

ユビキタス環境に適した宛先記述性の高い間接型通信の実現

松宮 健太¹ 徳田 英幸¹

¹ 慶應義塾大学大学院政策・メディア研究科

近年、家電やセンサなど多様な機器が高い計算能力を備え始めている。近い将来、これらの機器がユーザの周りに多数遍在し、そのサービスが協調してユーザを支援するユビキタス環境の実現が予想できる。しかし多数の機器が遍在する環境では、ホスト名や IP アドレスなどの識別子で通信相手を一意に指定し、一対一の通信を行う現在の通信モデルは適さない。識別子がサービス自体の情報を含まず、サービスとアドレスの対応を把握するのが困難であり、通信相手が利用できなくなった場合や、新しい通信相手が利用できるようになった場合に対応できない。本稿では機能の特長を表し、アプリケーションが指定可能な宛先と、宛先に対応する機器に対するメッセージの配送を提供する通信モデルを提案し、それを実現する通信ミドルウェア SBD(Service Buffer Distribution) の設計と実装を述べる。

1 はじめに

近年、計算機の高性能化により、計算能力を備えた多様な機器が我々の周りで増加している。市販されているものだけでも、携帯電話や PDA などの携帯端末、ハードディスクレコーダなどの情報家電、ゲーム機などがある。また、研究開発段階のものとしては腕時計やアクセサリなどより小型な機器から、センサや有機 EL を用いたディスプレイなど、新しい入出力機能を備えた機器がある。ネットワーク基盤の普及も著しく、ホームネットワークや無線ネットワークなどにより、これまで高速ネットワークが整備されていなかった環境にもネットワークが整備され始めている。このような環境では、機器上で動作するサービスが協調してユーザを支援できる。例えばテレビからレンジや洗濯機などの家電を操作する、隣の部屋に人が居るか、外で雨が降っているかなど様々な情報を光情報や壁に張りつけたディスプレイに表示し、視覚化するなどの利用方法が可能になる。

現在一般的に利用されている通信モデルでこれら機器の協調動作を支援するには、2つの問題がある。まず宛先の柔軟性が低い点がある。現在通信の宛先として一般的に用いられているのは、IP アドレスもしくはホスト名とポート番号の組み合わせである。これらの識別子はサービス自体の情報を含まず、サービスとアドレスの対応を把握するのが困難である。また、これらの識別子はサービスから指定できないが、自己説明的な宛先を用いる場合はサービス自体がその宛先を指定できる必要がある。DNS が割り当てるホスト名はサービスの情報を表すのに利用できるが、管理ドメインの情報が必要なため制約が強い。

また、通信が直接的な問題がある。現在一般的に利用されている通信モデルはユニキャスト通信である。

Indirect Communication with Expressive Addressing for Ubiquitous Computing Environment

¹ Graduate School of Media and Governance, Keio University
5322, Endo, Fujisawa, Kanagawa 252, Japan
E-Mail: <kenta@ht.sfc.keio.ac.jp>

ユビキタス環境では特定のサービスを提供する機器や同様のサービスを提供する別の機器が動的に環境に出入りするため、サービスを提供する機器が頻繁に変化する。そのため、機器と直接通信を行うモデルだけでなく、サービス自体に対して通信を行う通信モデルが必要である。

これらの問題を解決するため、本研究ではサービス自体がその宛先を柔軟に指定でき、それと実際の機器のアドレスを対応付ける通信モデルを提案する。同モデルでは、サービスの宛先はその特長を表す複数の属性から構成される。また、サービス同士は機器に対して直接メッセージを送受信するのではなく、通信ミドルウェアに登録されたサービスの宛先に対してメッセージを送受信する。これにより、通信相手の機器が変化した場合にも動的に対応する。

本稿では、上記の通信モデルを実現する通信ミドルウェアについて述べる。ユビキタス環境で利用される通信ミドルウェアの機能要件として以下がある。

単純性 多数の機器の協調動作を利用するアプリケーションは、通信ミドルウェアを多用することが予想できる。そのようなアプリケーションのプログラマを支援するため、通信ミドルウェアは単純なプログラミングインタフェースを提供する必要がある。

動的な環境の変化への対応 ユビキタス環境では機器が頻繁にネットワークに出入りすることが予想できるため、通信ミドルウェアは環境内で利用できるサービスの動的な変化に対応する必要がある。

管理の容易さ ユビキタス環境における通信ミドルウェアは、ホームネットワークなど、利用者が多様な環境でも利用するため、管理者が管理しなくても動作する必要がある。

効率性 多数の機器が存在する環境では、サービス間での通信量が増加することが予想できるため、効率的な通信ミドルウェアが必要となる。

本稿で述べる通信ミドルウェア、SBD(Service Buffer Distribution)は、文字列の配列を宛先として提供することで単純性を実現する。また、特定のサービスに送信されたメッセージを一時的にミドルウェア内に保持することで動的な環境の変化に対応する。SBDはそれぞれの機器上で動作し、アプリケーション層で動的にネットワークを構築する。新しい機器の参加や機器の削除に動的に適応するため、管理コストは低い。また、サービスの宛先を分散ハッシュテーブルで管理することで効率性を高める。

