第 XXV 部

Integrated Distributed Environment with Overlay Network

第 25 部
Integrated Distributed Environment with Overlay Network

第 1 章 Introduction

1.1 IDEON revisited

IDEON (Integrated Distributed Environment with Overlay Network) is a working group of researchers who try to approach realization of the integrated distributed environment (IDE) through construction of overlay networks (ON).

IDEON focuses on research, development and operation of overlay networks as an infrastructure to realize free and creative rendezvous, location and routing (Table 1.1).

Our research topics include, but not limited to, the following:

- Application-layer multicast
- Operable distributed hash tables
- Self-sustained trust management for distributed autonomous systems

1.2 Projects

Table 1.2 shows the list of specific development projects in IDEON as of January 2005.

For the detailed plans of those projects, please refer to the web pages of each project. The list of pages can be found in the URL below.

`http://member.wide.ad.jp/wg/ideon/?en%2FProjects`

1.3 Activity plans

Activities in IDEON can be outlined as follows:

$$\{understand \rightarrow build \rightarrow try\}* \rightarrow deploy$$

(where $*$ denotes repetition.)

Currently we are primarily at "*build* → *try*"

Table 1.1. Redefined terminology in IDEON

Terms	Meaning
Free	Having no restriction whatsoever as to with which peers one can communicate.
Creative	Being able to select a set of peers according to one's objectives, requirements, needs and contexts so that communication becomes most valuable for the participants.
Rendezvous	To identify such a peer.
Location	To locate such a peer in the overlay networks by the acquired identifier.
Routing	To deliver a message to such a peer on the acquired location.
Overlay Network	An application-specific virtual network of peers over the IP network to realize rendezvous, location and routing over an appropriate abstraction of entities.

Table 1.2. Specific development projects in IDEON (January 2005)

Project Name	Description
wija	A pluggable XMPP/Jabber client intended to be a testbed for pursuing all possibilities of communication.
libcookai	A Pastry-based distributed game library.
enhanced eigentrust	A trust management.
mchord	An implementation of Chord.
PuchiChord	An implementation of Chord as a research platform.
n_chord (provisional)	Dynamic optimization of distributed hash tables.

phase. Our goal is to bring all development projects into "deploy" phase by the end of fiscal year 2005.

1.4 Topics covered in this report

The rest of this report covers the following topics:

1. Discussions on Multi-Overlay Architecture over the Internet
   - Clarification of the philosophy put forward in IDEON.
2. Efficient Searching Using DHT Over the Unstructured Name Space on DNS
   - An application of DHT (Distributed Hash Table).
3. Distributed Scalable Multi-player Online Game Servers on Peer-to-Peer Networks
   - Description of the design of libcookai.
4. WOT for WAT: Spinning the Web of Trust for Peer-to-Peer Barter Relationships
   - Reasoning behind *i*-WAT, a complementary currency plug-in for wija.

第 2 章　Discussions on Multi-Overlay Architecture over the Internet

abstract

We argue that the overlay network approach is a solution to numerous issues concerning the Internet such as mobility support, identification of entities without computing resources, security and multi/anycasting. The major advantage is that an overlay solution is essentially independent of other overlay solutions. Thus, there is no risk of conflict between solutions. Moreover, overlay can introduce tailor-made addressing and routing schemes for each application.

To realize overlays in the form of modular components for network applications, we focus on the interface between overlays. We propose an interconnection model that enables an overlay network for network exchange to act as a market for other networks.

We conclude that overloading the network layer to resolve issues is not only difficult, but also inimical to other solutions. To let the network layer as commons for future innovations, it will be necessary to consider alternative approaches such as an overlay model before we decide to overload.

2.1 Overlay Networking as a Solution

This chapter discusses employment of overlay networks to resolve contemporary issues concerning the Internet. The issues include the mobility of entities that do not fit in a physical node, identification of targets without computing resources such as people, security in an untrustworthy network, trust management among a multitude of unknown nodes, and node-finding (rendezvous) in an ubiquitous networking environment.

Fundamental to our overlay approach is the management of identifier spaces that are independent of IP addresses. Such identifier spaces should be capable of providing applications with appropriate ways for rendezvous, location and routing.

We argue that decoupling identifiers and the Internet protocols is preferred to resolving the above issues within the network layer, which sometimes contradicts against the design of the layer and/or overload it.

Our Communication Model

The following is a description of the communication model we use at IDEON (Integrated Distributed Environment with Overlay Network), a working group in the WIDE Project that pursues autonomy in the designs of distributed systems.

We think that network designs should encourage self-generation of activities which utilize resources spread among different locations (hence, *integrated distributed environment*) by allowing spontaneous creation of layers of abstract network over the network layer (hence, *with overlay network*).

Putting more stress on autonomy changes

how the three ingredients of communication are performed:

1. Rendezvous (or how to identify the peer)
   The word "rendezvous" means a prearranged meeting place. In computer communications, such a meeting place can be a name space or a space for identifiers. Rendezvous performed autonomously allows spontaneous naming and resolution among the participating nodes.
2. Location (or how to locate the peer)
   This is to locate the node that represents the identifier. The node is typically identified by the identifier in the lower layer of the network, an IP address in most cases. Autonomous location involves identifying the closest copy of information among redundant copies spread over the network with help from participating nodes in the vicinity.
3. Routing (or how to reach the peer)
   This is to traverse the topology of the network so that a message can reach the peer. Location and routing can be done in the same procedure because it is necessary to traverse the topology of the network to locate the peer. Autonomous routing will involve creation of topology in an ad-hoc manner.

We propose alternative networking designs so that each of these can be performed in unrestrained and imaginative ways.

By "unrestrained and imaginative" we mean that no restraint should be imposed by the network as to which object can become the target for communication, without intervention of any authorities or privileged intermediate nodes, and that new ways of communication can be developed by the creativity of the participants of the network.

Since autonomy implies that there is no authority to guarantee the truthfulness of information (or that such an authority is weak), trust becomes an important issue.

In addition, the recent ubiquitous network environment makes the Internet heterogeneous and connected with real space. The only network we know of that has a variety of nodes is the current Internet, and the trend toward greater variety will continue as the mass of nodes increases. Real space is also integrated in the network. At least at 1991, we had a sensor connected to the worldwide network, the famous coffee pot[113] in Cambridge University. Many other projects, such as tangible bits[148], integrates real space and network.

2.2 Issues concerning the Internet

We think that an integrated distributed environment with overlay network constitutes an effective way of redesigning the current Internet. Thus, We consider issues concerning the current Internet to be opportunities to introduce new designs. In this section we focus on such problems and solutions.

2.2.1 Endpoint Naming, Locating, and Routing

There are many issues concerning message routing and endpoint granularity. In the current Internet architecture, almost all messages are routed into a physical network interface card (NIC), and the computer on the NIC hands the payload of the message to a process to which a port number is assigned.

Binding between message routing and physical NIC is an essential constraint of the Internet. To overcome it, Roma Project[291], for example, creates its own routing scheme to route the message to a person regardless of its connectivity. This is an outstanding example of an application that uses the overlay network to decouple the endpoint of a message from the underlay network endpoint to satisfy the needs of applications.

Another kind of application introduces a more symbolic and abstract location such as "temperature of the room" or "energy consumption of this building." In many cases, endpoints of this kind do not exist. Some works of sensor fusion, eg. IrisNet[106], create virtual and abstracted nodes (organizing agent nodes) decoupled from

physical sensor nodes. The virtual nodes collect data from one or many physical node dynamically. The data can be altered or integrated in the virtual node. And an overlay network structure provides naming, locating, and routing between client and such virtual nodes.

Decoupling of identifier spaces is a fundamental approach for node mobility, too. Mobile IP (MIP)[244] and LIN6[171] are efficient approaches for mobility and they decouple node addresses and network addresses. Although MIP is a standard of the Internet Protocol suite, MIP does not decouple name spaces of nodes and networks completely. On the other hand, LIN6 is designed to decouple node identifiers and network identifiers. MIP and LIN6 are efficient because their routing is tightly coupled with the IP routing.

For mobility, there are many alternative approaches that satisfy various needs. Due to strong relation with the IP-layer routing, MIP and LIN6 are not applicable to non-node mobility. For example, processes, contexts, or sessions can not move in the case of these solutions.

Some of contemporary approaches to node mobility uses an overlay network. Internet Indirection Infrastructure (i3)[287] is one of the best approaches for mobility using an overlay network. Their approach creates a concrete overlay network that is capable of naming, locating, and routing. In i3, each node forms a network of a distributed hash table and works as a message router over the network. Any nodes can "listen to" any points in the network. This approach decouples not only naming, but also locating and routing.

The major advantage over other mobility approaches is its flexibility. Because it has an independent routing and naming mechanism, not only physical hosts but also processes or other objects can handle its session as an overlay node. Moreover, ROAM (Robust Overlay Architecture for Mobility)[333], a mobility extension of the i3 approach, introduces a control mechanism of tradeoff between efficiency and privacy by controlling trigger insertion randomness.

2.2.2 Security and Overlay

Security and trust involve numerous issues that are difficult to resolve. In this chapter we focus on two issues concerning security and trust, namely, node quarantine to keep infected or malicious nodes out of the network, and service selection among unfamiliar and unknown nodes.

A node quarantine model for IPv6 node security was proposed in an Internet Draft[168]. It involves control of the datalink layer such as VLAN or Internet layer association (route advertisement in IPv6) to quarantine dubious nodes and protect other nodes from attacks. The approach is tightly coupled with the IP or datalink layer. Using some security audit mechanism, a node will be certified as "clean." And then it can enter the network.

Our approach can be applied in the same manner, but it utilizes a higher layer and decouple the secured state from the network layer. The regular network layer "outside" is considered to be dirty and the overlay network inside is kept clean.

Another issue concerning security and trust is service selection. To select a trustworthy service provider from all the providers around a client node, some sort of trust management among providers is necessary. At the same time, a service provider needs to distinguish malicious client nodes among accessing client nodes.

To satisfy these requirements, each node must be able to manage its own trust and that of others. A trust network, *i*-WAT[272], creates a peer-to-peer style overlay network to maintain and exchange trust between nodes. Using such a network, a node can decide which nodes to trust, or select a better one among unfamiliar nodes.

2.2.3 Overloading Causes Contradiction

As pointed out in the previous sections, the Internet involves numerous issues and needs. Clearly, the Internet Protocol is incapable of satisfying all those needs simultaneously. For example, mobility on or above the transport layer and addressing and locating a non-IP object such as a person are solved beyond the network layer.

Although design of network layer is limited to identification, locating, and routing of network interface, there are many demands to overload the Internet Protocol and IP addresses.

For example, an IPv6 address is a unique identifier in the Internet. Thus, re-using an IPv6 address as an identifier of a device, node, or person is an attractive idea. But such overloading is an abuse of the IPv6 address, which is not designed to support it.

