VIOPS-04 Virtualized Infrastructure Operators Group

DHT/分散ストレージの技術研究と実証実験

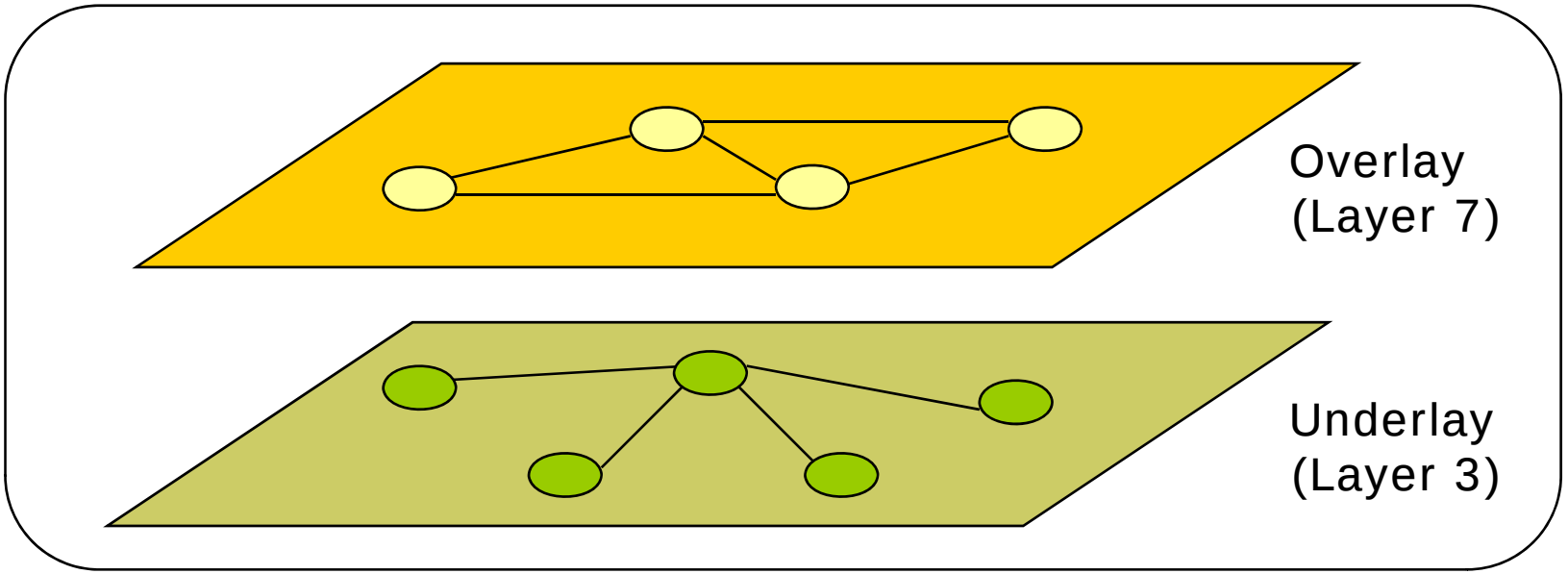
門林雄基
奈良先端科学技術大学院大学 (NAIST)
情報科学研究科 /
情報通信研究機構 (NICT)
情報通信セキュリティ研究センター
http://xdt.sourceforge.net/

