BSB: Towards Demand-Aware Peer Selection With XOR-based Routing

Qingyun Ji Darya Melnyk Arash Pourdamghani
Stefan Schmid
TU Berlin, Germany

Abstract

Peer-to-peer networks, as a key enabler of modern networked and distributed systems, rely on peer-selection algorithms to optimize their scalability and performance. Peer-selection methods have been studied extensively in various aspects, including routing mechanisms and communication overhead. However, many state-of-the-art algorithms are oblivious to application-specific data traffic. This mismatch between design and demand results in underutilized connections, which inevitably leads to longer paths and increased latency.

In this work, we propose a novel demand-aware peer-selection algorithm, called *Binary Search in Buckets* (BSB). Our demand-aware approach adheres to a local and greedy XOR-based routing mechanism, ensuring compatibility with existing protocols and mechanisms. We evaluate our solution against two prior algorithms by conducting simulations on real-world and synthetic communication network traces. The results of our evaluations show that BSB can offer up to a 43% improvement compared to two selected algorithms from the literature.

1 Introduction

Many modern networked and distributed systems rely on peer-selection algorithms, including (but not limited to) Amazon Dynamo [11], Apache Cassandra [19], and Ethereum [18]. Since peer-selection affects the latency and throughput of the system, designing efficient and simple peer-selection algorithms has been a major research topic over the past decades [24]. While many different algorithms have been proposed in the literature already, they are typically oblivious to the inherent patterns in demand. Recently, Avin et al. [4] have demonstrated that different types of applications have their own unique data traffic characteristics. These characteristics manifest, for example, in significant spatial (non-temporal) locality. Roughly speaking, high spatial locality means that communication happens frequently between specific pairs of nodes within the network. This work takes a step toward demand-aware peer-selection, by introducing a novel approach that utilizes information about the communication demand to optimize peer selection, while also utilizing XOR-based routing [23]. This change from demand-oblivious to demand-aware design promises a more efficient, scalable, and cost-effective network topology.

1.1 Our contribution

In this paper, we present two novel demand-aware peer-selection algorithms, under the class of *Binary Search in Buckets* (BSB). These algorithms use the fact that communication between peers reveals patterns that can be exploited by the peer-selection process. The algorithms are based on the structure of Kademlia [23]. From the perspective of a sending peer, the peers of the network are divided into buckets, where the number of peers in each bucket increases exponentially. The algorithms differ in how sending peers select which peers to communicate with inside the buckets.

- The HALF-SPLIT strategy considers the demand in each bucket and connects to a central peer based on the demand.
- The MAX-DEMAND strategy connects to the peer with the maximum demand in each bucket.

By design, our algorithms respect the local and greedy XOR-based routing and rely on a local view of the demand. Hence, they can replace most of the existing peer-selection algorithms without requiring a change in other parts of the system. We show the benefits of our algorithms by comparing them to previous algorithms on a range of synthetic and real-world datasets. We show that for the cases where the spatial complexity of demand is not high, our algorithms can perform better than the previous algorithms, which is observed in some real-world instances.

1.2 Related work

In this section, we first review the traditional algorithms for peer selection, and then discuss works related to demand-aware designs

Peer-selection algorithms. One of the key results in peer-to-peer (P2P) network design is Chord [29], a simple network design for efficient network location. The peer identifiers are arranged on a ring, each peer maintains up to $\log n$ connections, and key location is implemented via greedy routing. The Kademlia protocol [23] improves the key location by allowing the peers to select their connections in a randomized manner. Many more demand-oblivious P2P network designs have been proposed in the literature, including Viceroy [21], CAN [27], Pastry [28], and Tapestry [31]. For a detailed overview of P2P network architectures, please refer to [3] and [20].

