero.

E. Time Efficiency and Memory Usage

*1) **Time Efficiency**:* We test Dino's time consumption of building a Dino block and recovering the new block from a Dino Block. To build a Dino block, the sender has to construct PB and calculate the difference between PB and B, whose time complexity is $O(nlog(n))$, and the time complexity to recover a new block is $O(n)$. Fig. 7a is a scatter plot that shows the time consumption of Dino. For a block with about 3,000 transactions, it is very fast to recover a new block and build a Dino block.

*2) **Memory Usage**:* We collect data on nodes' memory usage of sending lists and receiving lists. For each transaction, a node only needs to contain its hash and a pointer to the transaction. The memory usage increases with the transaction hashes in the two lists, and when the nodes have collect transactions in the network, their memory costs will keep constant. It is observed that Bitcoin's mempool usually has about 60,000 transactions, and it costs about 2.7 MB to store those transactions. For a node with 15 connections, it cost about 40.5 MB. In Bitcoin Cash, there are usually 3000 transactions in a node's mempool, and it costs 0.15 MB to store transaction hashes. For a node with 15 connections, it cost about 2.1 MB. Fig. 7b shows a node's memory usage when using the Dino protocol to transmit blocks to different peers. We believe Dino's memory usage is acceptable in the current blockchain network.

*3) **Extra Bandwidth Consumption**:* To understand the bandwidth of Dino's *received transaction message*, we collect data of the bandwidth consumption of INV messages as well as the bandwidth consumption of *received transaction messages* in Dino. Fig. 7c shows bandwidth consumption of INV messages and *received transaction messages*. The result shows that as the process goes on, the INV message occupies large bandwidth consumption while the *received transaction messages* occupies less bandwidth. The bandwidth consumption of *received transaction messages* is only about 5% of the bandwidth consumption of INV messages in the transaction relay process. We believe that the extra bandwidth consumption is acceptable.

V. SIMULATION

As BTC and BCH's transaction generation rates and transaction volume are limited, we test Dino's scalability to larger transaction volume and higher transaction generation rates as well as its network effects in our simulator.

A. Simulator Setup

We modified an open-source Bitcoin Simulator [18] to support transaction relay and Dino protocol. There are two types of nodes in our simulation: full nodes (do not mine blocks) and miners. We set nodes number to 9000 full nodes [19] and 16 miners. For full node configuration, the bandwidth and network latency come from Verizon [20], [21]. Each full node connects to at least eight connections and has at most 125 connections, which is the default configuration in Bitcoin. For miners' configuration, their number, connections and bandwidth come from blockchain.info. We set the coinbase transaction as 400 bytes and the block interval as 10

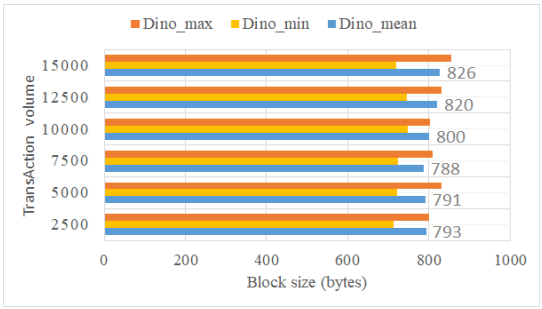


Fig. 8. Dino's scalability to larger transaction volume.

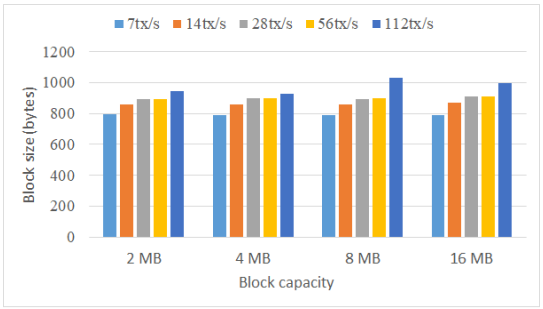


Fig. 9. Scalability to larger transaction generation rates.

minutes. We collect 100,000 transactions from the Bitcoin network to use their parameters in the simulation. In each experiment, we let miners generate 1,000 blocks to measure Dino's performance.

