オーバレイ構築ツールキット Overlay Weaver

首藤一幸† 田中良夫 関口智嗣

産業技術総合研究所 （AIST）
グリッド研究センター
† 現在、ウタゴエ（株）

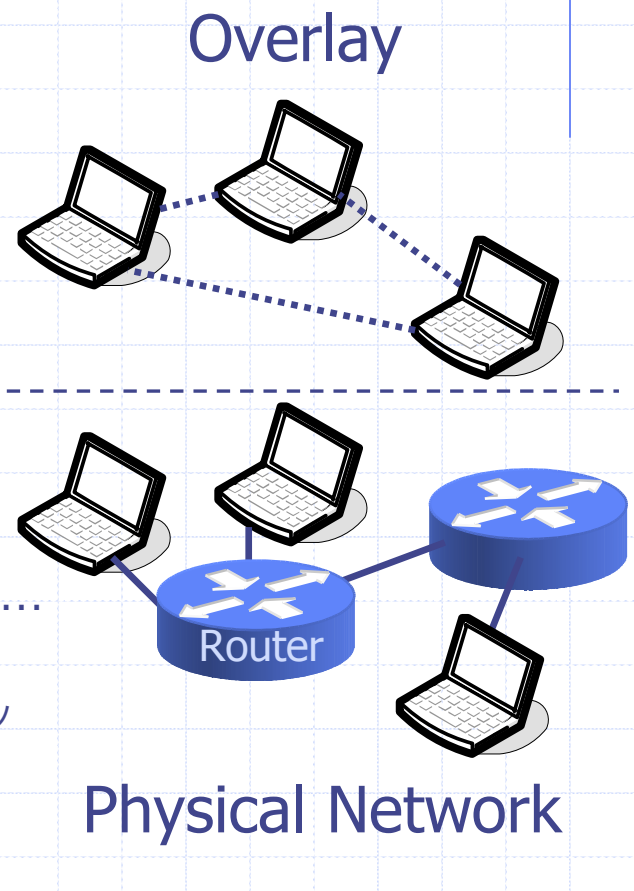
Overlay Weaver

http://overlayweaver.sf.net/

AIST

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、ネットワークオーバレイ or オーバレイネットワークと呼ばれる。



Overlay Weaver

Unstructured と Structured

- 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。
- 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
  - 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。

Overlay Weaver

Structured overlayのルーティングアルゴリズム：Accordion, Broose, CAN, Chord, D2B, DKS, GISP, Kademlia, Kelips, Koorde, ORDI, Pastry, Symphony, Tapestry, …

オーバレイ研究の問題

- アルゴリズム設計・研究から応用までの距離

設計・研究手法	*問題*
アルゴリズムの シミュレーション	応用のための実装は、別途、行う必要がある。
分散環境の エミュレーション	大規模実験が困難 → 設計・研究の手段として問題 例: PC 50台でようやく1,000ノードをエミュレート

- アルゴリズム実装の労力
  - コードは数千ステップになる。
  - 分散システムゆえ、デバッグが困難。
- アルゴリズム間の公正な比較の難しさ
  - シミュレーションの手段はある (e.g. p2psim)。
  - 実環境向け実装の良さは、アルゴリズム以外の部分に大きく依存する。

Overlay Weaver

オーバレイ構築ツールキット

Overlay Weaver

- Structured オーバレイのライブラリ
  - アルゴリズムその他を差し替え可能。
- 分散環境エミュレータ
- 周辺ツール
  - エミュレーションシナリオ生成器
  - 可視化ツール

Overlay Weaver

オーバレイ構築ツールキット

Overlay Weaver

- アルゴリズム設計・研究から応用までの距離を縮める。
  - １度の実装で、計算機１台上での大規模エミュレーションと実環境での動作が可能。
    - 30万ノードのエミュレーション、実機約200台での動作試験を行った。
  - 多ノードのエミュレーションで設計・実装・検証を進め、その実装を、そのまま応用。

Overlay Weaver

オーバレイ構築ツールキット

Overlay Weaver

- アルゴリズム実装の労力を減らす。
  - よく知られたアルゴリズム５種を、それぞれたかだか数百ステップで実装できた。
    - Chord, Kademlia, Koorde, Pastry, Tapestry
      - 加えて、Chord の亜種 ×２
    - Chord 619ステップ, Pastry 872ステップ (Java)
  - エミュレーションでの、多ノードの動作確認・デバッグ → 迅速な開発・品質向上
  - 関連研究　- Overlay Weaver とは異なるアプローチ
    - MACEDON: C, C++ に似た独自のオーバレイ記述言語を導入
    - P2: Prolog 似の宣言的オーバレイ記述言語 OverLog を導入

