第XVII部

オーバーレイネットワークによる統合分散環境

第 17 部
オーバーレイネットワークによる統合分散環境

第 1 章　はじめに

1.1 オーバーレイネットワークとは何か

1.1.1 IP とオーバーレイネットワーク

オーバーレイネットワークは、既存のリンクを用いて、その上位層における目的に応じて仮想的なリンクを形成し、構成するネットワークと定義できる。このワーキンググループでは、 主に IP の上位層で仮想化を行う例を扱う。

オーバーレイネットワークは特別なものではなく、インターネットを利用して何か目的を達成する際に、一般に用いられていると考えることができる。例えば、SMTP や POP/IMAP によるメール配送システムは、MTA(Message Transfer Agent) 間を結ぶメッシュと、メールサーバの周囲にクライアントが配置されたスター型のネットワークから成るオーバーレイネットワークとして捉えることができる。IRC やインスタントメッセージング等のポピュラーなアプリケーションについても同様と考えられるし、WWW は、ハイパーリンクにより構成されたネットワークのノードに、ブラウザが自由に接続するアプリケーションと考えることができる。

IP はグローバルな空間における到達可能性を提供し、後述するランデブーのための基盤を準備するが、それのみでは人間の目的は達成されない。IP が実現する到達可能性の上にオーバーレイネットワークを形成することで、初めて人間の目的に応じたアプリケーションの構築が可能となると言える。

人間の役に立つアプリケーションの開発のためには、実験的にオーバーレイネットワークを形成することにより素早くインターネット上に展開し、評価した後にやり直すといった行為を繰り返す必要がある。そのことを効率的に行うためには、個別にオーバーレイネットワークの形成のための努力を再投入するのではなく、様々な形態のオーバーレイネットワークの形成を支援し、容易にするツールキットの存在が望まれる。

1.1.2 オーバーレイネットワークの形態

上のような理解に基づくと、1980 年代後半から 1990 年代初頭にかけて盛んに研究された分散オペレーティングシステムは、今、改めてオーバーレイネットワークの形成を支援するプログラミング環境として捉え直すことができる。

そのような先行研究の 1 つ、ISIS Toolkit[31] の作者である Birman らは、1991 年に書かれた論文 [32] で、ISIS Toolkit がサポートするプロセスグループの 4 つの形態 (表 1.1) を紹介している。その上で彼らは、経験上、多くのアプリケーションではこれらのグループを組み合わせてシステムに必要な機能を実現していると報告している。

これは、人間の活動に応じて様々なネットワークの形態があること、そしてアプリケーションの開発においては、それらのグループを形成するためのプリミティブを用意しておくことが役立つということを裏付けていると言える。

表 1.1 に示した分類は、我々が今後、様々なオーバーレイネットワークを考えていく上での雛形として用いることができる。

1.2 IDEON ワーキンググループ設立の目的

IDEON(Integrated Distributed Environment with Overlay Network) とは、オーバーレイネットワークによる統合分散環境を意味する。このうち、統合分散環境 (Integrated Distributed Environment) は WIDE(Widely Integrated Distributed Environment) のサブセットである。IDEON という名称は、広域にこのような環境を構築するという WIDE プロジェクトの目的に対し、仮に形成するオーバーレイネットワークを通してアプローチするという実験的な精神を表している。

IDEON は次の目的をもって 2002 年に設立された。

1. 既存のオーバーレイネットワークの研究をサーベイし、系統立てる。
2. 実験的なオーバーレイネットワークの形成を通し、統合分散環境の実現のためのインフラスト

表 1.1.　ISIS Toolkit におけるプロセスグループ

番号	名称	定義と特徴
1	ピアグループ (Peer Groups)	対等なプロセスが協調して目的を達成するグループ。複製したデータを管理したり、タスクを分割したり、互いの状況をモニタする場合に用いられる。
2	クライアント/サーバグループ (Client/Server Groups)	多数のクライアントに対し、ピアグループを形成するサーバ群がサービスを提供するグループ。 要求/応答によるコミュニケーションを特徴とする。
3	拡散グループ (Diffusion Groups)	サーバが、全サーバおよびクライアントにメッセージをマルチキャストするような、クライアント/サーバグループの特殊例。多地点に情報を配信する場合に用いられる。
4	階層グループ (Hierarchical Groups)	複数のグループを木構造にまとめたグループ。大規模なグループコミュニケーションを実現する。

ラクチャを提案する。

目的 1 は学術的な意義を持つ。前節までの理解から、オーバーレイネットワークは人間がインターネットを利用する上で必須であり、その概念を整理し、後続する研究が活用できるようにすることは、学問の発展上、重要であると考えられる。この報告書ではその試案を作成した。

目的 2 はより実用的である。統合分散環境を実現するという目的の達成のためには、オーバーレイネットワークの利用の他にもアプローチが考えられる。特に、統合分散環境の部品となるマルチキャスト、エニキャスト、モビリティ等の実現については、IP 層に位置づけるという考え方があり、WIDE でも盛んに研究が行われている。こうしたアプローチの問題点は、インターネットにおける展開が実際には容易ではないという点である。既存のインフラストラクチャに手を加えるという方法は、そのインフラストラクチャのユーザが増えるに従って困難となる。既に人類全体が情報を共有するための基盤として成立しつつあるインターネットでは、その基幹部分への機能の追加や変更は困難を極めると言ってよい。

対して、オーバーレイネットワークは IP のアプリケーションであるので、インターネットのインフラストラクチャ自体を変更することなく、自由に試すことができる。この特徴を最大限に活かし、新しい才能が新しいアイデアを試すための手法として活用されることが望まれるし、そのためには、マルチキャスト、エニキャスト、モビリティ等を含め、様々な可能性を実際に実装することにより、広域な統合分散環境の実現に少しでも近づくと共に、このアプローチ自体の実効性についても実証していく必要がある。

1.3 peer-to-peer との関係

IDEON ワーキンググループの活動は、所謂 peer-to-peer とも関係が深い。

peer-to-peer は、文字通りの意味ではピアからピアへ (の通信) を意味する。ピアは対等な存在を意味するので、このことは表 1.1 に示したピアグループにおける通信を一般に意味すると考えられる。

表 1.1 では、ピアグループはすべてのプロセスグループの基礎となっている。このことを人間のネットワークに置き換えると、対等な仲間同士での協調作業が人間の活動の基礎となっていることを示している。インターネットを利用する活動を考えると、このことが重要な意味を持つことが分かる。例えば電子メールの交換は、本質的には立場が対等であるし、チャットにも同じことが言える。人間の活動にとっては、オーバーレイネットワークがピアグループを形成することが重要である場面が多い。

工学的に見ても、ピアグループは重要である:

耐故障性の実現:

サーバが故障しても、その事実をクライアントから隠蔽し、変わりないサービスを提供するためには、サーバの複製が存在し、その内部状態を何らかの方法により共有し、交換可能な状態にあることが必要となる。

規模拡張性の実現:

分散システムが、規模に依らずにその性質を保つためには、1 台のサーバに負荷が集中するのでは

なく、(できるだけ経路を共有しない場所に) サーバの複製が存在し、同等のサービスを提供できることが必要となる。

可塑性の実現:

分散システムの一部が破壊されても、残った部分が役割を代替することにより、その性質を保つためには、破壊されたサブシステムの役割が複製できることが必要となる。また、そのことが確実に行われるためには、システム内に固定された指揮系統があるのではなく、どのノードも自律的にネットワークの再構成に参加し、システムを回復させることができなければならない。

これら耐故障性、規模拡張性、可塑性は、動的に変化する様々な条件に依らずにシステムがその性質を維持できることと一般化できる。その実現のためには、システムが冗長な構成要素から成ること、すなわち、互いに交換可能なノードから成り、それらがネットワークで結ばれていて、機能や内部状態が複製できることが前提となる。

以上から、ピアグループを形成するオーバーレイネットワークは特に重要であり、第 4 章で紹介するプラットフォーム JXTA でも、その理由により Peer Group という抽象を基礎としていると考えることができる。

1.4 ランデブー

IDEON ワーキンググループの活動の基礎となる問題意識のひとつに、ランデブーという問題がある。特に、「自由で創造的なランデブー」が今のインターネットに求められているという問題意識を共有し、既存の研究をランデブーのためのオーバーレイネットワークという軸を用いて捉えなおす。

本節では、このようなランデブーにおける問題意識を述べる。ランデブーとは、通信を行う相手との「待ち合わせ」を示す。ここでは、インターネットに接続されたノードが、通信を行うノードを特定するための、通信の両端で共有する知識とサービスを指すものとする。

1.4.1 自由で創造的なランデブー

End-to-End で責任を持った通信を可能にするという原則 (End-to-End principle) が、インターネットアーキテクチャのデザイン原則の一つ [141] である。この原則を守る事で、各ノードが自分の要求に併せて自由にプロトコルを設計するというイノベーションが保証し、同時に、不安定なネットワークから通信固有の状態を分離する事による耐障害性を確保する事ができる。

自由に相互に通信可能なネットワークとしてインターネットは発展してきた。しかし、その発展に伴い、ランデブーは困難になってきている。

従来は DNS を利用したランデブーで十分だった。一方近年は、

1. 様々なランデブーを DNS に依存するための過負荷や文脈の破綻
2. DNS のとり扱える範囲外のランデブーに対する要求

の二つの問題が顕在化している。

この問題を解決するために、従来のランデブーを補完する新しいランデブーの枠組を考える必要がある。新しいランデブーは、ランデブーに利用する空間に対する制約が無い、つまり自由なランデブーである事が望ましい。一方、ランデブーとして、必要なノードと資源を効率良く、創造的に組み合わせる事が可能な、創造的なランデブーである事が望ましいのは言うまでも無い。