Meanwhile, multicasting and anycasting overloads addressing and routing of IP. In multicasting, an address is shared among some set of interfaces and a special routing protocol is required to send messages to all of the interfaces. This essentially means that identifier of NIC is overloaded as a group of NICs, and the overloading enforces special handling of the identifier in every node participating in the Internet.

For example, multicasting requires PIM or another multicast routing protocol to be usable between sender and listener. Because not all routers support PIM and there is no evidence that PIM scales well against Internet-size, enabled area of multicasting is somewhat limited, or tunneling, a very primitive overlay, is applied to bridge between those enabled areas.

Anycasting is similar but involves more complex overloading against addressing and routing. In anycasting, interfaces share an address, as they do in multicasting. The difference is that only one interface at a time can receive a message sent to the address.

In [231], it is pointed out that anycasting essentially involves issues of security and scalability. Although well-known anycast addresses for well-known services is an attractive idea for a service finding scheme, it also causes overloading of the IP routing.

Overloading is not always undesirable. However, it sometimes causes contradiction between two or more overloading technologies.

For example, ingress filter[98] conflicts with the earlier version of Mobile IP. Ingress filter prevents malicious attackers from spoofing its source address. However, a node with the earlier version of Mobile IP sends messages with its home address, and the messages are dropped by the filter. Due to this contradiction, Mobile IP for IP version 4 is required to use tunneling between a home agent and a mobile node to communicate. With redesign involving the network layer, Mobile IP version 6 behaves well against ingress filtering.

Great care must be exercised in overloading the network layer. Otherwise, contradiction with other technologies will undermine efficiency or, in the worst case, result in failure of deployment. At the same time, we believe this kind of overloading technology requires long and difficult standardization process (at present, mobile IPv6 draft revision is 24) to make sure that the technology is safe and harmless to other technology. Moreover, once standardized, the overloading technology becomes another obstacle to the introduction of new technologies.

2.3 Coordination of Multi-Thin Overlays

2.3.1 Multi-Thin vs. Single-Ultimate

It is necessary to investigate two architectures, namely, multi thin (MT) overlay network and single ultimate (SU) overlay network, and select whichever is better. The advantage of MT overlay network is its extendibility. For each new application, an overlay network can be created that has a naming and routing system optimized for the application.

At the same time, MT overlay introduces complexity. Lack of interconnectivity between MT overlays would be inconvenient. The resources on the overlay cannot be accessed from outside the overlay. To overcome this restriction, a syndication mechanism or a set of syndication mechanisms would be needed.

Pros and cons of SU overlay network are the verse of those of MT overlay. Since SU overlay introduces a widely applicable naming and routing framework, most applications would perform

well within the framework. At the same time, however, a user wishing to use an application beyond the framework must modify the application or construct a new framework suitable for the application. One outstanding example of SU overlay network is Project JXTA (see `http://www.jxta.org/`).

2.3.2 Future Network Architecture with MT Style Overlays

To envision future network architecture, we believe MT style overlays are required in order to solve issues without introducing any contradiction. However, isolated multi-thin overlays do not utilize resources on the net well. With MT overlays a node on an MT overlay cannot use resources on another overlay.

Thus, interconnection of MT overlay is required to enable users to find better resources from all over the network. There are two approaches to interconnect MT overlays, namely the gateway approach and the client-side approach.

A gateway approach such as FLAPPS[192] introduces a protocol translation gateway or a proxy node in an overlay to gain access to other overlays.

Although gateway approaches are advantageous in that interconnection is transparent to clients, and thus implementation of a client software becomes easy, there is also a disadvantage.

There is a risk of contradiction as same as overloading. If an application's requirement spoils another application's requirement for the gateway, implementation of the gateway becomes difficult.

In a client-side approach, everything is left up to the client process. There is nothing to interconnect multi overlays except user and pointer. URP (Uniform Rendezvous Pointer)[61] is a uniform pointer format for overlay networks. A URP is sufficient for a client to indicate which network to connect, how to search a resource in the network, etc. A URP has sufficient messages to make network be connected, to perform effective resource search, and so on.

A client program is capable of accessing resources on overlay network with plug-in software. When a client (or user) requires resource pointed by a URP, and it requires access to a network other than that which the client is currently on, it can obtain a new connection via corresponding plug-in. To trigger the interaction, resources in the overlays can contain URP to other networks.

The following is an example of a URP scenario. In the scenario, nodes A and B are on file exchange network X, and nodes A and C are on file exchange network Y. There is an *i*-WAT network to exchange values.

As node A gives its own work to node B, node A requests a certain amount of payment over the *i*-WAT network. The request contains a URP that describes how node B can access *i*-WAT node of node A. With the URP node B can complete its payment.

Then, node A wants another file from node C. Because node A has received a payment from node B, node A can pay for the file with node B's payment. With those interaction, *i*-WAT works as a market regardless of which file exchange network is used.

Another set of networks would be integrated to another kind of application. The advantage of client-side interconnection over the gateway approach is that an application developer can select a set of overlays including a newly created tailor-made network. Also, unlike in gateway approaches, since the interconnection point is isolated in client software, there is no risk of contradiction arising.

2.4 Conclusion and a Vision of the Future Internet

In this chapter we discussed a vision of the future Internet. Our model of the renovated Internet consists of many kinds of independent overlays to fulfill requests for applications in a satisfactory manner.

We have described our communication model,

which is a fundamental model of overlay, and have applied it to resolve various issues concerning the Internet.

Although overloading the network layer is an efficient way of resolving issues, we argue that overloading sacrifices innovation to some extent. Contradictions will arise between other technologies and the overloading technology, dictating the choice of an ineffective way of finishing the standardization process.

From our perspective, the Internet is sufficient if the followings are satisfied.

- Routability: messages sent to an interface should arrive at the interface
- Efficiency: routing between each interface reflects the structure of the datalink layer configuration as precisely as possible
- Scalability: more than 10 to 1000 times the Earth's population are able to connect to one another

The final item is rough estimation of the ideal ubiquitous network environment and sensor network scenario. In the ideal scenario, every location has intelligence to support the user's activity.

Technologies to overload the Internet Protocol, particularly in regard to routing, are inimical to such tremendous scalability. Prior to clarifying an approach capable of realizing scalability of this order, it will be necessary to consider alternative approaches such as an overlay model.

The software development process requires a modular structure to enable readability and suppress maintenance costs, rather than monolithic efficiency. Spurred by the recent introduction of more powerful computer hardware, networked applications can become more modularized to enable flexibility and simplicity. If the modularized approach proves to be effective, it will be necessary to focus on the interface between protocols, not on the protocol itself. The overlay approach is found to be highly effective for modularizing protocols. Thus, thorough investigation of the interface between overlays can facilitate the sound development of the Internet.

第 3 章　DHT を用いた DNS 上の非構造な名前空間の検索効率化の一検討

概要

モノの ID など、明確な構造を持たない（非構造）かつ、膨大な数の名前を扱う必要のある名前空間は、商品トレーサビリティの実現などに際して必要とされている。しかし、インターネットの標準的な名前解決機構である DNS（Domain Name System）では、この名前空間は必ずしも効率的に扱えない。一方、非構造かつ膨大な数の名前の取り扱いに適した名前解決機構の 1 つに DHT（Distributed Hash Table）があるが、DHT を効率的に利用するためにはクライアント側に対応するソフトウェアを導入する必要があり、DHT を適用する際の障壁の 1 つとなる。本研究では、この 2 つの問題に対する解決策として、DNS の特定のゾーンに DHT を効率的に結合し、DNS の一部として動作させる方式を提案する。このため、提案方式は DNS から DHT を呼び出す際に 2 つに分割したゾーンを用いる。その上で、最初のゾーンにおいては問い合わせ処理の負荷において最適化し、2 つ目のゾーンにおいては DHT ノードの並列度をそのまま用いることで、並列処理による最適化を行う。その結果、DNS と DHT がそれぞれ持つ、分散によるスケーラビリティ特性を損わず両者を結合する。同時に、本方式の問題点の 1 つである劣化キャッシュによる問い合わせ失敗という現象に関し、シミュレーションを通じて現象の発生を回避するための条件を検討する。

3.1 はじめに

3.1.1 ID を中心とした情報管理における課題

ユビキタスネットワーク、つまりどこにいてもコンピュータネットワークが利用可能な状態において有用と考えられる応用の 1 つに、コンピュータによる状態認識に関連する応用がある。このような応用の具体例には、物流の各段階においてその場での商品の状態や流通に関する情報を記録し、消費者や生産者からの追跡要求に対して情報を開示するサービス（トレーサビリティ）の実現などがある。

トレーサビリティシステムでは、たとえば図 3.1

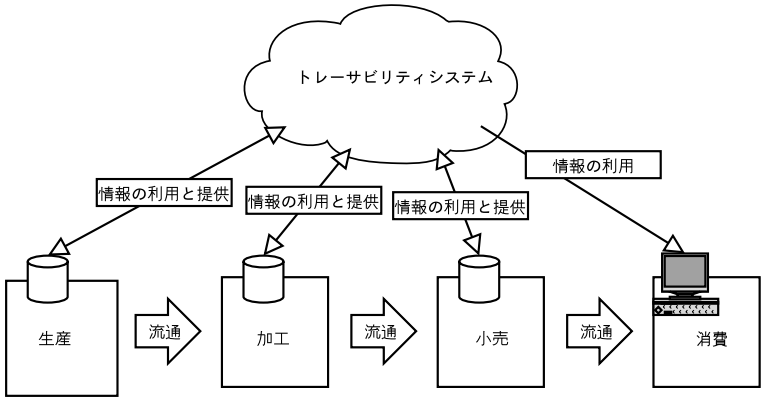

図 3.1.　トレーサビリティシステムのイメージ

のようなシステムが考えられる。生産された商品は、流通業者・加工業者・小売業者を経て最終的に消費者のもとに届けられる。消費者は商品に対して与えられた付加価値としてトレーサビリティシステムからさまざまな情報を取得可能であり、この商品がどのように生産され、加工され、輸送されたかを知る。なお、このような応用については文献 [353] にも詳しい。

トレーサビリティシステムにおいて重視すべきは、スケーラビリティである。たとえば、書籍の 2001 年の国内の出荷量は約 42 億冊（全国出版協会・出版科学研究所調べ）である。これらは再販制度や中古品市場があるため、ID 1 つあたりの問い合わせ要求は消費して終わる食品などよりも数倍多くなると予想できる。

その上、トレーサビリティシステムでは、多量の ID に関連する情報の取り扱いができるだけでなく、柔軟な資源追加・管理ができなければならない。たとえば、ある商品がヒット商品になると、事前の予測を超える規模の検索要求が発生する。このような場合でも計算機資源の柔軟な追加などにより対応できなければならない。