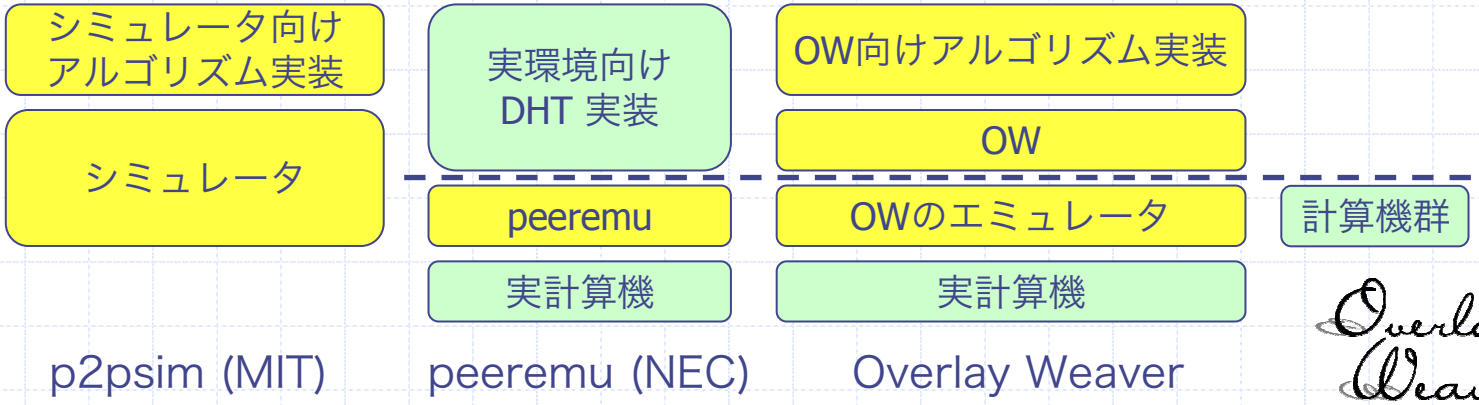
Overlay Weaver

オーバレイ構築ツールキット

Overlay Weaver

- アルゴリズム間の公正な比較を可能にする。
  - アルゴリズムだけを差し替えて動作させることができる。通信その他、アルゴリズム以外は同一の実装。
    → 公正に、複数のアルゴリズムを比較できる。
    - cf. 単一のアルゴリズムを実装しているライブラリ: Bamboo, Tapestry, FreePastry, Khashmir, SharkyPy, OPeN, ...
  - エミュレーション、実環境上、双方で可能。

アルゴリズム・実装間比較へのアプローチ



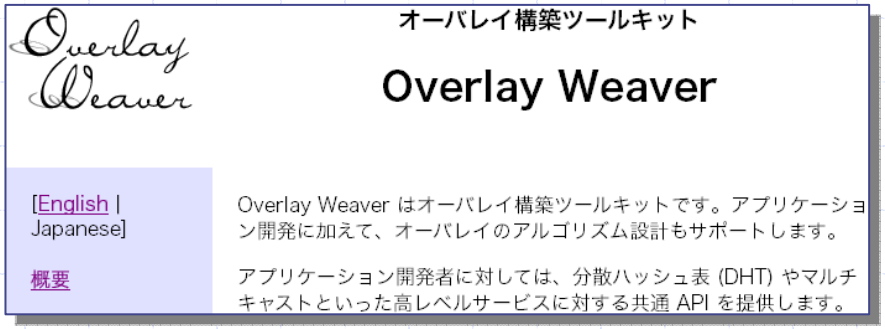
Overlay Weaver

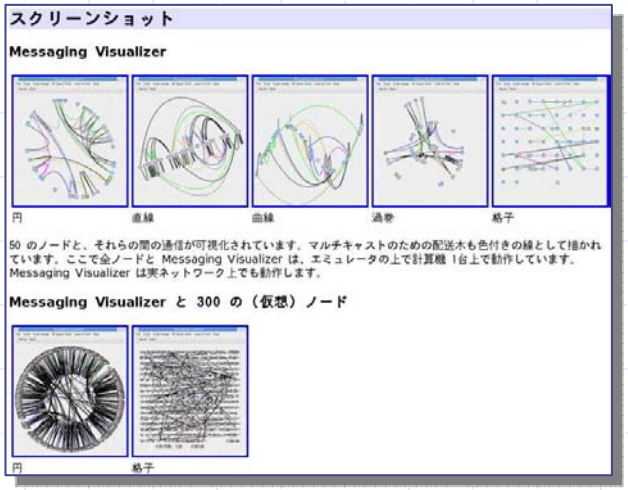
オープンソースソフトウェアとしての Overlay Weaver

- http://overlayweaver.sf.net/ (SourceForge)
  - 2006年１月17日、リリース。
  - Apache License 2.0

- 状況　(2009/10/14 12時)
  - ダウンロード 15,512件。
  - メーリングリスト登録数
    - 英語 91名, 日本語 54名
  - Mixi コミュニティのメンバ数 161名

- 想定ユーザ
  - アルゴリズム設計者 だけでなく
  - アプリケーション開発者
    - 例: RDF文書のP2Pストレージ by 的野 (産総研)
  - 卒論・修論での活用、大歓迎





ウェブサイト

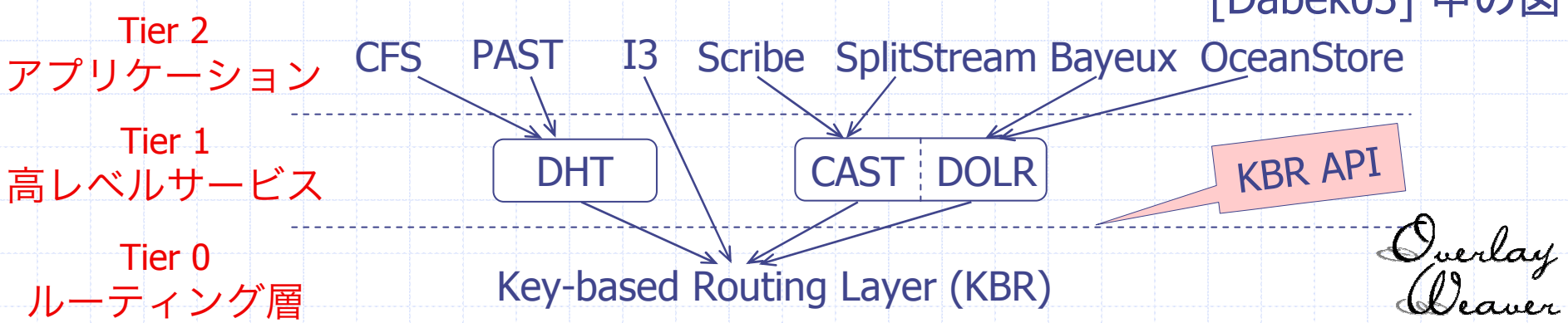
Overlay Weaver

アルゴリズム実装の容易化と差し替え可能化

従来研究：
ルーティングと高レベルサービスの分離

- [Dabek03] で提案された。
  - Frank Dabek et al., “Towards a common API for Structured Peer-to-Peer Overlays”, Proc. IPTPS’03, 2003.
- 分散ハッシュ表 (DHT) 等の高レベルサービスが共通して行うルーティング処理を、key-based routing (KBR) として抽象化した。
  - 宛先 ID に数値的に近い ID を持つノードに辿り着く。
- アプリケーション (Tier 2) や高レベルサービス (Tier 1) を、ルーティングアルゴリズム非依存にできる。
  - ルーティングアルゴリズム： Chord, Pastry, …