Demand-aware network design. Demand-aware network design has been pioneered by the work of [6], which gives a glimpse of the potential and challenges of demand-aware network design. Following this work, demand-aware network design has been studied in the context of datacenter networks [16, 25, 13], distributed hash tables [26], and other settings [1, 8]. One direction closely related to our work is presented in [5, 12, 10]. There, the authors consider the problem of designing *bounded-degree* demand-aware networks, i.e., demand-aware networks where each peer can only be connected to a constant number of other peers. More recently, authors of [14] provided algorithms for demand-aware augmentation of an existing network with a matching.

Demand-aware peer selection. In one of the early works on demand-aware peer selection, the authors of [2] focus on accounting for the underlying Internet

Protocol	Demand-awareness	XOR-based routing
Chord [29]	No	Yes
Permutations [30]	Yes	No
BSB [this paper]	Yes	Yes

Table 1: A summary of selected peer-selection algorithms compared to our work.

topology, and thus leveraging the cooperation with internet service providers. Among the most recent related works, we note [30], which introduces a demand-aware peer-selection algorithm for training deep neural networks. In this work, we refer to the algorithm in [30] as the *Permutations algorithm*. Intuitively, this algorithm consists of $\log n$ overlapping rings of peers on top of each other, and depends on coin-change routing. This routing is not in line with the traditional XOR-based routing, hence can not be easily integrated with existing protocols. Furthermore, as this work only focuses on overlapping rings, it does not fully utilize many other possible peer-selection scenarios. Lastly, we want to point out that in the context of blockchain systems, works such as [7, 22] optimize the peer-selection procedure to minimize latency between the broadcast of a transaction and its confirmation. Compared to these works, our objective of adjusting to any network traffic demand is more general.

In the empirical evaluation, we compare our work to Chord and to the Permutations algorithms. A comparison of the theoretical guarantees of these algorithms is presented in Table 1.

2 Model

In this section, we discuss the underlying model used in the design of our algorithm and in the empirical analysis.

The peer-to-peer network considered in this paper is an overlay topology operating on top of an underlying network. We consider an overlay network topology N consisting of n peers/nodes[1] (e.g., computers) with identifiers $0, 1, \ldots, n-1$. Without loss of generality, we assume that n is a power of two. Note that the algorithms presented in this paper assign nodes to buckets of different sizes. If n was not a power of two, the algorithms would simply assign the additional nodes to the largest bucket. Further, we assume that the nodes of the network communicate with each other by sending messages over the neighboring links and forwarding other nodes' messages.

Overlay network. We consider an overlay network that consists of two parts. First, to ensure the connectivity of the underlying network, we consider the existence of an overlay network structure where the n nodes are already connected to each other such that they form a ring. This is a common assumption in the literature [29]. Then, to achieve an efficient peer-to-peer network, we allow nodes to locally augment the network by adding up to $\log n$ additional links to their peers. We assume that the links of these two parts have the same properties: all having equal capacity and delay between their two endpoints. We denote the distance between nodes i and j by $\text{dist}_{i,j}$. The exact value of the distance depends on the underlying routing strategy, detailed below.

[1] From now on, we use nodes and peers interchangeably.

Routing mechanisms. We define routing on the overlay network (including the ring and the augmented links) and discuss two possible routing mechanisms. The first considered routing strategy uses the *shortest paths*. This is a routing strategy that always uses the smallest number of edges for communication. When computing the shortest path distance, we assume that all nodes are aware of the whole network. We only consider this type of routing as a benchmark for other routing strategies, since assuming the knowledge of the whole network is unrealistic.

The second routing strategy is *local routing*. Here, we assume that each node has a local view of the network. The nodes forward messages based on the messages they receive and their own connections to neighbors. As many local routing strategies depend on the peer-selection algorithm, we will specify these strategies later in the paper. Observe that local routing, unlike shortest path routing, only require nodes to keep track of their neighbors.

Demand matrix. To be able to design demand-aware peer-selection algorithms, we assume that the communication demand of the network is fully described by a demand matrix $D \in \mathbb{R}^{n \times n}$. Each entry i and j of the demand matrix shows the amount (or the percentage of) traffic between nodes i and j. Observe that the elements on the diagonal of the demand matrix always have zero values, as the nodes do not communicate with themselves. Moreover, since we consider directed networks, the demand matrix is not required to be symmetric.