SBDを用いた場合、メッセージの送受信が仲介されるため、直接的な通信よりも通信の遅延が増加することが予想される。従って動画や音声のストリーミングなど、遅延に敏感なアプリケーションやスループットを必要とするアプリケーションには適さない。逆に、小さなデータを特定のサービスグループに伝搬させるようなアプリケーションにおいて有用性が高い。例えばPDAやノートPCを用いてグループの予定を管理するスケジューラなどに利用できる。グループのメンバが予定を更新するとその更新が伝搬し、更新の際にネットワークに接続していなかったメンバも後からそれを受信できる。また、ユーザの周囲に遍在する機器に対してコマンドを伝搬するようなアプリケーションも考えられる。例えば、光の色で様々な情報を表すライトが、ある部屋の天井に設置されていたとして、新たに部屋の入口にライトを設置すると特別な設定を行うことなく同様の情報を表示するという利用方法が可能となる。

本稿では、2章で宛先記述性と間接型通信についてその実現方法を比較する。3章でSBD通信ミドルウェアの設計を述べ、4章でそのプロトタイプ実装について説明する。5章では基本性能の評価を行う。6章では関連研究について述べ、最後に7章で本稿をまとめる。

2 宛先記述性と間接型通信

本研究の通信モデルでは、サービスはその特長を表す宛先を通信ミドルウェアに登録し、他のサービスはその宛先に対してメッセージを送信する。メッセージは一定期間通信ミドルウェアに保持し、サービスが明示的にそれを読み込むか、あらかじめ通知要求を発行することでそれを受信する。図1の(a)にその概念図を示す。

宛先記述性が高いとは、宛先がサービスの特長を表す複数の属性で構成されることを指す。宛先記述性が高いことで、サービスを柔軟かつ直感的に指定できる。一般的に、宛先の記述性はネーミングシステムやディレクトリサービスによって提供されている。そのようなシステムの例としてDNS[1]やSDS[2]などがある。機器のIPアドレスが変更された際にその宛先との対応付けが更新されれば、IPアドレスを意識することなくメッセージの配送が行える。多数の機器が遍在し、それぞれに手動で固定IPアドレスを割り当てるのが困難

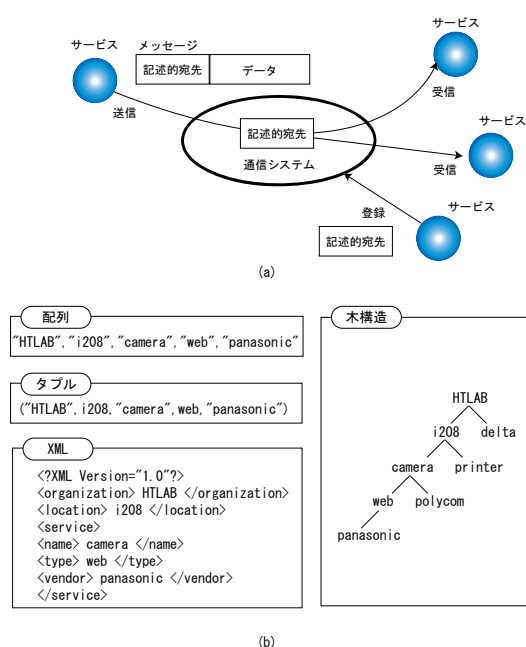


図 1: 通信モデルの概念図

になることが予想できるユビキタス環境では、このような特性が重要になる。

間接型通信とは、機器のアドレスに対して直接メッセージを配送するのではなく、中間的な宛先に対して一旦メッセージを送り、それを機器に配送することでエンドポイントの結び付きを弱める通信モデルを指す。IPマルチキャスト[3]やタブルスペース[4]は間接型通信を実現している。これにより、宛先に該当するサービスの一つが利用できなくなった場合も、同じ宛先を持つ別のサービスに動的にメッセージが配送される。

これらを実現する際に考慮すべき項目として、宛先の構造、宛先の名前解決、宛先の管理、メッセージの配送がある。以下ではそれぞれについてその実現方法を比較する。

2.1 宛先の構造

記述力の高い宛先を提供する場合、その構造を決定する必要がある。図1の(b)は宛先が取り得る構造の例を示す。構造として、単純な配列、タブル、XML、木構造などが考えられる。

宛先の構造として配列を用いる場合、各属性には型を与えない。この方法の利点は、単純さである。逆にこの方法の欠点は、検索効率や柔軟性が他の方法と比べて低いことである。各属性が型と順番を持った配列をタブルと呼ぶ。それぞれの属性が型を持つため、柔軟な宛先を比較的単純に記述できることである。XML(eXtensible Markup Language)を利用する利点は、柔軟にデータ構造が定義でき、かつスキーマを利用することでデータの正当性を検査できることである。タブルやXMLは柔軟性を提供する半面、名前解決や検索が複雑になる

