Kademlia:

A Peer-to-Peer Information System Based on the XOR Metric

Petar Maymounkov and David Marières, 2002
Lecture Notes in Computer Science
DOI: https://doi.org/10.1007/3-540-45748-8_5

2022/10/31　Bcali 輪読資料　B2 kekeho

1. 概要

- P2P 分散ハッシュテーブル(Distributed Hash Table: DHT)である**Kademlia**についての論文
- ノード間でお互いの状態を知るために必要な設定メッセージを、キー検索をする際に一緒に送ってしまうことで、メッセージ数を削減
- DoS攻撃に耐性を持つ
- キー: 160bit。ノードも160bitのキー空間にノードIDを持つ
- データ<key, value>は、keyに”近い"IDを持つノードに保存される
  - 近い: 距離$d(x, y) = x \oplus y$
- 証明可能な一貫性とパフォーマンス、レイテンシを最小限にするルーティング、対象的な単一方向性トポロジを組み合わせた最初のP2Pシステム

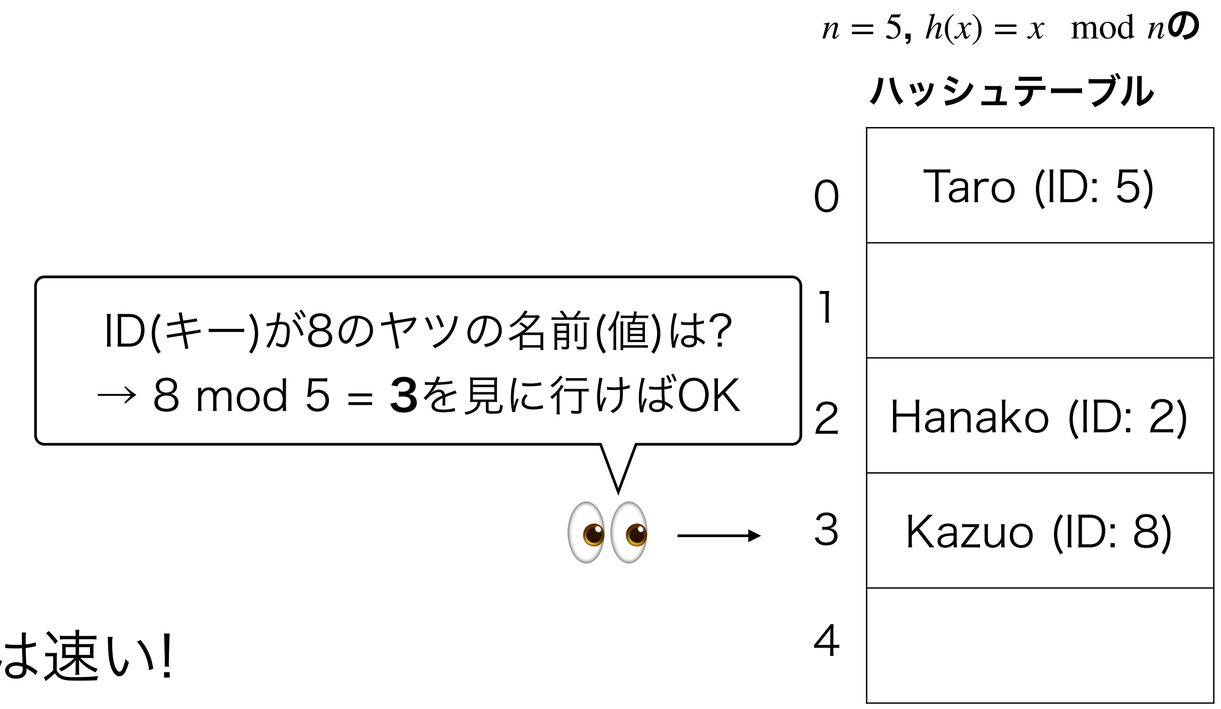
2. 前提知識: DHTとは

Hash Tableの概要

- **Hash Table**: キーと値を効率的に対応付けるデータ構造
- $\text{key} = h(\text{value})$

  hはハッシュ関数
- 例: ハッシュ関数$h(x) = x \mod n$

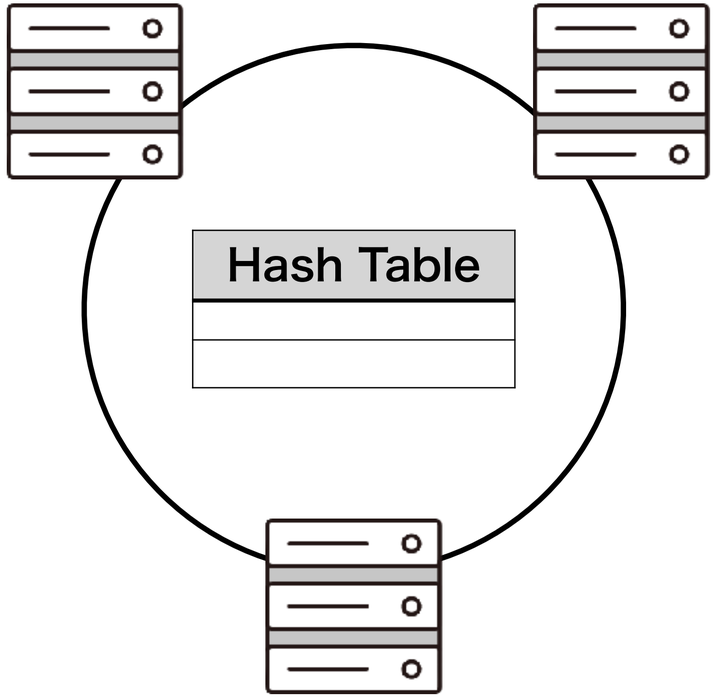
  (nは配列サイズ)
- 挿入・削除・取得(探索): $O(1)$
  - 配列で検索したら$O(n)$…。Hash Tableは速い!