1.4.2 オーバーレイネットワークによるランデブー

自由かつ創造的なランデブーを実現するために、DNS に代わる新しいシステムをインターネット全体に導入する、という考えは、End-to-End principle に背く恐れがある。しかしながら、何かの必要がある度にランデブーを独自に開発し、展開していたのでは、コストも高くつくし、それぞれのランデブー空間の中に含まれる情報の量が少なく、効率が悪い。

オープンなオーバーレイネットワークを導入し、その上でランデブー機構を実現する事により、このような問題を解決でき、「自由で創造的なランデブー」に一歩近づく。似た要求のランデブーはひとつのオーバーレイネットワークまとめる事ができ、かつ、オーバーレイネットワークは複数が共存可能なので、新しい機能が必要なイノベーションがあれば新しいオーバーレイネットワークを導入すれば良い。

1.5 本報告の構成

この報告の残りは次のように構成される:

第 2 章 オーバーレイネットワークの基礎技術

分散ハッシュテーブル方式を中心に、既存のオー

表 2.1.　分散ハッシュテーブルの特徴比較

Propertiles	Tapestry	Chord	CAN	Pastry
Parameter	Base b	None	Dimen d	Base b
Logical Path Length	$\log_b N$	$\log_2 N$	$O(d^* N^{1/d})$	$\log_b N$
Neighbor-state	$bLog_b N$	$\log_2 N$	$O(d)$	$bLog_b N + O(b)$
Messages to insert	$O(\log_b^2 N)$	$O(\log_2^2 N)$	$O(d^* N^{1/d})$	$O(\log_b N)$
Mutability	App-dep	App-dep	Immut	unknown
Load-balancing	Good	Good	Good	Good

バーレイネットワークの基礎研究を整理して紹介する。

第 3 章 分散ハッシュテーブルの応用

分散ハッシュテーブル方式を用いるオーバーレイネットワークのうち、特に Chord に注目し、その応用技術を考察する。

第 4 章 その他の技術

匿名性と情報の秘匿性を追求した技術である Freenet と、プラットフォーム技術である JXTA を紹介する。

第 5 章 結論と IDEON 発動に向けて

これまでのサーベイ結果をまとめ、IDEON からの提言を行う。

第 2 章　オーバーレイネットワークの基礎技術

2.1 分散ハッシュテーブル

オーバーレイネットワークの基礎技術として分散ハッシュテーブル (Distributed Hash Table) が提案されており、その具体例として Chord[150], CAN[131], Pastry[135], Tapestry[169] がある。これらの特徴の比較を表 2.1 に示す。

分散ハッシュテーブルを用いたネットワークでは、検索キーワードをハッシュ関数を用いてキー値に変換し、ネットワークに送信する。ネットワークからキー値に対応するデータの取得を行うことが可能であり、これを参照 (Key/Value Lookup) と呼ぶ。この参照では、フレーズ検索のようなあいまい検索や前方一致といったことができず、キー値に対応するデータが一意に決定される。これが一般的に言う検索と参照の違いである。

参照で得られるデータは、検索結果であるコンテンツ自体の場合もあるが、IP アドレスのリストなどが格納され、コンテンツの場所を示すポインタとしても用いられる。ユーザは IP アドレスのリストを取得後、その IP アドレスへ直接接続しコンテンツを取得するという 2 回の処理を行わなければならない。

2.2 Chord

ネットワークの中でデータを持つノードを探すこと (ランデブー) は大きな課題である。完全な分散システムで検索する方法として Chord ネットワークを紹介する。Chord ネットワークは Ion Stoica ら [150] によって提案されたオーバーレイネットワークの 1 つである。

Chord の実装は、DHash レイヤと Lookup Service レイヤに分けられる (図 2.1)。Lookup Service レイヤでは、検索の基本である参照処理を行う。キー値を service, 参照値を location とすることでリソースの発見を行う。DHash レイヤでは参照によるリソースの発見だけでなく、データの転送や複製、キャッシュ機構などによる信頼性や性能の向上、各ノードの負荷の分散を行う。

2.2.1 参照処理

Chord の参照処理において、キー値とデータのペアを保持するノードは、ハッシュ関数を用いること

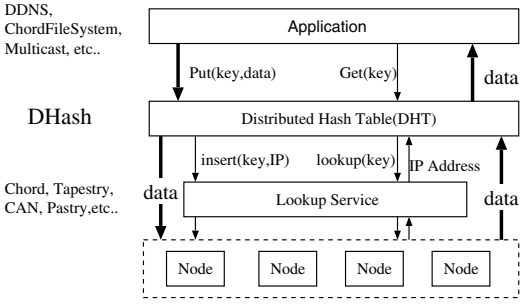

図 2.1.　DHash レイヤと Lookup Service レイヤ

により偏りなく分散され、ネットワーク内で一意に決定される。

検索者は検索キーワードをハッシュ化したキー値を Chord ネットワークに送信すると、キー値に対応したデータを保持するノードへ $O(\log N)$ でルーティングされる。そして、目的のデータを保持するノードが検索者に返信することにより、検索者はキー値に対応したデータを参照することが可能である。

2.2.2 Chord の特徴

Chord では参照やネットワークへの join/leave にサーバを必要とせず、各ノードが完全に分散して処理を行う。Gnutella 等の検索クエリをフラッディングするネットワークにおいて、検索可能ノード数 N の増加に伴い線形のパケットの増加が起こる。一方、Chord ではパケット数の増加は $O(\log N)$、ホップ数は $O(\log N)$ であり規模拡張性に優れている。また、FreeNeet や Gnutella において、検索時にデータが存在しないことを判断するには、検索の遅延を推測し判断しなければならない。Chord ではデータが存在しない場合であっても、明示的にデータが存在しないということを知ることが可能である。また、永続性に関して best effort 型であり、データの挿入や更新に保証がないという特徴を持つ。

ノード数 N の Chord ネットワークの性能を以下に示す。

- N 番目のノードが join/leave する際に、$O(1/N)$ のキー値とデータの移動が生じる
- ネットワークの維持には、他の $O(\log N)$ ノードの情報を持つ必要がある
- $O(\log N)$ のメッセージ数で参照が可能である
- join/leave する際のメッセージ数は $O(\log^2 N)$ である

2.2.3 Chord API

- insert(key, value)
  key/value をネットワークに挿入する
- lookup(key)
  Key に対応する Value を返す
- update(key, newval)
  key/newval をネットワークに挿入する
- join()
  Chord ネットワークに自分を追加する
- leave()
  Chord ネットワークから退出する

ChordAPI には削除操作は存在せず、作成者のみが update() を行うことでデータの書き換えを行っている。

2.2.4 ハッシュ関数の利用

各キー ID とノード ID は SHA-1 のハッシュ関数を用いて作成され、同じ ID 空間にマッピングされる (キー ID = SHA-1(キーワード), ノード ID = SHA-1(IP アドレス))。160 bit の ID 空間の場合 3.40×10^{38} の ID が存在することになり、コンテンツと IP アドレスをマッピングしても衝突しないことが前提とされる。もし衝突する場合には任意の数字とノード ID を加えた値 (anyID + SHA-1(ip)) が用いられる。Chord の ID 空間はリング状につながっており、ノード ID とキー ID がランダムに分散した状態となっている。ここでは説明しやすいように 7 bit(0–127) の ID 空間を用いて説明を行う。

Chord ネットワークにおいて、ID を持った各ノードは仮想的なリングを形成し、時計方向に接続を行っている。図 [2.2] では Node5→Node10→Node45→Node90→Node105→Node120→Node5 の様に接続されている。この接続はルータ同士の接続ではなく、ノード同士で作られる接続である。そのため、ノード間には複数のルータが存在し、ネットワークの近さとを全く考慮しない仮想的なネットワークである。

2.2.5 キー値のマッピング

各キー値とデータのペアは、時計回りに最も近い

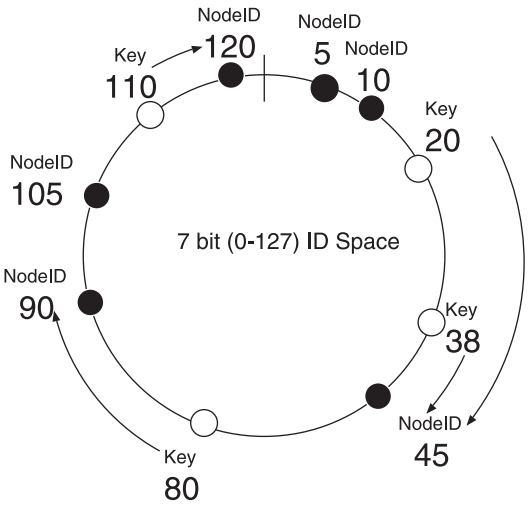

図 2.2. キー ID のマッピング

ノードに格納され、このノードを代表ノードと呼ぶ。Key20、Key38 は時計回りに最も近い Node45 に格納される。同じように、Key80 のデータは Node90、Key110 は Node120 に格納される。そして、キー値とデータのペアは、ネットワークに参加しているノードのみに保存される。

2.2.6 基本的な参照方法

図 2.3 は基本的な参照方法を表している。検索者である Node45 が、Key110 に対応するデータを取得したい場合、Node45→Node90→ Node105→Node120 へと検索クエリが転送される。Node120 は Key110 を保持しているため、Key110 対応するデータを Node45 へ送信する。