[Dabek03] 中の図



Overlay Weaver

ルーティング層の分解

- それでも、
  ルーティングアルゴリズムの実装は煩雑。
  - ノード間の通信や遠隔呼び出しから実装する必要あり。
    - 例: p2psim の Chord 実装は 2,835 ステップにもなっている。
- 実装が容易であることが望ましい。
  - 自分が設計するアルゴリズムだけでなく、比較対象となる他のアルゴリズム群も…

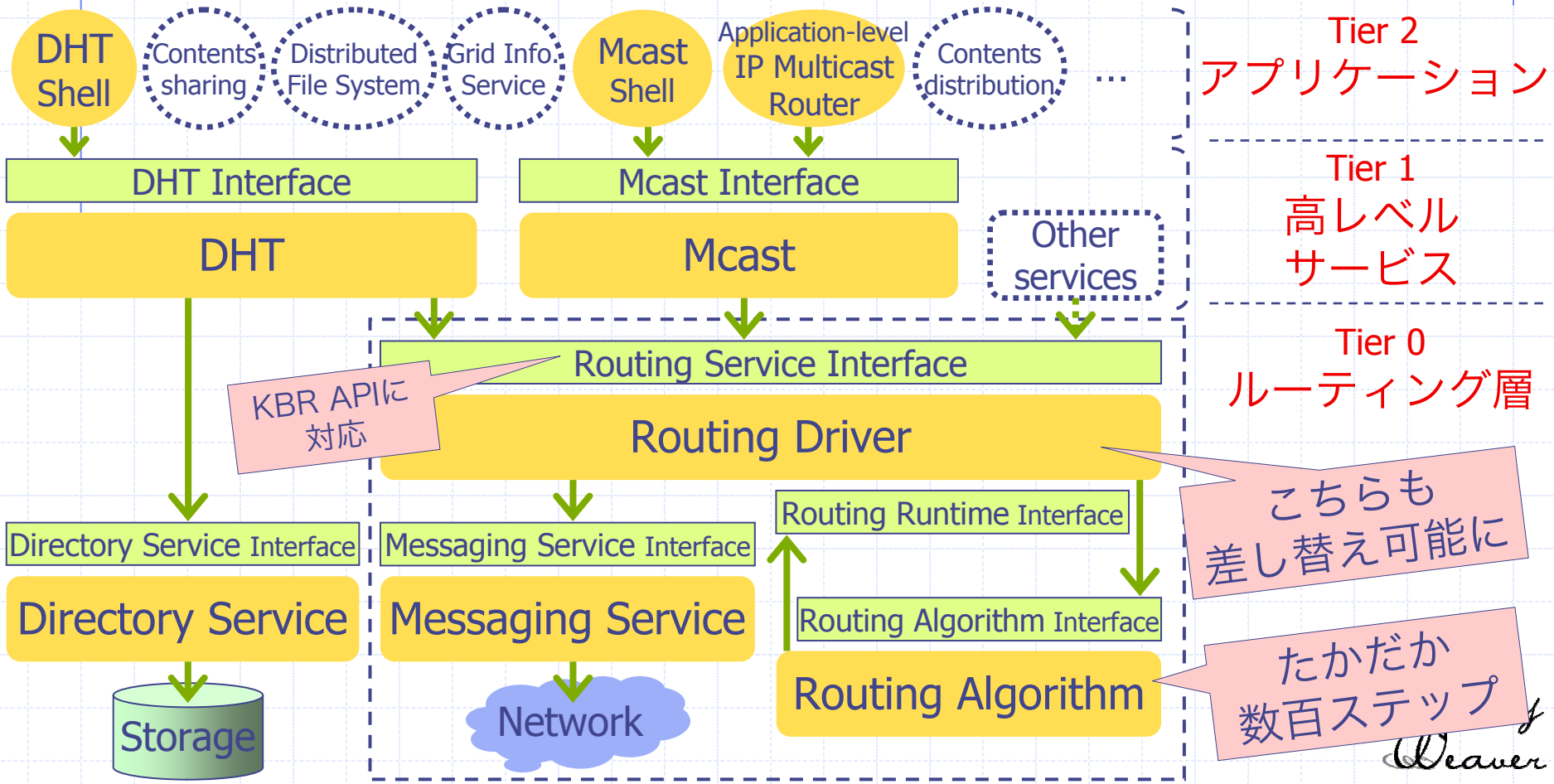
そこで

- ルーティング層から、
  各種アルゴリズムに共通する処理をくくり出した。
  - いわば、KBR 層のさらなる分解。

Overlay Weaver

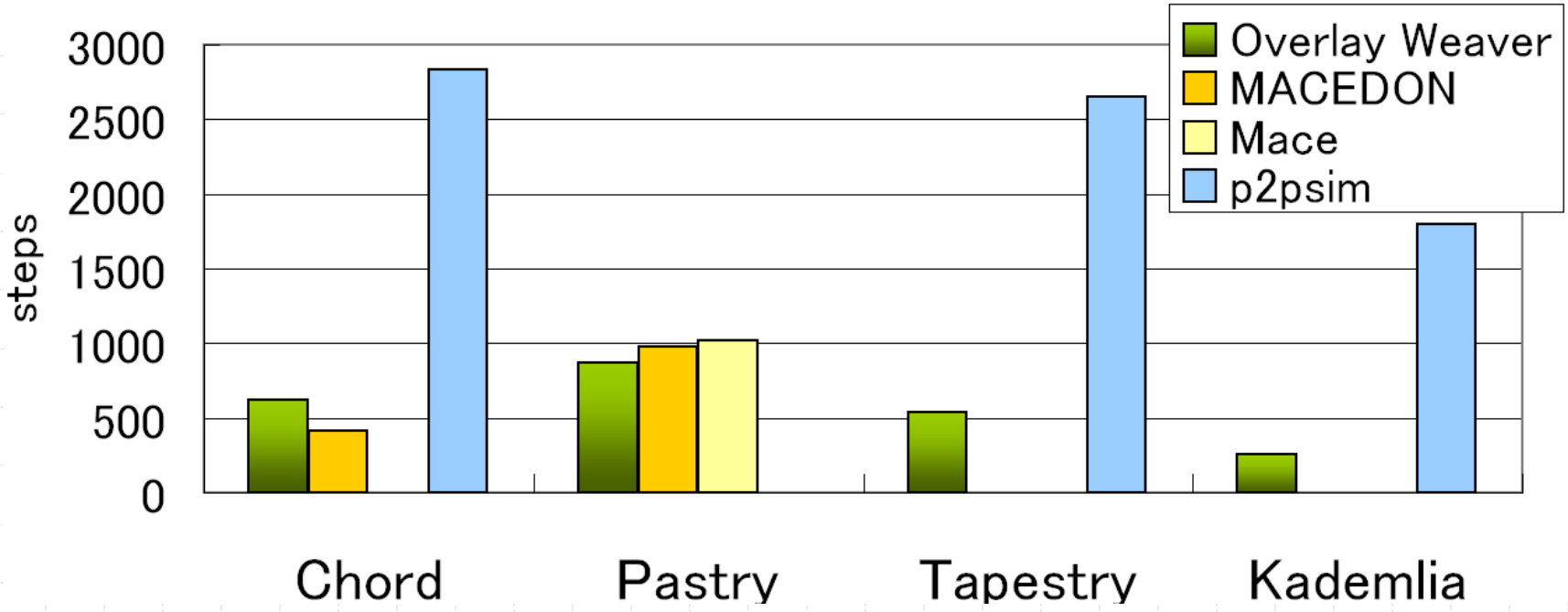
ルーティング層の分解

- アルゴリズムの実装が容易になった。
- ルーティング処理の実装が選択可能になった：IterativeとRecursive



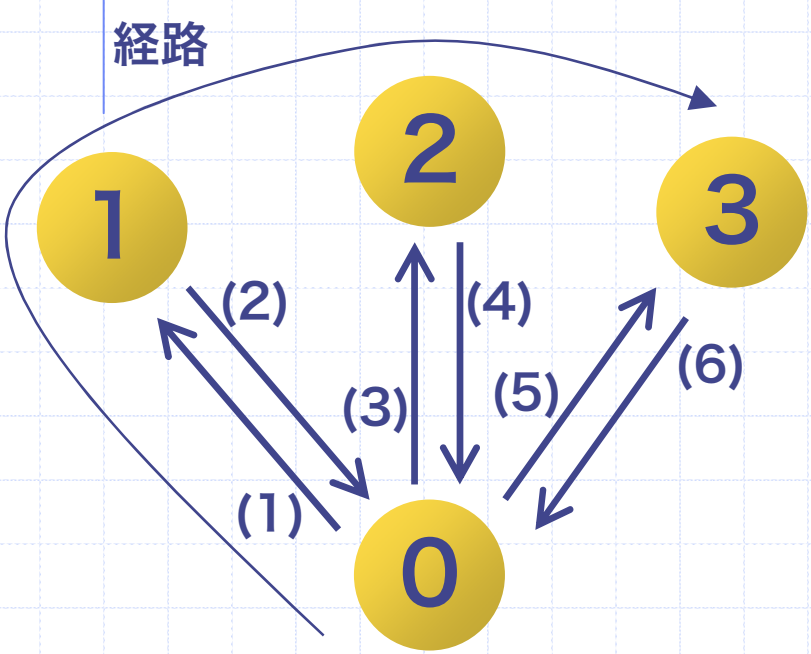
アルゴリズム実装の容易さ

- 各種のルーティングアルゴリズムを、
  たかだか数百ステップ (Java) で実装できた。
- 比較対象
  - MACEDON, Mace: 専用言語でアルゴリズムを記述し、変換して C++ コードを得る。
