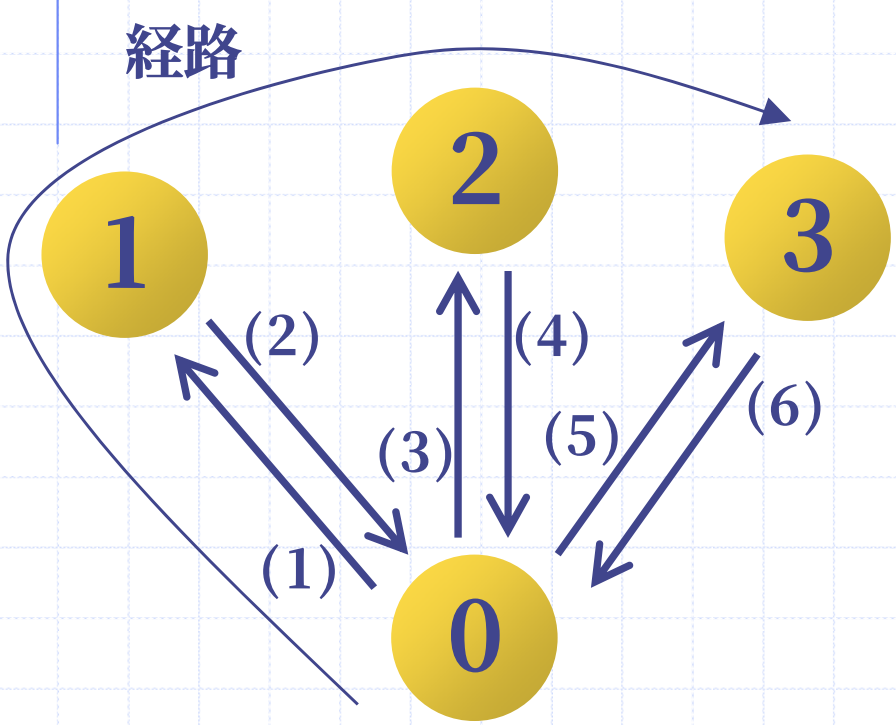
構造化オーバレイにおける反復探索と再帰探索の比較

首藤 一幸 (ウタゴエ)
加藤 大志 (NEC)
門林 雄基 (奈良先端)
土井 裕介 (東芝)

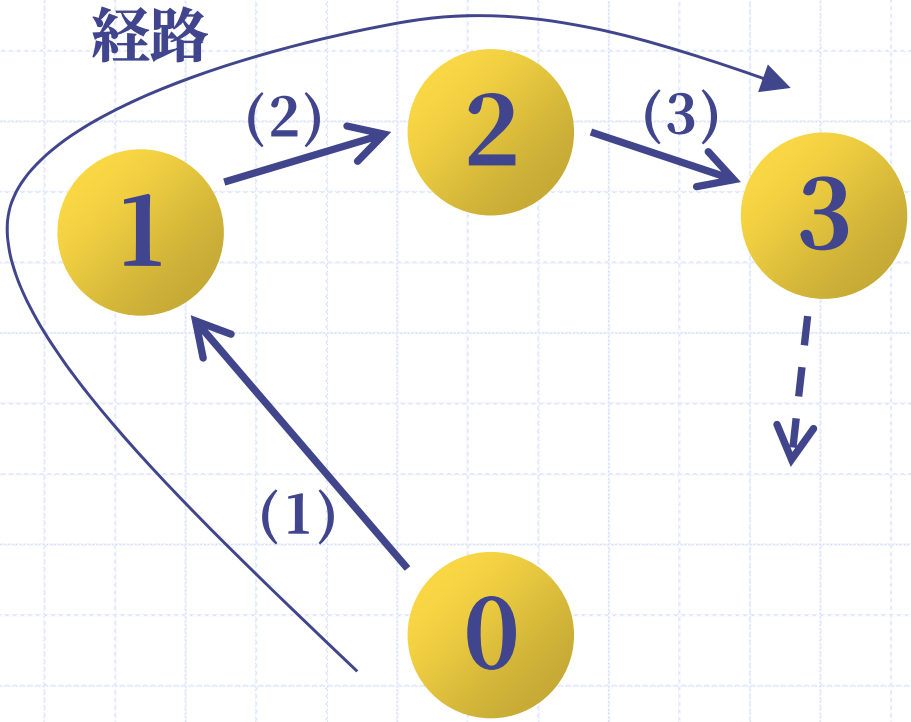
この研究は
WIDEプロジェクト
IDEONワーキンググループで
行われました。