Cost function. The objective of our model is to minimize the overall communication cost. This cost depends on the communication matrix and on the routing strategy in the augmented network. Formally, the cost function C_N for a network N is defined as

$$C_N = \sum_{\forall i,j} \text{dist}_{ij} \times D_{ij}$$

where dist_{ij} is the distance between the nodes i and j under the applied routing strategy.

3 Peer-Selection Algorithms

We now describe our demand-aware peer-selection algorithms. The overarching part of our algorithms is inspired by the Kademlia protocol [23]. In the routing tables of Kademlia, nodes are organized in a binary tree-like structure. For our algorithms, we instead use the cyclic structure.

From the perspective of a single source node, the entire key-space is first divided into $\log n$ buckets, as shown in an example in Figure 1. Bucket 0 only contains the successor node, while the largest and farthest bucket contains half of the nodes inside the network. As the nodes in a bucket always have keys with a specific length of common prefix, a bucket can be simply represented by a key range with its start point and end point. Algorithm 1 presents this overarching part as pseudocode. The next step is to select a peer in each bucket. We have designed two demand-aware methods for peer selection in a bucket. A summary of these two node selection algorithms is described below and is presented as pseudocode in Algorithm 2.

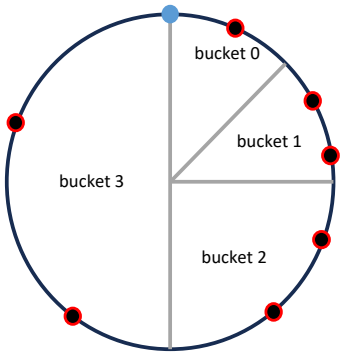


Figure 1: This figure shows the layout of the buckets on a cycle from the blue node's perspective. We assume that the blue node has the key 0000. The buckets from the farthest to the closest relative to the source node (the blue one) are correspondingly indexed from 0 up to $\log n - 1$. They are made up of all nodes with prefixes 0001, 001, 01, and 1 respectively.

Algorithm 1 BSB algorithm

$degree \leftarrow \log n$
for $i \leftarrow 1$ *to* n **do**
  for $j \leftarrow 1$ *to* $degree$ **do**
    $startpoint \leftarrow (i \oplus (1 \ll j)) \gg j \ll j$
    $endpoint \leftarrow startpoint + (1 \ll j) - 1$
    $demand \leftarrow D[i][startpoint : endpoint]$
    $peer \leftarrow LocateNode(n, i, demand)$
    $peer \leftarrow (indexOfMax(demand) + i)\%n$
    $G[i][peer] \leftarrow 1$

3.1 Half-split

Given the local communication demand from the source node s, from which the corresponding bucket-level demand can be extracted, node s chooses the peer in each bucket such that it splits the bucket-level demand approximately evenly. Note that bucket 0 only contains the successor node of the source node, so node s will directly add its successor in bucket 0 to the routing table. In other buckets, node s determines the node along the cycle that has approximately 50% of the total bucket-level demand on both sides, i.e., splits the bucket-level demand in half. We call this node the *mid-node*, and a link to this node will be added to the source's routing table.

3.2 Max-demand

Another demand-aware method for peer selection is to directly establish links to the node that has the highest inbound demand from the source node. Intuitively, this MAX-DEMAND method could help shorten the routing path length between node pairs that communicate frequently, since it could at least reduce the communication cost from the source node to peers within its routing table. Note, however, that finding the peer with the highest bucket-level demand is

Algorithm 2 LocateNode($n, i, demand$)

```
1: if HALF-SPLIT then
2:   total_demand ← 0
3:   for k ← 1 to n do
4:     total_demand += demand[k]
5:   half_demand ← total_demand / 2
6:   cumulative_demand ← 0
7:   for k ← 1 to n do
8:     cumulative_demand += demand[k]
9:     if cumulative_demand ≥ half_demand then
10:      node = (i + k) mod n
11: if MAX-DEMAND then
12:   max_demand ← −1
13:   for k ← 1 to n do
14:     if demand[k] > max_demand then
15:       max_demand = demand[k]
16:       node = (i + k) mod n
17: return node
```

more time-consuming than the previous method.