トレーサビリティシステムを実際の利用者に結びつけるためには、商品などに対応付けられる「モノの ID」をどのように管理するかが問題となる。とくに、膨大な ID が存在する空間の中で、分散して配置されたデータベースのうちどのデータベースがその ID に関する情報を保持しているかを発見する方法が技術的課題となる。本研究では、とくに ID を検索の鍵として、ID に関連する分散した情報源へのインデックス情報を取得する技術を総称して、分散インデックス技術と呼ぶ。

なお、このような技術領域をより一般的に捉えると地理的および管理ドメイン的に分散したデータベースに対して、ある ID に関連する情報がイベント駆動や受動的観測などにより蓄積されていく、というモデルとなる。この ID は世の中すべての事物に関連づけられる位に巨大な空間を持ち、また分散データベースの数は、ID の空間に比べれば小さいものの、世の中の観測・管理の主体となる存在（会社・ビル・行政地区など）の数に比例するため、簡単に全数を探索できるような数ではないだろう。このようなモデルはユビキタスネットワーキング応用の一般的なモデルの 1 つとして考えてもよい。

3.1.2 分散インデックス技術の実現

分散インデックス技術に求められる機能は、優れたスケーラビリティと柔軟な資源管理である事は 3.1.1 項で述べた。

ここでは、既存の名前管理システムである DNS を分散インデックス技術に利用する場合を考える。DNS は、木構造を持つ名前空間の管理に特化しており、物品のシリアル番号などのように構造を持たない名前空間の管理には向かない。理由は 2 つあり、第一に、構造を持たない名前空間の上位の空間の名前サーバに問い合わせが集中する点がある。第二に、集中を緩和するためのサーバの分散方式の設計と名前空間の結節点（ゾーンカット）の設計が密接に関係するため、動的な負荷に対応するための負荷分散方式の設計変更が困難である点である。

一方、分散インデックス技術が持つ要求に対して有効な技術の 1 つに、Stoica らによる Chord[288] といった、分散ハッシュテーブル（DHT）とよばれる一群の方式がある。ハッシュテーブルのような、一

連の $key \rightarrow value$ の対応関係を登録・検索するシステムを、完全に自律分散したノード群によって実現するための方式である。DHT では、分散した DHT ノードすべてが、同一のハッシュ関数を利用してメッセージを経路付けし、ハッシュ値の空間を分散分割統治する。

DHT では、名前空間の内容はすべての DHT サーバにほぼ一様に分布する。また、サーバの追加および削除への対処は自律的に行われるため、非構造な名前空間を柔軟に管理するという分散インデックス技術の要求を満たすことができる。

同時に、DHT はスケーラビリティ特性に優れる。全ノード数を N とした時の問い合わせの通信量は $O(\log_b N)$ である（b は方式によって決定される）。ノードを追加し、N を増大させることでシステム全体の容量（全体で利用可能な通信帯域・メモリ領域）が増加し、一方で通信量はそれほど増加しないことから、ノードの追加により性能が相対的に向上するものと考えられる。

しかし、DHT には 2 つの問題が存在する。第一に、DHT は既存の DNS を前提として発展してきた基盤とは異なる方式であり、クライアント側の対応が必要になる。第二に、DHT の名前空間は一様であることが前提であり、DNS と異なり、空間内の特定の部分を特別に扱うような処理に向かない。後者により、たとえばコストをかけて自社に関連する名前空間の問い合わせを向上させる（サービスの差別化）といったことは DHT では実現できない。

なお、本研究では、DHT に関して、Chord を前提としている。しかし、著者は本論文における貢献は DHT として総称される技術全般にあてはまると考える。

3.1.3 本研究の貢献

分散インデックス技術を実現する既存の技術としては、EPCglobal Inc. が提唱している ONS[13] のように、DNS を利用してデータベースと ID との対応づけを行っている方式が存在する。しかし、3.1.2 項で説明したとおり、DNS を利用した分散インデックス技術にはスケーラビリティの問題と柔軟性の問題がある。一方で、DNS は結節点を境界にして名前空間に固有の性質を与えられる（名前空間の多様性の許容）という利点がある。

一方、分散インデックス技術に DHT を用いる場合を考える。とくにトレーサビリティシステムを考えると消費者（一般家庭）がそのシステムの末端となる。分散インデックス技術の実現のためにすべてのクライアントの変更を前提とする DHT は、導入に高いハードルがあり、ONS では既存の基盤である DNS が最初の分散インデックス技術として提案されている。また、DHT は名前空間の性質は一様であることを前提としており、DNS の代替のような機能には適さない。

本研究ではこれらの背景を踏まえ、DNS の名前空間木の 1 つのゾーン（DNS 名前空間のサブセットの単位）として DHT 名前空間を導入し、ID を登録・検索する名前空間にのみ DHT が持つ特性を与える、という方式（名前空間マウント方式）を提案する。同時に、提案方式が持つ問題点を明らかにした上で、シミュレーションによりその問題が発生する条件を絞り込む。

本章は次のように構成する。3.2 節において、関連研究について述べる。3.3 節では、DNS から DHT をアクセスする場合の課題について述べる。課題を解決する手法と、名前解決が失敗する場合について、3.4 節で述べた上で、とくに名前解決の失敗に関して 3.5 節でシミュレーションを行う。3.6 節で本研究の評価を行った上で、本研究で触れられなかった話題について 3.7 節で述べる。最後に、3.8 節にて本研究の結論と今後の課題について説明する。

3.2 関連研究

関連研究として、Cox らによる、DNS の機能を Chord により代替させる研究 [47] が存在する。Cox らは、Chord の全域を用いて DNS の代替機能を提供するシステムを実装し評価した。このシステムは DDNS とよばれる。評価の結果、DDNS は次のような特性を持つとされる。

まず、DDNS では、Chord の持つ自己組織化機能により、名前サーバの管理が自動化され、また同時に良好な負荷分散が行われる。しかし、問い合わせに必要なメッセージの数が多いために遅延が全体的に多くなる。

とくに遅延に関しては、DNS と DHT が持つ遅延の特性の違いがある。DHT では、自動的に構成されるネットワークは比較的整った構造になる。その結果、ほとんどの問い合わせの遅延はある一定の範囲に収まり、上限を大きく上まわる可能性が少ないと

同時に、下限を大きく下まわる可能性も少ない。一方で、DNS では名前空間木の積極的な枝分かれにより、問い合わせの回数が比較的少数に抑えられる上、キャッシュの効果により繰り返される多くの問い合わせが短時間で解決される。一方、レスポンスが悪いサーバや、設定が誤っているサーバにも問い合わせを行うので、解決できない時はクライアントがタイムアウトするまで待ち時間が発生する。

また、DDNS 単体では、ホスト名とアドレスの分散した対応表以上の機能は果たせない。DNS はゾーンごとにサーバを分割可能なため、ゾーンに特有な機能、たとえば負荷分散機能などをサーバ側にて実現できる。一方で、DHT では空間それぞれとサーバの関係のハッシュ関数による切断によりスケーラビリティと自律性を実現しており、DNS のような名前空間ごとに異なる機能を実装するためにはクライアント側へ実装が必要になり、非現実的である。

本研究ではこの研究成果を踏まえて、DHT が持つ優れたスケール特性および柔軟性を、DNS が持つゾーン固有の特性として実現する方法を提案する。

3.3 DNS と DHT の名前空間接合における課題

3.1 節で述べたように、DNS の不均質性を活かして DNS の一部ゾーンの管理に DHT を用いることは、とくに既存のアプリケーション基盤に対してシリアル番号のような非構造かつ膨大なスケールの名前空間を導入する際に合理的である。そのためには、DHT と DNS の名前空間を接続する部位（名前空間接合部）において、DHT が持つスケーラビリティを損わないように注意する必要がある。

名前空間マウント方式には、2 つの課題が考えられる。第一に、3.1 節で述べた導入への障壁である。いずれかのサーバでプロトコルを変換し、DNS 名前空間の一部として DHT 名前空間を導入し、導入障壁を下げる必要がある。

もう 1 つの課題は、第一の課題に従属する。名前空間接合部では DNS と DHT のプロトコルを相互に変換する。DNS と DHT はそれぞれスケーラビリティに優れたシステムであるが、名前空間接合部がボトルネックとなるようでは DHT のスケーラビリティを活用できない。

たとえば DNS と DHT の接合の平易な実現方法として、あるゾーンを管理する名前サーバがプロトコル変換を行い、DHT の名前空間を検索する、という方法が考えられる。この方法は、DNS が持つ名前権限委譲（name delegation）機能を利用したプロトコル変換ゲートウェイ方式である。しかし、ほかのプロトコル変換ゲートウェイ方式と同様、ゲートウェイ自体がボトルネックとなり、DHT のスケーラビリティを損なう可能性が高い。

図 3.2 は、この方法におけるボトルネックの存在を示している。図の左側は名前空間に対応し、右側は実際の通信の模式図である。既存のリゾルバと DHT ネットワークは量的に大きい。ゲートウェイを設置

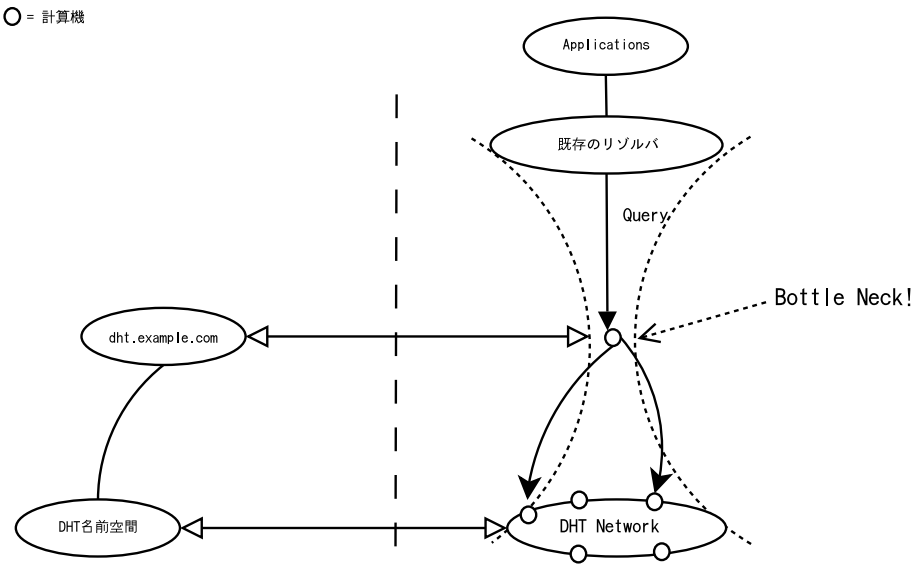

図 3.2.　ボトルネックの存在（左側は名前空間、右側は問い合わせの流れを示す）

し、DHT の名前空間をその直下に位置させることによってリゾルバからの問い合わせをゲートウェイに誘導したとして、量的に大きい二者の間を有限（図では 1 つ）のゲートウェイが仲介することになり、ここが深刻なボトルネックとなる。