The nodes start up as follows: (1) Miners connect to each other; (2) Each full node connects to eight random nodes. Every second, each node generates a transaction with probability p. If the node number is n, transaction generation rate is v, then $p = \frac{v}{n}$. It should be noted that we do not account for heterogeneous nodes and leaving nodes during the transaction relay phase (churn).

B. Scalability to Larger Transaction Volume

The block capacity is set to increase from 1 MB to 6 MB and we test Dino's scalability when the transaction volume increases from 2,500 to 15,000. To provide miners with enough transactions to fill the block, we set the transaction generation rate at 28 tx/s. Fig. 8 shows the maximum, minimum and mean bandwidth consumption of Dino blocks with different transaction volumes. The number near the blue bar shows the average bandwidth of Dino blocks. For a fixed transaction volume, the size of Dino blocks is stable. The result further proves Dino's good scalability to larger transaction volume.

C. Scalability to Larger Transaction Rate

In section III-G we claim that transaction generation rates have a significant impact on the size of Dino blocks. We set the transaction generation rate to 7 tx/s, 14 tx/s, 28 tx/s, 56 tx/s and 112 tx/s. For each transaction generation rate, we test the bandwidth consumption of Dino under block capacity with 2MB, 4MB, 8MB and 16MB, respectively. For a larger block capacity, we set a longer block interval to let miner have enough transactions to package. The results are shown in Fig. 9. For a fixed block capacity, the size of Dino blocks increases slowly with the increase of transaction generation rate because the bitset filters out many *redundant transactions*. When the transaction generation rate is 7 tx/s, the bitsets in Dino can filter out around 20 transactions. When the transaction generation rate is 112 tx/s, the bitsets in Dino can filter out at most 1000 transactions. The higher the transaction generation rate is, the more *redundant transactions* the two bitsets can filter out. The low costs of bitsets make it possible for Dino to have good scalability to higher transaction generation rates.

D. Block Propagation Latency

We measure block dissemination latency to reach a certain number of nodes in the network with different protocols. Fig. 10 shows that as the block size increases, the block propagation latency of Compact and Graphene protocols increase, while Dino's block propagation latency is almost constant. It takes much more time for Compact and Graphene blocks to reach the final 5% nodes that have poor bandwidth. However, as the Dino block is much smaller and its block size stays almost constant when the block increases from 1 MB to 32 MB, it takes less time for a Dino block to reach the last 5% nodes. When the block increases to over 8 MB, Dino has a substantial advantage over Compact and Graphene. Consider the time for a block to reach 95% nodes of the network(t_{95}), the probability for a fork to occur is approximately as:

$$P(fork|Interval = 600sec) = 1 - e^{\frac{-t_{95}}{600}} \quad (6)$$

When the block size is 32 MB, the forking rate under Dino protocol is 0.58%, which is still smaller than current Bitcoin's forking rate [18]. The result shows that Dino can help improve TPS without degrading system security.