Part I:
オーバーレイネットワーク/DHT
イントロダクション

オーバーレイ・ネットワークとは *

- 単一障害点が無い、
- フレームワークまたはアプリケーションが自律的に構成する、
- 目的志向のネットワーク

- サービスを分散した計算機群によって構成する手法

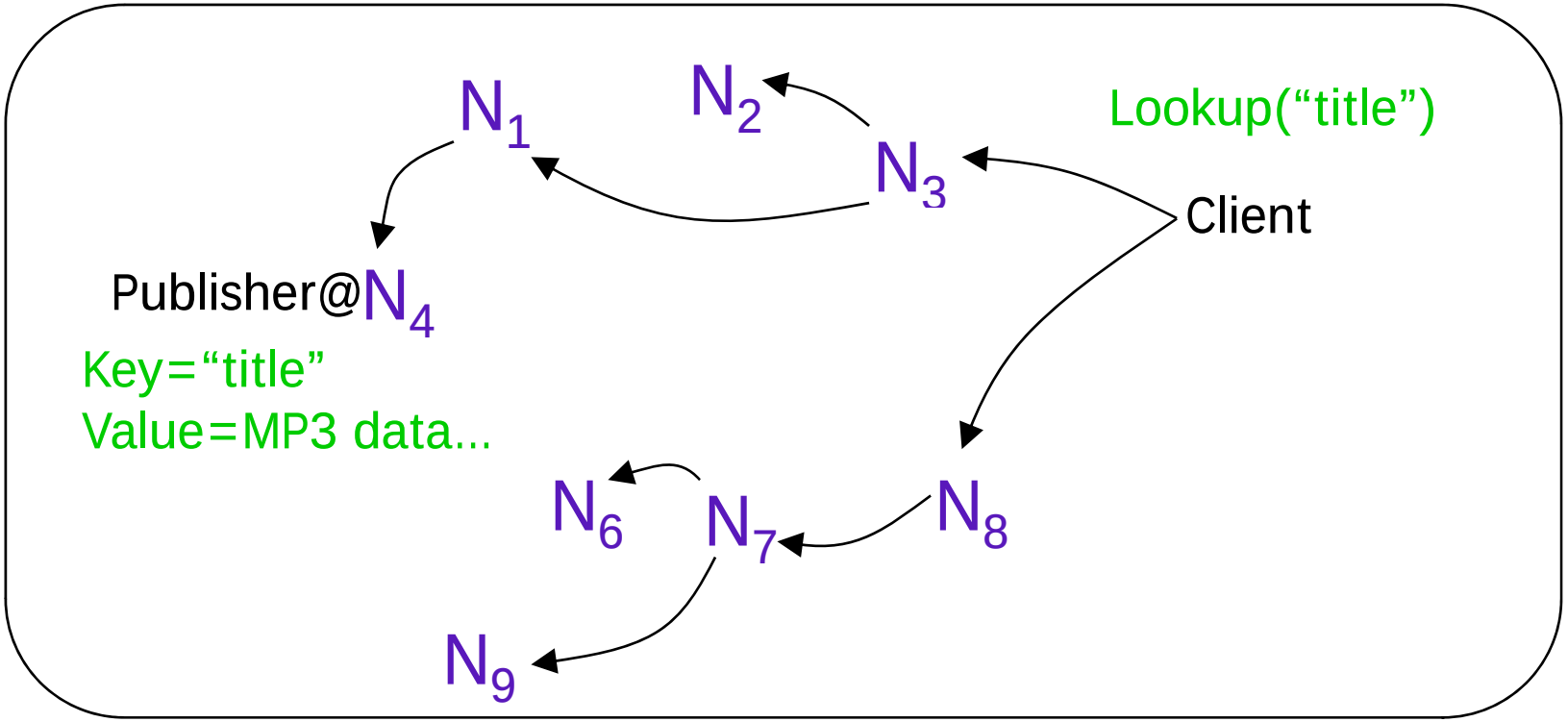


オーバーレイの類型 *

- 構造のないオーバーレイ (unstructured overlay)
  - “Peer-to-Peer”, “P2P”
  - ノード間のトポロジを規定しない
- 構造化オーバーレイ (structured overlay)
  - ノード間のトポロジとして一定の構造を規定することにより、資源探索や経路制御の効率を求めたオーバーレイ
- 説明のため、
  オーバーレイ上でのファイル探索問題を考える
  - ノード数、ファイル数ともに膨大

構造のないオーバーレイにおける探索
unstructured overlay

- ノード間のトポロジを規定しない
- 問い合わせをフラッディング



障害に強いが、探索時に最悪 O(N) のメッセージ

構造化オーバーレイ
structured overlay

- ノード間のトポロジとして一定の構造を規定することにより、資源探索や経路制御の効率を求めたオーバーレイ

- 問い合わせをルーティング
  - Chord を例として *

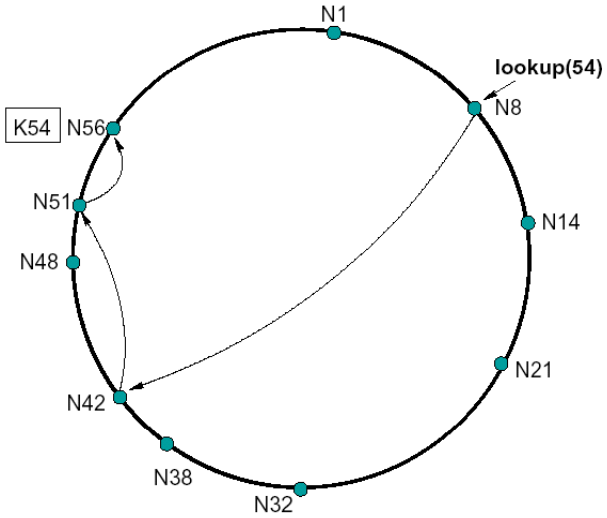
構造化オーバーレイの方式研究

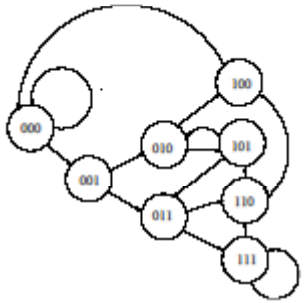
- Tapestry (Kubiatowicz et al., @ UCB, 2000)
- Chord (Stoica et al. @ MIT, 2001)
- Pastry (Rowstron et al. @ MS, 2001)
- Kademlia (Maymounkov et al. @ NYU, 2002)
- Koorde (Kaashoek et al. @ MIT, 2003)