2. 前提知識: DHTとは

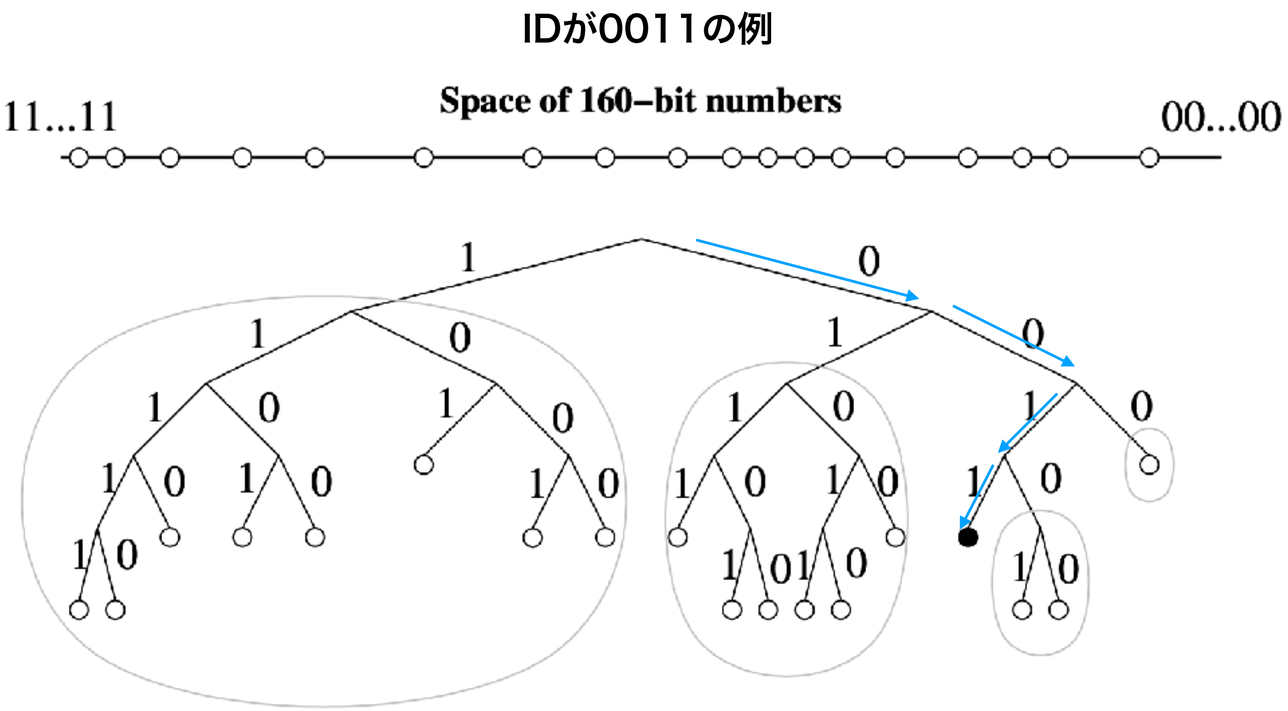
Distributed Hash Tableの概要

- **Distributed Hash Table (DHT)**: Hash Tableを複数のピアで管理する技術 [1]
- ユースケース
  - BitTorrent: P2Pファイル共有ソフト。ピアを見つける際にDHTを使用
  - Ethereum: ブロックチェーン。ノードを見つける際にDHTを使用



3. システムの概要

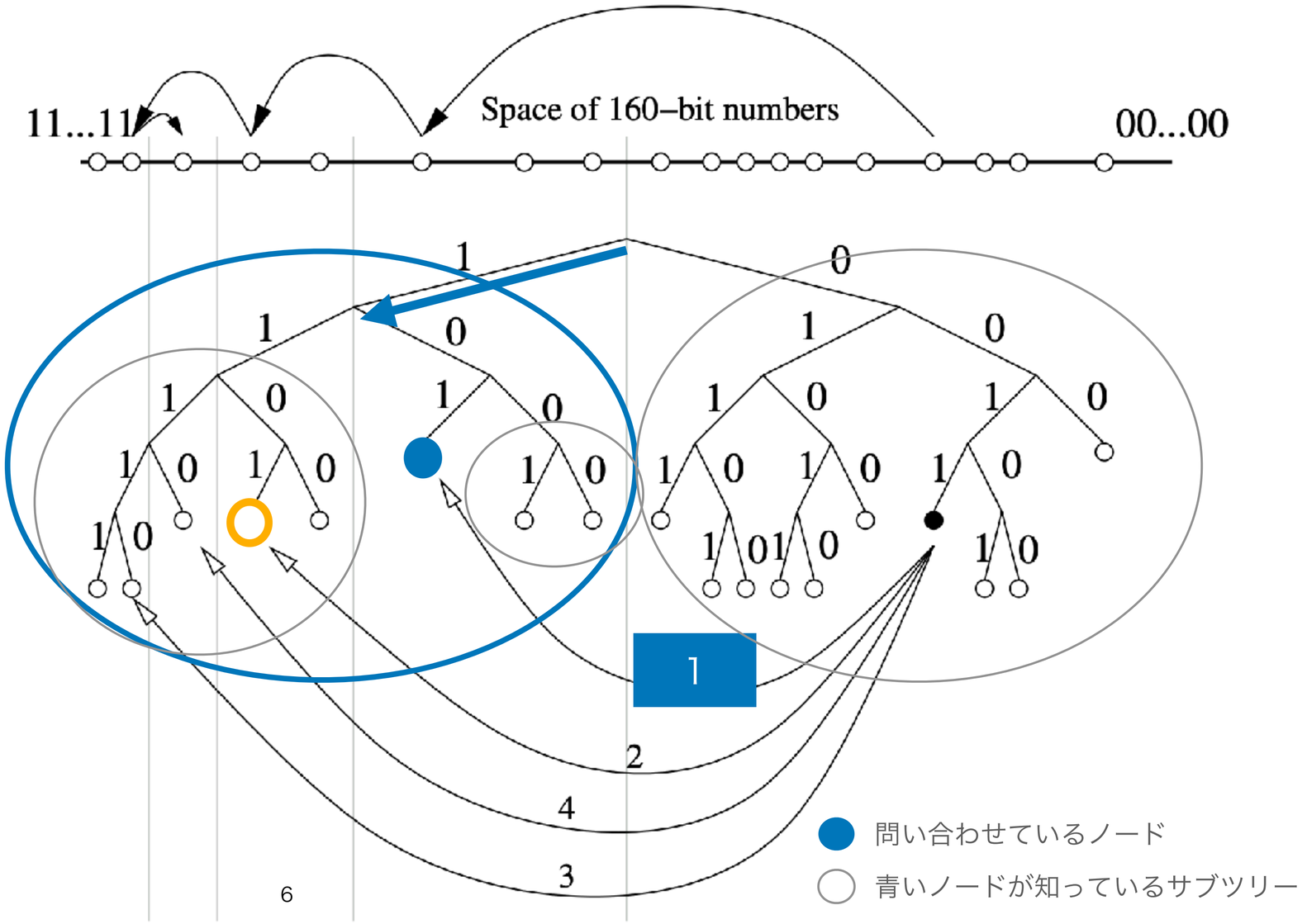
- ノードを二分木の葉として扱う
- あるノードに対し、そのノードを含まないサブツリーに分割していく
- 各ノードは、それぞれのサブツリー内の一つ以上のノードを知っている必要がある
- 値は、キーに近いIDを持つノードに保持される