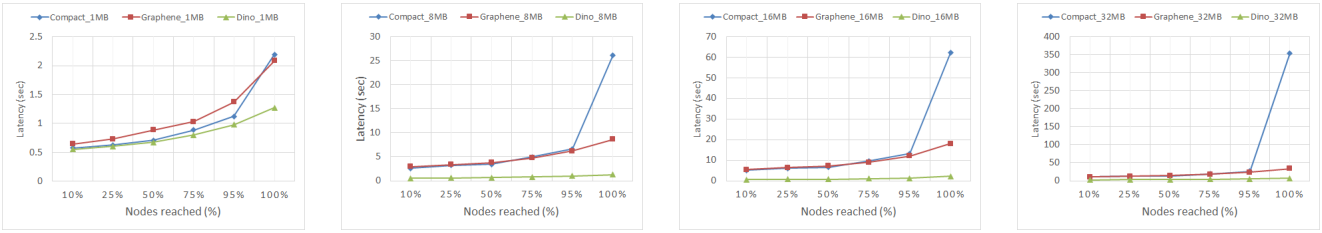
VI. DISCUSSION

A. Malicious Nodes

The fact that Dino needs two honest peers sounds like a strong assumption. However, current transaction and block transmission protocols all need two honest peers. A Dino node can still identify double-spending transactions, invalid blocks and DoS attacks by itself and disconnect from malicious nodes. Furthermore, Dino helps block propagate faster among the blockchain network, improving system security. We believe rational miners and nodes are willing to use it.

B. Privacy Leakage

Dino needs two peers to tell each other which transactions they have received in their receiving lists periodically, which may bring extra privacy concerns for a node whose peer nodes may know the time when it received a transaction [3] [22]. For nodes who are extremely concerned about their privacy, they do not need to send *received transaction messages*



(a) block with 1MB (b) block with 8MB (c) block with 16MB (d) block with 32MB

Fig. 10. Block propagation latency with different block capacities.

periodically. When a recipient asks for a block, it should send a block request message which contains a *received transaction message* to the sender. This variation sacrifices block transmission latency to obtain higher privacy and lower bandwidth consumption.

C. Network Churn

Dino is not suitable for those nodes who connect or disconnect with other nodes frequently. Previous work shows that 80% of connections among nodes keep for at least one day [23], and that the average share of peers connected for at least one day varies between 55% and 75% [24]. As long-time connections occupy a large proportion, Dino can produce a marked effect for those peers.

D. Backward Compatibility

Dino does not change the consensus protocol of Bitcoin or Bitcoin cash and can be deployed incrementally (as long as the two peers at both ends of the connection support it). Since we do not deploy Dino to other blockchains to test Dino's performance and compatibility, it is not guaranteed that Dino fits all permissionless blockchains, especially when miners gain less on-chain transaction fee but obtain much more off-chain profit by colluding with others.

VII. Related Work

A. Block Compression

The XThin [14] demands the recipient to provide a Bloom filter [25] that contains all transactions in its mempool to the sender. Once the sender receives the Bloom filter, it transmits an 8-byte short transaction hash list of all transactions and missing transactions to the recipient. The Compact [6] is similar to XThin. For the Compact in Bitcoin, the recipient does not need to send a Bloom filter, and the sender transmits a 6-byte short transaction hash list directly.

The Xthinner [8], an improved protocol based on XThin, contains two optimizations. One is compressing the short transaction from 8 bytes to 3 bytes. The other is using a state machine to encode a 3-byte transaction hash to save bandwidth further. The Graphene [7], which is better than Xthinner [26], combines Bloom filters and invertible bloom lookup tables (IBLT) [27]. In Graphene, the recipient needs to send transaction volume in its mempool to the sender when it asks for a block, and the sender responds with a Bloom filter and an IBLT to help the recipient reconstruct a new block. However, five messages are needed to transmit a block if the recipient misses some transactions in the new block, and its bandwidth consumption still grows with transaction volume in a block. Velocity [28] is a block propagation method that utilizes rateless erasure coding.

B. Network Topology

Kadcast [29] utilizes the structured overlay topology of Kademlia [30] and realizes an efficient broadcast operation with tunable overhead. However, it does not focus on reducing block size and can not be deployed incrementally. There are also some third-party relay networks, such as bloXroute [31], Marlin [32] and Fiber network [33]. Those proposals introduce the block delivery network (BDN) concept into the blockchain network from the traditional content delivery network (CDN). They are orthogonal to our work because we focus on reducing the bandwidth consumption of transmission blocks by reducing block size. In contrast, they focus on reducing block propagation latency by introducing relay nodes.