3.4 名前空間マウント方式

本節にて、DHT に対する DNS からのアクセス手法を論じる。説明のために DNS の構成要素を整理した上で、提案方式および具体例を述べる。

3.4.1 DNS の構成

DNS には、名前情報の提供および解決という対になる 1 つの機能がある。実際には、これら 1 つの機能は 1 つのプロセスで提供されている場合が多い。しかし本節では、名前情報の保持という機能（DNS-NS：名前サーバ機能）と名前情報の解決という機能（DNS-RS：リゾルバサーバ機能）は独立の構成要素として扱う。なお、アプリケーションは名前解決サービスが必要な場合、スタブリゾルバとよばれる専用のライブラリを利用し、1 つないし複数の DNS-RS に再帰的名前解決を依頼する。

DNS では、名前空間の接合には名前権限委譲（delegation）という機構が用いられる。名前権限委譲においては、上位のゾーンにおいて、直に接続する下位のゾーンの DNS-NS の名前を NS リソースレコード（以下 RR）という形式に記述することにより、下位のゾーンに対する権限を該当する DNS-NS に委譲する。必要な場合は IP アドレス（Glue A RR）を同時に記述する。DNS-RS は、この情報を頼りにして、木構造の名前空間を上層から下層に再帰的に探索する。

なお、DNS の詳細な動作は、RFC1034[206] などによって定義されている。

3.4.2 2 つのアプローチの併用による構成

通信方式の異なる 2 つの方式を接合する以上、何らかのゲートウェイの設置は避けられない。この際、ゲートウェイに対応する部分からボトルネックとなりうる要因をできるだけ排除する必要がある。本研究では、ボトルネックを「**高負荷**」が「**限られた装置に集中**」する現象と捉え、これらをそれぞれに排除する 2 つのアプローチを採用する。

1. 負荷削減アプローチ：名前空間接合部を構成する装置単体での負荷をできるだけ下げる
2. 分散化アプローチ：DHT とほぼ同等のスケーラビリティを持たせる

提案方式はこれら 2 つのアプローチを別個に追求するために、DNS と DHT の名前空間接合部において 2 段階のゾーンを設ける。本項では、この 2 段階のゾーンを利用して DNS と DHT のスケーラビリティを損なわずに両者を結合する手法について述べる。

図 3.3 に提案する名前空間マウント方式を示す。図中左側が DNS と DHT の名前空間を示しており、

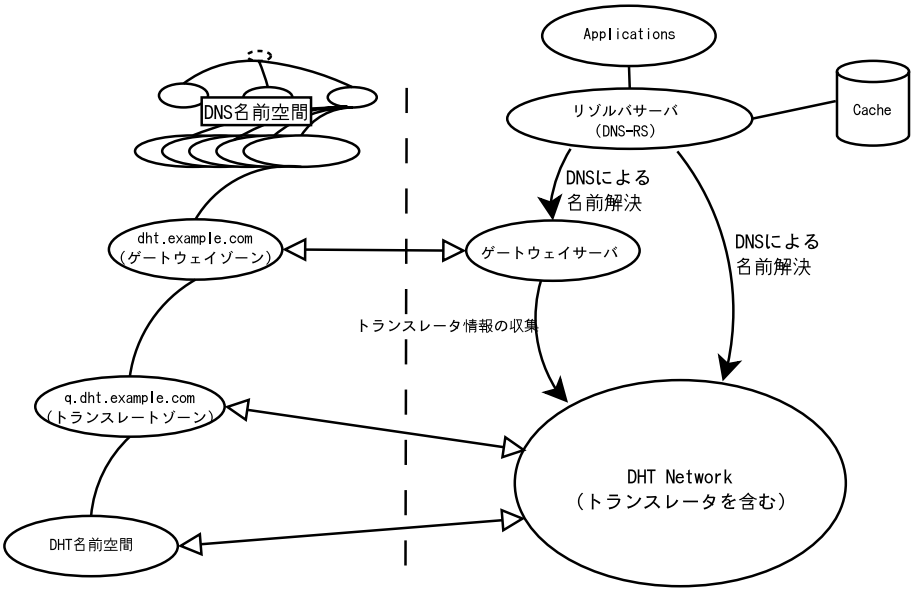

図 3.3. 2 層構造による名前空間マウント方式の実現（左側は名前空間、右側は問い合わせの流れを示す）

上層には DNS の名前空間が広がっている。DNS の名前空間は、ゲートウェイゾーン・トランスレートゾーンからなる名前空間接合部を経由して DHT 名前空間に接合される。図中右側には、実際のサーバなどの情報取得の関係が示されている。本提案では、1 段目のゾーン（ゲートウェイゾーン）において負荷削減アプローチを実現し、2 段目のゾーン（トランスレートゾーン）において分散化アプローチを実現させる。その結果、DNS から DHT への問い合わせにおいてそれぞれのアプローチにおいて最大限に最適化する余地が生まれ、ボトルネックを解消できる。

DNS の名前権限委譲においては通常、名前権限の委譲先は限られた静的なサーバ群であり、分散化アプローチの適用は難しい。そこで、負荷削減アプローチの適用を前提としたゾーンを設置する。このゾーンをゲートウェイゾーンと呼ぶ。ゲートウェイゾーンを構成するサーバの仕事は、次に述べるトランスレートゾーンを構成するサーバ群を調査し、これを DNS-RS に応答することである。

負荷削減アプローチを取るため、ゲートウェイゾーンでは資源を消費する複雑な処理は避ける。したがって、実際の DHT の問い合わせは DNS-RS からそれぞれの DHT ノードに依頼させる必要がある。加えて、DNS-RS からの要求をそれぞれの DHT ノードに振り分けるために、名前権限委譲の機構を用いる。そのために、トランスレートゾーンと呼ぶゾーンを、ゲートウェイゾーンの直下に設ける。

DHT ノードはそれぞれトランスレートゾーンに対する名前サーバとして動作する。トランスレートゾーンにおける名前解決は、DNS による名前解決を受信し、DHT の名前解決に翻訳して行う。ここで、DHT 自身が持つ量的な拡張性を利用する結果、前述の分散化アプローチが達成できる。

3.4.3 動作の流れ

`example.com.` の DNS-NSは、`dht.example.com.` に対する静的な権限委譲を行っている。たとえば 3 台の名前サーバが存在し、それらがゲートウェイサーバ（ゲートウェイゾーンの名前サーバ）となる。ゲートウェイサーバにより構成される名前空間 `dht.example.com.` がゲートウェイゾーンとして機能する。

ゲートウェイサーバは、それぞれ各個に開始点（種）となる DHT ノードを与えられ、DHT ネットワーク上の隣接ノードを順次探索して、DHT ノードの存在を自律的に学習し続ける。またこの際、各 DHT ノードと通信して、トランスレータとして動作可能かを問い合わせる。

トランスレータは DHT ノードのうち、ネットワーク資源やポート番号（DNS は 53 番）が利用可能なものにおいて起動される。そして、トランスレートゾーンに対して権限があるかのように振舞い、DNS によって受信した問い合わせを DHT の問い合わせに変換し、DHT による解決結果を DNS による応答に合成して返す。

問い合わせ時の動作の流れを、図 3.4 に示す。この例では、RFID あるいはバーコードによって読み出された ID を元に、NAPTR RR[190] を読み出し

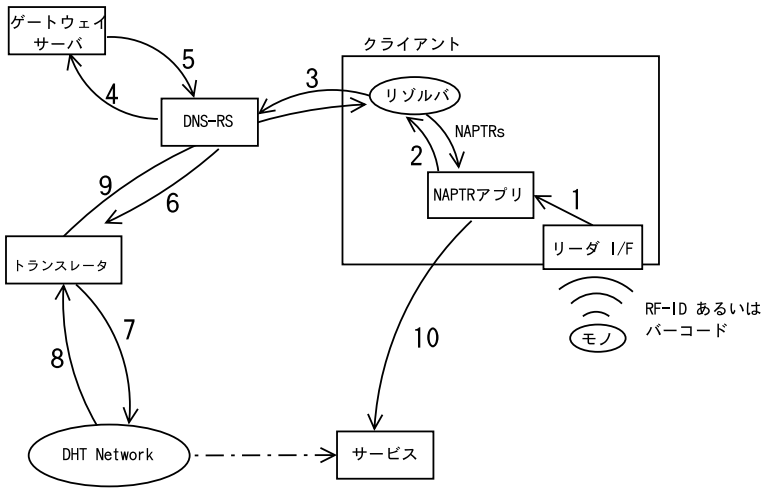

図 3.4.　ゲートウェイゾーンを経由した問い合わせの流れの例

て、サービスを利用するまでの流れを示している。

1. リーダから ID が読み出される
2. アプリケーションが RR の問い合わせを発行する
3. リゾルバ（ライブラリ）は、DNS-RS へ問い合わせを転送する
4. 権限委譲あるいはキャッシュにより、DNS-RS の問い合わせはゲートウェイサーバへ転送される
5. いくつかの DHT ノード（トランスレータ）に権限委譲するメッセージが返される
6. DNS-RS はトランスレータへ問い合わせを転送する
7. トランスレータは受信した問い合わせを分析し、対応する情報を DHT 側に問い合わせる
8. DHT 側が応答する
9. トランスレータによって DNS の形式に変換された情報が、DNS-RS 経由でアプリケーションに返される
10. DHT に格納された情報の内容に基づき、アプリケーションはサービスを発見し、利用する

ここで、12345678 というシリアル番号が RFID などから読み出された（図 3.4 中矢印 1、以下同様）場合、次のような NAPTR RR と合成し、12345678.q.dht.example.com というドメイン名を連鎖的に導出できる。アプリケーションは、リゾルバを用いてこのドメイン名からサービスへのポインタを得るための問い合わせを DNS-RS に送信する（矢印 2 および 3）。

```
example.com. IN NAPTR 10 10 "" ""\
"/(.*)/\1.q.dht.example.com./" .
```

（便宜上 2 行で記述したが、実際は 1 レコード）

example.com. からの名前権限委譲により、DNS-RS はゲートウェイサーバを dht.example.com.（トランスレートゾーン）に対する権限を持つ DNS-NS であると学習する。その結果、DNS-RS からの 12345678.q.dht.example.com. への名前解決要求はゲートウェイサーバに送信される（矢印 4）。

ゲートウェイサーバは、DNS-RS から受信した名前解決要求への応答として、q.dht.example.com.（トランスレートゾーン）に対する NS RR を返答する（矢印 5）。ゲートウェイサーバはこの NS RR に、学習した DHT ノードの中からトランスレータとして動作可能なノードを格納する。例外的な場合を除いて、NS RR によって応答したトランスレータ情報は一度で廃棄する。