3. システムの概要

ノードの検索: 1110を探せ

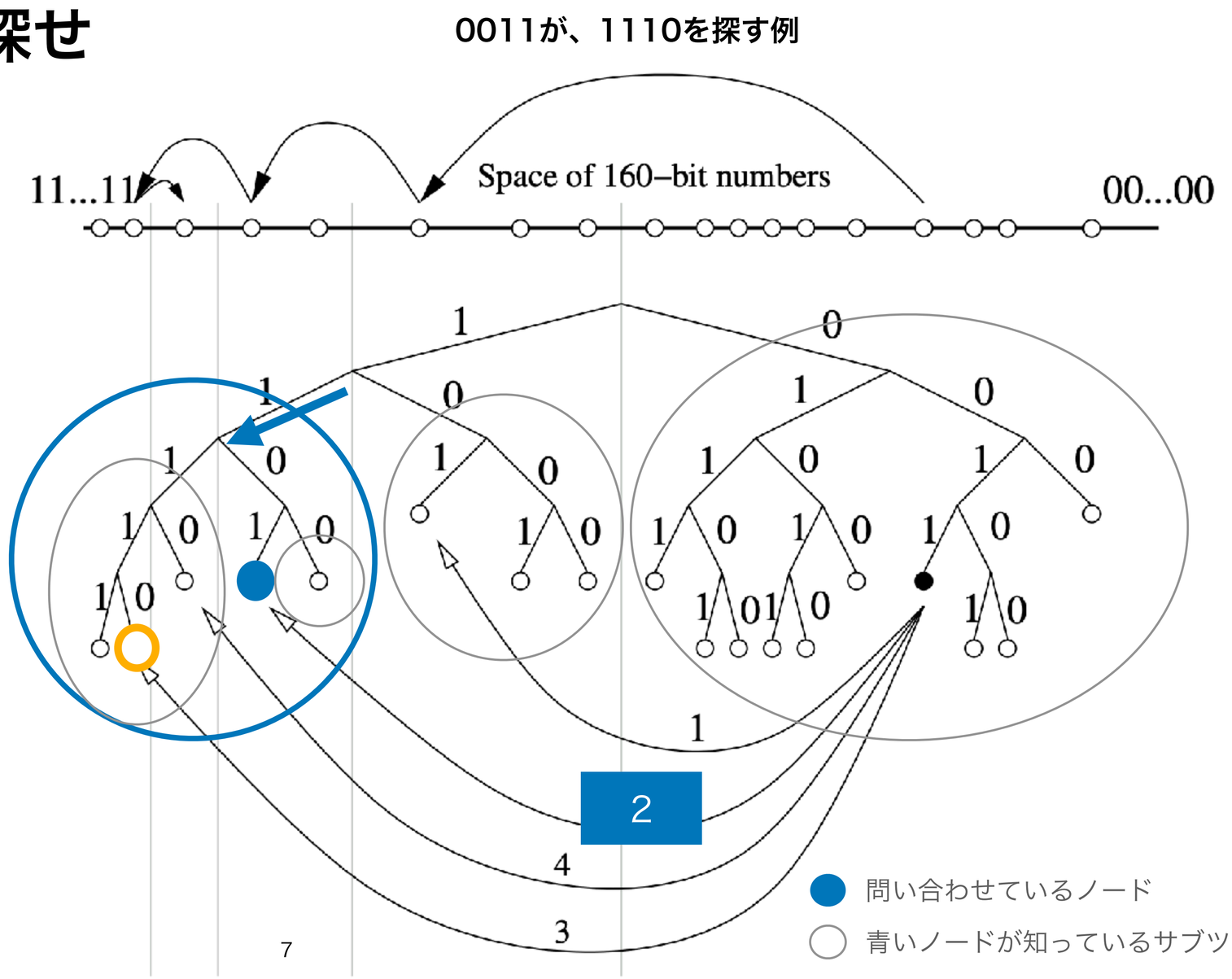
- 1*の中で知っている101に問い合わせ
- 101は、11*の中で知っている1101を教える