概要

- 構造化オーバレイの通信パターン、反復探索と再帰探索を、いくつかの観点から比較し、得失を整理する。



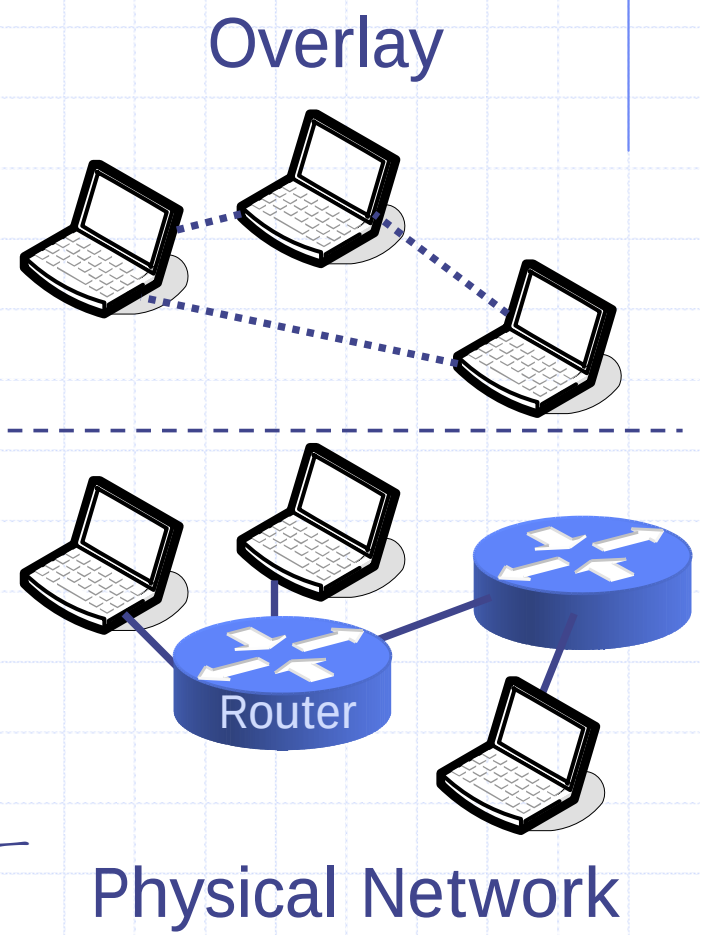
反復 (iterative)



再帰 (recursive)

オーバレイ (overlay)

- 他のネットワークの上に構築されたネットワーク
  - 例：電話ネットワーク上のインターネット
  - 例：ファイル共有ソフトのネットワーク
    FastTrack, eDonkey2K, Gnutella, …
    ノード数 100万以上
- ノード数が数万、数百万と増えてもなお性能・対故障性を保つため、
  自律的・非集中的に構築される
  アプリケーションレベルのネットワーク
  - 機能：検索, マルチキャスト, メッセージ配信, …
- そのトポロジは下位ネットワーク（インターネット）の物理的トポロジとは独立
  - それゆえ、オーバレイネットワークと呼ばれる。



Unstructured と Structured

- 非構造化 (unstructured) オーバレイ
  - 例： Gnutella ネットワーク, Winnyネットワーク
  - 誰を隣接ノードとするか、トポロジに制約がない。
  - 存在するオブジェクトは、発見できる可能性がある。
  - 一般に、効率は良くないが、柔軟な検索が可能。
- 構造化 (structured) オーバレイ

今回のターゲット

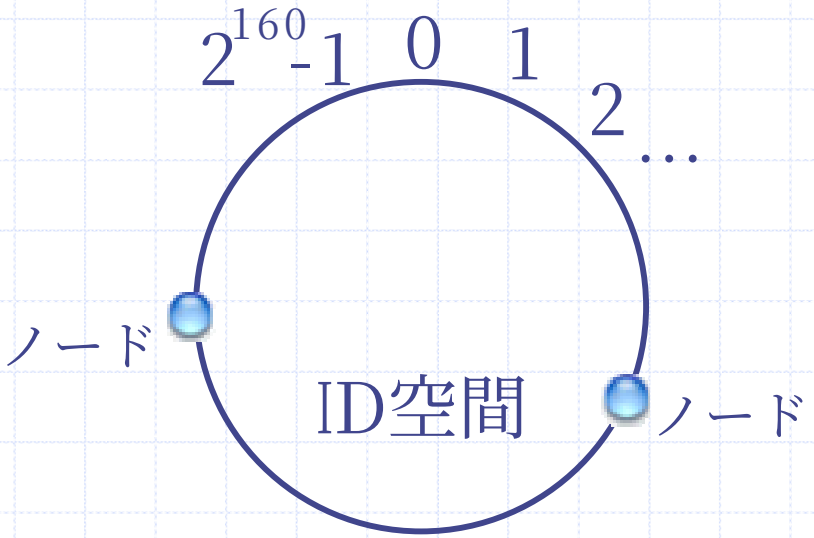
  - 例： DHT（分散ハッシュ表）のネットワーク
    - アルゴリズム: Chord, CAN, Pastry, Tapestry, Kademlia, …
  - 誰を隣接ノードとするか、トポロジに制約がある。
  - 存在するオブジェクトは、（たいてい）発見できる。
  - 一般に、効率は良いが、柔軟な検索が苦手。

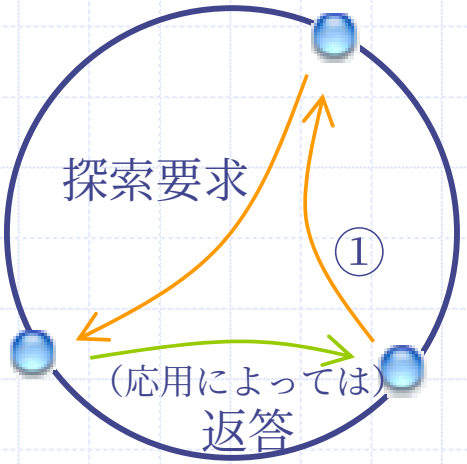
構造化オーバレイの応用

- 分散ハッシュ表 (DHT)
  - put(key, value), get(key), remove(...)
  - 例: Chord, Pastry, Tapestry, Kademlia, ...
  - 応用: もろもろの名前解決や位置解決
    - ホスト名 → IPアドレス, 名前 → 電話番号, 曲名 → 楽曲ファイルやそのURL, ...
- マルチキャスト
  - 構造化オーバレイで、配信木やその集合 (forest) を構築する。
  - 例: Scribe, SplitStream, ...
  - 応用: グループチャット, 音声/ビデオ会議, 放送, VPN, 分散処理, ...
- エニーキャスト
- ...