ゲートウェイサーバから権限を委譲されたトランスレータは、DNS-RS からの再帰問い合わせを受信する（矢印 6）。ここでトランスレータは、問い合わせを調べて DHT により解決する（矢印 7 および 8）。本研究ではトランスレータの動作方式については定義しないが、たとえば、問い合わせドメイン名のうち、トランスレートゾーン以下を DHT における問い合わせの key とみなし、対応する値を取得する、などといった方式が考えられる。ただし、応答を合成するために必要なデータをどのような形式で格納するかは、DHT 利用者とトランスレータの間で合意しておく必要がある。

トランスレータによって合成された NAPTR RR の応答はアプリケーションに返され（矢印 9）、これを手がかりにアプリケーションはサービスを発見できる。

なお、同じ DNS-RS を利用した二度目以降の問い合わせは、DNS-RS に存在するキャッシュの効果により直接同じトランスレータに送られる。したがって、ゲートウェイサーバが応答する RR の TTL によりキャッシュの効果時間が決定されるため、DNS-RS が再びゲートウェイサーバへ問い合わせを行うまでの間隔は制御可能である。ゲートウェイサーバの負荷を下げるためには、TTL は長いほうがよい。しかし、長い TTL は名前解決失敗の可能性を引き上げる（3.4.4 項にて述べる）。

また、ほかの DNS-RS を利用した問い合わせは異なる DHT ノード上のトランスレータに送信される。これは、ゲートウェイサーバが問い合わせに対して応答するトランスレータを変化させ、分散させるからである。

3.4.4 キャッシュが原因の名前解決失敗

ゲートウェイサーバにより返されたトランスレータの情報を、DNS-RS はキャッシュする。TTL が長ければキャッシュが有効に作用し、結果的にゲートウェイサーバの負荷を下げられる。しかし、長すぎる TTL を持つキャッシュは、トランスレータの DHT からの離脱ないし故障が原因で名前解決が失敗する可能性を高める。

情報源 X に対するキャッシュ C において、C の TTL 期間内で C が有効な間に、情報源 X が変化した結果 $X \neq C$ となり、一見有効な情報に見えるが実際には誤った情報を示すキャッシュを、本研究では劣化キャッシュ（stale cache）と呼び、キャッシュ

が劣化キャッシュとなることを劣化キャッシュ化と呼ぶ。劣化キャッシュに関する研究については、たとえば文献 [359] を参照されたい。

とくに本研究では、ゲートウェイサーバを情報源として DNS-RS へ通知されるキャッシュに着目する。この場合、ゲートウェイサーバにより通知されたトランスレータのキャッシュが劣化キャッシュ化する原因には、トランスレータの停止、故障、あるいはネットワークの不調による離脱などがある。

DHT では、アプリケーションが前提とする内容にもよるが、一般的に中央集権的な管理がなじまないアプリケーションにおいて多く導入されるものと考えられている。したがって、名前サーバに比べて、DHT ノードは頻繁に DHT への参加・離脱を行うものと考えられる。そして、DHT ノードすなわちトランスレータの離脱は、DNS-RS に存在するそのトランスレータに対応する NS RR および対応する Glue A RR などが劣化キャッシュ化するという結果を招く。

このような NS RR の劣化キャッシュ化は深刻であり、存在しないトランスレータへの問い合わせを継続するため、名前解決に失敗する。少なくとも ISC bind 9.2.3 で検証した限りでは劣化キャッシュ化したレコードは対応する DNS-NS に対するアクセスがタイムアウトしても有効であり続け、その結果、対応するゾーンへのアクセスが不能になる。

3.5 劣化キャッシュの影響評価

3.4.4 項で述べたように、劣化キャッシュの影響により名前解決が失敗する可能性がある。ここでは、DHT ノードが離脱するまでの時間と、ゲートウェイサーバが返す NS RR の TTL の関係を軸として、シミュレーションによる劣化キャッシュの影響評価を行う。

3.5.1 検証の焦点

本研究では、とくにノードの平均接続持続時間に対してゲートウェイサーバが DNS-RS に返す TTL をどの程度の長さにすると、無視できない程度の劣化キャッシュが発生するかを調査する。なお、本研究では、無視できない程度の劣化キャッシュの発生を、失敗した問い合わせの比率が 0.001 以上となる場合、と定義する。

シミュレーションの出力として R_f：失敗した問い合わせの比率を得る。名前解決の失敗は、問い合わせを行った際に DNS-RS が持つキャッシュのすべてが劣化キャッシュだった場合に発生する。

3.5.2 シミュレーションモデル

シミュレータは次の要素より構成される。DHT の要素は、ネットワークに参加しているノードの集合と、ノード自身である。1 つの DHT ネットワーク上にて、単位時間あたり一定数のノードが到着する。また、それぞれのノードは単位時間あたり一定の確率で離脱する。この結果、系全体のノード数はある程度の規模で安定する。また、DHT ネットワークに対して 1 つのゲートウェイサーバが存在し、DHT ネットワークから単位時間ごとに 1 つのノードを取得し、キューに保存する。

今回のシミュレーションによって明らかにする点は、ある DNS-RS における劣化キャッシュの作用である。したがって、実際には DNS-RS は多数存在するが、ここでは代表して DNS-RS が 1 つ存在するモデルを作成し、着目する。

DNS-RS に対応するクライアントは 1 つ存在し、クライアントは単位時間ごとに一定数の問い合わせを発行する。DNS-RS は問い合わせに際して、キャッシュ情報をチェックし、もしキャッシュに情報が存在する場合はそのキャッシュの内容に基づき、問い合わせが成功するか否かを判断する。一方で、キャッシュが存在しない場合はゲートウェイサーバへ問い合わせて、その結果をキャッシュに記入する。問い合わせの正否は、クライアントがログファイルに記録する。

また、単位時間ごとにキャッシュを検査し、キャッシュ全体の中で劣化キャッシュがいくつ存在するかを記録する。

3.5.3 パラメータと手順

シミュレーションのパラメータは次のとおりである。

- L：シミュレーションの長さ（単位時間）
- N_a：系全体に対する、DHT ノードの単位時間あたりの到着数
- R_d：単一 DHT ノードの、単位時間あたりの離脱確率
- M：ゲートウェイサーバが一度に返すノードの数、あるいは、トランスレートゾーンを提供するトランスレータの並列度

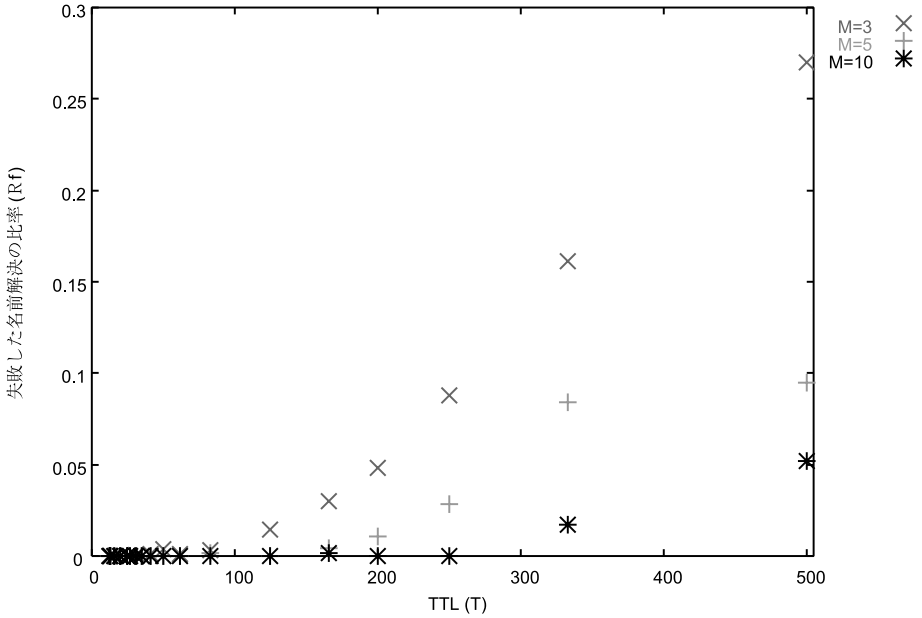

図 3.5. 失敗した問い合わせの比率（R_f）と T の関係（抜粋）

表 3.1. シミュレーションの条件

パラメータ	値
L	25000
N_a	5.0
R_d	0.004 ($\frac{1}{250}$)
M	3、5、10
T	25 ... 500
N_q	0.1

- T：ゲートウェイサーバが提供する NS RR の TTL
- N_q：単一の DNS-RS に対してリゾルバが発行する名前解決問い合わせの、単位時間当りの到着数

シミュレーションの条件を表 3.1 に示す。ここで、各ノードの平均接続持続時間を 250 と定め、L は平均接続持続時間の 100 世代分とした。また、R_d は平均接続持続時間の逆数として、DHT の参加ノード数が 1000 ノード程度で安定するよう N_a を定めた。M は、UDP に格納可能な上限と比較的近いと思われる値（$M = 10$）と、最低限の値である DHT ノードと 2 隣接ノード（$M = 3$）、および中間的な値（$M = 5$）を代表値として選んだ。T は等比数列に近くなるような値とした。N_q は単一の DNS-RS に対する問い合わせの到着頻度であり、今回のシナリオでは DHT ノードの取得速度に比べて十分に遅い 0.1 とした。

シミュレーションの手順は次のとおりである。

1. 初期状態（ノードが存在しない状態）の生成
2. ノード数が安定するのに十分な時間、ノードの到着・離脱を継続
3. 安定した時点で DNS-RS およびクライアントを生成し、問い合わせ成功率などを測定開始
4. 手順 3 より L 単位時間経過時点で終了

3.5.4 シミュレーション結果

以上の条件で行ったシミュレーション結果を図 3.5 に示す。縦軸が全問い合わせの試行のうち、劣化キャッシュが原因の失敗した問い合わせの比率、横軸がゲートウェイサーバが返す T の値である。

3.6 考察

3.6.1 本方式の効果と限界

3.3 節では、名前空間マウント方式の課題として既存のクライアントとの適合性と、名前空間接合部におけるスケーラビリティの確保を挙げた。

本方式による効果は、ゲートウェイゾーンとトランスレートゾーンの分割により、2 つの異なるアプローチにより名前空間接合部のボトルネックを軽減する点にある。極論すると、ゲートウェイサーバが返す NS RR の TTL が無限大であれば、世界中の

DNS-RS はゲートウェイサーバに 1 回しか問い合わせを行わず、DNS-RS とトランスレータは分散した対応関係を作り、ゲートウェイゾーンによるボトルネックは存在しなくなる。

トランスレータは必要に応じて追加可能であり、世界中の DNS-RS からの問い合わせ量が増大した場合でも処理を継続できると考えられる。

したがって、本方式により、DHT のスケーラビリティを損なうことなく DNS から DHT に対する問い合わせを解決でき、その結果既存の情報基盤に対して透過的に DHT を導入できた。これによりたとえば、トレーサビリティのようなアプリケーションが DHT の特性を必要とし、かつ、専用アプリケーションの導入が困難である、という場合に、問題を解決できる。