3.3 XOR-based greedy routing

Since the peers are chosen in a demand-aware manner, the global routing information is not available to the nodes. Therefore, a node can only determine the next hop based on the destination key. In other words, the message will be forwarded to the node whose key has the longest common prefix with the destination key in the current node's routing table. Unlike the shortest path routing in a peer-to-peer network, the greedy routing does not require individual nodes to know all links. At each step of forwarding a message, the node only considers the destination key to determine the next hop.

The pseudo-code for XOR-based greedy routing is presented in Algorithm 3. In this algorithm, each node at each step checks the peers in its routing table and chooses the peer with the longest common prefix with the destination key as the next hop, until the message arrives at the destination. Consequently, the length of the XOR-based routing path is upper bounded by $\log n$, as the key is represented by a binary value of $\log n$ bits.

4 Experimental Evaluation

The main goal of the evaluation section is to answer the following questions:

- **Question 1:** What is the running time of the presented peer-selection algorithms?
- **Question 2:** Under which parameters do the algorithms perform best?
- **Question 3:** How do our algorithms perform in a wide range of real-world datasets?

Algorithm 3 XOR-based geedy routing

Input n : Number of nodes in the network ($n = 2^m, m \in \mathbf{N}$)
Input G : Final topology with directed edges ($G \in \{0,1\}^{n \times n}$)
Output R : XOR-based routing path lengths in a matrix($R \in \mathbf{R}^{n \times n}$)

▷ *Compute the path length from each node-pair and add the corresponding routing cost to total cost*
for $i \leftarrow 1$ *to* n **do**
  for $j \leftarrow 1$ *to* n **do**
    $curr \leftarrow i$
    $next\_hop \leftarrow curr$
    $max\_common\_prefix\_len \leftarrow 0$
    $path\_length \leftarrow 0$
    ▷ *Stop when the destination node j is arrived*
    while $curr \neq j$ **do**
      for $k \leftarrow 1$ *to* n **do**
        if $G[curr][k] = 1$ **then**
          $common\_prefix\_len \leftarrow log_2 n - bit\_length(k \oplus j)$
          if $common\_prefix\_len > max\_common\_prefix\_len$ **then**
            $max\_common\_prefix\_len \leftarrow common\_prefix\_len$
            $next\_hop \leftarrow k$
      $curr \leftarrow next\_hop$
      $path\_length \mathrel{+}= 1$
    ▷ *Add the routing cost from node i to node j*
    $R[i][j] \leftarrow path\_length$

return R

4.1 Datasets

In our work, we consider two variants of datasets, a synthetic dataset that was created based on Zipf distribution, and also the realistic dataset from [4].

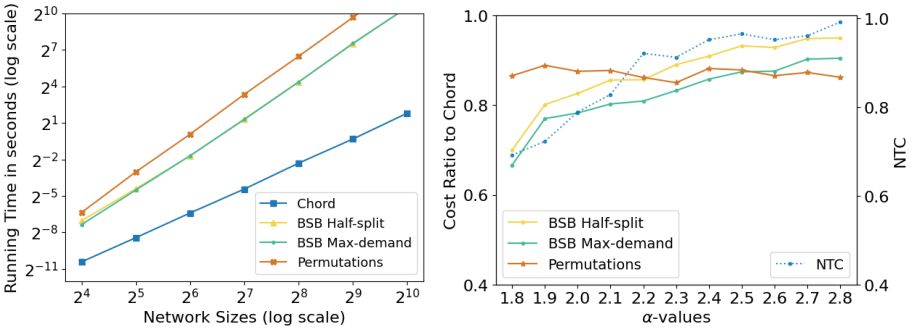
Zipf distribution. We generated 16 synthetic network communication data samples with different α-values, following the rules of the random Zipf distribution [9]. α is a hyperparameter in generating sequences based on Zipf distribution, which must be greater than 1. The probability density of every random variable becomes smaller as the α-value increases. That is, the skewness of the distribution becomes weaker with larger α-values.