欠点がある。

宛先の構造として木構造を用いる場合、枝を降りるごとに検索範囲が絞られるため、検索効率が高い。逆にこの方法の欠点は、システム側が提供する制約の強い木構造を用いると、それ以外に分類できる宛先を表すのが困難になる点である。例えば管理ドメインの情報がノードとなった木構造では、どの組織が管理しているのか曖昧な機器などの宛先を表すのが困難である。

2.2 宛先の名前解決

通信ミドルウェアが宛先を利用してメッセージの配送を行うには、それを具体的な機器のアドレスに変換する名前解決機能が必要となる。通信ミドルウェアの広域での運用や機器数の増加を考慮すると、宛先の検索を効率的に行う手法が必要となる。効率性を実現するには、まず、宛先の名前空間を複数のより小さい名前空間に分割する必要がある。個々の名前空間を小さくし、それぞれを異なるサーバに分配することでスケーラビリティが上がる。この際、複数のサーバに名前を分散する方法として、名前に情報を埋めこむ分配方法と、ハッシュ関数を利用した分配方法がある。

宛先に埋めこむ情報として、管理組織や位置の情報が利用できる。それにより、名前の構造を保持したまま分配できる。この方法の欠点は、名前に必須属性ができることである。最初からこのような情報を含む名前に対して有効であると言える。ハッシュ関数を用い宛先を分配する場合も名前空間を複数のサーバで管理する。ハッシュ関数を用いて宛先を固定長のビット列に変換し、各サーバに分配する。これにより名前に新たな情報を加えること無く分配できるが、宛先を固定長に変換するため、その記述性が失われる可能性がある。

2.3 宛先の管理

通信ミドルウェアは、宛先をシステム内で保持する必要がある。これらの情報を保持する方法として、集中管理で行う方法、IP ルータ型分散管理で行う方法、アプリケーションルータ型分散管理で行う方法が考えられる。

集中管理で行う場合、次に見る分散型とは異なり、複数のサーバ間で保持する情報の整合性を保つ必要が無い。しかし、保持する情報の増加に従ってアプリケーションからの要求処理や宛先の名前解決によるサーバの負荷が増加するため、スケーラビリティが低い。インターネットのように複数のルータで分散して宛先やメッセージを管理する方法では、一つのサーバで宛先やメッセージを管理する場合に比べて負荷が分散され、宛先の検索効率も上がるため、スケーラビリティが向上する。しかし、宛先やメッセージの整合性を保つための機構が必要となる。

また上の2つの方法では、サーバを他の機器とは別に保守管理する必要がある。サービスが動作している機器自体が宛先やメッセージを管理する方法をアプリケーションルータ型分散管理と呼ぶ。この方法では、

各機器が分散して宛先やメッセージを管理する。この方法の利点は、ルータが機器上で動作するため、サーバやルータを別に管理する必要が無いことである。また、スケーラビリティや耐故障性も確保できる。しかし、実装は複雑になる他、機器に高い処理能力が必要となる。

2.4 メッセージの配送

送信者が受信者に対してメッセージを送ったときに、通信ミドルウェアはそれを受信者に配送する機能を提供する必要がある。配送の方法としてフォワーディングを行う方法と共有空間を用いる方法がある。

IP ネットワークでのパケット配送のように、通信ミドルウェアに渡されたメッセージを適切なサービスに逐次配送する方法をフォワーディングと呼ぶ。この場合、通信ミドルウェアはまず宛先の名前解決を行い、配送先を決定する。配送先が決定した後、その機器に対してメッセージを送る。この方法の利点は、メッセージが逐次受信者に送られるので、遅延が少ない点である。この方法の欠点は、次に挙げる共有空間を用いた方法と異なり、メッセージを通信ミドルウェア内に保持できない点である。これにより、受信者が短期間故障した場合などにメッセージが失われる。

共有空間を用い、メッセージを保持することで、過去に送信されたメッセージを受信できる。これにより、サービスが短期間故障した際にその間のメッセージを取得できる。また、マルチキャスト通信において、新しく追加されたサービスにメッセージを再送する必要が無い。この方法の欠点は、メッセージ受信者がメッセージがあるかどうか一定間隔で検査する必要があるため、遅延が増大する可能性があることである。しかし、実際に利用する際には、メッセージが通信ミドルウェアに渡された際、そのメッセージを待っているサービスに即座に通知する機能が提供される場合も多い。

3 SBD の設計

本稿では、SBD を実行している機器を **SBD サーバ** と呼ぶ。アプリケーションやサービスは、自身が動作する機器の SBD サーバを利用して、メッセージの送受信、サービスの登録、メッセージの通知要求を発行する。サービスに対して送信されたメッセージは、一つの SBD サーバに一度保持され、サービスがそれを読み込むことでメッセージ交換を実現する。サービスに対するメッセージが保持されるメモリ領域を **サービスバッファ** と呼ぶ。