構造化オーバレイの基本

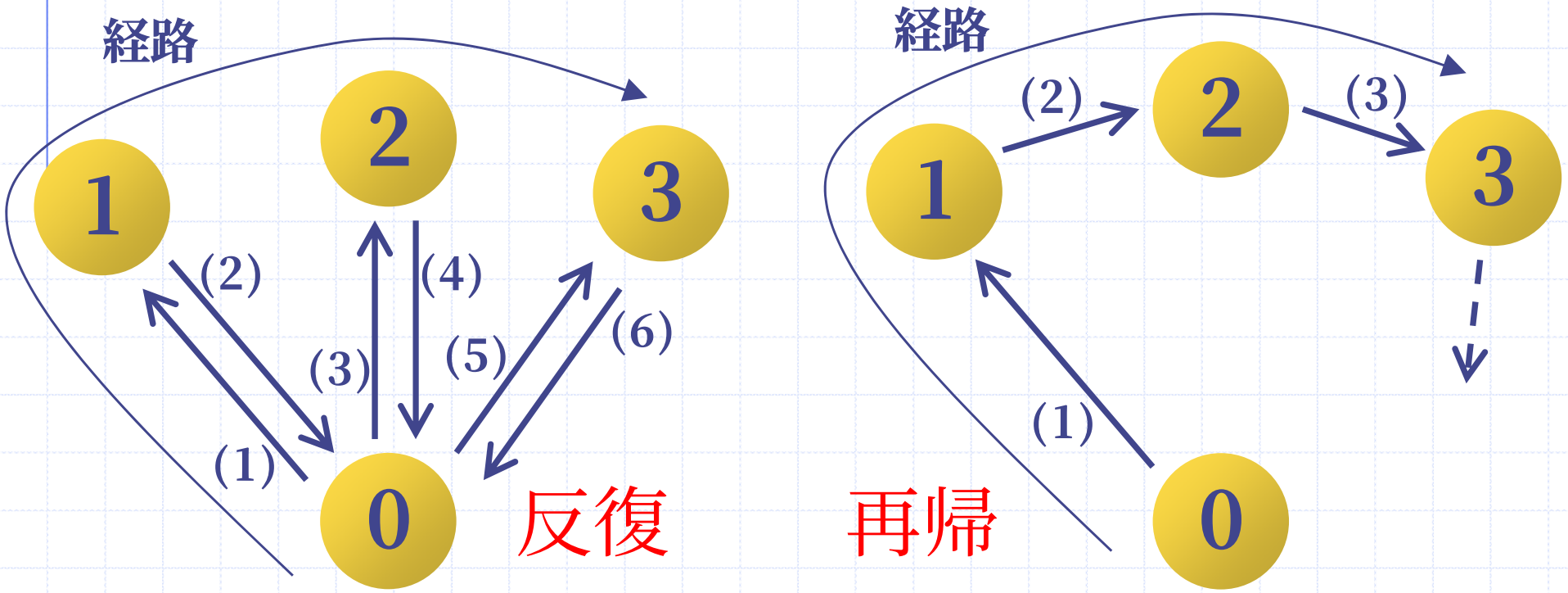
- ノード（計算機）とオブジェクトの両方にIDが振られる。
  - IDはたいてい整数値。160ビットだったり 128ビットだったり。
  - オブジェクト：任意の文字列だったり、ファイルだったり、プロセスだったり。
- ノードは、ID空間中のある範囲を担当する。
  - だいたい、ノードのIDと数値的に近い範囲を受け持つ。
- IDを宛先として担当ノードの探索 (ルーティング & 転送) が行われ、担当ノードに行き着く。





探索様式 (lookup styles)

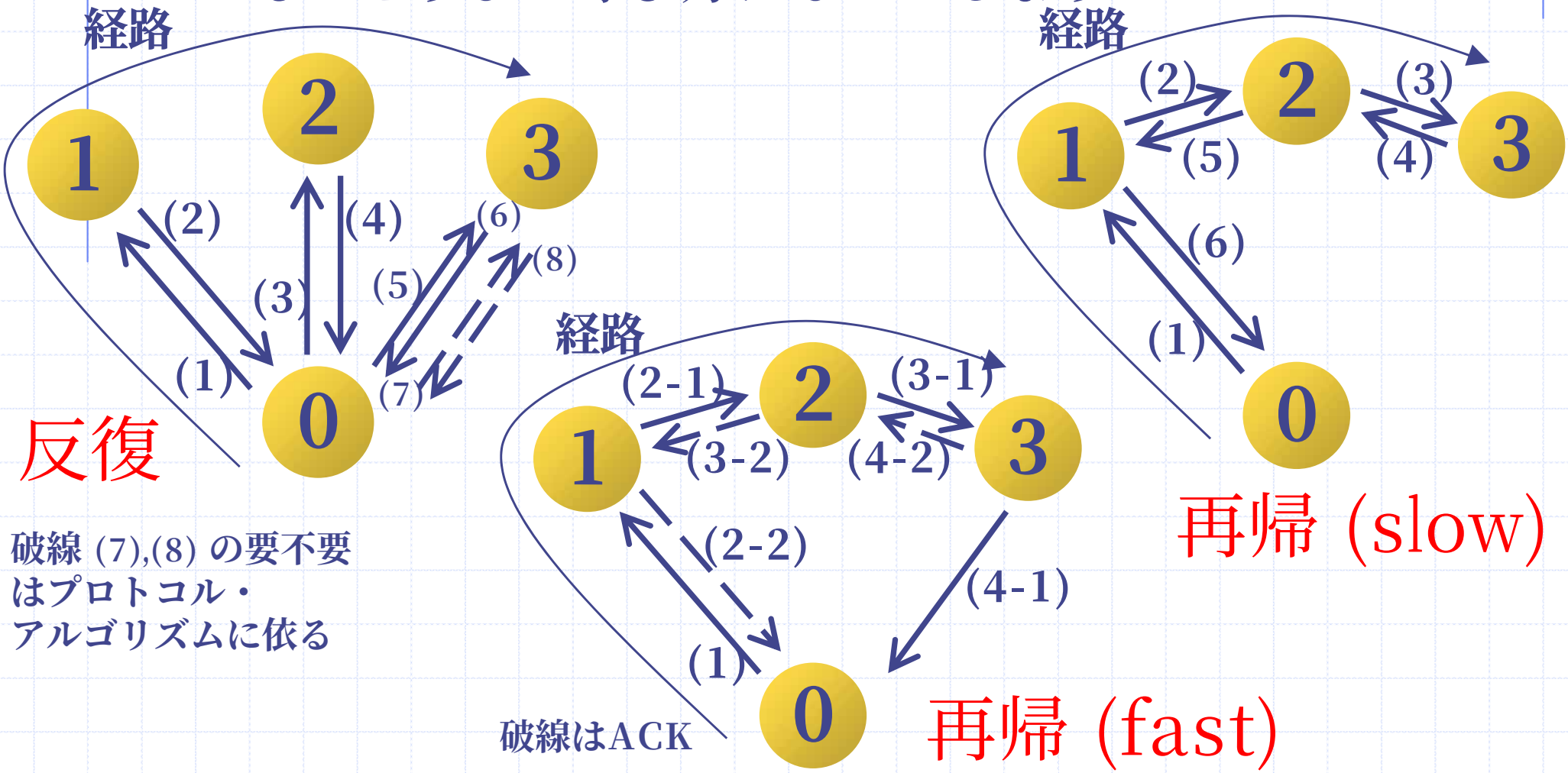
- 反復探索 と 再帰探索: iterative lookup と recursive lookup
  - 探索アルゴリズム (Chord, ...) とは独立。きっと。