一方で、本方式における明らかな性能の限界を決定する要素は、ゲートウェイサーバが利用可能な通信帯域であることがわかった。ゲートウェイサーバとトランスレータの関係において、性能の限界が発生しうる代表的なシナリオを以下に示す。

1. 劣化キャッシュ限界シナリオ：ゲートウェイサーバの帯域を使い切りそうになり、対応して TTL を長くした結果、劣化キャッシュによる問い合わせエラーが頻発
2. トランスレータ負荷限界シナリオ：ゲートウェイサーバの帯域を使い切りそうになり、トランスレータの情報の取得の並列度を減らした結果、新しい情報の取得が間に合わず、同じトランスレータを多数の DNS-RS へ応答し、その結果トランスレータに性能以上の負荷が集中

説明のため、ゲートウェイサーバの設計例を、図 3.6 に示す。ゲートウェイサーバは、名前サーバとして動作し受け答えを行う部分（DNS サービスモジュール）と、DHT のノードを収集する部分（ノード情報収集モジュール）の 2 つから構成される。詳細には以上 2 つの部分に加えて、応答する NS RR の TTL を決定する機能（TTL 決定モジュール）と、収集したノード情報を管理する機能（トランスレータ情報管理モジュール）が存在する。

DNS サービスモジュールは、通常用いられている名前サーバと比べて特殊な処理を行うものではなく、通常の使用条件であれば計算量、ネットワーク入出力などにおける問題はないと考えられる。

一方、ノード情報収集モジュールは、各 DHT ノードとのやりとりが通信のすべてである。すなわち、DHT ノードに接続し、その隣接ノードの情報を収集し、セッションを切断した上で、次のノードに接続する、というプロセスの繰り返しである。通信を暗号で守るなどの計算量の多い処理を行わない限り、最低限の通信処理だけでノード情報を収集できる。

ただし、通信に必要な時間、とくに RTT や TCP の場合ネゴシエーションにかかるオーバーヘッドがあるので、DHT ノードを収集する速度には一定の上限があると考えられる。DHT ノードの収集速度（node/sec）に比べて問い合わせ到着速度（query/sec）が高い場合は、収集した DHT ノードを複数回使うか、余剰帯域があれば DHT ノードの収集を並列化して対処する。一方、DHT ノードの収集速度が十分に高い場合は、古い情報を使い劣化キャッシュのリスク上昇を避けるため、新しい DHT ノードの情報と入れ替える。ここで、問い合わせ応答に帯域を消費し、DHT ノードの収集の速度あるいは並列度が低下した場合、前述したトランスレータ負荷限界シナリオが発生する。

また、ゲートウェイサーバは、DNS-RS に返す NS RR の TTL の調整によって負荷を調節可能であると

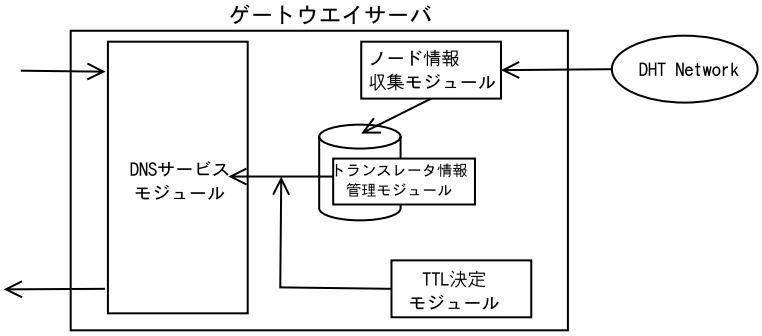

図 3.6.　ゲートウェイサーバの設計

考えられる。具体的には、ゲートウェイサーバの負荷が高い時にはTTLを長くすることにより、DNS-RSからゲートウェイサーバの問い合わせ頻度を減少させられる。一方で、TTLを長くすることにより3.4.4項で述べた劣化キャッシュの問題が発生する、というトレードオフの関係にある。ここでゲートウェイサーバの負荷が上がるにつれTTLを延ばしすぎることで、前述した劣化キャッシュ限界シナリオとなる。

3.6.2 劣化キャッシュの影響

$T = 250$（ほぼ平均接続持続時間）の時点で、$M = 3$では8.8%、$M = 5$では2.8%の名前解決に失敗しており、$M = 10$ではほぼすべて成功していることが読み取れる。また、ほぼすべての名前解決に成功するTの範囲は、今回の実験では$M = 5$の時に$T = 125$、$M = 3$では$T = 35$程度が上限であった。

今回の結果から、それぞれのMにおけるTの上限を推測可能である。また、$M = 3$程度では平均接続持続時間に比べて15%程度のTしか確保できない。これに対して、Mを向上させることによりTの上限を改善でき、$M = 10$であれば平均接続持続時間に比べて100%程度のTが利用可能になる。

一方、劣化キャッシュ限界シナリオを考えると、Tはこの上限を下まわる範囲で制御する必要がある。今回の結果はR_dの値があってはじめて適用可能な結果であり、実環境ではR_dの正確な推測が必要である。しかし、筆者の知る限り、効率的な観測方法は確立していない。その結果、簡単な測定に頼ることになり、R_dの推測値の精度は低くなる。当然、安全のためのマージンを多く取る必要が出てくるため、Tの限界は短くなる。したがって、Tの上限を長く取るためには、高精度かつ低コストにR_dの期待値を測定する方法が必要である。

3.7 議論

提案した名前空間マウント方式は、3.4.4項で述べた劣化キャッシュの問題以外にも、以下のような問題や、調査が必要な曖昧な点を抱えている。

第一に、プロトコル上の問題がある。RFC2181[74]によると単一のRRSet（あるドメイン名に対する特定のTYPEのRRの組）は単一のTTLを持っている必要がある。たとえば、複数のトランスレータを記録しておき、それぞれの古さや信頼性、処理能力や現在の負荷に応じて動的にTTLを変更してトランスレータの負荷を均衡化する方式が考えられるが、この方式は明確にRFCに違反してしまう。

また、動的にNS RRの組を合成しているが、この場合DNSSEC[69]の適用が困難になる。NS RRの組に対して毎回署名を行うとすると、署名の計算量が多いためゲートウェイサーバの負荷が上がり、その分スケーラビリティは低下する。一方で、NS RRを再利用し、再生成の回数を抑えればゲートウェイサーバの計算量負荷は下がるが、これはトランスレータの並列度の実質的な低下に繋がる。また、トランスレートゾーンの鍵をどう管理するか、という問題もある。ただし、DNSSECを適用する場合はDHT自身に同等あるいはそれ以上の強度を持つセキュリティ機構が必要となる。

第二に、負荷制御がどこまで能動的に行えるかという論点がある。TTL制御による負荷制御は、前述のとおりトランスレータの負荷の均衡化に用いる場合はプロトコル上の注意が必要である。一方、ゲートウェイサーバの負荷調整の目的であればRRSetのTTLを統一できるので適用は容易である。ここでは、TTL制御に対して問い合わせ頻度がどの程度の追従性を持ち、どの程度時間的余裕を見てTTL制御を行うべきか、などの論点が考えられる。

第三に、DNSの権限の考え方に起因する問題がある。具体的には、ゲートウェイサーバがトランスレートゾーンの権限を持つサーバとしてDNS-RSに送信するNS RRの組と、NS RRに格納されたトランスレータ自身がトランスレートゾーンに権限を持つサーバとしてDNS-RSに送信するNS RRの組が食い違った場合、トランスレートゾーン自身に権限を持つトランスレータの情報が優先される。

したがって、DNS-RSにとってNS RRの組とはトランスレータから返される値であり、DNS-RSに返すNS RRの組を保持管理すべきはゲートウェイサーバではなくてトランスレータである、ということになる。と同時に、トランスレータ自身は、自身のDHT隣接ノードのうち適当なノード（トランスレータ機能を有するノード）をNS RRの組として返せなければならない。具体的には、M個に1つの割合で中心となるノード（仮にSOAノードと呼ぶ）を決定し、これを中心としたDHTノードの集合を自律的に形成した上で、ゲートウェイサーバはSOAノードから情報を収集する、という方式にする必要がある。

3.8 おわりに

本研究では、製品 ID などのような非構造な名前空間の管理に適したデータインデックス手法の 1 つである DHT を DNS から利用可能にする方式（名前空間マウント方式）に関する検討を行った。これにより、既存の DNS を用いた利用者環境から、非構造な名前空間を効率的に検索できる DHT の特性を透過的に利用できる。

名前空間マウント方式の特徴は、スケーラビリティの上限を決定づける 2 つの要素、即ち処理・回線負荷の高低と、並列度の高低を分解することで、プロトコル的な制約から脱し、制約下におけるトレードオフを追求する必要性をなくした所にある。

しかし、名前空間マウント方式は、DNS のプロトコルの意図とは異なる動作、つまり問い合わせに応じて名前空間の権限委譲を動的に行う。同時に、それぞれのトランスレータはネットワークから離脱する可能性があり、その結果劣化キャッシュが発生し、名前解決失敗となるリスクがある。本研究では、名前解決失敗のリスクをシミュレータを通じて検証し、一定の条件下でどの程度のリスクが発生するかを明らかにした。

今後は、3.7 節で述べた課題の解決や、平均接続持続時間の計測方法の研究、ゲートウェイサーバの帯域を考慮したスケーラビリティの考察、実動作環境における実験などを行う必要がある。最終的に、商品トレーサビリティシステムや ONS のような枠組に DHT を適用し、多数の ID の取り扱いを安価・効率的に行える環境の構築を目指す。

第 4 章　Distributed Scalable MOG Servers on P2P Networks

abstract

Today's Multi-player Online Games (MOGs) are challenged by infrastructure requirements because of their server-centric nature. Peer-to-peer overlay networks are an interesting alternative if they can implement the set of functions that are traditionally performed by centric game servers. In this paper, we propose a *Zoned Federation Model* (ZFM) to adapt MOGs to peer-to-peer overlay networks. We also introduce the concept of *zone* and *zone owner* to MOGs. A zone is some part of the whole game world, and a zone owner is a game sever of a specific zone. According to the demands of the game program, each node actively changes its role to a zone owner. By dividing the whole game world into several zones, workloads of the centric game server can be distributed to a federation of zones. In order to reduce response latency overhead on data exchanges between a zone owner and its clients, we limit the use of a Distributed Hash Table (DHT) to the rendezvous point of each zone; actual data exchanges are carried out through direct TCP connection between a zone owner and its members. We also use the DHT as backup storage media to cope with the resignation of a zone owner. We have implemented this zoned federation model as a middle layer between the game program and the DHT, and we evaluate our implementation with a prototypical multi-player game. Evaluation results indicate that our approach enables game creators to design scalable MOGs on the peer-to-peer environment with a short response latency which is acceptable for MOGs.