  - p2psim: シミュレータ。遠隔メソッド呼び出しから記述されている⇒ コード量多め

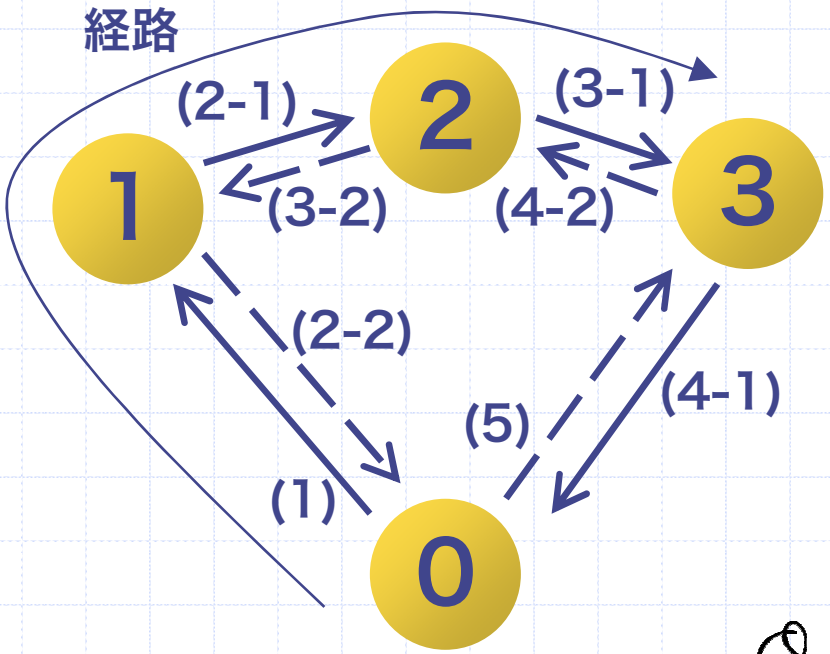


選択可能なルーティング処理部

- Iterative / Recursive ルーティングを、
  各アルゴリズムと組み合わせることが可能になった。
  - ルーティング層を分解したことの成果。
  - 両様式にはそれぞれ強み / 弱みがある。



Iterativeルーティング



Recursiveルーティング

Overlay Weaver

アルゴリズム側 インタフェース

- ルーティングアルゴリズムの実装が、ルーティング処理部に対して提供するインタフェース
  - closestNodes (ID target, int maxNumber)
    - 与えられた ID に最も近い ID を持つノード(群)を返す。
    - この結果を元に、ルーティング処理部は次ホップを決める。
  - adjustRoot (ID target)
    - ルーティングの最終段階で、ターゲットIDに最も近いIDを持つノードに対して呼び出される。
    - ルーティングの宛先 (ゴール) を返す。
    - Chord で必要。
  - 他 6メソッド