Real-world dataset. We conducted our simulations using two of the three datasets available: cluster A and cluster C. They are collected from two different applications, namely Database and Hadoop applications. Every split data chunk includes network packets in a 20-minute time interval, where the time intervals of different data chunks do not overlap with each other.

Besides the Facebook main datasets, we evaluated our work on datasets of smaller sizes from various applications, like Microsoft, pFabric, and ProjecToR.

4.2 State-of-the-art algorithms

In the following, we provide a short overview of how we implemented two peer-selection algorithms from the literature, Chord [29] and the Permutations algo-



(a) Running time

(b) Cost compared to Chord

Figure 2: Figure 2a demonstrates the running time of the peer-selection algorithms that we focus on. Figure 2b shows the correlation between the communication costs calculated with different peer-selection algorithms and the non-temporal complexity (NTC) of the network traffic data.

rithm [30], and also how the coin-change routing algorithm works.

Chord. Chord employs a cyclic topology of nodes. In this case, nodes simply select peers using the bitwise XOR operator. That is, the k-th peer of node with ID i is $(i \oplus 2^k)$. Moreover, the routing between any pair of nodes on a cyclic network of size n is guaranteed within $\log n$ steps.

Permutations algorithm. This algorithm is similar to the standard Chord protocol in the sense that the peers are selected based on a group of predefined unidirectional distances to the source node. The difference is that while the distances in the Chord protocol are powers of two, those in the permutation algorithm are calculated based on the given demand matrix. The permutations are selected from the integers in $[1, n-1]$. Then the GCD-filter (greatest common divisor filter) is applied to reduce the number of candidate permutations. The final permutations that define the augmented communication links are chosen one by one from the candidate list. The network is then updated with additional edges after each selection. The GCD-filter ensures the connectivity of the network during the process of permutation selection, which is essential for the calculation of communication costs.

Coin change algorithm. The goal of the coin change algorithm is to find a way to deliver a target amount of money using the minimum number of coins. The available coins refer to the selected permutations, and the target amount of money refers to the unidirectional distance from the source node to the destination node. The coin change algorithm computes the minimum number of permutations needed to have all possible distances ranging from 1 to $n-1$. It correspondingly stores the combinations of permutations that make up the routing paths associated with valid distances. Dynamic programming is used to reduce the computational cost of repeatedly calculating the routing paths for each source-destination pair. Once the calculation is finished, routing becomes quite fast by looking up the associated linear combination of permutations based on distances in a list of $n-1$ elements.

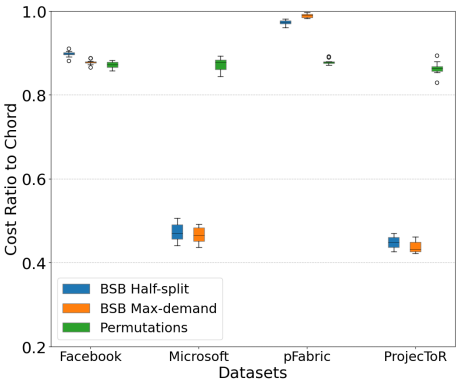


Figure 3: Communication cost ratio of BSB and Permutations algorithms compared to Chord, considering various small networks (with 64 nodes in each).