4.1 Introduction

Today's Multi-player Online Games (MOGs) are constructed in a server-centric model. To achieve scalability, MOGs usually employ clusters of game servers. In spite of scalability, clustering technologies cost game creators co-location fees. Therefore, starting a new MOG for small or medium enterprises is difficult. Also, in today's server-centric solutions, users cannot play the game when the centric game server stops its services.

In this chapter, we try to achieve an alternative MOG infrastructure by using peer-to-peer overlay technologies. The peer-to-peer overlay network is a highly distributed network or computing environment. Nodes on a peer-to-peer overlay network

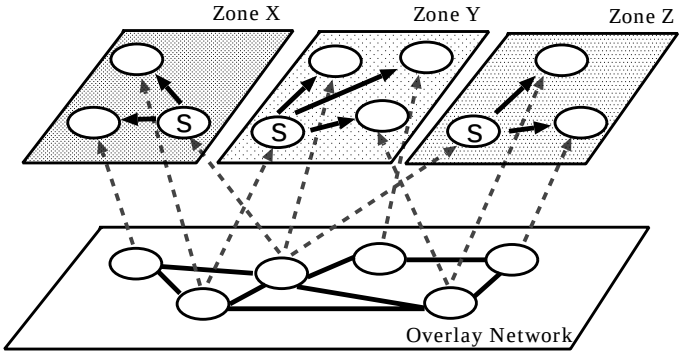

Fig. 4.1. Zones on the overlay network

compose a network on the application layer and share resources of each node.

To construct a peer-to-peer overlay infrastructure for MOG, we model a "Zoned Federation Model (ZFM)". ZFM is a latency optimizing approach based on peer-to-peer overlay technologies. The key idea of the ZFM is as follows: numerous participating user nodes on an peer-to-peer overlay network compose a large game server cluster. The ZFM creates several client-server groups on a peer-to-peer overlay network, as shown in Figure 4.1. Constructing a small-scale client-server model in each zone, the ZFM can achieve as short a response latency as that of the server-centric MOG solutions.

In the ZFM, the tasks of a centric-server are distributed into a peer-to-peer overlay network. We partition the game world into several *zones* by the locality of the game world or the locality of the game data's features. In addition, we let participating nodes play a cluster of the game server on each zone. The node playing a cluster of the game server establishes a direct connection to each client node in the same way that the direct connection between the server and clients in the client-server model is connected. Hence, each server node can serve the latest game status with almost the same performance as that of a centric-server. Each zone is independent from each other; therefore, a user node can become the server on several zones and play the client of some zones.

The cluster of the game server in the ZFM is maintained by participating user nodes autonomously. According to the peer-to-peer overlay environment or the game sequence, each participating user node changes its role to either a cluster of the game server on some zone, the client of the game on other zones, or to resources for constructing a large network or storage. Of course, each participating node can easily resign from the game server task, stop the client role, or leave from the peer-to-peer overlay network. Employing a peer-to-peer overlay network as a backup storage media, the ZFM provides a mechanism for all participating nodes to record the latest game status serialized by the game server role node, to inherit the tasks of the old server role node or the game sequence completely, and to continue serving the game data.

We have implemented the ZFM as a library of a zoning layer, a middle layer between a game program layer, and a TCP/IP stack, or between the game program layer and the Distributed Hash Table (DHT), which is a technique for constructing a peer-to-peer overlay network. We also evaluated the performance of the ZFM implementation with a prototypical multi-player game.

The rest of this chapter is organized as follows: we describe the features of today's MOGs and server-centric solutions in section 4.2. We mention details of the ZFM, and describe its implementation in sections 4.3 and 4.4, respectively. Section 4.5 shows the evaluation result of our ZFM implementation with a focus on the response latency.

We refer to related work in section 4.7, and

finally, we discuss future work and conclude this chapter in sections 4.8 and 4.9.

4.2 Multi-player Online Games

MOGs are such games such that several game users share a game world and play on the game world by exchanging shared game data. We call such shared game data, Global Status Data (GSD). There are several types of MOGs, such as racing, First Person Shooter (FPS), Real Time Strategy (RTS), or Role Playing Game (RPG), etc. Some MOGs, called Massive Multi-player Online Games, are played by thousands or even tens thousands of users.

Typically, a MOG requires a short response latency and a consistency of GSD among users. Short response latency is needed to provide a comfortable game response without stress for users[10, 11, 16, 114, 238, 253], and the consistency of GSD is required to produce the same game world for all users, as well as to prevent cheating or unfairness in play[116]. Today's MOG is usually constructed in the client-server model to manage the consistency of GSD, and several clustering or distributing techniques are employed to achieve a short response latency and scalability.

4.2.1 Global Status Data

In MOG, users have to share GSD to play in the same game world. GSD can be changed according to the game players' demands or the game sequence. Changes of GSD should be serialized and should be synchronized among game players to keep to the consistency of the game world. Therefore, MOG requires some authoritative nodes to provide serializability of state changes and to ensure the consistency of changes. In the client-server model, a centric game server works as this type of authoritative node.

The GSD of the MOG has several localities of interest, and several large-scale MOGs employ interest management techniques[214]. The examples of localities of interest on a game world are as follows: locality of the network infrastructure, locality of part of game world, locality of the group sharing specific GSD, locality of personal information, etc.

Distributing or clustering techniques use these localities to partition the tasks of a single centric server or the whole GSD. In other words, partitioning GSD or server tasks into several groups by locality enable a reduction in overhead on a single server machine, to achieve scalability or responsibility. Contents Distribution Network technologies focus on the locality of network infrastructure, and SimMud[165] partitions the game world into several regions with focus on the locality of the area of the game world. Server clusters are constructed by the locality of the group sharing specific GSD, the locality of personal information, or the locality of the numbers of access.

In modeling the ZFM, we focus on the locality of GSD, that is, the locality of the group sharing specific GSD, the locality of personal information, and the locality of the number of access.

Also, GSD should keep its consistency by using an authority to produce the same game world for all users. In the client-server model, a centric server or centric server clusters judge conflicts among users, modify the GSD according to game sequence, serialize changes, synchronize updated GSD on all users to prevent a mismatch, and audit GSD to find cheating or unfairness. Processing these tasks on a distributed environment such as on peer-to-peer overlay networks is difficult. Some distributed agreement protocol[62] can resolve inconsistency on peer-to-peer overlay networks, but such distributed agreement protocol is likely to be complex.

4.3 Zoned Federation Model

We design the Zoned Federation Model (ZFM) as a model of a MOG on an overlay network by employing five techniques to construct the ZFM: *zone*, the *Distributed Hash Table* (DHT), *zoning*, *mapping*, and *zoned federation*. A zone is a judgment and data transfer block of the GSD to distribute whole GSD into a peer-to-peer overlay

network. DHT is a peer-to-peer overlay network technique we employ to construct the ZFM. Zoning is a framework of partitioning whole GSD into several zones to use the DHT as a backup storage media and as a rendezvous point to zones. Zoned federation is a mechanism to let participating user nodes manage each zone and the whole game world autonomously.

4.3.1 Assumptions

Before describing the ZFM in detail, we assume the following:

- GSD on the DHT overlay network never disappears
- No malicious user nodes join to the MOGs
- Network infrastructure provides short and stable network delay over the inter-domain networks.

4.3.2 Zone

First, we introduce the concept of a *zone*. A zone is a piece of the whole GSD partitioned by locality of interest. When dividing the whole GSD into zones by locality, the GSD on some zones is required by some of all the participating nodes. We define a zone as a judgment block and a data transfer block. According to this definition of zone, we define a server role as existing on each zone, and design a mechanism to manage the server role and the GSD on each zone.

As in the ZFM, the SimMud[165] partitions whole GSD by locality of interest. However, SimMud divides the game world into several regions only by locality of area in the game world. The ZFM partitions whole GSD into several zones not only by locality of area in the game world, but also by locality of group sharing specific GSD or the locality of personal information. The ZFM can produce a flexible data structure for any type of MOG.

Along with the concept of *zone*, we introduce node status. Basically, each user node has one of three statuses for each zone: *independent*, *zone member*, and *zone owner*.

The zone owner represents the server role on a zone, and the zone member represents the client role on a zone. When a user node wants to receive updates of the GSD of a particular zone, the node changes its status as a zone member. A zone member node can request the zone owner node to modify some GSD on the zone. If a user node wants to modify the GSD on the zone when there is no zone owner, then the node tries to change its role to zone owner. Although the zone owner node has a right to change all GSD on the zone, the zone owner node has responsibility for judging conflicts among requests from zone members, modifying GSD along with the requests, serializing GSD according to the game sequence, and announcing any updates of GSD to all zone members.

If a user node is not interested in the GSD of some zone, the node then has an independent status to the zone. Next, we add a new definition for "zone." When we describe "a node joining to a zone" or "the zone owner leaving from the zone", we use "zone" to represent the membership who manages the zone owner role and GSD.

Along with the basic three node statuses, we add one extra node status: *data holder*. In the ZFM, whole GSD is distributed into a peer-to-peer overlay network representing several zones. Futuremore, some nodes contribute their own local storage to part of the shared storage on the peer-to-peer overlay network. We call the node contributing its storage for storing GSD of some zone, a data holder. Therefore, a data holder node has all GSD of some zone. Data holder nodes are selected by the data sharing algorithm of the employed overlay technique.

4.3.3 Distributed Hash Table

In the ZFM, we employ the DHT as at technique to construct an peer-to-peer overlay network. The DHT is a distributed data placement and a data lookup algorithm for an overlay network. Basically, the DHT stores the mapping between a key and a value, and the DHT appears as an ordinary hash table for each user node. When a node

tries to search some data, the node looks up its own DHT as a hash table. A key points to the identifier of some node, then, the node pointed to by the key contains true data on the hash table. Therefore, a user node refers the true data to some other node whose identifier is obtained through the DHT. Variations of DHT, e.g., Chord[289], CAN[254], Pastry[265], Tapestry[332], etc, exist.

Using the DHT, all nodes on the same DHT overlay network use one hash table, that is, all nodes share the same data. DHT ensures the pairing of a key and the corresponding data or the identifier of some data holder node; therefore, the mismatching of data among nodes never occurs.

However, DHT has several drawbacks. First, the DHT has accessibility to data, that is, every node can modify every piece of data on the DHT. Therefore, some nodes can modify some data when the data is not a match for the node. The DHT does not have a judge to avoid conflicts of changing data among user nodes. Also, the DHT ensures every node will share the same data; hence, the accessibility of the DHT can easily provide inconsistency of data on a time sequence.

Second, the DHT has a tradeoff between the size of the routing table and routing hops. If the order of the routing table size is $O(\log N)$ at most, then the order of the routing hops becomes $O(\log N)$[265, 289]. Hence, read or write operations to the DHT require $O(\log N)$ routing hops. Several Application Layer Multicast (ALM) methods using the DHT are proposed[35]; these ALMs require several routing hops on the application layer. Therefore, using the DHT as a data transfer method may provide long network latency when the overlay network topology is large. Such long network latency may be a critical overhead for some MOGs, racing games or first person shooter games.