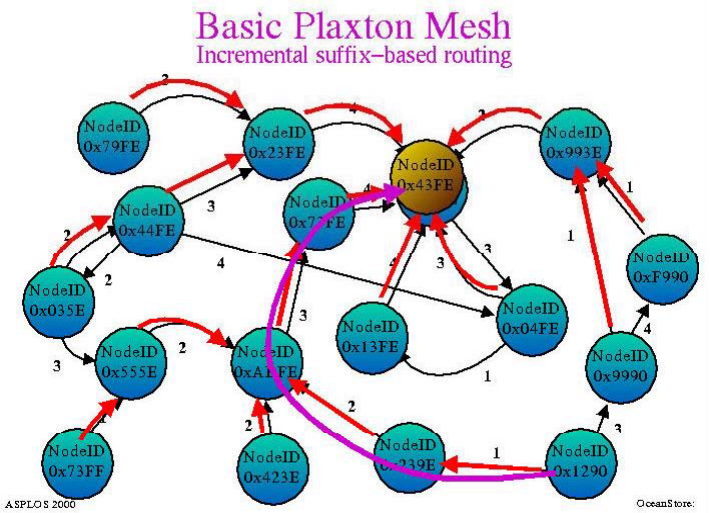
  …

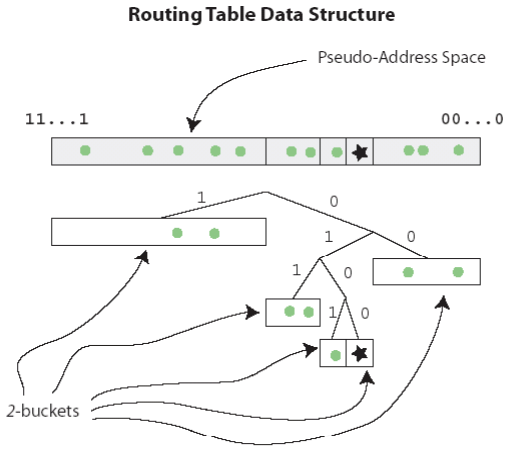
- トポロジや経路制御手法に差異
- 通信量、計算量、対故障性などについて比較評価がなされている

構造化オーバーレイ:さまざまなトポロジ









Part II:
オーバーレイ上へのサービス構築

構造化オーバーレイの3側面 *

- Rendezvous
  - **オーバーレイへの加入および近接性の維持方式**
  - ノードがオーバーレイに加入するときに踏むべき手続きと、加入後に隣接ノードとの通信を維持するために必要な手続きを規定する
- Location
  - **資源探索方式**
  - 探索する資源を識別子空間に射影する方式と、アプリケーションに対する資源探索のサービスモデルを規定する
- Routing
  - **経路制御方式**
  - ある識別子にあててメッセージを送るアルゴリズムを規定する