サービスがメッセージを受信するには、まずその宛先を SBD サーバに登録する。SBD サーバはサービスの宛先として単純な文字列の配列を提供する。宛先を登録すると、そのサービスに宛てられたメッセージを保持するためのサービスバッファが作成される。複数のサービスは同一の宛先を登録でき、任意の数のサービスが特定の宛先に宛てられたメッセージを受信できる。どの SBD サーバにサービスバッファを作成するか

は、ハッシュ関数を用いて宛先をハッシュし、それに基づいて割り当てる。

宛先を登録したサービスは、必ず一つのSBDサーバと対応付けられる。本論文ではこのSBDサーバをサービスに対する**中間サーバ**と呼ぶ。サービスは一つの中間サーバに対応付けられるが、中間サーバは任意の数のサービスに宛てられたメッセージを保持する。耐故障性や負荷分散を向上させるため、複数の中間サーバを対応付け、メッセージを複製する方法も考えられるが、本論文では最も基本的な設計としてサービスに対するメッセージは一つの中間サーバで管理する設計を述べる。

SBDサーバ同士は分散ハッシュテーブルを提供するネットワークをアプリケーション層で構築する。サービスとその中間サーバとの対応付けはこの分散ハッシュテーブルを用いて行う。分散ハッシュテーブルは検索鍵と値を複数の機器で分散して管理する構造である。それぞれのSBDサーバは全体の検索鍵と値の一部を管理し、検索アルゴリズムを用いて効率的に値を検索できる。本ミドルウェアが構築する分散ハッシュテーブルは、サービスの宛先を検索鍵とし、その宛先の中間サーバのアドレスを値として対応付ける。

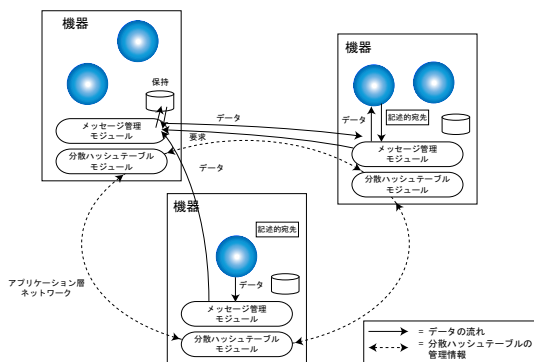


図 2: SBD の全体図

図 2 に SBD の全体図を示す。SBD は 2 つのモジュールから構成される。

メッセージ管理モジュール アプリケーションやサービスに対してプログラミングインタフェースを提供し、メッセージ送信要求、メッセージ受信要求、サービス登録要求、メッセージ通知要求を処理する。また、これらの要求を中間サーバで動作するメッセージ管理モジュールにフォワードする。別のメッセージ管理モジュールから要求をフォワードされると、それを処理する。

分散ハッシュテーブルモジュール アプリケーション層で動的に分散ハッシュテーブルを構築し、それを管理するメッセージの送受信と処理を行う。同モジュールは、記述的宛先と機器を対応付けるテーブルを保持する。初期化時にはこのテーブルを作成し、

その後機器が追加されるたびに更新する。また、検索要求を処理する際もこのテーブルを用いて検索を行う。

3.1 宛先

SBD では、宛先の構造として文字列の配列を用いる。サービスの宛先は、以下のように表される。

“HTLAB” “i208” “camera” “web” “panasonic”

文字列の順序は区別しない。宛先をバイト列に変換する際には、それぞれの文字列を統一的な手法で並び変え、連結した一つの文字列をハッシュ関数を用いて固定長のバイト列に変換する。ハッシュした宛先は、中間サーバを検索する際に用いる。この際完全一致を用いて検索を行う。これは、ハッシュされた宛先を比較する際に部分一致を用いるのが困難だからである。

このような構造を用いることで、サービスの単純な指定が可能となる。逆にこの方法の制約として、宛先に意味を付加できないことと、名前の衝突が起きやすくなることがある。宛先に構造や型を持たせれば、その意味を通信ミドルウェアが解釈し、それに従って異なる扱いができる。例えば組織を示す属性を付加できれば、中間サーバとしてその組織で管理されている機器を割り当てられる。また、順序や型が異なる宛先同士を区別することで、サービスはより多くの宛先を利用できる。本ミドルウェアでは、宛先の意味はアプリケーションとサービス間で共有し、通信ミドルウェアはこれを利用しない。また、複雑な宛先を用い、順序や型を持たせると、宛先の僅かな違いでアプリケーションが意図したサービスにメッセージが配信されないことも考えられる。これを防ぐためには柔軟な一致を用いる必要があるが、その場合異なるサービスを区別するには注意が必要となり、ミドルウェア側の負担が増える。

3.2 プログラミングインタフェース

SBD はアプリケーションプログラマに send, receive, register, requestNotify のインタフェースを提供する。send, receive はそれぞれメッセージの送信と受信を行い、register はサービスの登録を行う。requestNotify は、メッセージの受信を高速化するため、最新のメッセージを即座に通知する要求を発行する。全ての要求に対して宛先を指定する必要がある。要求を処理する際、まずその宛先を一つの文字列に変換する。次にそれを分散ハッシュテーブルモジュールに渡し、中間サーバを検索する。それぞれの要求は検索された中間サーバに対して送信される。