3. システムの概要

ノードの検索: 1110を探せ

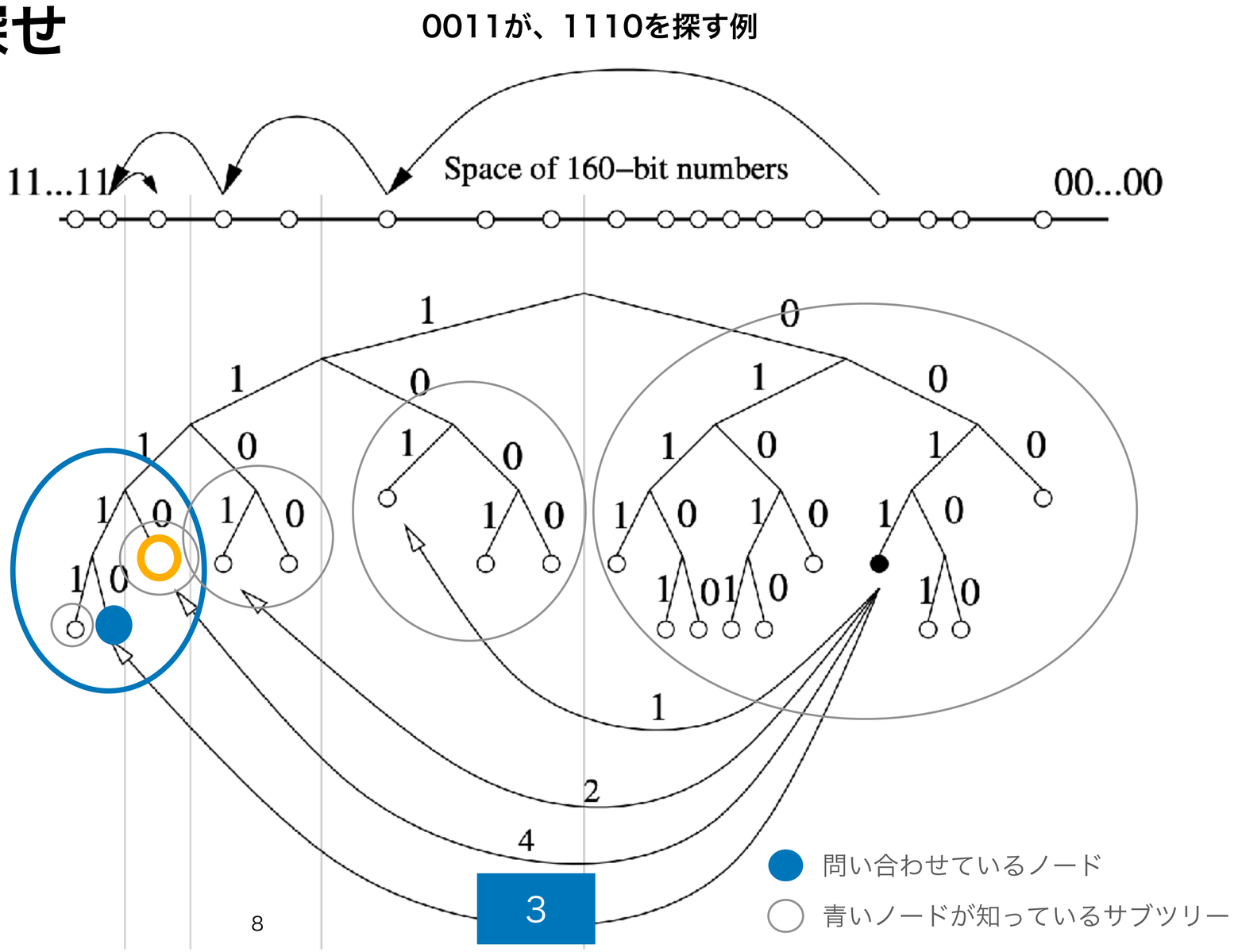
- 1101に問い合わせ
- 1001は、111*の中で知っている11110を教える



3. システムの概要

ノードの検索: 1110を探せ

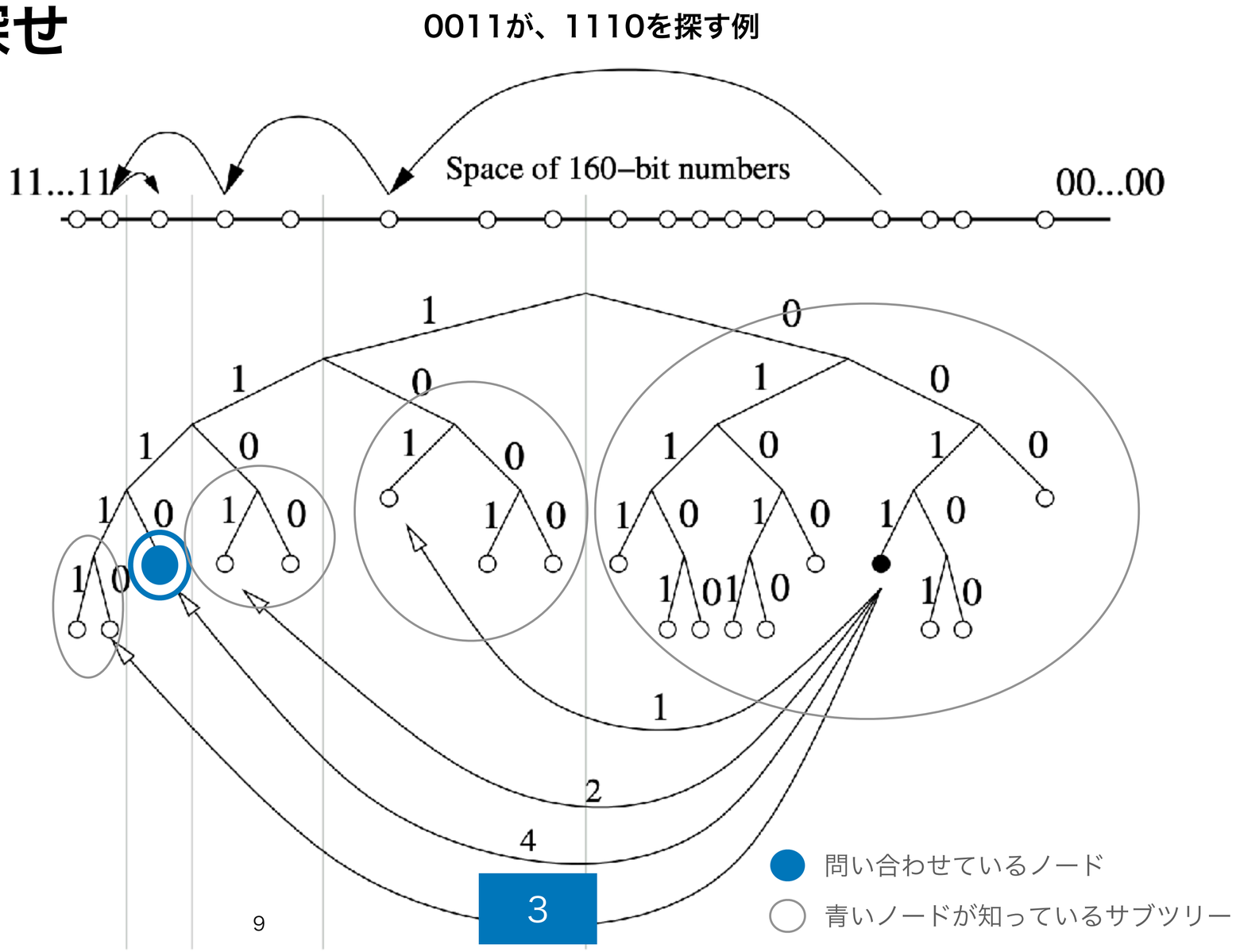
- 11110に問い合わせ
- 11110は、1110*の中で知っている1110を教える