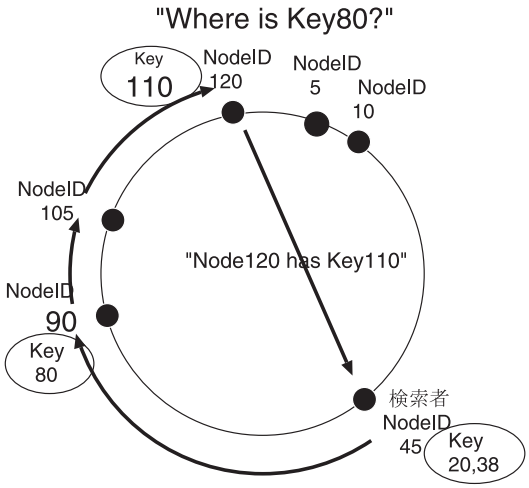

図 2.3.　基本的な参照処理

2.2.7 高速な参照方法

基本的な参照方法では、N ノードで構成されるネットワークの検索に $O(N)$ のホップ数がかかることになる。Chord では、性能を改善するために、高速な参照方法を定義している。7 bit のハッシュ値を用いて 128 の ID 空間が存在する場合、図 2.4 の様に自ノードの ID より 2^K $(0 \leq K < 7)$ 先のノードの IP アドレスを常に保持し、検索キーをショートカットしクエリの転送を行う。これにより $\log(N)$ のルーティングテーブルを保持することになるが、最大 $\log(N)$ 回のホップ数で高速に検索を行うことができる。また、Node45 が Key 44 の参照を行った場合、ワーストケースである 7 ホップとなってしまう (図 2.5)。

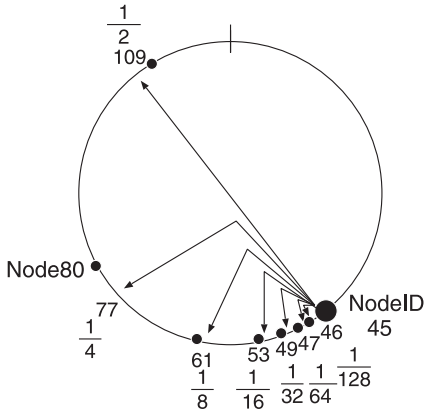

図 2.4.　$\log(N)$ 回の hop 数で検索

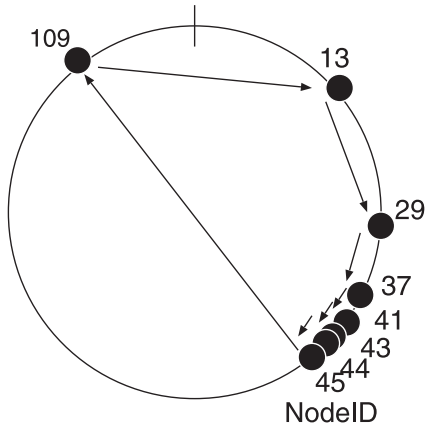

図 2.5.　ワーストケース

2.2.8 ネットワークへの参加

図 2.6、図 2.7 は、Node45 は Key 20、38 のキーペアを保持しており、Node30 がネットワークに参加する場合を表している。まず Node30 は自分の挿入すべき場所である Node45 を探し、Node45 の保持する Key20 を自分にコピーする。Node10→Node30→Node45 へリンクの繋ぎ直しを行う。前に述べたように 1 ノードを参照するには $\log(N)$ メッセージ数 (ホップ数) かかる。そのため、ネットワークの参加へは自ノードを参照される $\log(N)$ 個のノードに変更を行うため、参加時に生じるメッセージ数は $O(\log^2(N))$ となる。

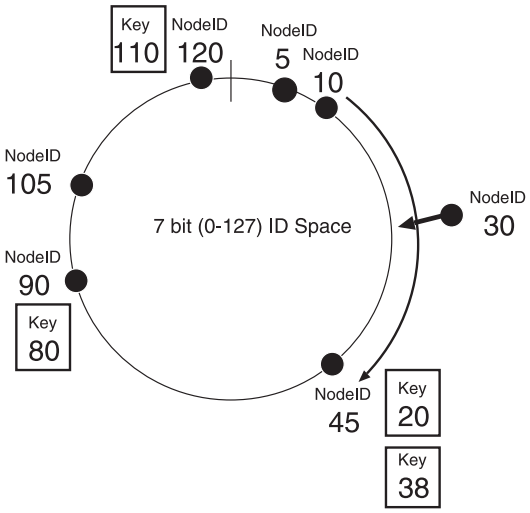

図 2.6. ネットワークへの参加

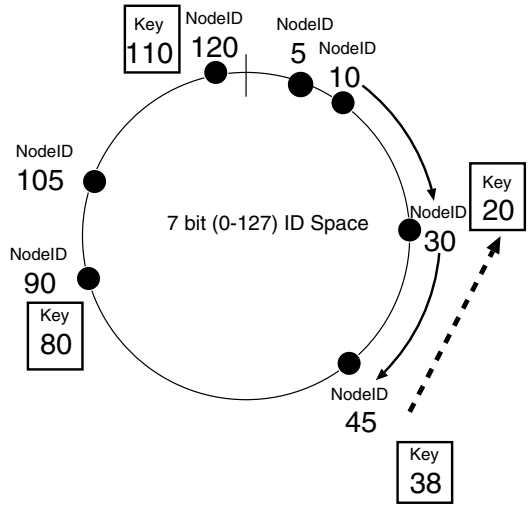

図 2.7. ネットワークへの参加 (続き)

2.2.9 負荷の均一化

SHA-1 のハッシュ関数を用いることでキー値はランダムにマッピングされ、各ノードへ均等に分散される。また参照の多いキー値を各ノードがキャッシュしておくことにより、参照手続きの回数を削減することができる。各ノードは $\log(N)$ 個のノードへパケットを送信できるということは、$\log(N)$ 個のノードからパケットを受信することを意味する。不特定多数のノードからではなく $\log(N)$ 個のノードからだけパケットを受信するためパケット数の制限をかけることが可能であり負荷の上昇をコントロール可能である。

2.2.10 安定性

それぞれのノードは通常ルーティングテーブルに $\log(N)$ ノード情報を持っている。さらに、これと同時に時計回りに次の r 個のノード情報も保持している。冗長性を持たせるために、この r 個のノードに自分の保持するキー値とデータのペアのレプリカを作っておくことにより安定性を高めている。

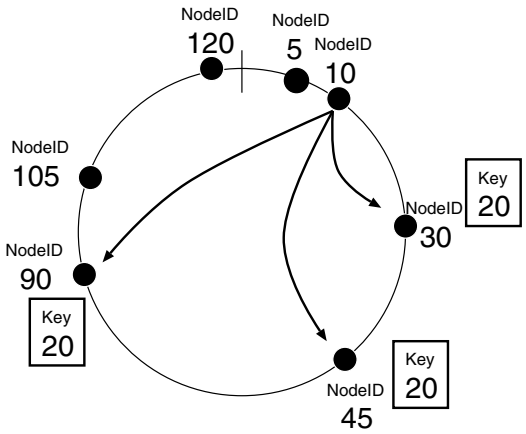

図 2.8. 安定性の向上

2.2.11 考察

Chord ネットワークは、ハッシュ関数を用いて各ノードに規則的に役割を持たせている。そのためアドホック的なネットワークに比べ状態遷移が多く、ネットワークの維持にかかる処理は複雑化することが予測される。

分散アプリケーションではどこにデータを格納するか問題となるが、その解決方法の 1 つとして、分散ハッシュテーブルである Chord を用いた参照処理が利用できる。そのため、best effort 的なサービスではあるが、今後様々な分散アプリケーションに適用できると考えられる。

2.3 Tapestry と Plaxton 構造

Tapestry[169] は、カリフォルニア大学バークレィ校で開発されている分散ハッシュテーブル方式のオーバーレイネットワークのインフラストラクチャである。Tapestry は Plaxton, Rajamaran および Richa により考案された分散データ構造 [120](以下 Plaxton

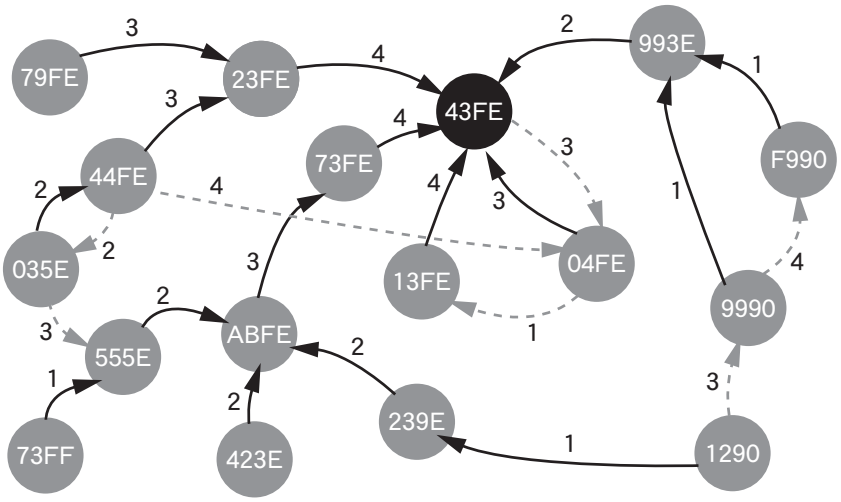

図 2.10.　Plaxton 構造の例: ノード 43FE を目指す経路

構造) に基づいており、局所性の追求を特徴とする。

Plaxton 構造は経路表を用いるが、同様に経路表を用いるオーバーレイネットワークの関連研究にはマイクロソフト社/ライス大学による Pastry[135] がある。