Overlay Weaver

ツールキットの構成

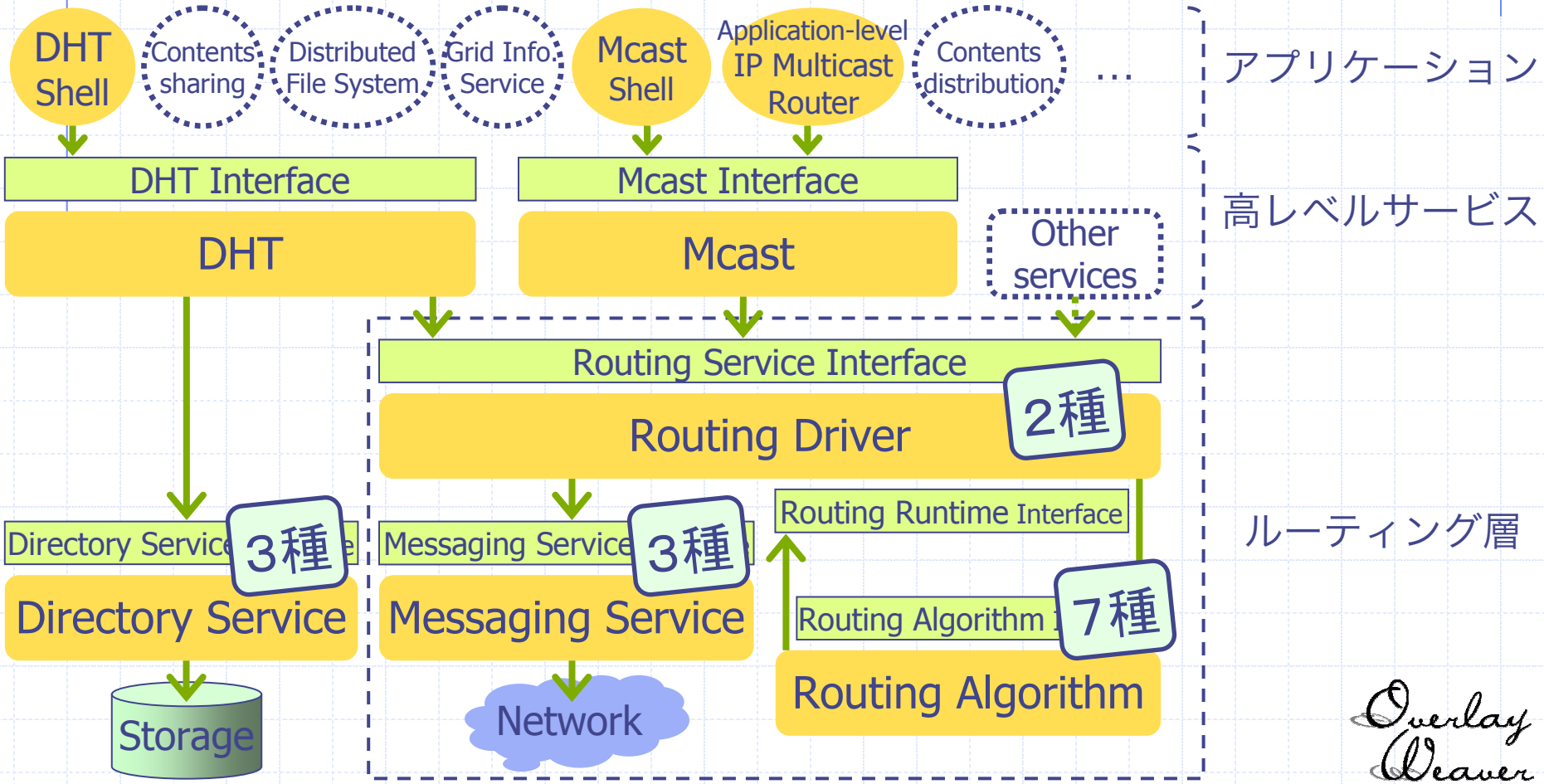
構成

- 実行系（runtime）
  - 後述
- ツール群
  - 分散環境エミュレータ
  - シナリオ生成器
  - メッセージカウンタ
  - Visualizer (可視化ツール)

Overlay Weaver

実行系の構成

- 各コンポーネントに複数の実装があり、実行時に選択が可能。



Overlay Weaver

下位コンポーネント

- ルーティング処理部 (driver)
  - アルゴリズムの実装に問い合わせを行いながら、ルーティングを行う。
  - iterative / recursive の２種類を提供。
- ルーティング アルゴリズム
  - ルーティング処理部に、次ホップを決めるために必要な情報を提供する。
  - Chord, Kademlia, Koorde, Pastry, Tapestry, Chordの亜種×２ の実装を提供。
- メッセージング サービス (通信部)
  - オブジェクトの送受信を受け持つ。one-way / round-trip 通信をサポート。
  - UDP, TCP, エミュレータ用の３種類を提供。
- ディレクトリ サービス
  - ローカルなハッシュ表
  - オンメモリ実装, オンメモリだが定期的に二次記憶装置にチェックポインティングを行う実装, Berkeley DB実装の３種類を提供

Overlay Weaver

高レベルサービス

- DHT
  - 分散ハッシュ表: Distributed Hashtable
    key-valueペアの put / get が可能
- Mcast
  - マルチキャスト / グループ管理
    - グループIDを指定してのグループへの参加 / 離脱
    - グループ内ノードへのマルチキャスト
  - グループごとに配送木が構築される。
    - Scribe と同様の方法

Overlay Weaver

サンプル アプリケーション

- DHT シェル
  - DHTサービスをコマンドで操作できる。
- Mcast シェル
- ↑ エミュレータと組み合わせて使うことで、アルゴリズムの動作試験や比較を行うことができる。
- IPマルチキャストルータ