3. システムの概要

ノードの検索: 1110を探せ

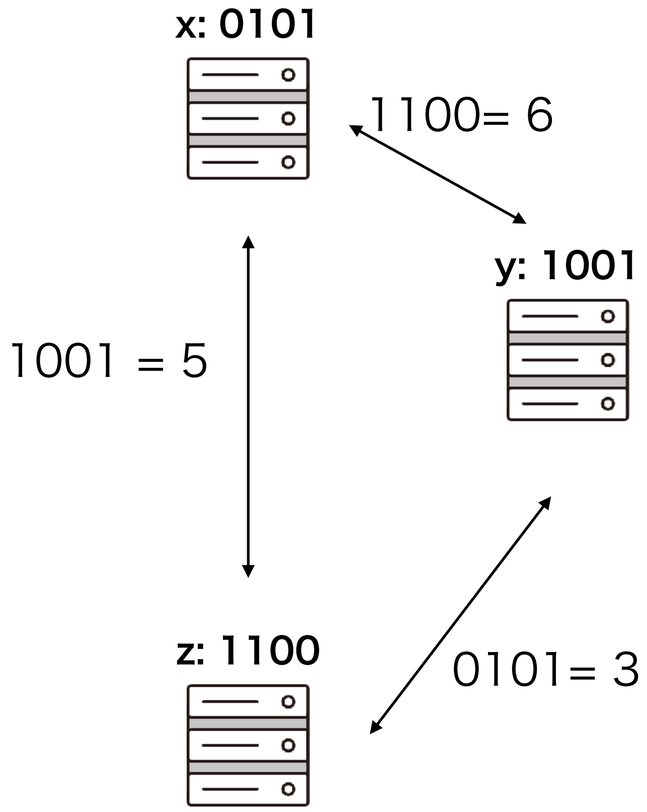
- 1110に問い合わせ
- ビンゴ!



4. システムの詳細

XOR距離関数

- 各ノードとキーは160bitの識別子を持つ
- ノードが送信する全てのメッセージにはノードIDが含まれる
- 2つの識別子間の距離$d(x, y) = x \oplus y$
  - XOR($\oplus$)を用いて距離の概念を示せる
    - $d(x, y) = d(y, x)$
    - $x \neq y$かつ$\forall x, y : d(x, y) = d(y, x)$であれば、$d(x, x) = 0,\ d(x, y) > 0$が成り立つ
    - XORは三角特性をもつ: $d(x, y) + d(y, z) \geq d(x, z)$
      - $d(x, y) \oplus d(y, z) = d(x, z)$および$\forall a \geq 0, b \geq 0 : a + b > a \oplus b$なので



4. システムの詳細

XOR距離関数

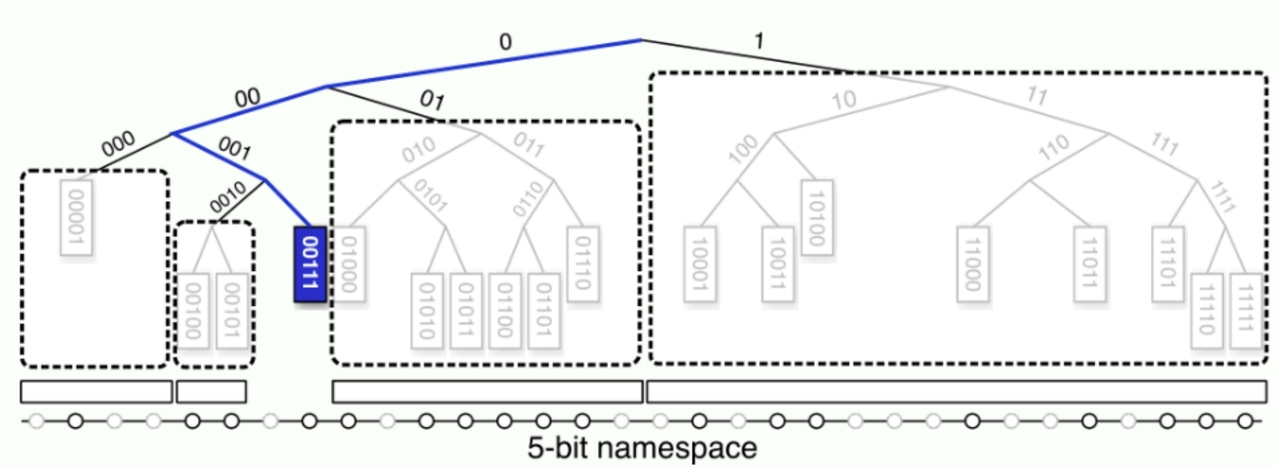
- **二分木ベースのシステムにおいても距離の概念を捉えている**
- 単方向性: 任意の点xと、距離$\Delta > 0$に対し、$d(x, y) = \Delta$となるような点yが一つだけ存在する
  - 同じキーのすべての検索が同じ経路に沿って収束する → キャッシュが有効

0

4. システムの詳細

ノードの状態

- 各ノードは、クエリメッセージをルーティングするためにお互いのコンタクト情報を保存する
- k-bucket: $0 \geq i > 160$ごとに、自分の位置に対して2^iから2^{i+1}-1の距離にあるノードのIPアドレス・UDPポート・ノードIDを保持する
  → サブツリーごと