2.3.1 Plaxton 構造

Plaxton 構造はスケーラブルで高速なオブジェクト探索および経路制御を、各ホップで固定サイズの経路表を参照することにより実現する静的なオーバーレイネットワークである。Plaxton 構造では、すべてのノードがサーバであり、ルータであり、クライアントである。すなわち、オブジェクトを格納し、経路制御を行い、探索要求を発行できる。

他の分散ハッシュテーブル方式と同様に、Plaxton 構造でもすべてのノードおよびオブジェクトに識別子が割り当てられ、SHA-1 等のハッシュアルゴリズムにより均質に名前空間に分布することが前提となる。

0642	○	*042	○	*02	○	*0	○
1642	○	*142	○	*12	○	*1	○
2642	○	*242	○	*22	○	0642	○
3642	○	*342	○	*32	○	*3	○
4642	○	*442	○	0642	○	*4	○
5642	○	*542	○	*52	○	*5	○
6642	○	0642	○	*62	○	*6	○
7642	○	*742	○	*72	○	*7	○

図 2.9.　Plaxton 構造の例: ノード 0642 の持つ経路表

Plaxton 構造におけるオブジェクト探索の基本的なアイデアは、オブジェクト毎に、そのネットワーク上の位置を知る「ルートノード」を決定的なアルゴリズムにより求め、探索の際には当該ノードを頂点とする木構造を逆に辿るというものである。この際の経路制御は、各ノードと近隣ノードとの論理的なリンクを表す経路表を参照することにより行われる。図 2.9 は、一例として、あるオーバーレイネットワークにおけるノード 0642(8 進表現) の持つ経路表を示している。

経路表のエントリは ⟨ 目的地, ○⟩(ただし ○ は目的地の記述に一致する近隣ノードへのポインタ) という形式を持つ。∗ はワイルドカードを表し、与えられた目的地の識別子と、サフィクス (現在の Tapestry ではプレフィクス) が最長一致する近隣ノードに中継することで、最終的に目的地に到達する。

図 2.10 は、一例として、あるオーバーレイネットワークにおいて、ノード 43FE(16 進表現) に向かう経路 (実線) をすべて示している。

図中の矢印の脇の数値は、リンク先の識別子の何桁目が自己と不一致かを示している。

この方法では、N 個の可能な要素から成る名前空間で、識別子の基数を b として解釈する場合、最大でも $\log_b N$ ホップ (オーバーレイネットワーク上のホップ) で目的のノードに到達でき、経路表のサイズは $b \cdot \log_b N$ の固定サイズとなる (図 2.10 で、矢

印脇の数の最大値が 4 であることに注意されたい。識別子 43FE と 1 桁目が異なるノードを起点としても、最大 4 ホップで目的のノードに到達できる)。

オブジェクトが探索可能になるためには、当該オブジェクトを Plaxton 構造上に公開する手続きが必要となる。これはオブジェクト O を格納するサーバ S から、O のルートノードに対してメッセージを送ることにより実現される。この際、O を S にマッピングする情報を各中継ノードに残していく。これにより、オブジェクト探索の要求元 C は、S と C を葉として持つ部分木の頂点に探索要求が達した段階で O の位置を知ることができる。このことの重要な帰結は、オブジェクト O が複数のサーバ $S_1, S_2, \ldots, S_n$ に格納されている場合、探索元 C はこれらの中でもネットワーク的に近傍のサーバを知ることができるということであり、局所性が実現できる。

局所性は、遅延を短縮し、帯域の有効活用を促進すると共に、システムの信頼性/可用性を向上する上でも重要である:通信の経路が長ければデータが失われたり、壊れる可能性が増大するし、データやサービスが近傍に存在していれば、世界の他の部分から分断されたとしてもアクセスできる。

ただし Plaxton 構造には、構成に際してグローバルな知識を必要とする静的な構造であること、ルートノードが single point of failure となること、等の難点がある。

これらの問題の解決を図ったのが Tapestry であるといえる。

2.3.2 Tapestry による拡張

Tapestry では、オーバーレイネットワークの動的な構成を可能としてる他に、次のように Plaxton 構造の欠点を補っている。

オブジェクト探索においては、各オブジェクトに少数の決定的で独立な複数の GUID(Globally Unique ID) を生成し、すべての GUID に対してオブジェクトが可用であることを公開する (すなわちオブジェクトは複数のルートノードを持つ)。そしてすべての GUID に対して並行に探索を行うことにより耐故障性を実現し、かつ性能の低下とばらつきを防止している。

メッセージ伝搬においては、ランドマーク経路制御 [170] と呼ばれる方式により不要な (IP レベルの) 広域ホップを削減し、Plaxton 構造における局所性を更に補完している。この方式では、ランドマークとなるスーパーノード間を (オーバーレイネットワーク的に) 直接結び、AS 間転送を高速化している。

2.3.3 Tapestry の応用

Tapestry は現在、同じくカリフォルニア大学バークレィ校にて開発されている分散ストレージシステム OceanStore[92] の Java コードベースの一部として実装されており、OceanStore の下位レイヤとして利用されている。

Tapestry の応用としては他に次がある:

- Bayeux[171](アプリケーションレベルマルチキャスト)
- Berkeley SDS(Service Discovery Service)
- DNS++(分散スケーラブル DNS)
- Shuttle(非集中耐故障インスタントメッセージング)

このうち、Shuttle では、メッセージのリレイにサーバを用いず、ユーザを可用なオブジェクトとして Tapestry 上で公開することにより、プレゼンスの取得やメッセージの伝達を行っている。

第 3 章　分散ハッシュテーブルの応用

3.1 i3: Internet Indirection Infrastructure

本節では、Chord アルゴリズムを用いたアプリケーションの一例として i3(Internet Indirection Infrastructure)[149] を紹介する。

3.1.1 i3 の概要

i3 の基本的なアイディアは、id から受信者 IP アドレスへの変換、ならびに受信者へのパケット転送をおこなうインフラストラクチャ (以下、i3 ネットワークと呼ぶ) を用意し、送信者に id を宛先としてパケットを送らせることによってモビリティ、マルチキャスト、エニキャストを実現しようというものである。アプリケーション層マルチキャスト等の先行研究と異なり、これら三つの通信モデルを同時に実現できる点が本研究の特徴である。

i3 では、id と受信者アドレス R の組 (id, R) をトリガーと呼ぶ (図 3.1)。トリガーは i3 ネットワーク

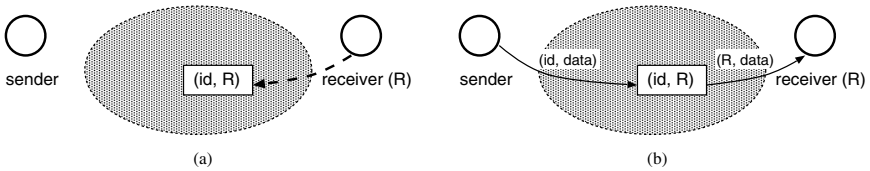

図 3.1.　トリガーの挿入 (a) とパケットの転送 (b)

上に分散して保持される。i3 ネットワークは Chord アルゴリズムによって構成されるオーバーレイ・ネットワークである。複数のトリガーを Chord アルゴリズムを用いて分散して保持することで、i3 は規模拡張性、耐故障性、負荷分散といった特性を実現している。なお、受信者アドレスはトランスポート層のアドレス、すなわち IP アドレスとポート番号である。

トリガーを用いることで、様々な通信モデルを簡潔に取り扱うことが出来る。

モビリティを例にとると、受信者アドレスが R から R' へ変わった場合、(id, R) を (id, R') に更新すればよい。送信者は受信者の IP アドレスを使わず id を宛先として通信しているので、受信者のアドレス変更を知る必要がない。

単一の id に対し、複数の受信者を定義することも可能である。例えば (id, R_1)、(id, R_2) という二つのトリガーを i3 のネットワークに挿入することにより、マルチキャストが実現できる (図 3.2)。この場合、トリガーの挿入はマルチキャストグループへの加入と同等であると考えることができる。ただし、ここでいうマルチキャストとは送信者および受信者を限定しない any-source multicast であることに注意されたい。

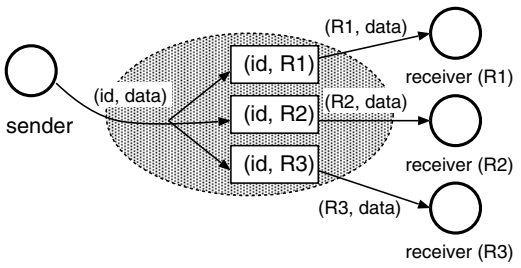

図 3.2.　i3 を用いたマルチキャストの実現

エニキャストを実現することも容易である。id をプレフィクス部とサフィックス部に分け、id を最長一致で検索することでエニキャストが実現できる。例えば共通のプレフィクス id_p と一意なサフィックス $s_1, s_2, \ldots$ をもつトリガー $id_p|s_1, id_p|s_2, \ldots$ を個々のサーバに割り当てることができる (図 3.3)。また、サーバが負荷に応じてトリガー数を増減することで、動的な負荷分散を実現することもできる。例えば、負荷が高くなってきた場合に $id_p \mid s_2$ を i3 ネットワークから削除し、他のサーバに処理を任せることができる。