3.3 メッセージの送受信

アプリケーションやサービスは、メッセージを送受信する際、自身が動作する機器のメッセージ管理モジュールに送受信要求として宛先やデータを指定する。要求を受け取ったメッセージ管理モジュールは、まず分散ハッシュテーブルモジュールに宛先を指定し、中間サー

バの IP アドレスを検索する。次に、中間サーバに要求をフォワードする。

送信要求の場合、中間サーバは宛先に該当するサービスバッファにデータを追加する。該当するサービスバッファが存在しない場合はエラーメッセージを返す。データにはタイムスタンプとシーケンス番号を付加する。タイムスタンプは有効期間を検査する際に利用する。検査する際の時間が、タイムスタンプが示す時間と有効期間の和を越えた場合にデータは破棄される。シーケンス番号はデータの順序を管理するのに利用する。

受信要求の場合、中間サーバは宛先に該当するサービスバッファを検索し、それに含まれるデータを取得する。

サービスバッファは、サービスが登録要求を発行すると中間サーバにおいて作成される。これはメッセージ管理モジュールが行う。既に存在するサービスに対して登録要求が発行された場合は新たに作成せず、その有効期間を更新する。

それぞれのサービスバッファは、対応する宛先に送信されたメッセージのデータを保持する。サービスバッファとそのデータはどちらも一定期間保持された後に破棄される。サービスバッファは、有効期間内に登録要求が再度発行された場合に有効期間が更新される。

3.4 分散ハッシュテーブルアルゴリズム

分散ハッシュテーブルは複数のノードで構成されるハッシュテーブルであり、特定の検索鍵と値を一つのノードに割り当てる。検索鍵を指定するとそれを割り当てられたノードを特定できる。本研究では、一つの SBD サーバを一つのノードとして扱う。ノードの検索を効率的に行うためのアルゴリズムとして Chord[6], CAN[7], Tapestry[8] などが提案されている。これらのアルゴリズムは検索を効率的にし、ノードの動的な追加と削除に対応する。それぞれ共通するのは、複数のノードで仮想的な識別子空間を構築し、各ノードに均等に検索鍵を割り当てることである。本研究では Chord を利用している。

Chord では、 m ビットにハッシュされた検索鍵と、同じく m ビットにハッシュされたノード ID を、同一の仮想的な環状の名前空間に割り当てる。ノードとノードの間にある検索鍵は、時計回りに見て一つ先にあるノードに割り当てられる。また、各ノードは自身の次のノードを把握する。これにより、全ての検索鍵について、それを担当するノードを検索できる。検索をより効率的にするため、各ノードは検索鍵とそれを担当するノードを対応付けるテーブルを保持する。各ノードが保持するテーブルは、 m 個のエントリで構成され、自身の ID を n とした場合、それぞれのエントリは $n + 2^{k-1} \bmod 2^m (1 \leq k \leq m)$ の検索鍵に対する担当ノードの情報を含む。図 3 に示した例では、名前空間は 3 ビットで構成され、ノードが 3 つ存在する。ノード 0 のテーブルには、 $0 + 2^{k-1} \bmod 2^3 (1 \leq k \leq 3)$ の

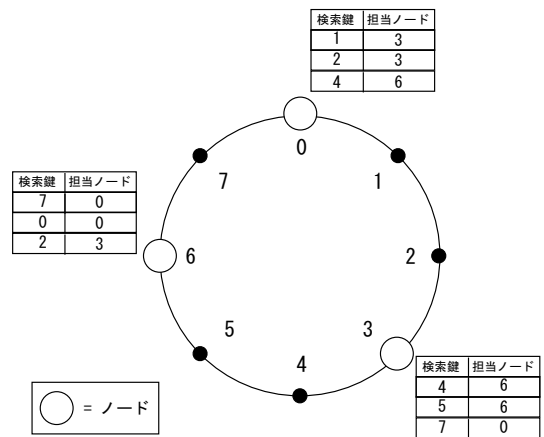


図 3: Chord 名前空間の例

検索鍵に対する 3 つのエントリがある。検索鍵 1, 2 については、その次のノードがノード 3 であるため、それが対応付けられている。検索鍵 4 については、ノード 3 を越えるため、さらに次のノード 6 が対応付けられる。検索鍵 3, 5, 6, 7 の検索はノード 3 にフォワードされる。検索鍵 0 については自身が担当するため、把握する。

自身が新たに参加されたことを他のノードに通知する際は、 $n - 2^{k-1} \bmod 2^m (1 \leq k \leq m)$ の検索鍵の前のノードに、 k 番目の検索鍵に対する担当ノードを自身に設定するように要求する。