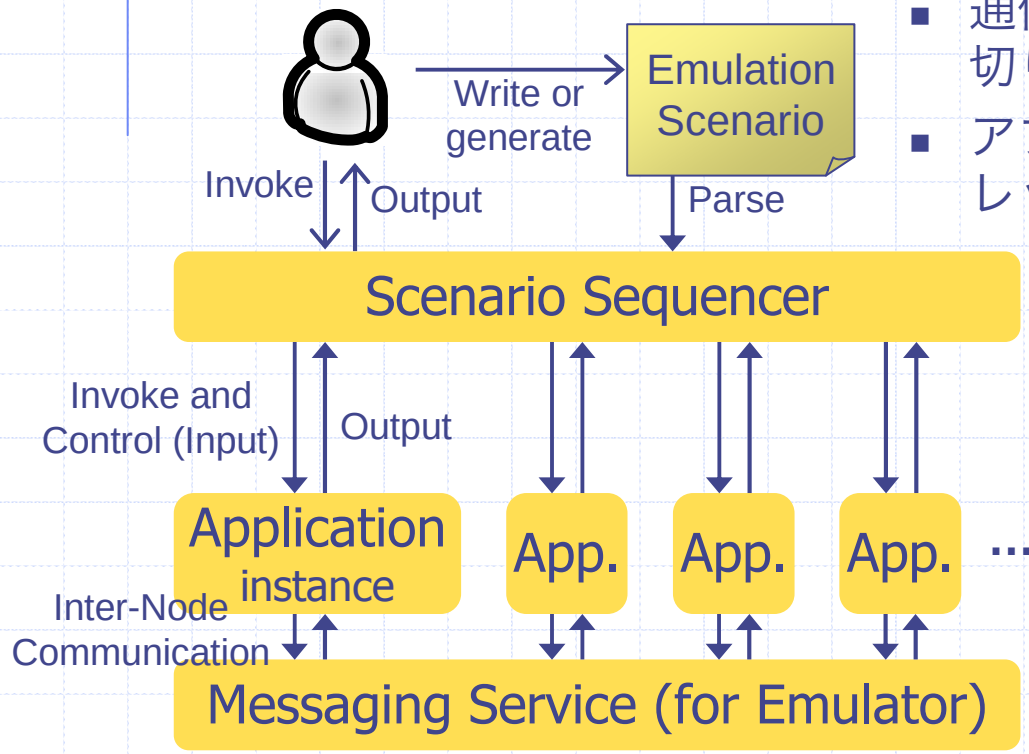
```
% telnet hoge.hpcc.jp 10000
…
Ready.
help
init <host> [<port>]
get <key> [<key> …]
put <key> <value> [<value> …]
remove|delete <secret> <key>
[<value> …]
        ...
Ready.
put aKey aValue
Ready.
get aKey
key:   a5d08205fa1e881fda8334d1f79317fa
value: aValue 592
Ready.
```

DHT シェルの使用例

Overlay Weaver

分散環境エミュレータ

- シナリオを読み込み、それに従い、複数のアプリケーションを起動 / 制御する。
  - アプリだけでなく、アルゴリズムの試験、デバッグに極めて有用。
  - 通信部は、強制的にエミュレータ用実装に切り替えられる。
  - アプリケーションインスタンスごとにスレッドを生成する。

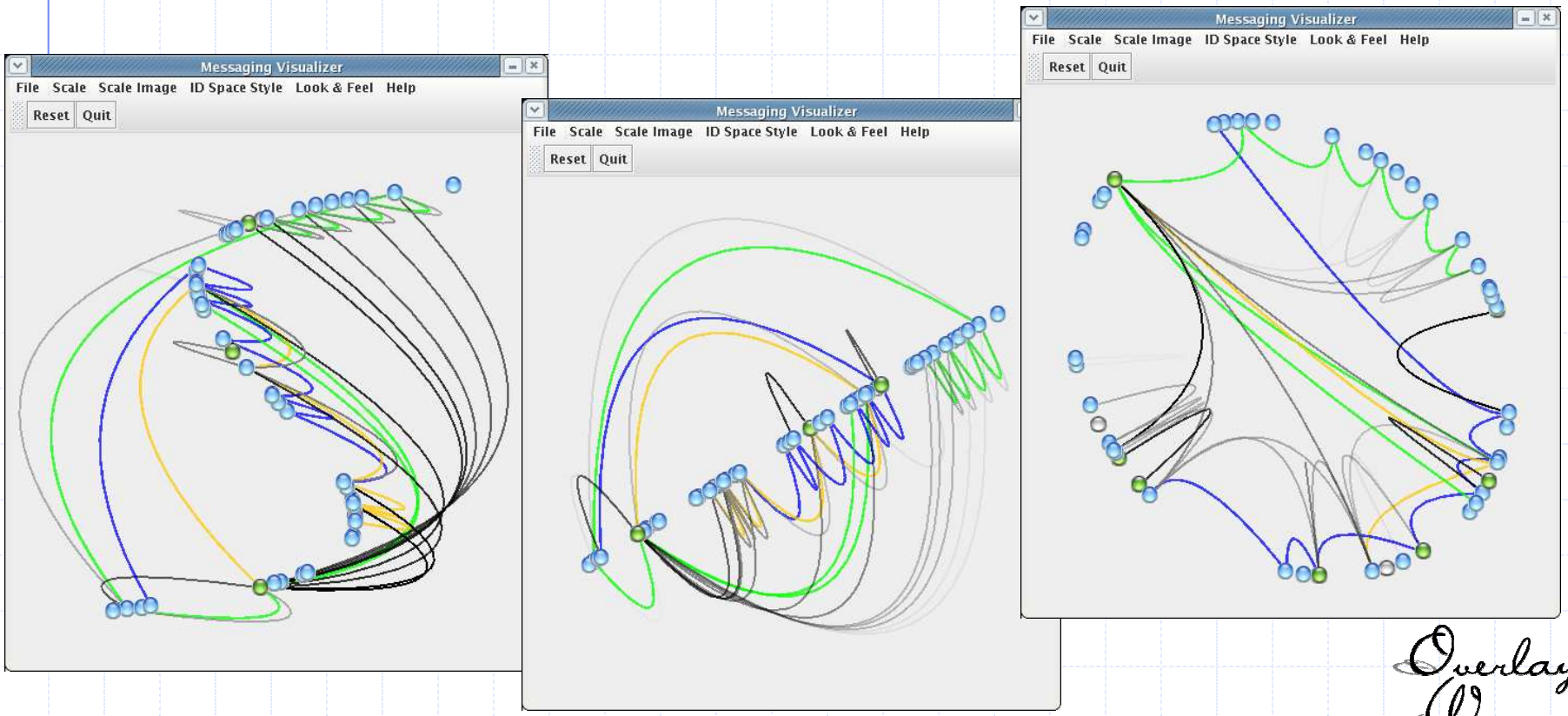



```
# invoke 1000 instances
class ow.tool.dhtshell.Main
arg -p 10000
schedule 0 invoke
arg ptp00.hpcc.jp
schedule 500,10,999 invoke
# put & get
schedule 5000 control 123 put foo bar
schedule 6000 control 234 get foo
```