- ○○ routing と呼ばれることが多いが、これは不正確。
  - routing とは「経路情報の構築」「次ホップの決定」を指す。
    次ホップへの転送は forwarding。

細かく分類すると

- 反復, 再帰 (fast), 再帰 (slow)
  - もっとうまい呼び方はないでしょうか？



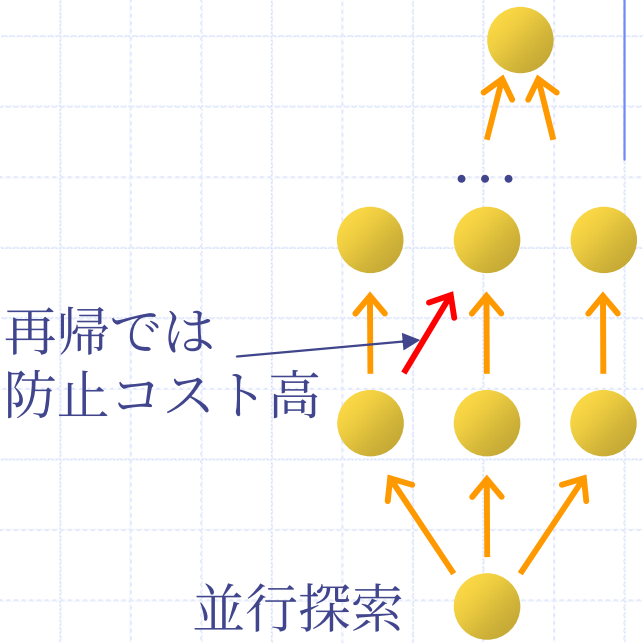
各実装による採用状況

- Chord [Stoica01] 反復
  - 論文には「どちらでも実装可能」と書かれている。
- Kademlia [Maymounkov02] 反復
  - 再帰探索での実装の方が、おそらく、厄介。
- Bamboo [Rhea04] 再帰 (fast)
- MS Pastry [Castro04] 再帰
- Overlay Weaver [首藤06] 反復 & 再帰 (fast)

- 選択は暗に行われてきた。
  - Chord (2001年) 時点で、言及はされていた。
  - Bamboo (2004年) でも明に比較検討は行われておらず。
  - 例外: secure routing の論文 [Castro02] は反復探索を切り捨てている: ルーティング攻撃という脅威を想定し、Pastry を前提とした場合、反復探索には安全性のメリットはなくコストは再帰 (fast) の 2 倍。

比較の観点

- 遅延 (応答性)
- プロトコル効率
  - メッセージのサイズ・数の小ささ
- 攻撃への耐性
  - 安全性



- 今回扱えなかった観点
  - 実装容易性
    - 考察・議論したが、まだ結論に至らず。
    - 状態 (既コンタクトノード, ブラックリスト等) の転送量や、非同期並行処理の記述量に依るか？
      - 再帰、特に再帰 (fast) は非同期処理多し。
    - フレームワークやツールキットの助けにも大きく依存する。
  - 並行探索の効果・効率・実装し易さ

遅延

- 均質な理想環境では、ホップ数に比例。
  - 均質: 通信遅延が一定, ノードの能力が均一
- 再帰 (fast) が良い。低遅延。
  - 再帰(fast) の遅延は反復の 0.6倍 [Dabek04] という結果も。

	反復	再帰 (fast)	再帰 (slow)
要求元に返答（例:DHT）			
ホップ数	**2n**	**n + 1**	**2n**
	or **2(n + 1)**		
メッセージ数	同上	$2n+1$	同上
返答なし（例:メッセージ配送）			
ホップ数	**2n − 1**	**n**	**n** (同左)
メッセージ数	同上	$2n$	$2n$ (同左)