特定の検索鍵の担当ノードを検索するには、自身のルーティングテーブルを利用する。ルーティングテーブルに指定された検索鍵のエントリが存在する場合は対応付けられたノードが返される。該当するエントリが無い場合は、検索鍵の直前のノードに検索要求がフォワードされる。このときテーブルの性格により、検索鍵との距離は少なくとも半分になる。そのため、 $O(\log N)$ の検索効率で検索が行える。

3.5 分散ハッシュテーブルの管理

分散ハッシュテーブルの構築は、SBD サーバが初期化される際に行う。既に構築された分散ハッシュテーブルに参加する場合、分散ハッシュテーブルモジュールの初期化時にそれを構成する SBD サーバ一つのアドレスを指定する。新しい SBD サーバはそれを利用し、自身のテーブルの初期化と他の SBD サーバのテーブル更新を行う。まずテーブルの各検索鍵を初期化し、それを担当する SBD サーバの検索要求を発行する。既存の分散ハッシュテーブルに参加しない場合は、自身を唯一のノードとして新たに構築する。

表 1 にテーブルの例を示す。番号はテーブル更新要求を処理する際に利用される。各検索鍵はそれを担当するノードの ID とアドレスに対応付けられる。

分散ハッシュテーブルモジュール同士は、新たな SBD サーバの追加や検索鍵を担当する SBD サーバを検索

表 1: テーブルの例

番号	検索鍵	ノード ID	アドレス
1	67492142	67492142	133.27.170.2
2	67492143	74596704	133.27.170.79
3	67492145	74596704	133.27.170.79

するため、管理メッセージをやりとりする。表 2 にその形式を示す。

表 2: 管理メッセージの形式

送信元アドレス	宛先アドレス	種類	データ
---------	--------	----	-----

3.6 アプリケーション記述例

以下に、SBD を利用したアプリケーションの記述例を示す。サービスとして、i208 という名前の部屋に設置された、異なる色を表示できるライトの機能を提供する LightService を考える。また、そのライトを制御する LightController アプリケーションを考える。どちらもまず、getInstance メソッドを用いて、MessageManager のインスタンスを取得する。LightService は同一インスタンスに対してサービス登録要求とメッセージ通知要求を渡す。また、メッセージが通知された際に実行する messageNotified メソッドを実装する。LightController は、MessageManager のインスタンスに対して“BLUE” という文字列の送信要求を渡す。同文字列は LightService に対して通知され、LightService はこれを処理する。

```
import jp.ac.keio.sfc.ht.kmsf.sbd.*;

public class LightService
    implements MessageListener{
    MessageManager manager;

    String[] myDescription =
        {"i208", "light", "color"};

    LightService(){
        manager=MessageManager.getInstance();
    }

    void register(){
        manager.register(myDescription);
        manager.requestNotify(myDescription);
    }

    void messageNotified(Message msg){
        processMessage(msg.data)
    }
    ...
}
```

```
import jp.ac.keio.sfc.ht.kmsf.sbd.*;

public class LightController{
    MessageManager manager;

    String[] lightDescription =
        {"light", "i208", "color"};

    LightController(){
        manager=MessageManager.getInstance();
        turnLightBlue();
    }

    void turnLightBlue(){
        manager.
            send(lightDescription,"BLUE");
    }
}
```

メッセージの宛先として文字列の配列を利用するため、単純な方法でサービスを指定できる。また、配列の各要素の順序が異なっても同一の宛先として扱われるため、アプリケーションプログラマはサービスの宛先に含まれる要素の順序を考慮する必要が無い。

4 SBD の実装

Java 言語を用いて SBD のプロトタイプ実装を行った。全体のソースコードは約 2300 行で構成される。SBD サーバ同士は UDP を用いて通信を行うため、それぞれのモジュールごとに 2 つのポートを利用する。メッセージ管理モジュールは、アプリケーションやサービスの要求を、分散ハッシュテーブルモジュールは、モジュール間で管理メッセージを送受信する。

本プロトタイプ実装では、それぞれの機器に一つの分散ハッシュテーブルモジュールを作成する。従って、複数のアプリケーションやサービスの要求は同一の分散ハッシュテーブルモジュールによって処理される。分散ハッシュテーブル内での SBD サーバの ID は、機器の IP アドレスをハッシュして生成する。

メッセージ管理モジュールは、サービスバッファの追加を効率的に行うため、それらをハッシュテーブルで保持する。また、それに含まれるデータはリストで保持する。

5 基本性能評価

本節では、プロトタイプ実装の基本性能を測定した結果を示す。メッセージに含まれるデータ量に対する応答時間の変化と SBD のサーバ数に対する応答時間の変化を測定する 2 つの実験を行った。どちらも SBD の実行には同種類の計算機を用いた。表 3 にその測定環境を示す。

図 4 にデータ量に対する応答時間の変化を示す。本測定では、機器 2 台で構成された SBD でのメッセージの配送時間を測定した。数値を記録する側の機器は、サービスの登録とメッセージ通知要求を送った後、自身の記述的宛先を指定したメッセージを送り、その通