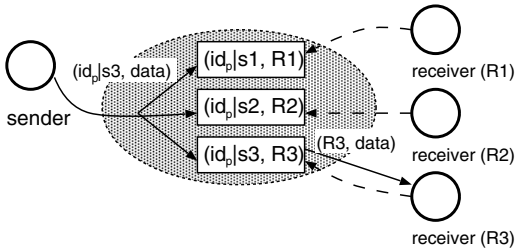

図 3.3.　i3 を用いたエニキャストの実現

マルチキャストにおいて受信者数が増えた場合に対応するため、id を別の id に変換するトリガー (id_1, id_2) が導入されている。これによって、(id_1, id_2)、(id_2, R) といったトリガーの階層化が可能となる。受信者数が多い場合にトリガーを階層的に構成し、単一トリガーあたりのパケット複製回数を減らすことによって、あるトリガーを持つノードの負荷を軽減することができる。

3.1.2 i3 におけるセキュリティ

本節では、i3 における既知のセキュリティ上の問題点と対策方法を述べる。

まず、i3 では既知の id に対してだれでもトリガーを挿入できるため、盗聴が問題となる。この問題への対策として、公知のトリガー (public trigger) と私的トリガー (private trigger) が導入されている。公知とする必要がないトリガーは私的トリガーとし、2^{256} の番号空間のなかから id をえらぶことによって盗聴を防ぐというものである。また、私的トリガーの id を定期的に変更することも可能である。

私的トリガーの id_p は公知のトリガーを通じて伝えられる。このとき、通信相手の公開鍵を用いてペイロードを暗号化することによって、ペイロード中に書かれた id_p の盗聴を防ぐことができる。

また、id が公知である場合、第三者がトリガーを削除してしまう可能性がある。この攻撃への対策として、id からサーバ S に直接射影するトリガー (id, S) を使うことを避け、(id, x)、(x, S) という二つのトリガーを用いる方法が挙げられている。x をサーバ S のみが知る秘密とすれば、第三者が x を類推することはできず、トリガーを削除することはできなくなる。

トリガーをリフレクタとして悪用する攻撃も考えられる。例えば、被害者 V を受信者として (id, V) を挿入し、id あてにパケットを投げ続けるといった攻撃である。この問題への対策として、(id, V) のトリガー挿入時に必ず V からの確認応答を要求する方式が挙げられている。

このほか、公知のトリガーに対する DoS 攻撃として、トリガーの階層を構成し、そのパケット増幅効果を用いてサーバにたいし DoS 攻撃をかけるというものがある。この攻撃への対策として、i3 ネットワーク内でのパケット転送において Fair Queueing を使う、単一ホストが挿入できるトリガー数を制限する、という 2 つの方式が挙げられている。

また、id 変換トリガーを連鎖させトリガーのループを形成することにより、i3 ネットワーク自体に DoS 攻撃が加えられる可能性がある。この問題への対策として、id 変換トリガー (id_1, id_2) を受け入れる前に乱数を記した検査パケットを id_2 あてに送り、ループを検出する方式が挙げられている。

3.1.3 考察

i3 の問題点

前節で述べた対策方式は完全なものではなく、いくつかの問題点をはらんでいる。ここでは筆者らが気づいた点について述べる。

i3 では盗聴対策として、公開鍵の存在を前提としている。エニキャストやサーバ負荷分散などの用途において、通信相手の公開鍵が既知であることを前提とした場合、PKI の運用形態に影響を与える可能性がある。複数のホスト間で秘密鍵を共有することになるからである。

また i3 では私的トリガーの秘匿性が確率的にしか保証されない。私的トリガーの id を定期的に変更した場合、ある id で継続して盗聴している攻撃者にとっては盗聴がより容易になる。

しかしながら、上記は現在の i3 アーキテクチャにおける問題点であり、さまざまなトレードオフの結果であることは言うまでもない。これらの問題点は、i3 ネットワークが保持する状態や、ノード間でやりとりするメッセージの数を増やすことによって解決できる可能性がある。

i3 の可能性

インターネットにおいて「間接アドレッシング」を導入するという考え方はこれまで全くなかったわけではないが、単一障害点や負荷集中といった点が問題となっていた。i3 では id から IP アドレスへの変換テーブルを Chord を用いて分散させることで、耐故障性や負荷分散を実現しつつ「間接アドレッシング」を提供することに成功している。

id は IP アドレス空間とは独立した番号空間であるため、既存のアプリケーションとの互換性の問題は残る。しかしながら、既存のインフラストラクチャ上でモビリティやマルチキャストを実現したまったく新しいアプリケーションを構築する場合、i3 のアプローチは有用であると考えられる。例として、新しい IM(Instant Messaging) サービスにおいて移動透過性やグループ通信機能を提供する場合を想定するとわかりやすい。

Chord やその他の DHT 方式は様々なアプリケーションに応用することができると考えられる。例えば、ディレクトリサービスや NAT 等の従来、単一のサーバで構成されていたシステムを、耐故障性を有し、かつ負荷分散可能なシステムとして構成することが可能となる。

3.2 DDNS

本節では、Chord を利用したアプリケーションの例として、DDNS を紹介する。DDNS は、Chord の DHash をバックエンドとして用いた DNS 代替方式である。

3.2.1 DDNS の概要

目的

DNS には、管理が困難であるという問題がある。関連研究 [79] より、DNS でやりとりされるメッセー

ジのおよそ 35%が、DNS の管理不備により、無意味なものとなっている事が明らかにされた。この観測は、DNS の管理の困難さを間接的に示している。

DDNS[45] では、オーバーレイネットワークを利用してサービスの提供を、名前空間の権限を持つ主体 (Authority) から切り離す。これにより、名前空間の利用をより容易に行える。同時に、オーバーレイネットワークの機能により、ルートネームサーバを利用せずに済み、負荷分散の効果も得られる。

方式

DDNS のオーバーレイネットワークには、Chord の DHash を用いる。DHash に対して、従来の DNS でも用いられている RRSet 単位でデータを挿入する。同時に、名前に対する Authority の確保のために、RRSet 単位の署名検証 (SHA-1) を行う。ある RRSet の署名 (SIG RR 相当) はその上位ドメインの鍵により署名され、検証可能である。Root に相当するドメインの署名はシステムに既知なものとする事で、DNSSEC と同様の検証が可能である。

評価

Jung[79] が採取したデータを利用し、評価を行っている。1000 ノードから成る Chord リングを構築し、その上に DDNS の機能をシミュレートした。なお、lookup を開始するノードはランダムに選択する。

成功する名前解決に関しての評価は以下の通り。Jung のデータから、成功した問い合わせ一日分 (約 260,000 クエリ) を採取し、シミュレータに与えた。前もってシミュレータには問い合わせに対応するデータを格納してある。

結果は、負荷分散において良好な効果を示している。これにより、DNS の実際のデータを元にしたシミュレーションに対しても Chord(DHash) の特性は損われないという事がわかる。

失敗するクエリに対する評価のためには、DHash ノードにデータを何も入れず、一日分の失敗する問い合わせ (約 220,000 クエリ) をシミュレータに与えた。ネガティブキャッシュを用いておらず DDNS にとって不利な条件であるにも関わらず、負荷分散や問い合わせ結果の取得に対して成功する問い合わせと同じ傾向を得た。

問い合わせの遅延時間の評価は、実際に Internet 上に展開された DHash ノード群を持たないため、Jung のデータを元にした推定値をもって評価としている。

その結果、DNS と比較して (原文 Figure 5 および Figure 6) 遅延に関して以下の特性が判明した。

- DHash の自動再構成機能の効果により、高遅延のノードが排除されるため、ほぼ 1200 ms 以内に問い合わせは完了する。
- 一方、DNS では遅延のメジアンが 43 ms であるのに対し、DDNS による名前解決の遅延のメジアンは 350 ms である。

3.2.2 DHash を用いる事による利点と問題点

DDNS により、従来の DNS で観測された管理上の問題は解決できる事が期待できる。

一方、DNS で利用されている動的なレコード生成は、DDNS では利用できない。動的なレコード生成は、RRSet の順序入れ換えによるアプリケーションサーバの負荷分散等で使われている。

一方、DDNS で実現している機能は、DHash で実現可能な機能の枠内という制約を受ける。DHash で動的なレコード生成を行えない以上、DDNS は分散した `host.txt` 以上のものではない。

DDNS で負荷分散等を行う場合は、クライアントサイドを制御する事により可能になる。しかし、何か新規機能を追加する度にクライアント全てを更新する事は現実的ではなく、問題となる。

また、DDNS、あるいは DHash に新機能を追加する必要がある場合、サーバ全てを矛盾無く更新する必要があり、困難である。

3.2.3 考察

オーバーレイネットワークの提供する機能とアプリケーションの要求

IP はホスト間の通信を担う。そして、IP における通信はデータの位置を常に意識したものである。現在、最も良く使われているだろう応用の WWW において、データの識別子が URL(Universal Resource **Locator**) で表現される点に、その一端が表われている。

これに対して、DHash という層を狭んで、データの位置と内容の関係の再抽象化を行ったアプリケーションの一つとして、ここで DDNS を紹介した。

従来の DNS では、データの位置と内容が密接に相関している。DNS の名前空間の中では、NS レコー

ドにより名前を管理するサーバの位置を指示する。一方、Chord をオーバーレイネットワークとして利用する事で、この 2 つは完全に分離できる。

オーバーレイネットワークの性質と、アプリケーションの要求は合致しなければならない。DDNS では、オーバーレイの性質のうち自動構成機能、自動負荷分散機能、耐故障性を、アプリケーションを実現する上での「利点」とした。一方で、アプリケーションが求める特性のうち、TENBIN[181] 等の、名前空間毎に差別化したサービスは、Chord そのままでは現実的には実現できない。従って、DDNS をそのまま現在我々が利用している DNS の代替として利用する事は困難である。