4.3 Results

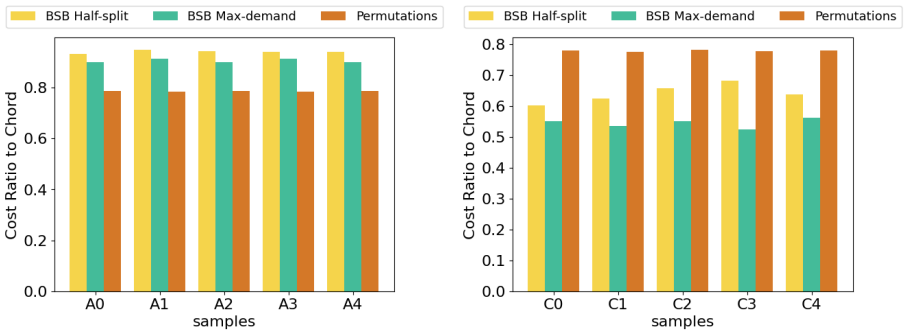
To simulate our system, we have used python 3.10 with networkx [15] and Matplotlib [17] libraries. Our programs were running on 16 AMD Opteron™ processors with a clock frequency of 2.0GHz, and 15 GB of RAM.[2]

Answer 1. Running time. As shown in Figure 2a, our BSB algorithms have an advantage over the Permutations algorithm due to faster computational speed, which makes them stand out in some specific domains.

Answer 2. Non-temporal complexity. In order to compute the non-temporal complexity of the datasets, we prepared three data files per network and compressed the files, respectively, following the methodology of [4]. Firstly, the original file stores the source-destination pairs associated with the packets transmitted across the network in their original sequence. On the basis of the original file, another file, the shuffled file, is generated by randomly shuffling the rows in the original file. At this point, the temporal structure of the network traffic is eliminated. The third file consists of the same number of rows as the original and shuffled file, while the source-destination pairs are randomly generated from the node set using the uniform distribution. During this step, the non-temporal structure is then removed. We name it the random file. Therefore, the non-temporal complexity is measured as the ratio of the size of the compressed shuffled file to that of the compressed random file. Figure 2b depicts a strong correlation between the cost ratios of BSB algorithms and the non-temporal complexity of communication demand, which indicates that our BSB algorithms are targeting cases of low non-temporal complexity.

Answer 3.1. Small real-world dataset. Figure 3 exhibits the simulation results on smaller datasets. We can see the four groups on the x-axis, namely Facebook, Microsoft, pFabric, and ProjecToR. The original datasets are all based on a 100-node peer-to-peer network. To simplify the process of XOR-based routing, we reduced the number of nodes to 64 in each group. We performed the filtering as follows: each of the original 100 nodes is randomly assigned a node identifier that ranges from 0 to 99. Then the nodes with identifiers from 0 to 63 were selected. After that, we walked through the whole dataset to remove

[2]The source code of this paper can be found in `https://github.com/inet-tub/BSB`



(a) Facebook cluster A

(b) Facebook cluster C

Figure 4: This figure shows the communication cost ratio of BSB and Permutations algorithms compared to the Chord algorithm, which are simulated on 4096-node networks generated from Facebook data samples. Figure 4a shows results for the database cluster (A). Figure 4b shows results for Hadoop cluster (C).

the rows (i.e., network packets) where either the source node or the destination node is out of the filtered scope. For each application, we performed 10 random ID assignments to avoid the situation where a single result is not sufficiently representative for our study.

Similarly, the performance of the Permutations algorithm does not vary much on different datasets, with a cost ratio of 0.86 on average. Nevertheless, the simulations of our BSB algorithms show completely different results in different types of applications. On Microsoft and ProjecToR networks, our BSB algorithms clearly outperform the Permutations algorithm with an average cost reduction of 55% (compared to Chord). On the other hand, they have only a slight advantage over Chord on Facebook and pFabric networks, underperforming compared to the Permutations algorithm. This phenomenon is again related to our hypothesis that the non-temporal complexity of the network flow could affect the efficiency of our BSB peer selection. In other words, we infer that the frequency distribution of network communications in Microsoft and ProjecToR datasets seems to be more skewed than that in Facebook and pFabric datasets.