構造化オーバーレイのサービスモデル

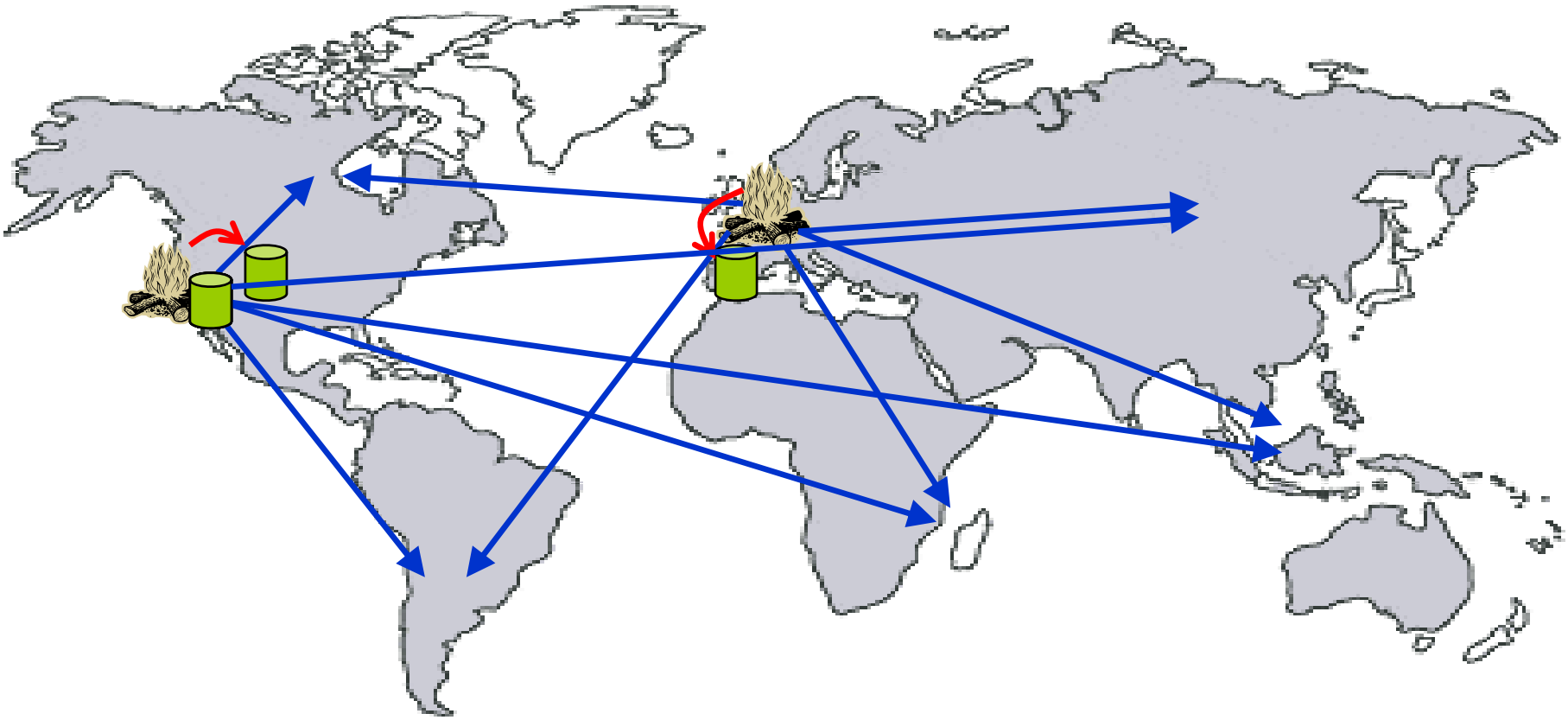
- DHT
  - Distributed Hash Table
  - ハッシュ表を分散して保持することにより分散ストレージを構成
  - put(key, value);　value = get(key);
- DOLR
  - Decentralized Object Location and Routing
  - オブジェクトの識別子を指定し、任意のメッセージをへ送る
    - 近傍のコピーへ
  - sendMessage(objectID, message);
- CAST
  - N個のノードが、グループを指定して参加・離脱
  - N=1: Anycast,　N>1: Multicast

構造化オーバーレイのAPI (Common APIの発表資料から)

Distributed Hash Tables (DHT)	Decentralized Object Location / Routing (DOLR)	Multicast / Anycast (CAST)
put (key, data)	*publish (objectId)*	*join (groupId)*
remove (key)	*unpublish (objectId)*	*leave (groupId)*
value = get (key)	*sendToObj (msg, objectId, [n])*	*multicast (msg, gId)* *anycast (msg, gId)*