経路長 n とした場合の
ホップ数 (とメッセージ数)

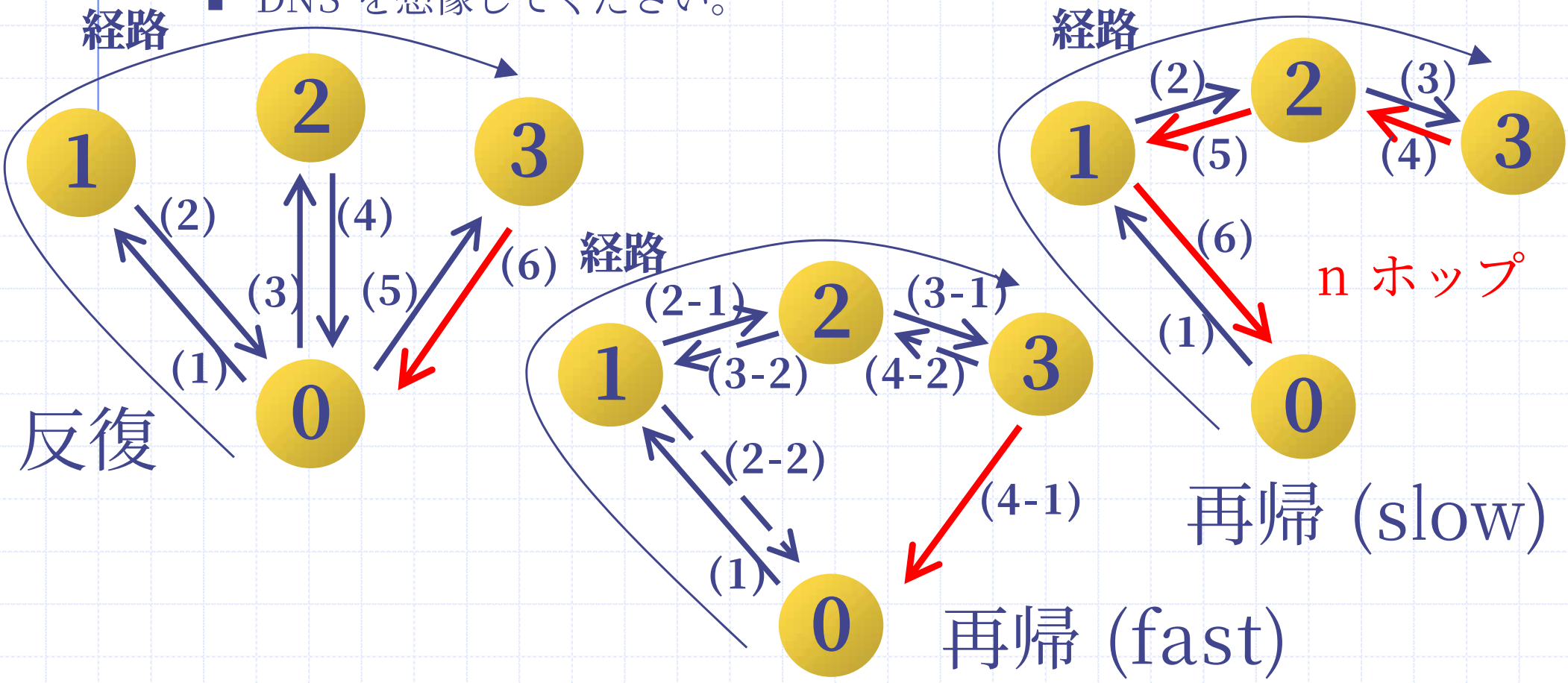
遅延

- 現実環境, 応用, 最適化が、
  各様式の良し悪しに影響する。

  - 応用: 返答データのサイズ
    - 大きいと再帰 (slow) 不利。
  - 最適化: 返答データのキャッシュ
    - 再帰 (slow) で効果的。
  - 現実環境: ノード故障
    - 故障検知の方式、探索のアルゴリズムによって、
      有利不利あり。

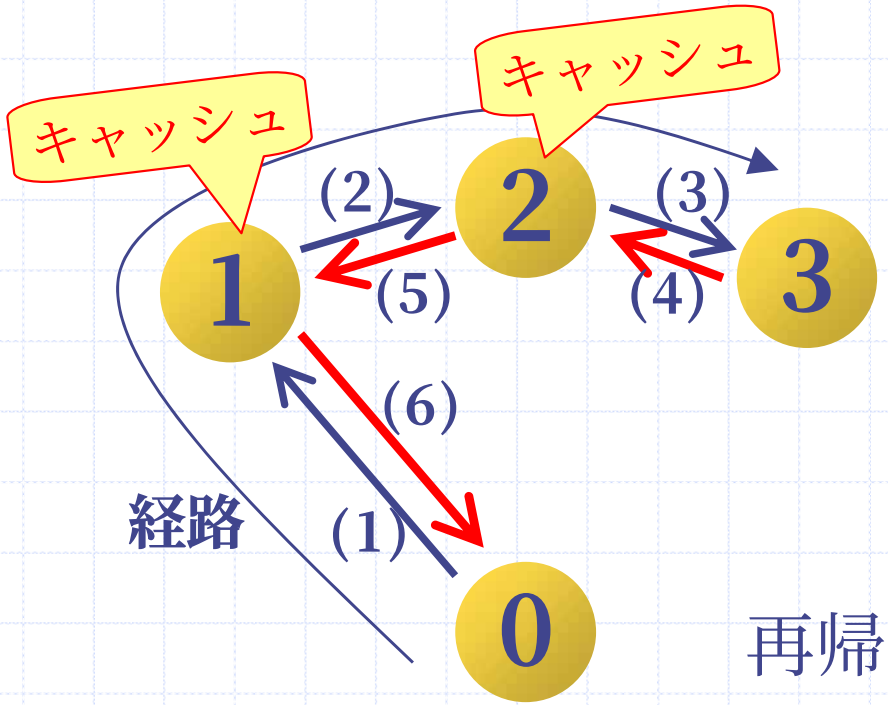
遅延: 返答データのサイズ

- 再帰 (slow) では、返答データが n ホップ転送される。
- 返答データが大きい場合、遅延大。
  - 問い合わせは小さく、返答が大きいような応用が多いだろう。
  - DNS を想像してください。