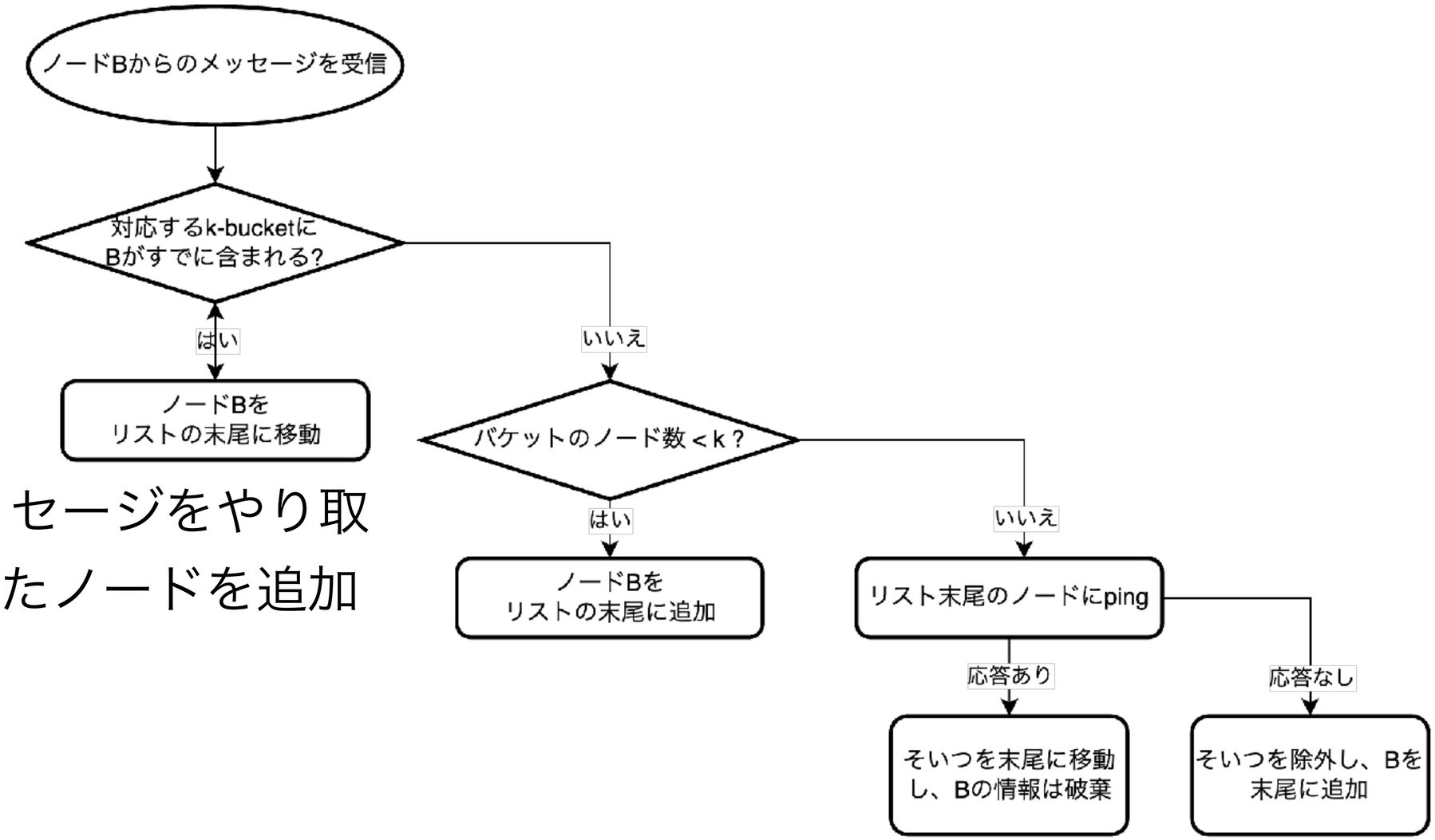
Routing Table of 00111

branch	k-bucket (k=5)				
1*	10001	10011	11011	11110	11111
01*	01010	01011	01100	01101	01110
000*	00001				
0010*	00100	00101			

https://hazm.at/mox/distributed-system/algorithm/distributed-hash-table/kademlia/index.html

4. システムの詳細

ノードの状態



- k-bucketに、メッセージをやり取りする中で発見したノードを追加していく
- 論文中では$k = 20$

4. システムの詳細

ノードの状態

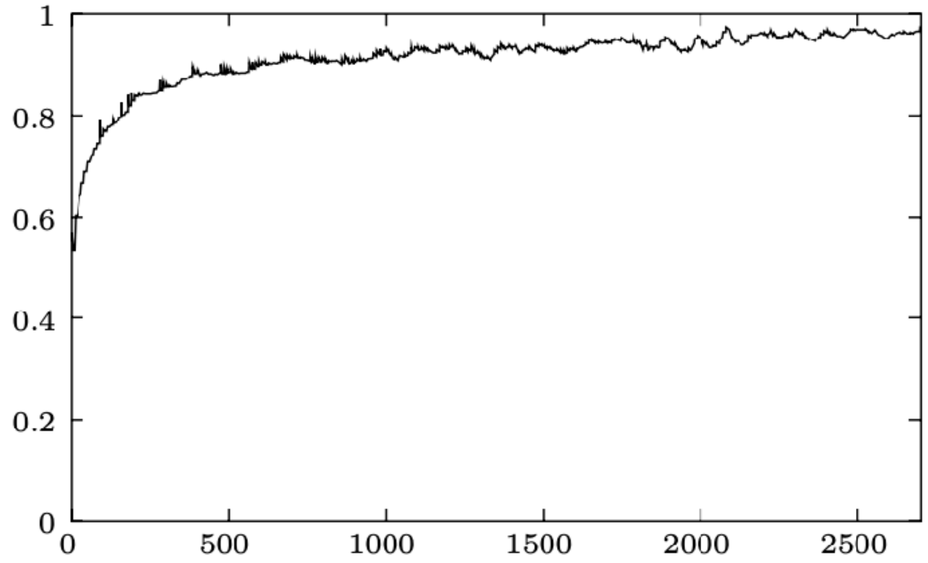


Fig. 3: Probability of remaining online another hour as a function of uptime. The x axis represents minutes. The y axis shows the the fraction of nodes that stayed online at least x minutes that also stayed online at least $x + 60$ minutes.