F. Dabek et al., “Towards a Common API for Structured Peer-to-Peer Overlays”, IPTPS’03.

DOLR vs. Distributed Hash Table（Tapestry発表資料から）

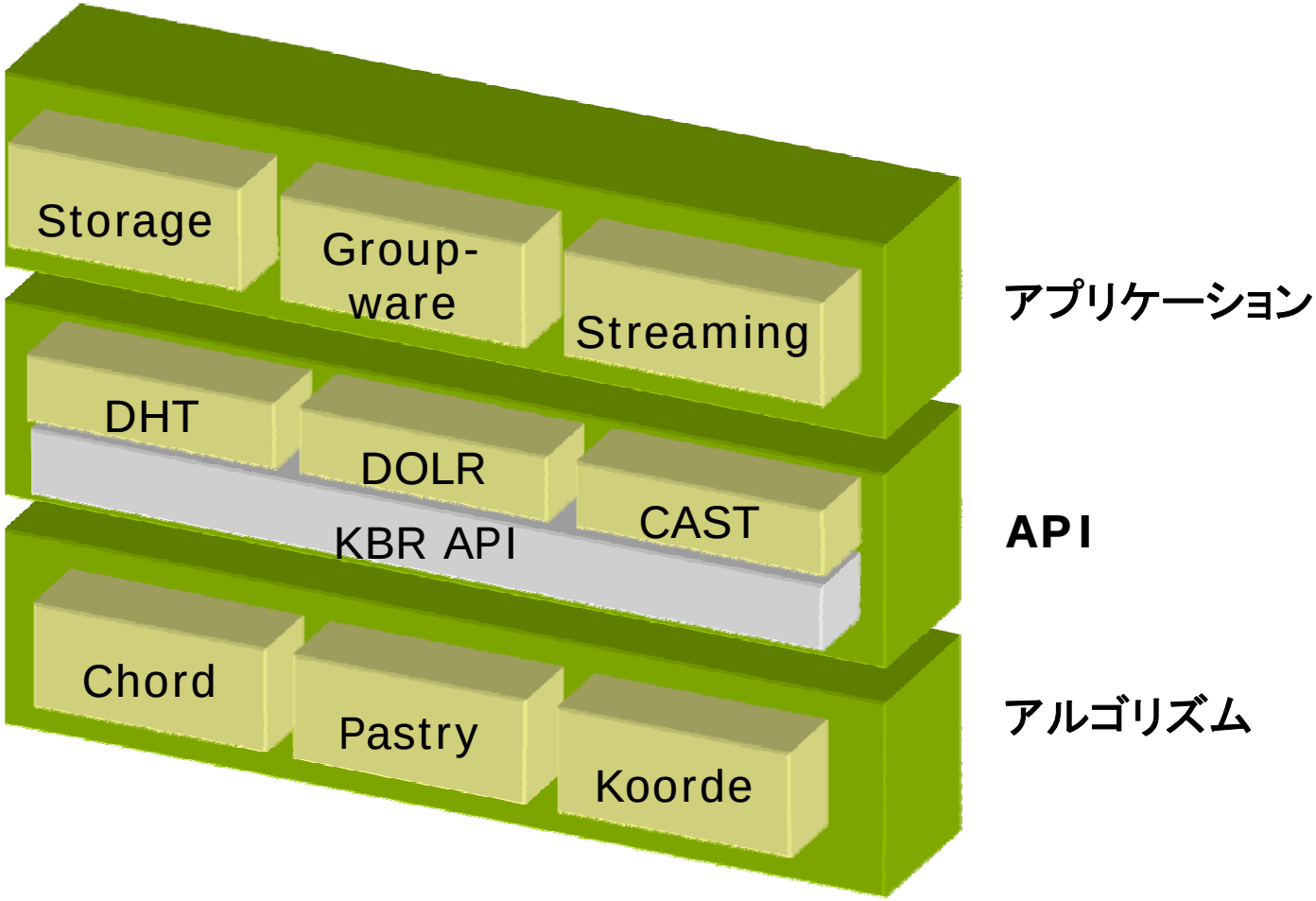


- DHT: hash content → name → replica placement
  - modifications → replicating new version into DHT
- DOLR: app places copy near requests, overlay routes msgs to it

KBR API で DHT, DOLR 等を実現

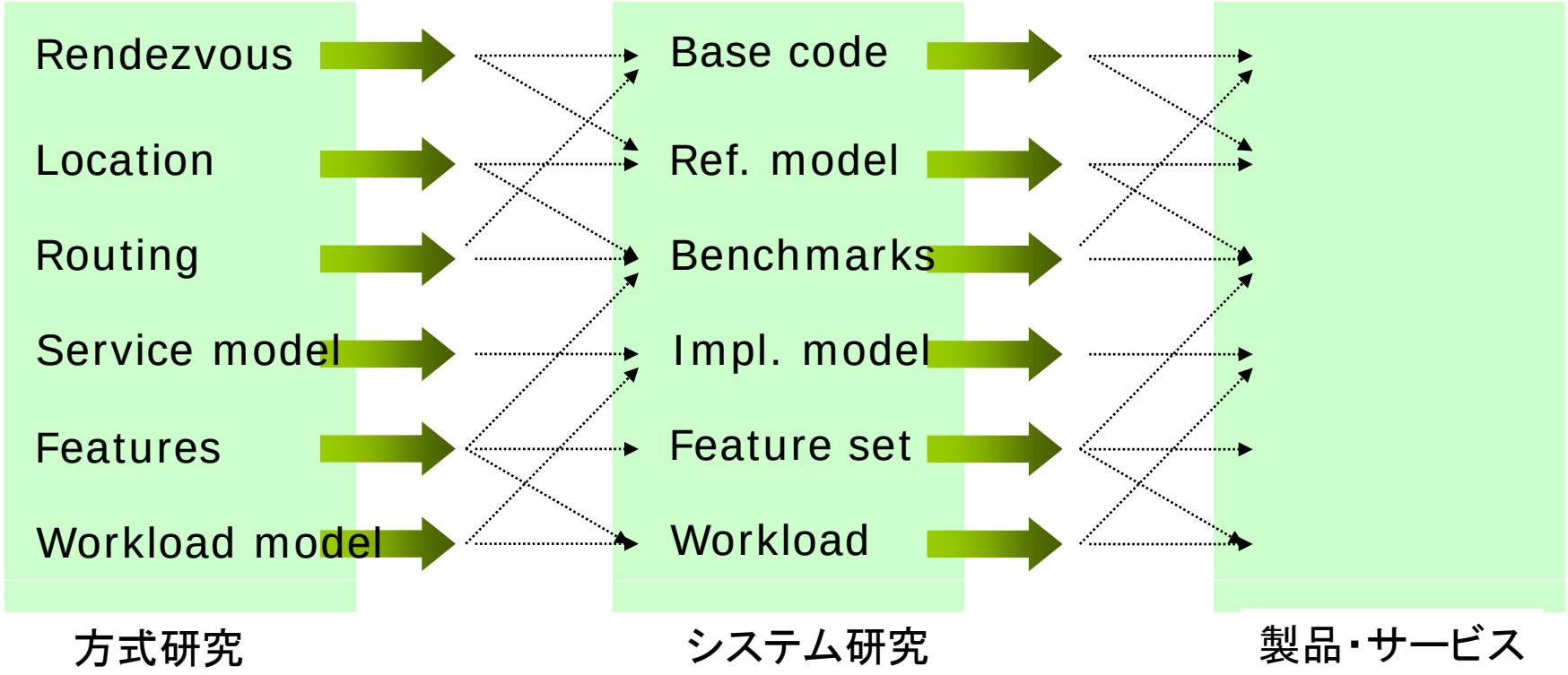
- **Deliver(key, msg)**
  - Delivers a message to application at the destination
- **Forward(&key, &msg, &nextHopNode)**
  - Synchronous upcall with normal next hop node
  - Applications can override messages
- **Update(node, *boolean* joined)**
  - Upcall invoked to inform application of a node joining or leaving the local node's neighborSet