遅延: 返答データのキャッシュ

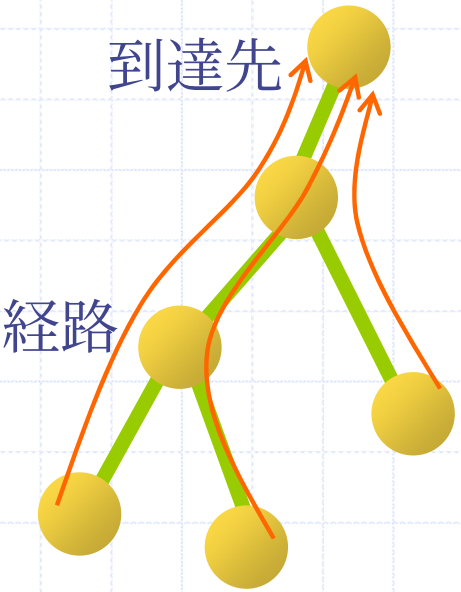
- キャッシュヒット → ホップ数減 → 遅延減
- 再帰 (slow) では、
  経路上のノードで自然にキャッシュできる。
  - 他の探索様式では、キャッシュするための明示的な通信が必要。
- 明示的な通信を伴う「複製」は、ここでは対象外。



再帰 (slow)

遅延: 返答データのキャッシュ

- キャッシュの効果を左右する要因:
  - 探索アルゴリズム (e.g. Chord) :
    つまり、経路長や、経路の収束具合い。
    - e.g. 経路長: Chord > Pastry
  - 応用:
    多数のノードから、少数の固定的な目標 ID 群に対して問い合わせが発生するような応用において、効果的。
  - 例: BitTorrent クライアント
    - トラッカーレス: 昨今の BitTorrent クライアントは、ファイル片の位置管理を DHT でも行うことができる。



同一 ID に向けた探索

- ↑ 探索様式の選択時には
  ここまで考慮することが理想。

遅延: ノード故障

- メッセージ送信先ノードが故障
  → 他の次ホップ候補へ送信し直し
  → 遅延増大

- 送信先ノードの故障は
  タイムアウトで検知される。
  - 場合によっては数秒以上 → 探索遅延への影響大
    - 探索自体はたいてい ～数百ミリ秒で済む。

- 故障ノードへの送信を防ぐことが重要。

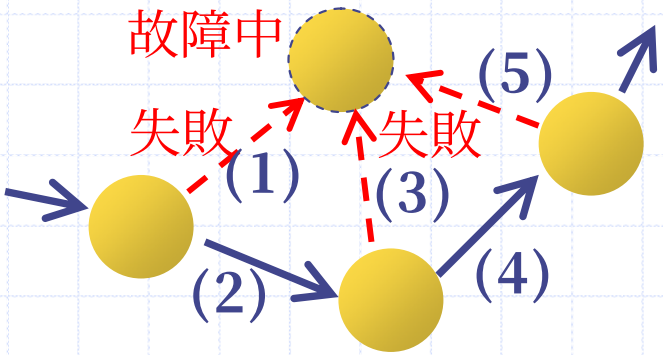
遅延: ノード故障

- 故障検知の方針によって、
  反復/再帰の有利不利が変わってくる。
  1. 通信の失敗だけをもって、故障を検知する。
  2. 能動的に故障検知のための通信を行う。
     - 生存確認, ハートビート, ...

- 1. は再帰が不利となる要因がある。
- 2. は反復が不利となる要因がある。

遅延: ノード故障

- **通信の失敗だけで故障を検知する場合**
  - 再帰が不利となる要因あり。
- 故障ノードへの送信が繰り返される恐れがある。
  - 送信先が故障 → 次善の次ホップ候補へ送信 → その次善のノードも、故障ノードに対して送信 → ...
  - タイムアウトが複数回。遅延増大。
  - Kademlia (サブセット) + 再帰 (fast) で経験した。
- 考察
  - 探索の終盤で起きやすい???
  - アルゴリズムによっては、一度故障ノードをまたいだら、二度と通信せずに済む???
  - 再帰: 少数ノードに大きな被害
    反復: 多数ノードに小さな被害?

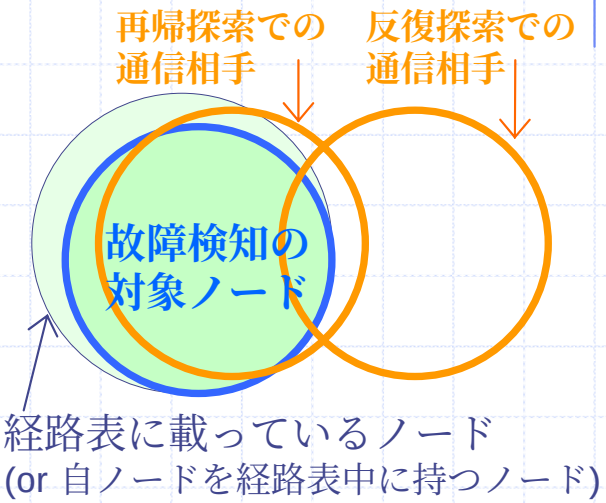


遅延: ノード故障

- 故障ノードへの送信が繰り返される恐れ
  - 防止策の例: 故障ノードの表をメッセージとともに転送していく → 転送コスト増大
- 反復探索なら防止は容易。コスト増なし。
  - 故障ノードの一覧表は、探索要求元だけが保持・管理すればよい。転送不要。