- **長く稼働しているノードをなるべくk-bucketに残す戦略**

- GnuTerraのトレースデータによると、ノードの稼働時間が長い位ほど、さらに1時間稼働する可能性は高くなる

- 古いノードが離脱するときのみk-bucketが更新される→新しいノードを大量に投入するDoS攻撃に対する耐性をもたせることにも繋がっている

4. システムの詳細

Kademliaプロトコル

- Kademliaのプロトコル: PING・STORE・FIND_NODE・FIND_VALUE
  - **PING**(contact): ノードに対して疎通確認を行う
  - **STORE**(key, value): キーと値のペアを保存するように指示する
  - **FIND_NODE**(key): 160bitのIDを引数にとる。受信者は、目的のIDに最も近いk個のノードのIPアドレス・UDPポート・ノードIDを返す
  - **FIND_VALUE**(key): 受信者は、指定されたキーに対応する値を持っていればそれを返し、そうでなければFIND_NODEと同様の動作を行う

4. システムの詳細

Kademliaプロトコル

- **値の保存**
  - FIND_NODEでキーに最も近いk個のノードを取得し、それらにSTOREを送る
  - 1時間毎に再発行
- **値の取得**
  - FIND_VALUEでキーに最も近いIDを持つk個のノードを見つける。その際、任意のノードが値を返すと終了
  - キャッシュ: 観測されたキーに最も近いが、値を返さなかったノードにSTOREを送る
  - 特定のキーの人気が高い場合、オーバーキャッシングが起こる可能性がある
    - ノードのDB内における有効期限を、自身のノードとキーに最も近いノードの間のノード数に応じて、指数関数的に反比例させる

4. システムの詳細

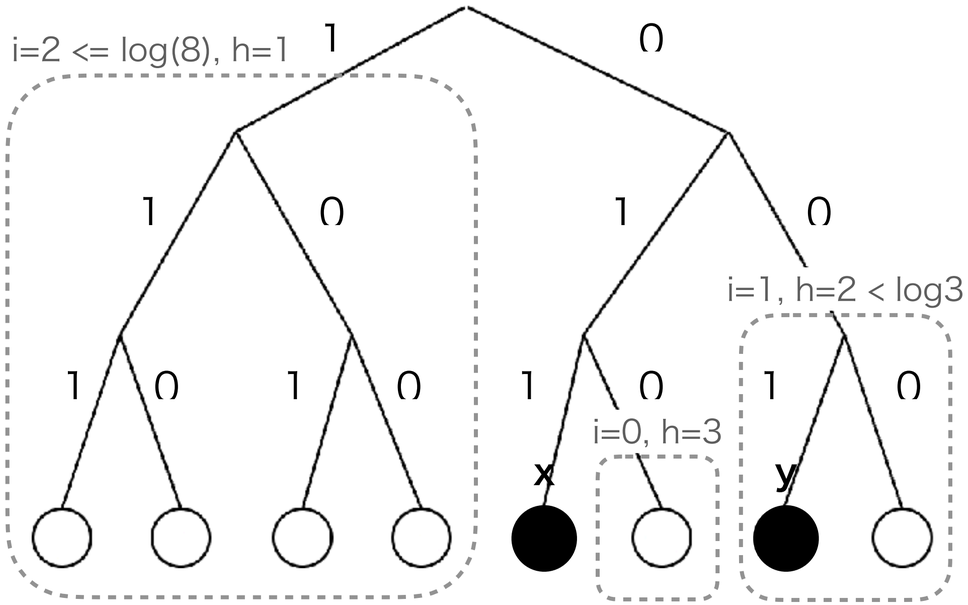
Kademliaプロトコル

- **ノードの新規参加**
  - ノードID: ランダムな160bitの値
  - ノードは、自身のノードIDに近いk個のキーと値のペアを保存したい
  - 接続先のノードに対し、自身のノードIDをkeyとしてFIND_NODEすることで近隣のノードを把握
  - 問い合わせを受けたノードは、自分より問い合わせ元のほうが持つべき値を、問い合わせ元に対してSTORE

5. 証明の概略

殆どの操作が対数時間でできる

- いくつかの定義
  - 距離範囲$[2^i, 2^{i+1})$をカバーするk-bucketについて、「bucketのインデックス」をiとする
  - 「ノードの深さ」hを$160 - i$とする
  - iは空ではないバケットの最小のインデックスとする
  - ノードxにおけるノードyの「バケットの高さ」を、xがyを挿入するであろうバケットのインデックスからxの空のバケットの最も小さいインデックスを引いたものとする

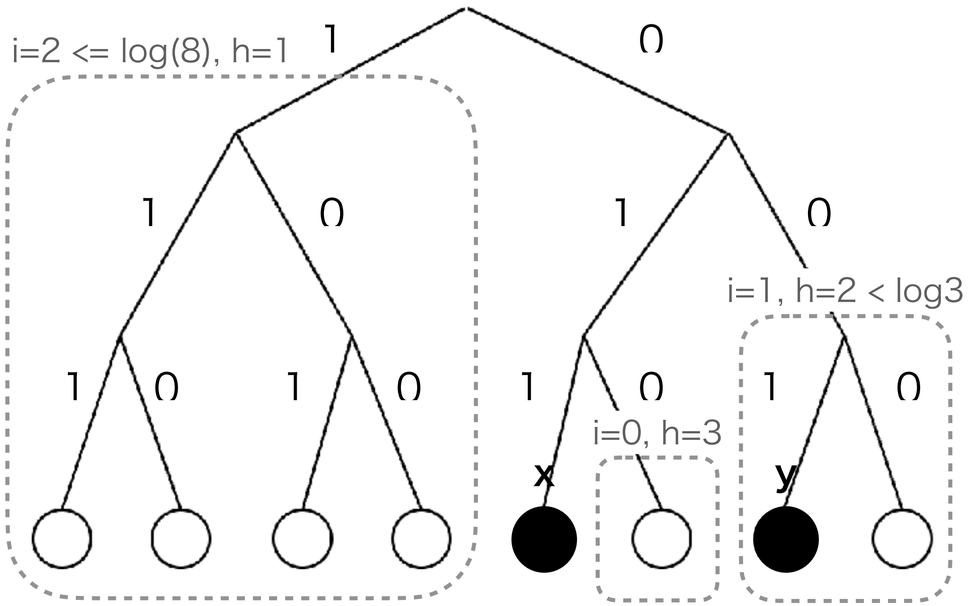


5. 証明の概略

殆どの操作が対数時間でできる

- ノードIDはランダムに選ばれるので、不均一な分布にならないとすると、ある任意のノードの高さ(バケットの高さ?)は、n個のノードを持つシステムでは$\log n$に収まる