KBR API を中心とした構造化オーバーレイのフレームワーク



ヒント: 研究成果の活かし方

- 方式研究 → システム研究 → 製品・サービス
  - 各論文はバラバラな条件設定でシステムを作っている
  - そのまま製品・サービスに転用できるわけではない



Part III:
Failure modes of DHT
DHTの故障モード

ノードの離脱 (Node churn)

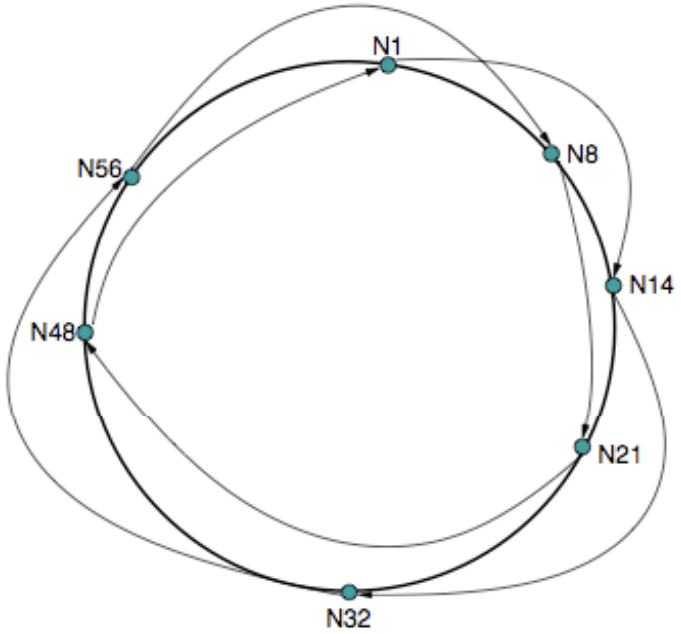
- DHTの設計段階では、ノードは離脱する前に他ノードに「お知らせ」して担当データを移譲することになている。(非現実的)

- Node churn
  - 「お知らせ」せずに突然いなくなる
  - これが普通では？(オペレータのセンスでは)

- ノードの離脱への万能な対応は難しい

一貫性のない状態

- 様々な理由により、一貫性のない状態になりうる:
  - ノードの離脱
  - 新規ノード
  - 不安定なリンク
  - 過渡状態
- メビウスの輪
  - 始点によって見え方が違う！



D. Liben-Nowell et al., “Analysis of the Evolution of Peer-to-Peer Systems”, PODC’02.

資源の枯渇（実装依存）

- Open files が無くなる
- Ephemeral port number が無くなる
- Shared memory
- Thread pool
- メモリリーク

- … 大規模Webサーバと同様
- 枯渇状況が見えないリソースもあるので注意

競合状態 (Race condition)