遅延: ノード故障

- **能動的に故障検知のための通信を行う場合**
  - 反復が不利となる要因あり。
- 「故障を検知できる相手」と「通信相手」の間に相関が
  - ある (再帰)
  - ない (反復)
    - 故障ノードへの送信を防ぐことが困難。



- 理由
  - 生存確認できる相手は、自ノードの経路表に載っているノード（または、自ノードを経路表中に持つノード）である。
  - 再帰探索では、通信相手は自ノードの経路表に載っているノードであり、反復探索では、主に他ノードの経路表に載っているノードである。
    → 反復では、自分の経路表に載っているノードと通信相手の間に相関がない。通信相手の予測も一般には困難。

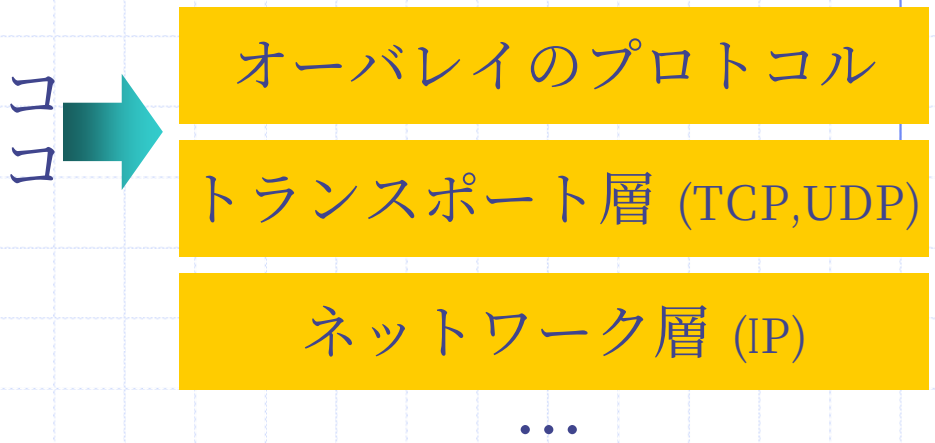
遅延: 近傍ルーティング

- 近傍ルーティングの効果が異なる。
  - 近傍ルーティング (proximity routing)
    - 実ネットワーク上の近接性を考慮したルーティング。
      メッセージ送信先として自らに近いノードを選び、通信遅延を小さくする。
    - 次ホップを選択する際に近接性を考慮する手法 (PNS, PRS) と、ノードの ID を決める際に近接性を考慮する手法 (PIS) がある [Gummandi03]。

- PNS,PRSでは 反復が不利。効果を得ることが困難。
  - 前ページと同じ理由。
  - 反復では、経路表に載っているノードと通信相手の間に相関がない。
    通信相手が多い。近接性の計測・保持が困難。

プロトコル効率

- 基準
  - メッセージ数の少なさ
  - メッセージサイズの小ささ



- 前提
  - 信頼性のない下位通信層 (例: UDP) を前提とする。
    - 信頼性のあるプロトコル (例: TCP) は ACK, NACK などを勝手に送受信する。
  - または、オーバレイのプロトコルより下の層については、ここでは考えない。

プロトコル効率: メッセージ数

- どれも似たようなもの。2n 前後。

	要求		返答		ACK※
再帰 (fast)	n	+	1	+	n
再帰 (slow)	n	+	n		
反復	n	+	n		
	or $(n+1)$	+	$(n+1)$		

※下位通信層が信頼性のないもの(例: UDP)である場合に必要。
信頼性ありの場合(例: TCP)にはそもそも下位通信層が追加の
メッセージ (ACKやNACK) を送受信している。今回は考慮せず。

- 再帰 (fast) には、ACK省略 → 信頼性を犠牲にして
メッセージ数を減らす自由度がある、とも言える。

プロトコル効率: メッセージサイズ

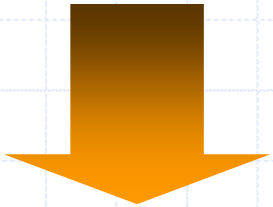
- 一概にどれが良いとは言えない。
  - 応用にも依る。
    - 問い合わせ内容/結果の大きさ。
- メッセージの識別子, 要求元アドレスを要求メッセージに含める必要性
  - ✗ 再帰 (fast)
    - 必ず含める必要あり。返答メッセージを戻す先として。
  - ○ 反復, 再帰 (slow)
    - UDP, TCP では省ける。
- 再帰 (fast) では、メッセージ数の半数を占める ACK が小さい。
  - 該当するのは、反復では複数の次ホップ候補, 再帰 (slow) では問い合わせ結果。どちらも ACK より大きい。

プロトコル効率: メッセージサイズ

- 応用依存
  - 再帰 (slow) は、問い合わせ結果が大きいと不利。
    - n 回転送する必要がある。
  - 反復は、問い合わせ内容が大きいと有利。
    - 1度の転送で済むから。
- 構造化オーバレイのパラメータ依存
  - 反復は、次ホップ候補を複数含む返答メッセージを n回転送する点、不利。
    - 次ホップ候補: IP(v4)アドレス + ポート番号で６バイト。これを複数個含む。数はパラメータ。

プロトコル効率: メッセージサイズ

- なんだかんだ言って、
  MTU に収まってさえいれば同じじゃん。
  - MTU: TCP/IP ではたいてい 1500オクテット弱?