Answer 3.2. Real-world dataset. Each full dataset is partitioned into data chunks based on the timestamps of the network packets, where each data chunk includes the network traffic data within a non-overlapping 20-minute time interval. Although the network size is reduced to 4096 in each data chunk using a key-filter, the cost calculation is still a time-consuming task, especially for the Permutation peer-selection algorithm. Therefore, we conducted simulations only on five data chunks per Facebook cluster due to the time limit.

Although our BSB algorithms and the Permutations algorithm are both based on the real-world network traffic, their peer-selection logics are quite different. As we discussed in the previous section, the peer-selection method is highly dependent on the routing mechanism. What we can now observe from the two plots is that the Permutations algorithm shows a stable performance on both datasets. Its cost ratio stays around 0.78, which indicates that it can reduce the communication cost by 22% compared to the Chord algorithm.

However, our BSB algorithms do not present results similar to the Permutations algorithm. They perform much worse than the Permutations algorithm on the Database cluster, whereas they perform much better on the Hadoop cluster. The BSB Half-split algorithm exhibits a cost ratio of approximately 93% on the former dataset and 60% on the latter, where the BSB Max-demand algorithm shows lower cost ratios of around 89% and 55% respectively. A reason for the unstable performance of our BSB algorithms could be the different frequency distributions of communication demand in various applications.

4.4 Discussion

BSB algorithms versus the Permutations algorithm. Intuitively, if there is a high communication demand between a source-destination pair, the easiest way to ensure low cost is to connect the source node to the destination nodes directly. Since the BSB algorithms use the "local view", simply connecting such source-destination pairs would be possible. Especially in the Max-demand version, the direct channels from the source node to its most frequent contacts in each bucket could be very helpful for reducing the total overhead. However, selecting peers using the "global view" seems to be more complicated, as nodes have to ensure that behaviors benefit themselves are also compatible with the collective interests. Nodes in the Permutations algorithm share the same group of permutations, which makes it restrictive to establish a connection for a single source-destination pair, unless those pairs have significantly higher communication demand among other pairs of nodes.
Comparing two BSB algorithms. The reason why Max-demand performs better than Half-split in most of the cases seems to be straightforward: Max-demand tends to build a direct link from a source node to the node that it has highest demand to. Recall that the total communication cost can be viewed as the sum of weighted path lengths. Hence, by shortening the path lengths between these pairs of nodes that communicate frequently, the communication cost is expected to be heavily reduced. Although Max-demand is more beneficial for improving the efficiency of P2P network communication, it has longer runtime than Half-split does on selecting peers at the bucket level.

5 Conclusion

In this work, we took initial steps toward a demand-aware peer-selection algorithm, that supports XOR-based routing, and aims to minimize the shortest path, weighted by the communication demand between peers. In particular, we introduced a class of algorithms under the general name of BSB, and gave a comparative performance analysis of this class considering a wide range of real-world and synthetic inputs. Our results indicated that when the communication demand is skewed, BSB algorithms achieved significantly reduced communication cost compared to state-of-the-art algorithms, by up to 43%. In the future, we aim to study extended variants of BSB algorithms, in particular, algorithms that incorporate randomization. We also aim to extend our evaluation to other applications where peer selection is relevant, for example, in blockchain networks.

Funding. This project has received funding from the European Research Council (ERC) under grant agreement No. 864228 (AdjustNet), 2020-2025.

References

[1] Addanki, V., Pacut, M., Pourdamghani, A., Rétvári, G., Schmid, S., Vanerio, J.: Self-adjusting partially ordered lists. In: INFOCOM. pp. 1–10. IEEE (2023)

[2] Aggarwal, V., Feldmann, A., Scheideler, C.: Can isps and p2p users cooperate for improved performance? ACM SIGCOMM Computer Communication Review (2007)

[3] Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. (2004)

[4] Avin, C., Ghobadi, M., Griner, C., Schmid, S.: On the complexity of traffic traces and implications. Proc. ACM Meas. Anal. Comput. Syst. (2020)

[5] Avin, C., Mondal, K., Schmid, S.: Demand-aware network designs of bounded degree. Distributed Comput. **33**(3-4), 311–325 (2020)