- 以下の呪文を同時に唱えると鬼門が開く:
  - スレッド
  - Mutex
  - Reference counting
  - マルチコア・プロセッサ

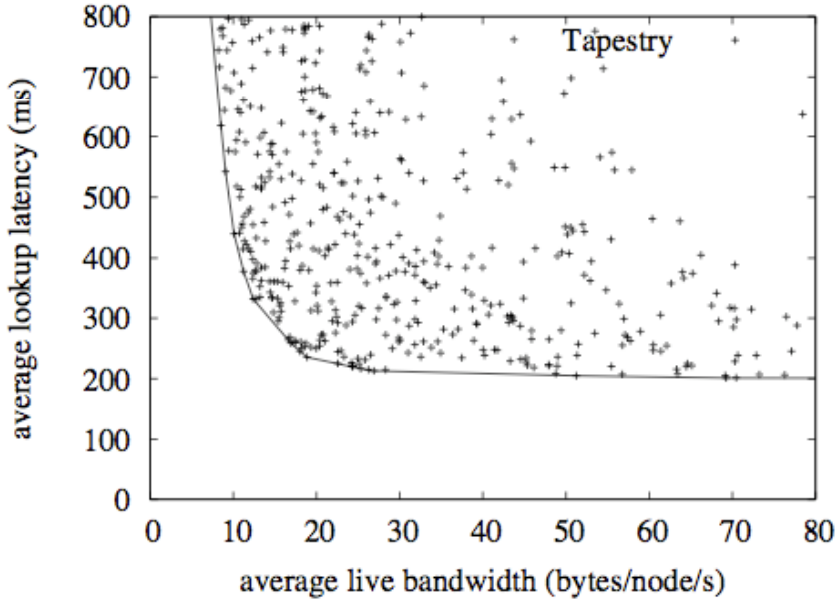
- Heisenbug
  - 観測しようとすると再現しない
  - 実運用レベルまでスケールを上げないと再現しない

  - … Welcome to our club!!

Part IV:
DHT Myths and Realities
DHTの幻想と現実

レスポンスタイムに関する幻想と現実

- DHTの応答時間は短い？
  - RTT，利用帯域，負荷に依存する
- スケールアウトする一方で、応答時間は保証しない
  - 応答時間を優先するなら KVS では？
  - 応答時間が最も重要なら SLB では？



J. Li et al., “Comparing the performance of distributed Hash tables under churn”, IPTPS’04.

メモリ容量に関する幻想と現実

- DHTはいたずらにメモリを消費する？
  - 実装言語による
- Java等のガベージコレクション:
  - malloc/free の5倍のメモリを使えば性能は互角
  - malloc/free の3倍のメモリ: 17% の性能ペナルティ
  - malloc/free の2倍のメモリ: 70% の性能ペナルティ
- M. Hertz, E. Berger, “Quantifying the Performance of Garbage Collection vs. Explicit Memory Management”, OOPSLA’05.

DHTはチューニングできるのか

- YES

- ハッシュのビット長
- Finger table エントリ数
- Successor リスト数
- M
- K
- P
- Alpha
- ….

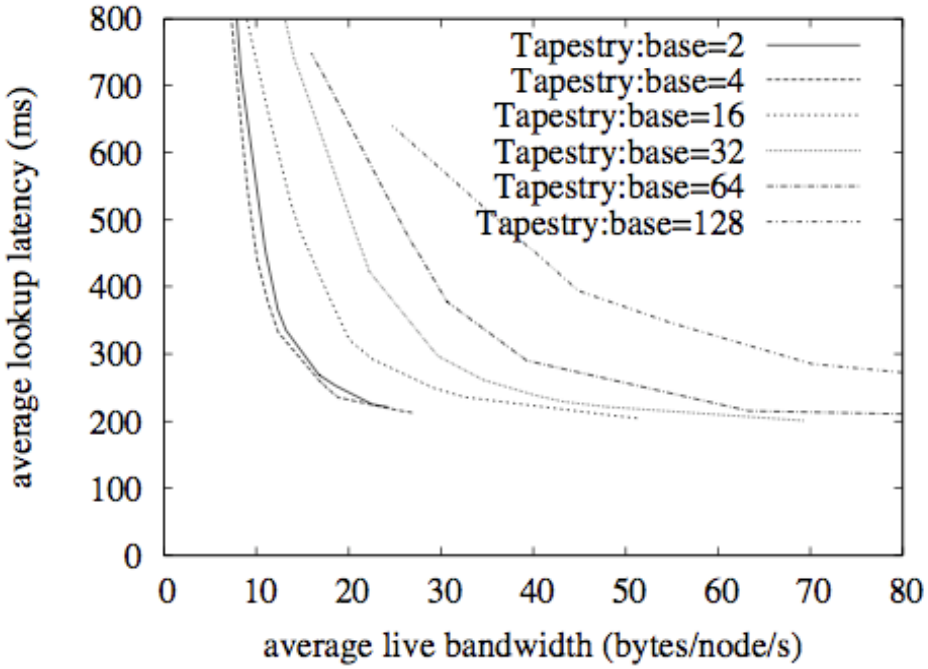


Figure 3: The effect of base in Tapestry.

J. Li et al., “Comparing the performance of distributed hash tables under churn”, IPTPS’04.

- 用途、想定負荷・規模に合わせた最適化をしましょう

DHTは信頼できるのか

- ノードの加入・離脱を制御できるならば YES
  - ネットワーク、OSの信頼性含む
- 制御できないなら NO
- S. Zhang, I. Stoica, R Katz, “The Cost of Inconsistency in DHTs”, UCB/CSD-5-1394.