このように、オーバーレイネットワークを利用する事によって、従来のシステムに無かった特性を持つ事も可能になる一方、アプリケーションの要求を完全に満たす事が困難な場合もある。Chord のような分散ハッシュテーブルを利用する場合、どうしても一様な性質の空間が前提となり、困難が生じた。

オーバーレイネットワークの更新

加えて、分散ハッシュテーブルを用いたオーバーレイネットワークの実用において障害となりうる点を本論文は指摘している。オーバーレイネットワークにおいて、特殊な機能、実験的な機能、あるいは単なる追加機能を実装する場合を考える。機能はクライアントあるいはサーバのどちらか側に実装する事となる。通常は、クライアント全てをその機能に対応させるために更新する事は一般に困難なので、従来のシステムではサーバを更新する事により対応を行うケースが一般的である。しかしながら、分散ハッシュテーブルを利用している場合はオーバーレイネットワーク全てを更新する必要がある。

第 4 章　その他の技術

4.1 Freenet

Freenet は、英エジンバラ大学の Ian Clarke が 1999 年に投稿した論文 “A distributed decentralized information storage and retrieval system” [40] を元に、オープンソースで開発が進められてきた分散型ファイル共有・検索システムである。

他の P2P 型のファイル共有システムに比べ、Freenet では参加者の匿名性を重視している点に大きな特徴がある。Napster や Gnutella 等が目的のファイルを保有するノードを発見し、そのファイルを取得するまでを対象としたアプリケーションであるのに対し、Freenet ではファイルの挿入、クエリーの発信元、ファイルの保有主など個人を特定できる要素を取り除き、匿名性を元に全てのファイル分散・共有に関するプロセスが行われることに設計の重点が置かれている。

Freenet の基本的なアーキテクチャは同じく Ian Clarke らが出した論文 “A Distributed Anonymous Information Storage and Retrieval System” [41] にまとめられている。本節ではこの論文を元に、Freenet の理念と設計についてその特徴的な部分を考察する。

4.1.1 設計理念

Freenet の設計理念は以下の 5 点に集約されている。

- 情報の生産者および利用者双方の匿名性
- 情報の保管者への否認性
- 情報へのアクセスを阻害する第三者の攻撃に対する抵抗力
- 情報の効率的な動的保管とルーティング
- すべてのネットワーク機能の分散化

以下にこれらを実現するためのメカニズムについて考察していく。

4.1.2 ファイルキーについて

Freenet 中のファイルは、ハッシュ機能の適用 (現在は 160 bit SHA-1) により得られたバイナリファイルキーにより識別される。このバイナリファイルキーは目的と特性の違いにより、以下の異なる 3 つのタイプのファイルキーが利用される。

KSK(keyword-signed key: キーワード署名キー)

ユーザがファイルに対して名付ける任意の短縮記述テキスト・ストリングから派生するファイルキーである。(論文 [41] 中には、ユーザが戦争に関する論文を示すテキスト・ストリングの一例として、“text/philosophy/sun-tzu/art-of-war” という記述が挙げられている。) このようなキー・ストリングは公開/秘密鍵のペアを生成するための入力として使用され、生成された片方の公開鍵はその

後ハッシュ化されファイル・キーとなる。非対称鍵ペアの片方の秘密鍵は、検索されたファイルがそのファイル・キーと一致すると最小限の整合性チェックを行い、ファイルに署名するために使用される。

SSK(signed-subspace key: 署名部分空間キー)

KSK において、一度使用された短縮記述テキスト・ストリングを他の者が使用することは出来ない。また、ファイルの挿入者は他者にファイルの検索を可能にするために、記述ストリングを公表する必要がある。これにより、普及した記述ストリングにジャンク・ファイルを挿入するような “key-squatting” 攻撃の可能性を残す。このような問題を解決するものとして SSK がある。SSK(signed-subspace key: 署名部分空間キー) は個人の名前空間を確保するものであり、名前空間の識別をおこなう公開/秘密鍵のペアを任意に生成することによってユーザ固有の名前空間 (以降、「部分空間」と呼ぶ) を作成する。SSK の作成は、ユーザが選んだ短縮記述テキスト・ストリングのハッシュと、上記で生成した公開鍵を XOR 演算によりまとめ、その値からファイルキーを生成するために再度ハッシュする事により行う。他者によるファイル検索を可能とするために、ユーザはその部分空間の公開鍵と一緒に記述ストリングを公表するが、データの格納には秘密鍵を必要とするため、部分空間の所有者だけがそこにファイルを加えることが可能となる。

CHK(content-hash key: コンテント・ハッシュ・キー)

第 3 のキーとして、CHK(content-hash key: コンテント・ハッシュ・キー) がある。CHK は、対応するファイルの内容を直接ハッシュすることにより簡単に生成される。ファイル自体は任意に生成された暗号鍵によって暗号化されているため、解読鍵を持ったユーザ以外はファイルの中身を知らないままファイルを保有することになる。一方、CHK をファイルキーにすることで解読鍵を持たない保有者も現在保有しているファイルと送られてきたクエリーの照合を行うことが出来る。

また CHK は SSK と共に用いることで、ファイルの更新と分割の実装に有用である。ユーザは上記の手順で作成した CHK をファイルキーとして (これを CHK2 とする)Freenet 上に挿入する。次にその CHK2 を示した間接ファイルの CHK (これを CHK1 とする) を SSK の下に挿入する。検索者はまず CHK1 で表される間接ファイルを探索し、次にそこに記された CHK2 を探索するという二段階の検索を用いて目的となるファイルを獲得することになる。ファイルの挿入者はこの間接ファイルを書き換えることで、ファイルの更新 (CHK2→CHK2') や分割 (CHK2、CHK3、CHK4. . .) を容易に行うことが出来る。

4.1.3 ファイルの検索と取得

Freenet では上記で述べたファイルキーを元に検索を行うため、ファイルの検索には検索者は最初にファイルキーを獲得するか、もしくは計算する必要がある。まず検索者はファイルキーと Hops-To-Live 値[1]から成るリクエスト・メッセージを作成し、自らの出所を明かさないよう、他のノードからリクエストを受けたとき同様に、最初に自身のノードにリクエスト・メッセージを発行する。Freenet 内のノードは、ファイルキーとノード・アドレスの組合せから成る経路表を保有しており、リクエスト・メッセージを受けた (そのファイルを直接保有しない) 中継ノードは、リクエスト・メッセージにあるファイルキーをその経路表に照合し、最も近いファイルキーに対応するノードにリクエストを転送していく。

検索が成功した場合、要求したファイルはリクエスト・メッセージ転送の経路を同様の経路を逆に辿ってファイル保有者から検索者まで転送される。その際、ファイルを転送する経路上にある中継ノードは、要求されたファイルキーとファイルの出所を元に経路表を更新し、以降そのファイルキーと辞書的に近い「類似キー」(ファイルキー AH5JK2 の場合、AH5JK1、AH5JK3 等) に対するリクエストは、以前に成功したデータの出所へ転送される。また中継ノードはファイル転送の際に、自身のノード内のローカル・キャッシュに転送するファイルの複製を保存していくことで、以降同じファイルキーによるリクエストは中継

[1] IP における TTL(Time-To-Live) のようなもの。IEEE Internet Computing(2002 年 2 月–3 月) に Ian Clarke 等により投稿された “Protecting Free Expression Online with Freenet” では TTL と表記されているが、ここでは元の論文 [41]) に習って Hops-To-Live の名称を使用する。

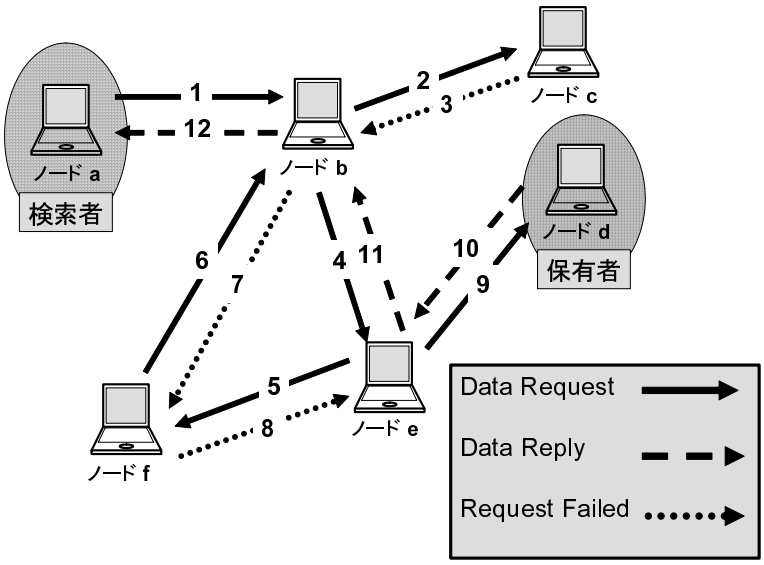

図 4.1. 典型的なリクエスト・シークエンス

ノードのローカル・キャッシュから転送が開始されることになる。

類似キーのリクエストをそのキーに対応するノードが集中的に受けること、及びファイル転送の際に中継ノードがファイルの複製をローカル・キャッシュに蓄積していくことによって、以下を実現することを主張している。