表 3: 測定環境

項目	測定環境
ハードウェア	Compaq EVO D500SF
CPU	Pentium4 1.6GHz
OS	FreeBSD 4.0
メモリ	256MB
ネットワーク	100Mbps Fast Ethernet

知が届くまでの時間を記録した。メッセージ自体はもう一方の機器に保持された。機器が2台だけなので、ルーティングのオーバーヘッドは無く、測定値は純粋なデータ配送にかかる時間を示す。

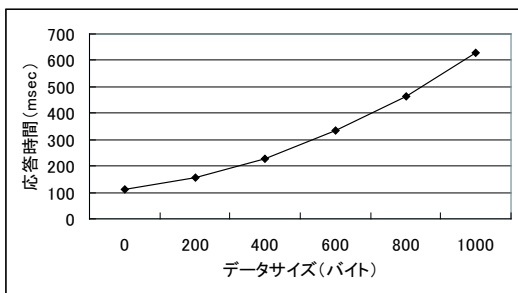


図 4: データ量に対する応答時間の変化

グラフでは、データサイズの増加に対して応答時間が増加している。これは予想された結果と言える。データサイズが0バイトでも100ミリ秒かかっており、配送自体の遅延が大きい。

図5にデータ量が0バイトのメッセージの配送時間を、異なる数のSBDサーバを用いて測定した結果を示す。宛先はランダムに生成し、メッセージ送信者がメッセージを送信してからそれがサービスバッファに保持されるまでの時間を記録した。

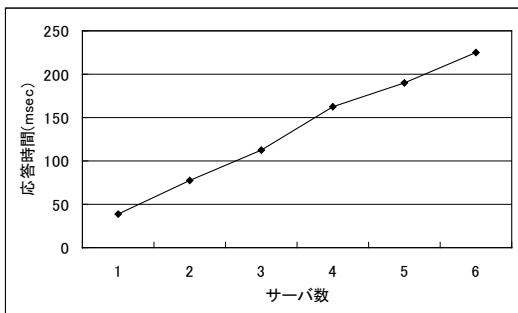


図 5: サーバ数に対する応答時間の変化

グラフでは、サーバ数に対して応答時間が増加している。これはルーティングに要する時間の増加を示している。サーバ数が少ないため、応答時間は単調増加

している。今回の測定に用いたサーバ数でもメッセージの送信に最大で200ミリ秒以上かかっており、ここでもメッセージ配送自体の速度面での改善が必要であると言える。

6 関連研究

SBDに関連する研究として、ネーミングシステムとメッセージ配送システムが挙げられる。

SBDは、サービス発見に利用した場合、ネーミングシステムの役割を果たす。現在一般的に利用されているネーミングシステムとしてはDNS[1]やLDAP[9]がある。DNSはインターネット上のホストに対して、管理組織の名前を含む木構造の名前を割り当て、それをIPアドレスと対応付ける。LDAPでは、ホストを表すプロパティを指定することでホストを発見し、DNSよりも多様な検索要求が発行できる。LDAPでも各プロパティを連結することで木構造の名前を提供できる。DNSやLDAPでは、アプリケーションの外部からホストの情報を明示的に登録する必要がある他、アプリケーションが割り当てられる名前に対する制約が強い。

Sun MicrosystemsのJava Spaces[10]は、複数のサービス同士がデータを交換するための共有空間である。サービスは、Entryと呼ばれるオブジェクトの集合をこの共有空間を介して交換する。サービスはEntryの書き込みと読み込みの操作が行える。多くの場合EntryにはJava RMIによるリモートオブジェクトへの参照が含まれる。Entryを読み込んだプロセスはそれを利用して直接リモートオブジェクトの機能を利用できる。実際の通信はこのように多くの場合RMIを利用して行うため、Java Spacesは共有空間を用いてデータを交換するよりも、サービスの検索を行うことを重視していると言える。それに対して本研究では、共有空間を介してデータ通信を行うことを目的としている。また、本研究では分散ハッシュテーブルを用いることにより宛先の効率的な分配を行っている。

MITで開発されているINS(Intentional Naming System)[5]は、宛先を用いてサービスの検索とメッセージの配送を同時に行うメッセージ配送システムである。宛先として、木構造を持った名前を利用する。宛先に含まれるノードはサービスが定義できるため、DNSのように要素に対する制約が高いシステムよりも柔軟である。また、リゾルバが動的にネットワークを構築するため、管理コストも低く、効率性を実現する機構も提案している。本研究との相異点は、INSがメッセージ配送にフォワーディングを用いているのに対して、本研究では共有空間を用いている点である。これにより、サービスの追加と削除により動的に対応できる。

U.C. Berkeleyで開発されているI3(Internet Indirection Infrastructure)[11]は、サービスと識別子に対応付け、その識別子を介して通信を行うことで間接型通信を実現するメッセージ配送システムである。サービスの識別子は、本研究と同様分散ハッシュテーブルを用