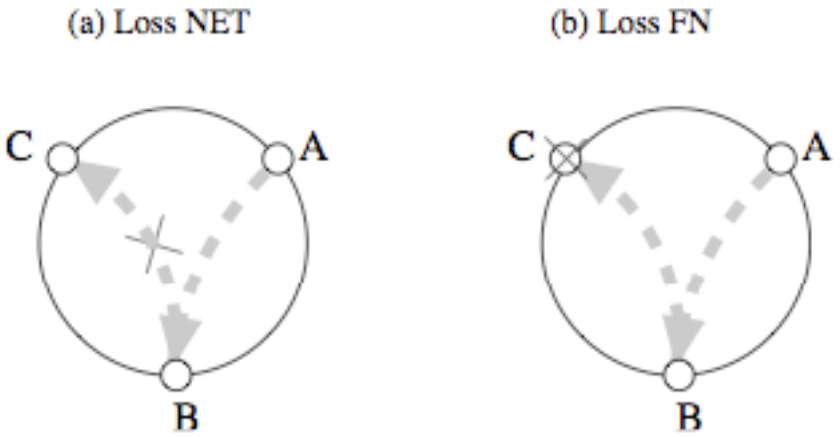


Figure 2: Possible scenarios of a lookup loss in Chord.

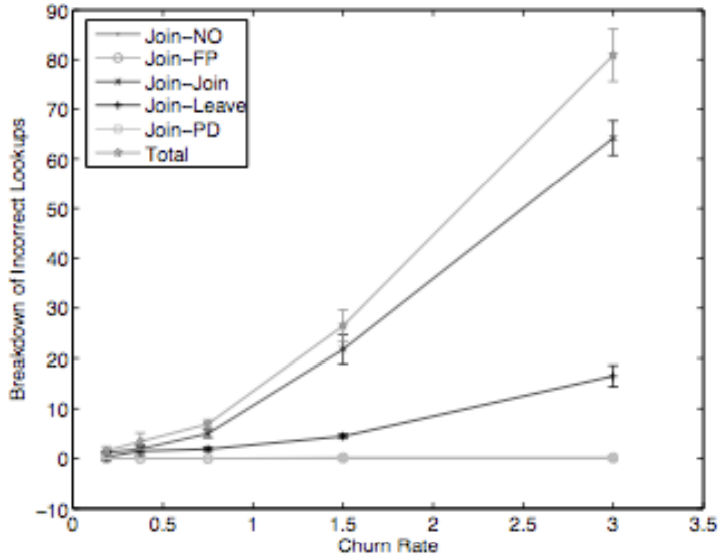
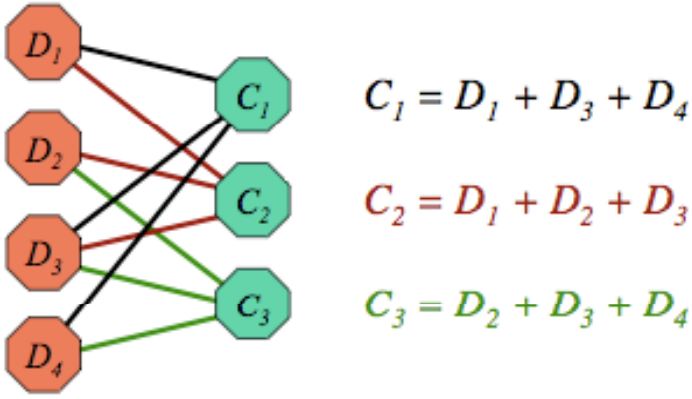


Figure 10: Breakdown of incorrect lookups vs. churn rate

DHTの可用性は保証できるのか

- ストレージの冗長性を制御できるならば YES
- 制御できないなら NO
- 「ネットワークRAID」は実現できる
  - → LDPC（低密度パリティ検査符号）

- R. Rodrigues et al., “High Availability in DHTs: Erasure Coding vs. Replication”, IPTPS’05.



DHTで何が起きているのか説明できるか

- システムを再構成できるなら YES
- 再構成できないなら NO

- Debug level
- Dtrace (Solaris), Systemtap (Linux)
- Oprofile

まとめ

- DHT/KVSが独り歩きしているようですが…
  - Rendezvous
  - Location
  - Routing

  } これらすべてで工夫の余地あり

- 産業界からのフィードバックなければ研究の余地なし
  - 応答時間の要件
  - 可用性の要件
  - 一貫性の要件
  - データ量の要件 その他リソース制約
  - Design space があまりにも広い！

- ベンチマークを設定すれば改善の余地あり

  → クラウドコン 2010