- 応用や下位通信層まで想定して計算しないことには意味がない？

今後の課題

攻撃への耐性: 議論の前提

- 探索様式 (反復/再帰) だけで安全性を確保できるわけではない
  → 「探索様式が」安全性に与える影響を考察する。
  - 他の要因: オーバレイへのjoin手順, ノードへのID付与方式, 経路表の構築/保守方式, メッセージ転送方式, …

- ここで対象とする脅威モデル:
  少数の敵対者が、特定の組織や地域を攻撃対象とする。
  1. 敵対者は少数である。
     - 多数の場合、そもそも守れない。
       例: Sybil attack, Eclipse attack
  2. 敵対者は、オーバレイそのものの破壊を目的としない。
     - オーバレイの破壊が目的であるなら、無差別なルーティング攻撃が可能。この場合の安全性に探索様式は影響しない。
  3. 敵対者は選択的な攻撃を行う。
     - オーバレイそのものが攻撃対象でない以上、特定の組織や地域を攻撃対象とすると想定される。

攻撃への耐性: 想定する脅威

- 前提
  - 敵対者は、パケットの操作だけでなく、盗聴、改竄が可能
- 探索の要求元ノード、中継ノード、到達先ノード、オーバレイ外部のノードが、それぞれ何を行えるかを整理

- 探索様式が影響するもの → 考察の対象
  - トラフィック増幅攻撃
    - メッセージ数の増幅、巨大な返答メッセージなどによるサービス妨害
  - 転送妨害 (ルーティング攻撃)
    - blackhole, grayhole: 転送をしない。
  - トラフィック解析
    - ノードIDとIPアドレス（→組織や地域）を対応付ける。
      → 特定組織や地域を狙った攻撃につながる。
- 探索様式が影響しないもの → ここでは扱わず
  - サービス不能攻撃
  - 返答拒否

探索様式ごとの耐性

	反復	再帰 (fast)	再帰 (slow)
トラフィック増幅攻撃	○ 困難	△ 脆弱	△ 脆弱
転送妨害の発見	△ 容易	× 困難	× 困難
トラフィック解析	× 脆弱	× 脆弱	○ 困難

← 抑制手法あり

← Secure routingで無力化

オーバレイに参加せずとも tap 可能

- トラフィック増幅
  - 反復: 全問い合わせを探索要求元が行う → 増幅効果なし。
  - 再帰: 1つの問い合わせが複数のメッセージを発生させる。
    ただし、抑制手法あり。
- 転送妨害
  - 反復: grayhole, blackhole は容易に検知できる。
  - 再帰: それすら検知は厄介。
    ただし、この種のルーティング攻撃は secure routing [Castro02] で無力化できる。
- トラフィック解析
  - 反復, 再帰 (fast): すべてのノードとの直接通信が起きる
    → ノードIDとIPアドレスの対応付けが可能。
  - 再帰 (slow): 経路表に載っているノードとの通信しか起きない。
  - 参加/離脱を繰り返してノード情報を集める攻撃も考えられるが、防止手法あり。

まとめ

- 遅延
  - 一般に — 再帰 (fast) ○
  - 返答データのサイズ — 再帰 (slow) ×
  - 返答データのキャッシュ — 再帰 (slow) ○
  - ノード故障
    - 通信失敗のみで故障検知 — 再帰 ×？
    - 能動的に故障検知 — 反復 ×
  - 近傍ルーティング — 反復 ×
- プロトコル効率
  - メッセージ数 — 似たようなもの
  - メッセージサイズ — 一概には言えない
- 攻撃への耐性
  - トラフィック増幅攻撃, ルーティング攻撃 — 反復 ○
    - ただし、他の様式でも対策はある (例: secure routing)。
  - トラフィック解析 — 再帰 (slow) ○

まとめ

- 単一の探索様式ですべての要件を満たすことは難しい。
- 応用、要件やその優先順位を考慮して選ぶべき。
- 長所
  - 再帰探索 (fast)
    - 遅延が小さい。近傍ルーティングの効果を得やすい。
    - メッセージ数を n+1 に減らすことが可能。
  - 再帰探索 (slow)
    - 遅延は小さくないが、近傍ルーティングの効果は得やすい。
    - ノードIDに対応するIPアドレスが漏れにくい。
  - 反復探索
    - 作りやすい？
    - 並行探索の効率が良い？
    - ルーティング攻撃を検知しやすい。
      (が、再帰でも secure routing によって防げる。)

大雑把に言った利点：

性能・効率

安全性

作りやすさ？

- 方式提案の論文で書き尽くされたわけじゃない。実装までもっていくと、まだまだやることが一杯ある！元気な若者よ来たれ！ (by 門林)

参考文献

- [Stoica01] I. Stoica et al., "*Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*", 2001.
- [Maymounkov02] P. Maymounkov et al., "*Kademlia: A Peer-to-peer Information System Based on the XOR Metric*", 2002.
- [Castro04] M. Castro et al., "*Performance and dependability of structured peer-to-peer overlays*", 2004.
- [Rhea04] S. Rhea et al., "*Handling Churn in a DHT*", 2004.
- [Dabek04] F. Dabek et al., "*Designing a DHT for low latency and high throughput*", 2004.
- [首藤06] 首藤一幸 et al., "*オーバレイ構築ツールキット Overlay Weaver*", 2006.
- [Castro02] M. Castro et al., "Secure routing for structured peer-to-peer overlay networks", 2002.
- [Gummadi03] K. Gummadi et al., "The Impact of DHT Routing Geometry on Resilience and Proximity", 2003.