VIII. Conclusion

This paper presents a block dissemination protocol that transmits block construction rules instead of block content. We show that current block propagation protocols suffer from increasing bandwidth consumption brought by large transaction volume. Further, we also discuss the necessary conditions for a block transmission protocol that scales to a larger transaction volume. Using these conditions, we propose Dino and give a quantitative analysis of its performance. We deployed Dino into Bitcoin and Bitcoin Cash to compare it with the state-of-art protocols and tested its performance in simulation experiments. Our results illustrate that Dino has a substantial advantage of scaling to larger transaction volume and higher transaction generation rates. Finally, our work points out a new direction of block transmission that is promising to overcome the contradiction between blockchain networks' performance and security.

Acknowledgment

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the National Natural Science Foundation of China under Grant No.61872397. The contact author is Zhen Xiao.

REFERENCES

[1] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[2] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 906–917.

[3] M. Grundmann, T. Neudecker, and H. Hartenstein, "Exploiting transaction accumulation and double spends for topology inference in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 113–126.

[4] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security*. Springer, 2014, pp. 436–454.

[5] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the delivery of blocks and transactions in bitcoin," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 692–705.

[6] M. Corallo, "Bip152: Compact block relay," 2016. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[7] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: efficient interactive set reconciliation applied to blockchain propagation," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 303–317.

[8] J. Toomim, "Benefits of ltor in block entropy encoding," 2018.

[9] P. Tschipper, "Buip010: Xtreme thinblocks," in *Bitcoin Forum (1 January 2016). https://bitco. in/forum/threads/buip010-passed-xtreme-thinblocks*, vol. 774, 2016.

[10] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International conference on financial cryptography and data security*. Springer, 2016, pp. 106–125.

[11] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 1–12.

[12] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems (TOCS)*, vol. 17, no. 2, pp. 41–88, 1999.

[13] (2015) Bitcoin core commit 5400ef. [Online]. Available: https://bit.ly/2Q2Djux

[14] P. Tschipper. (2016) Buip010 xtreme thinblocks. [Online]. Available: https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/

[15] "C++ bitset." [Online]. Available: http://www.cplusplus.com/reference/bitset/bitset/

[16] J. Toomim. (2018) Block propagation data from bitcoin cash's stress test. [Online]. Available: https://medium.com/@j_73307/block-propagation-data-from-bitcoincashs-stress-test-5b1d7d39a234

[17] "Dsn bitcoin monitoring." [Online]. Available: https://www.dsn.kastel.kit.edu/bitcoin/?ref=hackernoon.com#propagation

[18] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 3–16.

[19] "Global bitcoin nodes distribution." [Online]. Available: https://bitnodes.io/

[20] F. Armknecht, J.-M. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter, "Outsourced proofs of retrievability," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 831–843.

[21] Verizon. (2021) Verizon latency. [Online]. Available: http://www.verizonenterprise.com/about/network/latency/.

[22] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "Txprobe: Discovering bitcoin's network topology using orphan transactions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 550–566.

[23] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 817–831.

[24] T. Neudecker, *Characterization of the bitcoin peer-to-peer network (2015-2018)*. KIT Karlsruher Institut für Technologie, Fakultät für Informatik, 2019.

[25] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[26] J. Toomim, "Block propagation data from bitcoin cash's stress test," 2018.

[27] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2011, pp. 792–799.

[28] N. Chawla, H. W. Behrens, D. Tapp, D. Boscovic, and K. S. Candan, "Velocity: Scalability improvements in block propagation through rateless erasure coding," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 447–454.

[29] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 199–213.

[30] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[31] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, "bloxroute: A scalable trustless blockchain distribution network whitepaper," *IEEE Internet Things J.*, 2018.

[32] M. Labs. (2019) Design and analysis of a decentralized relay network. [Online]. Available: https://www.marlin.pro/whitepaper

[33] J. Toomim, "Fast internet bitcoin relay engine (fibre). 2017. homepage," August1, 2017.