- 辞書的に近いファイルキーを持つファイルは、経験的に特定のノードに集約されていく
- リクエストの多いファイルは多数のノードに複製・散布されることによって Single point of failure を防ぎ、検索に要する Hop 数を節約することに繋がる

図 4.1 にファイル検索における典型的なリクエスト・シークエンスを示す。

検索者は、ノード a でリクエストを始める。ノード a は、ノード b へリクエストを転送し、それはノード c へ進む。ノード c は、他のどのノードとも連絡を取ることができず、ノード b へ “リクエスト失敗” の後戻りメッセージを返す。その後、ノード b は、ノード f へリクエストを転送するための二番目の選択ノード e を試みる。ノード f は、ループを検知し後戻り失敗のメッセージを返すノード b へリクエストを転送する。ノード f は他のどのノードとも連絡を取ることができず、さらにノード e に、1 ステップ引き返す。ノード e は、二番目の選択のデータ保持するノード d へリクエストを転送する。ヒットしたファイルは、検索者であるノード a にノード e とノード b を経由してノード d から戻され、途中経路にあるノード e、ノード b は自身のローカル・キャッシュに複製を保存する。

4.1.4 ファイルの挿入

ファイルの挿入も、ファイルの検索と同様の手法に従って行われる。ファイルを挿入するノードは、まず挿入するファイルのファイルキーを作成し、そのファイルキーと Hops-To-Live 値を指定した挿入メッセージを自身のノードに送る。自身のノードに格納されていないことを確認した後、挿入メッセージは検索のときと同様、Hops-To-Live 値の限界まで検索を行い、既に使用されているファイルキーとの衝突の有無を確認する。ファイルキーの衝突が無かった場合、最初の挿入メッセージの経路に沿ってファイルの挿入が行われる。経路上の中継ノードは挿入されたファイルを複製してキャッシュし、新しいファイルキーのエントリーを自身の経路表に追加する。

4.1.5 ファイルの管理

Freenet では、参加するノードのデータストアの集合が Freenet 内に存在する全てのファイルの記憶装置となる。各ノードのデータストアは、最も新しく発生したリクエストの時間、あるいは挿入時間を元にソートされたデータ・アイテムが置かれる Least Recently Used(LRU) キャッシュとして管理され、

使用頻度の低い領域のデータを消去し新しく挿入されたファイル用の領域を解放していくことによって、データストア内のファイルが更新されていく。このように Freenet ではファイルの存続は保証されないが、需要の少ない情報は消滅し、需要の多い情報ほど多くの複製を生みながらより長く存続するということで Freenet 全体のデータの自律的なライフサイクルを実現している。

4.1.6 ノードの追加

Freenet に参加する新しいノードは、まず公開/秘密鍵のペアを作成する。この公開鍵は Freenet 内でのノード ID となる。新しいノードは既に Freenet に接続されているノードをなんらかの形 (公開されている Web サイト、または通常のコミュニケーションなど) で探し出し、この公開鍵とノード・アドレスの組を、指定した Hops-To-Live 値によって届く範囲に通知する。この時点で新しいノードには担当するファイルキーの ID 空間が割り当てられる。

4.1.7 プロトコルについて

Freenet のアプリケーションにおけるプロトコルはパケット指向型であり、トランスポートのメカニズム (TCP、UDP、パケット・ラジオなど) に依存しない自己完結型のメッセージを使用する。個々のメッセージは、64 bit のトランザクション ID を保有し、ノードの状態 (挿入、または検索の状態にあるのか) を追跡できるような仕組みになっている。物理的な所在を表すノードアドレスは、例えば　tcp/192.168.1.1:19114 のように、転送方法とその転送方法に必要な識別子 (IP アドレスとポート番号など) から成り立っている。

全てのメッセージは先に述べた 64bit のトランザクション ID の他に、Hops-To-Live 値と Depth 値が記述されている。Hops-To-Live 値はパケットの無限ループを防ぐために、中継ノードによって 1 ずつ減算されていく。この逆に Depth 値は中継ノードによって 1 ずつ加算されていき、この値は、最終的にリクエストに答えるノードが検索者に到達させるために充分な Hops-To-Live 値を決定するための指標としてを使用される。しかしこれらのことは、Hops-To-Live 値や Depth 値を逆算すれば検索者や保有者の物理的な所在が明らかになってしまうことを表している。そこで Freenet では各主体の物理的な所在を不明瞭にするために、「Hops-To-Live 値が 1 になったパケットは限定された確率の元に転送が継続される」「検索者は Depth 値をランダムな値に指定し直せる」という 2 つの仕組みを採用している。

4.1.8 考察

以上、Freenet の基本的なアーキテクチャについて説明してきた。

オーバーレイネットワークの特徴の一つとして、オブジェクトをネットワーク上のロケーションから切り離したことが挙げられる。このことは全てのノードが分散してファイルを保有することにより、ファイルの保有主を特定できない (つまりファイルの検索者と保有主が明確に分けられておらず、そのファイルが自分のものであるということを否定も肯定も出来ない) 状況を容易にし、ひいてはデータの保管と交換における「匿名性」という古くて新しい命題をエンドユーザのニーズとして再び喚起させたのではないかと考える。Freenet では匿名性を維持したままファイルキー自体の告知や検索をどのようにサポートしていくかといった問題や、悪意ある参加者による DoS 攻撃の可能性などいくつかの重要な課題も残されてはいるが、分散ハッシュテーブル方式によるキー分散と二重鍵技術の巧妙な組合せにより、匿名性を実現するアーキテクチャを提供していることは注目に値する。このように、ユーザサイドに生まれた (または喚起させた) ニーズに柔軟に対応していける利点もオーバーレイネットワークの特徴の一つなのではないだろうか。

4.2 JXTA

JXTA とは、Sun Microsystems が提唱する peer-to-peer のプラットフォームである。また、JXTA は同時に、ピアグループというピアの任意集合を利用したオーバーレイネットワークでもある。要素技術としては特に目新しいものは無い一方、peer-to-peer およびオーバーレイネットワークに関する標準的なモデルを作ろうとしており、注目すべき技術として本報告書に掲載する。

JXTA は現在、Sun Microsystems によって提供された、Project JXTA[2]によって、修正された Apache

[2] http://www.jxta.org/

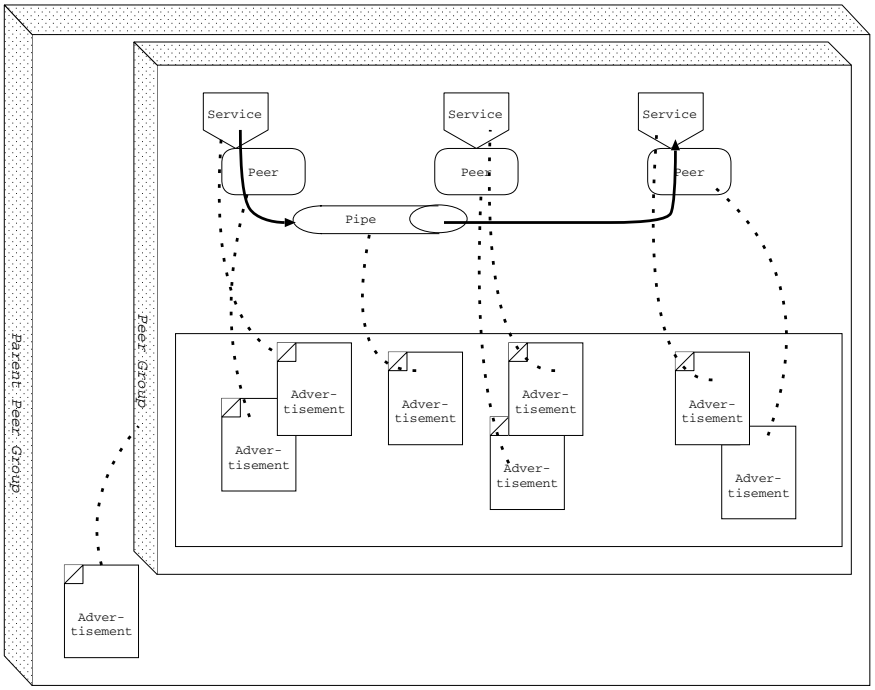

図 4.2. 主な構成要素同士の関係

License に基づき参照実装および周辺のサービスの開発が進められている。参照実装は、Java(J2SE, Personal Java) および C 言語によるものが存在する。

なお、JXTA 標準で用いられている用語は太字 (例: **Peer**) とし、一般的な技術としての用語 (例: ピア) と区別する。

4.2.1 概要

JXTA は、peer-to-peer と呼ばれる技術を包含する共通のアーキテクチャ、および対応する標準プロトコルのセットである。なお、JXTA では、標準を Core および Standard に分割しており、Core ではモデルの定義と、ピアが相互を発見する上で最低限のプロトコルの定義が含まれる。一方 Standard は、Core で定義された標準的なモデルを利用するためのプロトコルの定義が含まれる。

peer-to-peer 技術で求められる機能を多数のピア間で協調させるために、JXTA は以下の機能を標準化する。

- ピアの相互発見
- ピアグループへの組織化
- ネットワーク資源の広告 (**Advertisement**) と発見 (**Discovery**)
- ピア同士の通信 (**Pipe**/**Message**)
- ピアの監視

一方、JXTA はプロトコルで構成されているので、根本的には特定の実装言語や OS へ依存していない。同時に、トランスポートプロトコルに対する仮定も最小限に抑えてあり、TCP/IP 以外の環境でも利用可能な構造になっている。