シナリオ

Visualizer: 可視化ツール

- ノード、メッセージ、マルチキャストの配送木を、実行中に可視化。
- このツールに対する通信の報告自体もメッセージングサービスを用いて行われる。
  - ⇒ エミュレータ / 実ネットワーク双方で使える。



Overlay Weaver

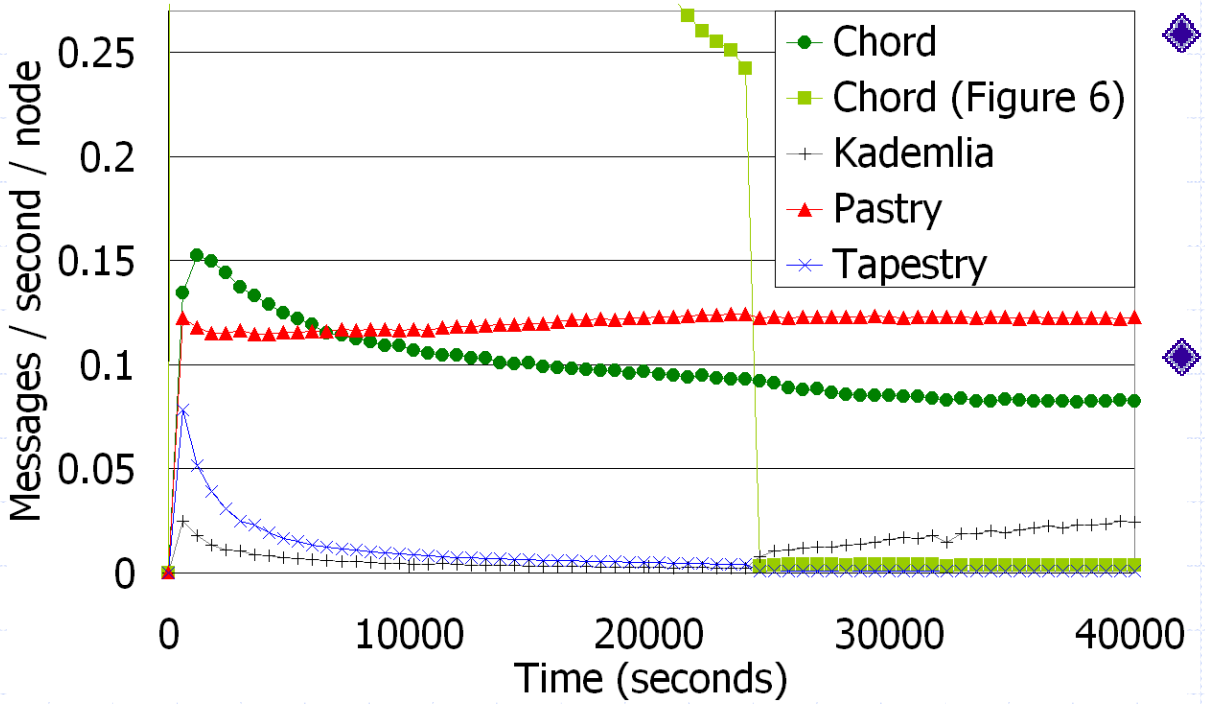
評価

評価項目 / 方法

- 目標
  - アルゴリズム設計から応用までの距離短縮
- 評価項目
  - アルゴリズムの実装が容易であること
  - 多ノード（エミュレーション）でアルゴリズム実装の動作を確認できること
  - アルゴリズム間の比較を公正に行えること
  - 実環境で動作すること
- 方法
  - コード量（前掲）
  - 4,000 ノードのエミュレーション
  - 実機約 200台での動作

4,000ノードのエミュレーション

- 単位時間あたりのメッセージ数を計測
  - シナリオ：put 4,000回, get 4,000回
- 各アルゴリズムの性質は論じない。
  - 動作確認、比較が可能であることを示すことが目的。
  - 結果はパラメータ (例: 処理の時間間隔) にも依存する。