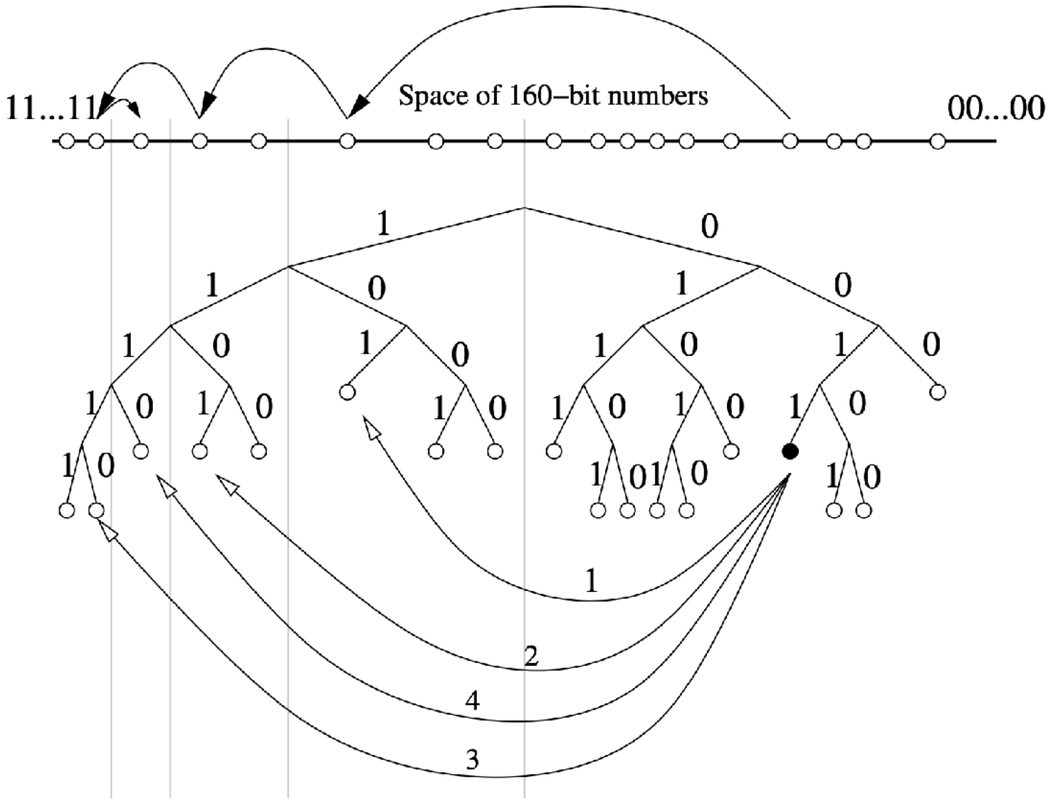
- k番目に近いノードIDに最も近いノードのバケットの高さは、ほぼ$\log k$に収まるだろう



5. 証明の概略

殆どの操作が対数時間でできる

- ノードが適切な範囲に存在する場合、全てのノードの全てのk-bucketに少なくとも1つのコンタクトが含まれると仮定する
- ターゲットIDに最も近いノードの深さをhとする
- このノードの最上位h個のk-bucketがいずれも空でない場合、検索手続きはステップごとに半分に絞っていける → $h - \log k$ステップでそのノードを探し出すことになる



5. 証明の概略

システムが確かな確率でキーに対応する値を返す

- <key, value>ペアが公開されると、k個のノードに送られる
- 1時間毎に再発行される
- (GnuTerraの観測データによると)新規ノードであっても1時間後に1/2の確率で持続する

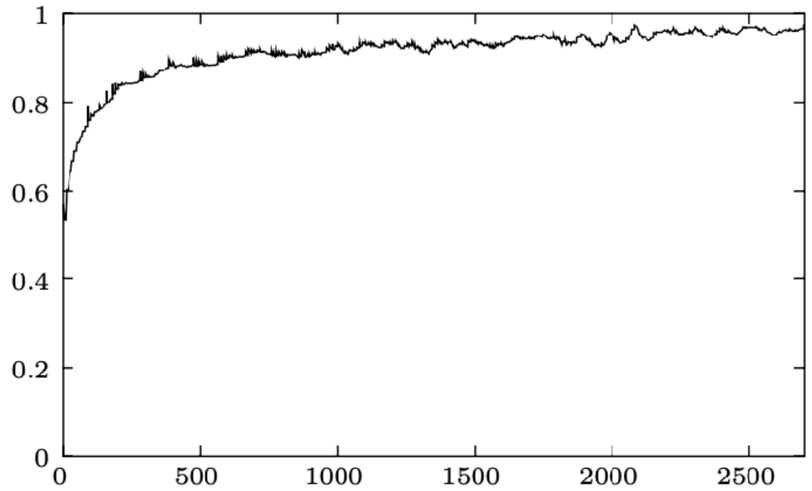


Fig. 3: Probability of remaining online another hour as a function of uptime. The x axis represents minutes. The y axis shows the the fraction of nodes that stayed online at least x minutes that also stayed online at least $x + 60$ minutes.

- 1時間後には、<key, value>ペアはキーに最も近いk個のノードの一つに$1 - 2^{-k}$で存在している

参考文献

1. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001. "A scalable content-addressable network". SIGCOMM Comput. Commun. Rev. 31, 4 (October 2001), 161–172. https://doi.org/10.1145/964723.383072