4.2.2 アーキテクチャの構成要素

JXTA アーキテクチャは、主に以下の構成要素から成立する。主な構成要素同士の関係は、図 4.2 に示す。図 4.2 では、ピアグループの中に 3 つのピアがそれぞれ **Service** を広告しており、また 1 つの **Pipe** を使って二つの **Service** が通信を行っている。実際には、ピアグループもまた一つの **Service** としてより上位のピアグループ (*Parent Peer Group*) に広告している。

Peer(ピア)

Peer とは、あるデバイス上に存在するノードの抽象である。JXTA そのものは、ピアが何か、という事は定義しない。ピアは自身の存在と接続点 (**Endpoint**)、およびそのピアが提供するサービスを広告する。

PeerGroup(ピアグループ)

ピアの集合。資源の共有はピアグループ単位で

行われる。つまり、資源の広告の伝播範囲は、単一のピアグループであり、また広告された資源はそのピアグループに所属しているピアからのみ利用可能である。

Network Service/Module

JXTA では、資源のやりとりを行う主体を **Network Service** と呼ぶ。ピアは、標準で用意される **Discovery Service** の利用によって、他の **Service** を発見し利用する。また、より上位の概念として、**Service**、**Application** 等を包含する、**Module** という概念が存在する。

Pipe

Pipe は、JXTA における通信路の抽象化である。**Pipe** は **Pipe Advertisement** によって広告され、利用者はその **Advertisement** を解釈して利用する。

Advertisement

Advertisement は、利用可能な資源 (**Network Service**, **Pipe**, ピア, ピアグループ等) を広告する。資源を利用するためのメタ情報を XML によって記述する。

ID

JXTA において識別する必要があるものに対して割り当てられる識別子。形式は URN であり、Spec による規定には、Format および Type に関する規則がある。ID Format は、ID の表現形式を区別するための文字列である。Core Spec では uuid 形式と jxta 形式 (一部の定数) のみを定義し、他の形式の定義は実装者に任されている。ID Type は、その ID が何を表現するかを判別するための ID の「型」である。

どの ID 形式でも定義する事を推奨 (一部は必須) されている ID Type には、**PeerGroup ID**/**Peer ID**/**Codat ID**/ **Pipe ID**/**Module Class ID**/**Module Spec ID**/ **Module Impl ID** の 7 種類がある。

4.2.3 ピアグループ

本節では、JXTA アーキテクチャの特徴でもあるピアグループについてより細かく説明する。

JXTA におけるピアグループは、ピアの任意の集合である。各ピアは任意のピアグループに作成・発見・参加・脱退が可能である。また、各種資源はピアが参加しているピアグループに広告される。また、ピアグループは親を持ち、木構造になる事ができる。

ピアグループの作成および参加は以下のような手続きによって行われる。

1. 現在参加しているピアグループ ($pg_{current}$) を親として、新規ピアグループ (pg_{new}) を生成する。
2. pg_{new} の **Advertisement**(adv_{new}) を、$pg_{current}$ に広告する。
3. 他のピアは、$pg_{current}$ にある adv_{new} を検出し、pg_{new} の複製を生成する。

さて、手順 3 において、「同一のピアグループの複製」が生成された事になる。ここで、それぞれのピアにおけるピアグループの複製を、*PeerGroup* と記述して区別する。

JXTA では、ピアの集合 (ピアグループ) を実現するために、それぞれのピア上で実体化される *PeerGroup* 上に、共通のサービスの組 (**PeerGroup Service**) と、そのサービス実体の間でとり交わされるプロトコルを定義している。**PeerGroup Service** の作用により、同じピアグループ に参加しているピアを相互に発見し、資源を共有する。

なお、ピアは起動時に、WorldGroup および NetPeer Group の 2 つの規定のピアグループ に参加する。

WorldGroup は、最低限の機能のみを実装し、全ての「親」となる一方、**Advertisement** の交換等は行わず、ピアグループの階層構造のルートに相当するものである。

NetPeer Group はデフォルトのピアグループであり、**Advertisement** の交換なども行う。アプリケーションは、起動時に独自の NetPeer Group を定義して、一般の JXTA ピアとは孤立したネットワークを作る事も許されている。

4.2.4 考察

JXTA では、ピアグループという概念をアーキテクチャの中核に置いている。また、ピアの具体的な定義をしておらず、利用者がかなり自由にピアを再定義する事ができる。その結果、JXTA のアーキテクチャは基礎的なフレームワークとして応用可能性の高いものとなる。

利用者は、その応用にあわせてピアを定義する。ピアはひとつの機器かもしれないし、より細かく機

器の構成部品単位で定義しても、あるいは抽象的に「利用者」というような定義を利用する事も可能である。システムはその定義に応じて、ピア ID を割り振り、ピア同士を相互に通信させれば良い。

さて、「ピア同士の通信」と簡単に書いた。しかしここでは、ランデブーが大きな問題となる。つまり、ピア同士の通信といっても、どのピアがどのピアと通信するのが適切か、という判断は一般的な状況において自明ではない。従って、通信アーキテクチャには何らかのランデブーの枠組を用意する。その枠組の性質によって、アーキテクチャの性質が大きく変化する (節 1.4.1)。

JXTA は、ピアグループという単位でピアを組織化する事が可能である。その上で、ピアグループ内にはランデブー機構が存在し、ピアグループの内部に広告された資源は相互に発見可能となる。つまり、ピアグループはランデブーの境界を決定づける。

一方、ピアグループとは何かという定義はアプリケーション側に任されており、その結果、柔軟なランデブー境界の設定が可能である。一般的に考えられるピアグループの定義を、以下に列挙する。

1. 権限: ある資源にアクセスするための権限を持つピアの集合
2. 協調作業: 作業状態の交換を行う環境 (エディタなど) をピアと定義し、ピアグループを協調作業の場として定義
3. 位置情報: 特定の場所 (座標や緯度経度ではなく、部屋などの文脈的な場所) に含まれるデバイスをピアと定義し、その部屋全体をピアグループとして定義
4. アプリケーション: 特定のアプリケーションが、相互にアプリケーションの利用者を発見するためにアプリケーションごとのピアグループを定義 (インスタントメッセージなど)

このように、JXTA は、そのアーキテクチャの上で、アプリケーション主導で任意のランデブー境界を設定できる。かつ、単一のアーキテクチャの上に様々なアプリケーションを実装可能であり、ランデブー空間の多様性を実現する。

以上、JXTA におけるピアグループというモデルについて述べた。ピアグループは、アーキテクチャが厳密にランデブーの方法を規定する方式 (DNS 等) と、ランデブーの方法が自由だが構造を持たない方式 (Gnutella、Freenet 等) との中間的な方式である。ユーザ側にランデブー方法の決定を委ねつつその機構・モデルを標準化する。

一方、JXTA における問題は、ピアグループそれ自身発見の手法に乏しい事である。発見 (ピアグループレベルのランデブー) が容易かつ柔軟になれば、JXTA アーキテクチャの利点が一層高まると考えらえる。

第 5 章　結論と IDEON 発動に向けて

5.1 まとめ

この報告では、オーバーレイネットワークの概念を整理し、その基礎技術として分散ハッシュテーブル方式と呼ばれる手法を紹介した。更に、分散ハッシュテーブルを用いたオーバーレイネットワークの具体的なデザインの代表例として、数値的に探索空間の分割を行う Chord と、経路表を用いる Tapestry について解説した。また、Chord の応用例として、間接参照により通信上の様々な機構を実現する i3(Internet Indirection Infrastructure) と、ドメイン名システムを改良する DDNS を紹介し、考察した。

その他の技術としては、匿名性の実現に重きを置く Freenet と、プログラミングプラットフォームとしての JXTA について解説した。

我々が特に分散ハッシュテーブル方式に注目する理由は、その簡潔さと効率の良さである。分散システムにおいて、耐故障性、規模拡張性、可塑性を実現する上で、冗長性の確保が必要であることは第 1 章で述べた。冗長性を確保すると、内部状態を共有するための通信のオーバヘッドの面でも、ハードウェア資源が冗長に用いられるという面でも、システムの効率を下げる傾向がある。これに対し、分散ハッシュテーブル方式では、ハッシュテーブルという簡潔なアイデアをネットワーク上に拡張し、基本的には代替可能な機能を各ノードが分担して行うことで、巧みに負荷の分散を行っているという点で注目に値すると考える。

その一方で、この報告により、分散ハッシュテーブル方式を広く用いていく上では、セキュリティについて更なる検討が必要であり、また、あらゆるイ

ンフラストラクチャと同様に、実際に広く展開された後では変更が難しくなるという指摘が行われた。

5.2 IDEON からの提言

IP 層が提供する、グローバルな空間における到達可能性は、人間が広域に渡って自由に、創造的にランデブーし、協調して活動することにより目的を達成する上で非常に重要である。

到達可能性の提供と比較すると付加的な機構であるマルチキャスト、エニキャスト、モビリティ等については、オーバーレイネットワークによる実現が可能であり、そうすることでより容易に実験を繰り返し、完成度を高めることができる。

また、ピアグループを意識してオーバーレイネットワークを構成することにより、あらゆるアプリケーションにおいて、システムの耐故障性や規模拡張性、可塑性を改善できる余地がある。デザイン上、サーバへの問い合わせが集中する箇所の存在を見直し、この報告で紹介した手法等を適用できる可能性がないか、是非、検討して欲しい。

以上を IDEON ワーキンググループからの提言とし、この報告を終える。