いて、複数の機器に分配、検索される。分散ハッシュテーブルは複数の機器が動的に構築する。識別子の生成方法は指定されていないが、上位のシステムで本研究同様サービスの記述をハッシュすることでも生成できる。また、分散ハッシュテーブルの利用により管理の容易さと効率も実現される。しかし、I3 では宛先を識別子としており、その構造を定義していない。また、メッセージ自体は保持されないため、動的な環境への対応が充分でない。

NTT 未来ねっと研究所で開発されている SIONet[12] は、意味情報を宛先として利用したイベント配送システムである。イベント受信者は、SIONet に対して興味のあるイベントの種類や適合条件を指定するフィルタを登録する。SIONet に対して送信されたイベントは、意味情報スイッチ (SI-SW) と呼ばれるソフトウェアがそのフィルタに従って適切な受信者に配送する。異なる SI-SW はアプリケーション層で動的にネットワークを構築する。SIONet では SBD と同様に柔軟な宛先を用いており、フィルタに複雑な適合条件を指定できる。しかしそれにより適合のオーバーヘッドが発生し、また異なる SI-SW 間のイベント配送にはフラッディングを用いているため、効率性を実現するのが困難である。

7 おわりに

本稿では、ユビキタスコンピューティング環境において現在の通信ミドルウェアで問題となる記述性の低い宛先と直接的な通信を解決するため、宛先記述性の高い間接型通信を提案した。同通信モデルは、複数の属性から成る記述性の高い宛先に対して、任意の数のサービスを対応付けることで、機器数の増加やその可用性の動的な変化に対応する。

本研究で設計、実装した SBD 通信ミドルウェアは、文字列の配列で表される単純な宛先を提供した。また、複数のサービスをこの宛先に対応付けることで、機器の動的な変化に対応できる。それぞれの機器上で動作する SBD は、アプリケーション層で動的に分散ハッシュテーブルを構築することで、複雑な管理を必要とせず、宛先の効率的な検索を実現した。

今回の設計、実装では、SBD の基本的な機能を重視したため、セキュリティについては考慮しなかった。しかし、任意のサービス同士の通信を考慮する場合、メッセージ送信者の認証とメッセージの暗号化が重要になる。ユニキャスト通信でメッセージ送信者の認証を行う場合は、メッセージの送信者と受信者が鍵を共有する。しかしメッセージの受信者が複数存在する通信で同様の手法を用いると、受信者が独自にメッセージを作成し、別の受信者に送信者と偽ってメッセージを送信できるため、独自の認証方法が必要となる。

また今回の設計と実装では、宛先の名前解決方法として完全一致を用いた。しかし部分一致を用いることでより柔軟な検索が可能となる。これを実現する際の問題は、ハッシュされた値を用いての部分一致が困難

な点である。一つの方法として、宛先を構成する配列の各要素を異なる組み合わせごとにハッシュし、もとの宛先の間サーバへの参照と対応付ける方法が考えられる。しかしこのような方法は通信ミドルウェアをより複雑にし、効率性が失われる可能性があるため、注意深く設計する必要がある。

参考文献

- [1] Mockapetris, P. V. and Dunlap, K.: Development of the Domain Name System, *Proceedings of SIGCOMM '88*, pp. 123–133 (1988).
- [2] Czerwinski, S., Zhao, B. Y., Hodes, T. D., Joseph, A. D. and Katz, R. H.: An Architecture for a Secure Service Discovery Service, *MobiCom '99* (1999).
- [3] Deering, S. E. and Cheriton, D. R.: Multicast routing in datagram internetworks and extended LANs, *ACM Transactions on Computer Systems*, Vol. 8, pp. 85–110 (1990).
- [4] Carriero, N. and Gelernter, D.: The S/Net's Linda Kernel, *Transactions on Computer Systems*, Vol. 4, No. 2, pp. 110–129 (1986).
- [5] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H. and Litley, J.: The design and implementation of an intentional naming system, *Symposium on Operating Systems Principles*, Kiawah Island, SC, ACM (1999).
- [6] Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, *SIGCOMM '01 Conference*, San Diego, California, ACM, pp. 149–160 (2001).
- [7] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S.: A Scalable Content-Addressable Network, *SIGCOMM '01 Conference*, San Diego, California, ACM (2001).
- [8] Zhao, B. Y., Kubiatowicz, J. D. and Joseph, A. D.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Technical Report UCB/CSD-01-1141, UC Berkeley (2001).
- [9] Smith, M. and Howes, T.: *LDAP – Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, Macmillan Technical Publishing (1997).
- [10] Sun Microsystems, Inc.: JavaSpace Specification (1998). <http://java.sun.com/products/jini/specs>.
- [11] Stoica, I., Adkins, D., Zhaung, S., Shenker, S. and Surana, S.: Internet Indirection Infrastructure, *SIGCOMM '02 Conference*, Pittsburg, Pennsylvania, ACM, pp. 73–86 (2002).
- [12] 星合隆成: P2P の理念およびその実現技術: SIONet の全貌, *Jnutella Workshop* (2002).