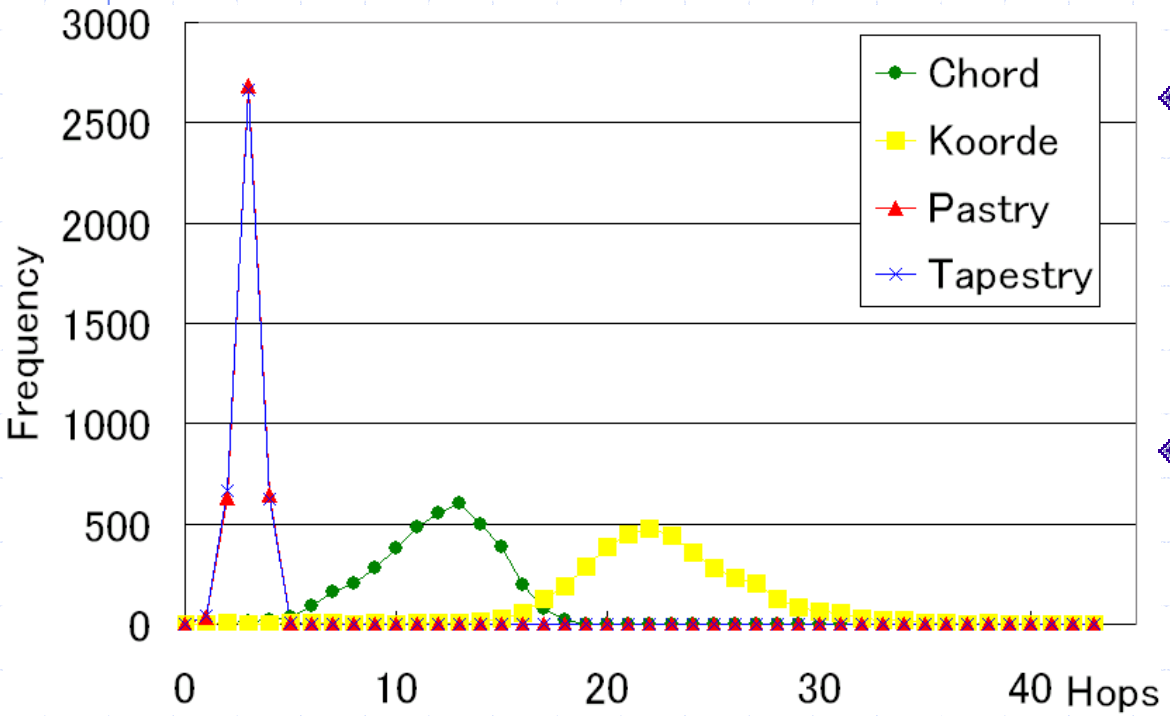


- 実験環境 – ふつ〜の PC
  - 3.4 GHz Pentium 4
  - メモリ 1 GB
  - Linux 2.6.15
  - Java 2 SE 5.0 Update 6
- 注
  - 「Chord (Figure 6)」は、オーバレイ参加時に経路表を完成させてしまう版。
  - Iterative ルーティング

Overlay Weaver

4,000ノードのエミュレーション

- ルーティング時のホップ数とその頻度
  - シナリオ：put 4,000回, get 4,000回

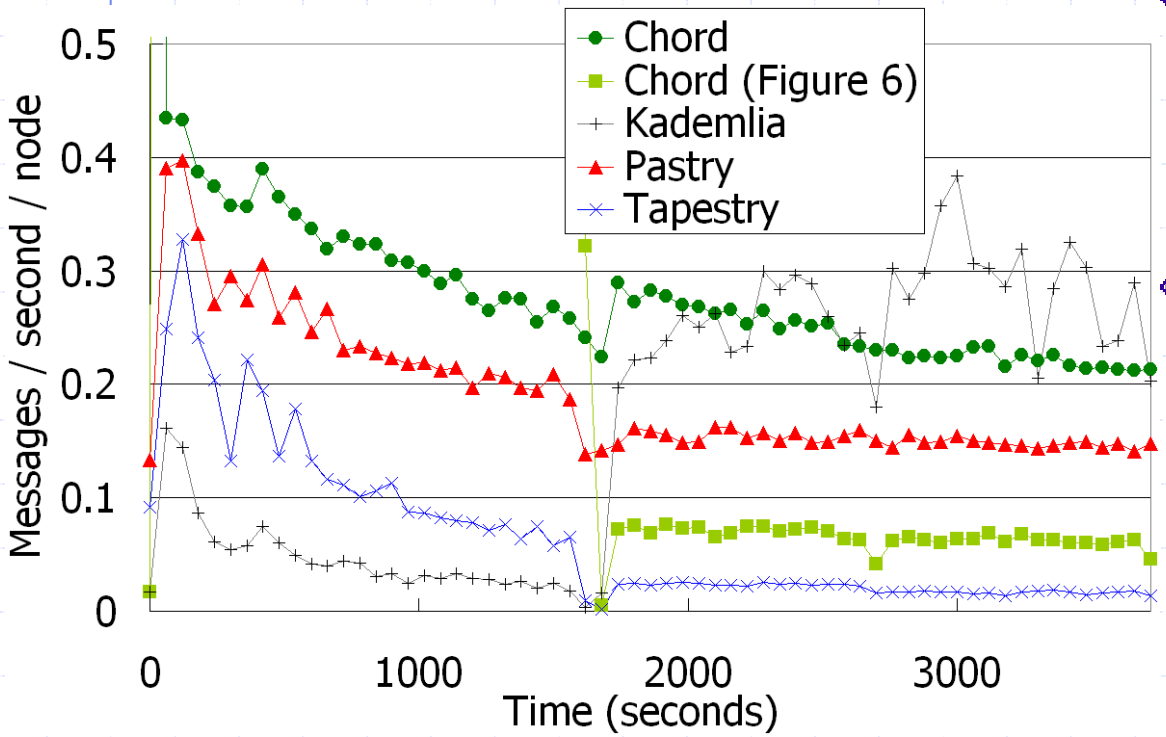


- 実験環境 – ふつ～の PC
  - 1.7 GHz Pentium M
  - メモリ 1 GB
  - Linux 2.6.15
  - Java 2 SE 5.0 Update 6
- 注
  - この図は論文にはありません。

Overlay Weaver

実機約 200台での動作

- 単位時間あたりのメッセージ数を計測
  - シナリオ：put 500回, get 500回



- 実験環境 – AISTスーパークラスタの一部 197台
  - 3.06 GHz Xeon
  - Linux 2.4.24
  - Java 2 SE 5.0 Update 6
- 注
  - エミュレーションより値の揺れが大きいのは、計測間隔が異なるため。
    - エミュ 10分, 実機 1分
  - アルゴリズムごとの傾向はノード数にも依存する。
  - Iterative ルーティング

Overlay Weaver

まとめ

- オーバレイのアルゴリズム設計から応用までの距離を短縮するツールキットを開発した。
  - 研究成果を直接応用に結びつける。
- ルーティング層からルーティング共通処理をくくり出し、アルゴリズムの実装を容易にした。
  - たかだか数百ステップで実装可能。

Overlay Weaver

今後

- 課題 & 今後
  - 他のアルゴリズムを実装し、アルゴリズム実装インタフェースの妥当性をより確かなものとする。
  - より大規模な実験を行う。
    - ボトルネック：スレッド数, メモリ容量, 仮想メモリ空間 (スタックサイズ), 処理性能、…
  - Unstructured オーバレイの実装をサポートする方法を検討する。
  - エミュレーションに加えてシミュレーションも可能となるようなアルゴリズム記述方法を検討する。
  - テストベッドの構築？
    - cf. OpenDHT on PlanetLab
  - アプリケーションの開発？
  - アルゴリズムの比較を行う？

Overlay Weaver