[6] Avin, C., Schmid, S.: Toward demand-aware networking: a theory for self-adjusting networks. Comput. Commun. Rev. **48**(5), 31–40 (2018)

[7] Babel, K., Baker, L.: Strategic peer selection using transaction value and latency. In: DeFi@CCS. pp. 9–14. ACM (2022)

[8] Bentert, M., Franke, M., Melnyk, D., Pourdamghani, A., Schmid, S.: Demand-aware multi-source ip-multicast: Minimal congestion via link weight optimization. In: International Federation for Information Processing Networking Conference (IFIP Networking) (2025)

[9] Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-like distributions: Evidence and implications. In: IEEE INFOCOM (1999)

[10] Dallot, J., Caldeira, C., Pourdamghani, A., Goussevskaia, O., Schmid, S.: Laslin: A learning-augmented peer-to-peer network. arXiv preprint arXiv:2509.11904 (2025)

[11] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: ACM SOSP (2007)

[12] Figiel, A., Korhonen, J.H., Olver, N., Schmid, S.: Efficient algorithms for demand-aware networks and a connection to virtual network embedding. In: OPODIS. LIPIcs, vol. 324, pp. 38:1–38:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)

[13] Figiel, A., Melnyk, D., Milentijevic, T., Schmid, S.: Distributed construction of demand-aware datacenter networks. In: IPDPS. pp. 162–172. IEEE (2025)

[14] Figiel, A., Melnyk, D., Nichterlein, A., Pourdamghani, A., Schmid, S.: Spiderdan: Matching augmentation in demand-aware networks. In: SIAM Symposium on Algorithm Engineering and Experiments (ALENEX) (2025)

[15] Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkx. In: Proceedings of the 7th Python in Science Conference (2008)

[16] Hanauer, K., Henzinger, M., Schmid, S., Trummer, J.: Fast and heavy disjoint weighted matchings for demand-aware datacenter topologies. In: INFOCOM. pp. 1649–1658. IEEE (2022)

[17] Hunter, J.D.: Matplotlib: A 2d graphics environment. Comput. Sci. Eng. (2007)

[18] Kiffer, L., Salman, A., Levin, D., Mislove, A., Nita-Rotaru, C.: Under the hood of the ethereum gossip protocol. In: Borisov, N., Diaz, C. (eds.) FC (2021)

[19] Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Oper. Syst. Rev. (2010)

[20] Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Commun. Surv. Tutorials (2005)

[21] Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: a scalable and dynamic emulation of the butterfly. In: ACM PODC (2002)

[22] Mao, Y., Deb, S., Venkatakrishnan, S.B., Kannan, S., Srinivasan, K.: Perigee: Efficient peer-to-peer network design for blockchains. In: PODC. pp. 428–437. ACM (2020)

[23] Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A.I.T. (eds.) IPTPS (2002)

[24] Naor, M., Wieder, U.: Novel architectures for P2P applications: the continuous-discrete approach. In: SPAA. ACM (2003)

[25] Pourdamghani, A., Avin, C., Sama, R., Schmid, S.: Seedtree: A dynamically optimal and local self-adjusting tree. In: INFOCOM. pp. 1–10. IEEE (2023)

[26] Pourdamghani, A., Avin, C., Sama, R., Shiran, M., Schmid, S.: Hash & adjust: Competitive demand-aware consistent hashing. In: OPODIS. LIPIcs, vol. 324, pp. 24:1–24:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)

[27] Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S.: A scalable content-addressable network. In: Cruz, R.L., Varghese, G. (eds.) ACM SIGCOMM. ACM (2001)

[28] Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) IFIP/ACM Middleware (2001)

[29] Stoica, I., Morris, R.T., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. (2003)

[30] Wang, W., Khazraee, M., Zhong, Z., Ghobadi, M., Jia, Z., Mudigere, D., Zhang, Y., Kewitsch, A.: Topoopt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In: USENIX NSDI (2023)

[31] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.: Tapestry: a resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. (2004)