To reduce the number of nodes access the GSD through the DHT, we use the DHT as a backup storage media, and as a rendezvous point of zones; that is, as the routing map to zones.

4.3.4 Zoning and Mapping

We introduce zoning and mapping methods for using the DHT as the rendezvous point to zones. To reduce message forwarding through the DHT, an independent node should know the latest membership of each zone by using only a single access to the DHT. In other words, the ZFM requires a mechanism by which an independent node can learn which node is the zone owner and which nodes are zone members when the independent node accesses to the zone data on the DHT overlay network.

Zoning makes each zone a judgment and a data transfer block of GSD by adding the information about the membership on each zone. Each zone contains some piece of the whole GSD (see section 4.3.2). We add *candidate list* and *members list* to each zone by zoning. The candidate list shows which node is the current zone owner, or tells which nodes are the candidates for a new zone owner. On the other hand, the members' list expresses which nodes have the zone member state for the zone. Figure 4.2 shows how the information about each zone looks on the user side of the DHT.

Data of each zone and data holder nodes on the DHT overlay network correspond to mapping methods. A DHT algorithm selects data holder nodes randomly, and assigns the key of each zone to the identifier of a corresponding data holder node (Figure 4.3). Because each key on the user side of the DHT points to the identifier or the address of the corresponding data holder node, the user node can use the DHT as the routing map to each zone.

Figure 4.4 shows message forwarding on the DHT. On the user node side, when a user node wants to read zone data, the user node gets the key generated by an employed hash function and a key word, and then user node can look up the identifier of the data holder node pointed out by the key. The user node then sends a message to the data holder node via several routing hops, and receives the zone data from the data holder

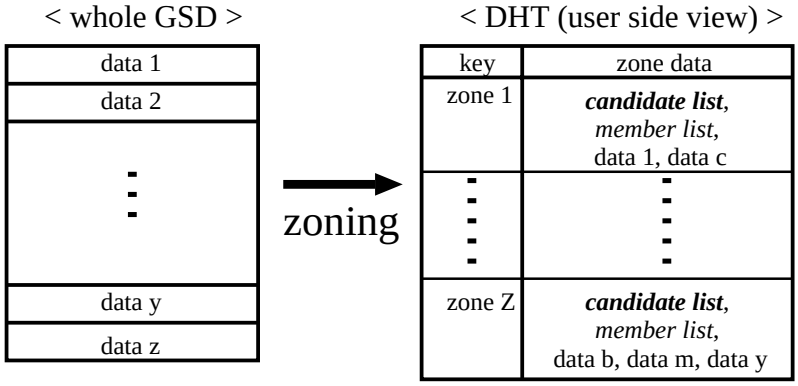

Fig. 4.2. Zoning

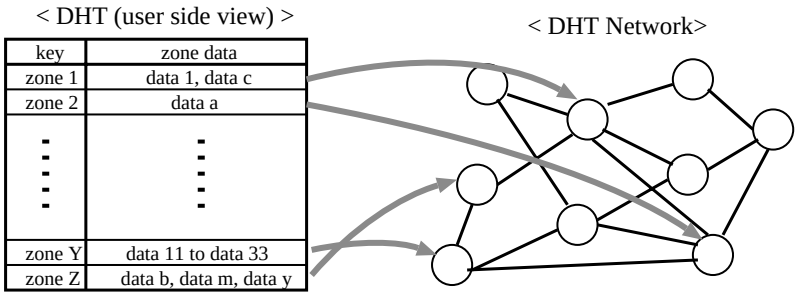

Fig. 4.3. Mapping

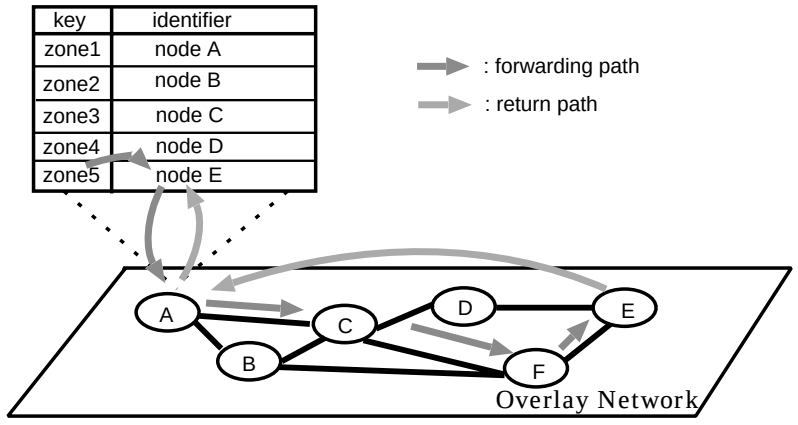

Fig. 4.4. Message Forwarding on DHT

node. If some user node wants to change some zone data, the user node sends a message to the data holder via several forwarding hops, and the data holder node changes the zone data according to the received message.

4.3.5 Zoned Federation

A zoned federation is a mechanism to manage the information of membership and consistency of the GSD on each zone. Zoning and mapping methods enable each participating node on the DHT overlay network to access each zone and to grasp the zone data of the accessed zone. For the nature of the DHT overlay network, the ZFM should let each participating user node join and leave the DHT overlay network easily. Therefore, the procedures to inherit the information of the zone data on the zone from the zone owner node must be tolerant of the frequent changes of the zone owner role node. The zoned federation

provides a mechanism to inherit the current situation of each zone and to manage each zone autonomously.

Data backup

To succeed in taking the current GSD from the old zone owner, the ZFM uses the DHT overlay network as a backup storage media. A zone owner is the centric game server on particular zone; therefore, the zone owner accepts zone members' requests, judges conflicts among requests, modifies GSD, serializes the changes of GSD, and announces any updates of GSD to all zone members. Also, the zone owner updates the current GSD on the corresponding data holder node whenever any updates about GSD occur.

By recording the current zone's GSD on the DHT overlay network, all participating user nodes can get the latest information about each zone from the DHT even when no zone owner exists in a specific zone. If a participating user node changes its status to the zone owner on some zone, the node can know all zone member nodes and the latest GSD from the zone data on the corresponding data holder node; therefore, the new zone owner node can accomplish all the tasks left by the old zone owner.

Requests about changes to GSD and announcements of the latest GSD are transmitted between the zone owner and zone members directly; in other words, no intermediate hop on the application layer level is employed in transmitting GSD-related messages in the ZFM. On the other hand, updating GSD on the data holder node is processed through the message forwarding mechanism on the DHT overlay network. Using the message forwarding of the DHT requires several routing hops; however, the updated zone data on the data holder node is processed in parallel by announcing the latest GSD to each zone member. Hence, backing up the zone data to The DHT overlay network doesn't influence the response latency of message exchanges between the zone owner and the zone members.

Zone membership management

In the ZFM, each user node accesses zones where the GSD required by the user node is stored. Also, the node playing the zone owner role is changed dynamically. The candidate list and the member list on each zone provides management mechanism of the zone owner role and grasping current zone members.

Each user node changes its status regarding each zone as illustrated in like Figure 4.5. An user node becomes the zone owner from the independent state or the zone member state by the *step up* procedure. An independent state node can join the zone as a zone member by following the *join* procedure. The zone member state node leaves from the zone by the *leave* procedure, and the zone owner role node can resign its game server task by the *step down* procedure and change to the independent state. Basically, these status changes and procedures are announced to other nodes by the candidate list and the members' list of each zone.

The candidate list shows which node is the current zone owner, or which node has the right to play the zone owner role. When there is no zone owner in a zone and a user node tries to become the new zone owner of the zone, the user node writes its identifier into the candidate list of the zone and reads the latest zone data through DHT message forwarding. If the identifier of the user node is listed on the top of the loaded candidate list, the user node can become a new zone owner, and then the user node can start working as the

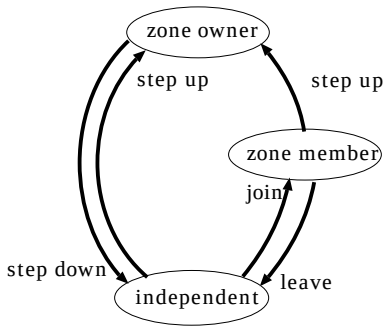

Fig. 4.5. Node Status Changes

zone owner. When the zone owner wants to leave the zone, the zone owner has to remove its identifier from the candidate list on the data holder node. After removing its identifier from the candidate list, the zone owner can leave the zone.

On the other hand, when a user node wants to join a zone as a zone member, the user node writes its identifier into the members' list on the data holder node, reads the latest situation about the zone from the DHT overlay network, and sends a *join* message to the zone owner who is listed on the top of the candidate list. When the zone owner receives a join message from a new zone member, the zone owner adds the identifier of the new zone member into its own members' list. Next, the zone owner and the new zone member establish a connection and exchange messages directly. When a zone member node tries to leave the zone, the zone member node removes its identifier from the members' list on the DHT, that is, the members' list on the data holder node. The zone owner can realize the disappearance of the zone member when the connection between the zone owner and the zone member is closed.

The details of the procedures for managing the membership of zones and GSD on each zone are described in section 4.4.3.

4.4 Implementation

In this section, we describe our implementation of the Zoned Federation Model (ZFM). We have inserted a *zoning layer* as a middle layer between game programs and TCP/IP stacks, and between game programs and the DHT layer.

The zoning layer covers both the DHT layer and the TCP/IP stacks (Figure 4.6). By using the zoning layer, game programs don't have to consider whether a message should be exchanged through the DHT or not.

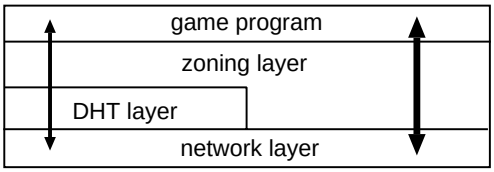

Fig. 4.6. Zoning layer

Our ZFM optimizes the response latency on a MOG. In the implementation, we introduced several techniques to reduce or to optimize response latency.

We implemented this zoning layer as a C library; we call this zoning layer library a *libcookai*, and a C library of Pastry customized for the ZFM. We also implemented a sample MOG using libcookai.

4.4.1 Pastry for ZFM

Table 4.1 shows APIs of our pastry implementation in C language. These APIs are the interfaces of the Pastry DHT overlay network for the zoning layer (Table 4.1).

Each user node also participates in the Pastry DHT overlay network by the *join* function. By using the *query* API, the zoning layer reads current zone data from the data holder node on the Pastry DHT overlay network. If any node tries to change the same zone data asynchronously, some inconsistency of the zone data may occur. To avoid inconsistencies of data, we divide the 'write operation' of the zone data into *set* and *delete*. The zoning layer calls the *set* function to add a new data value of zone data on the DHT, and calls the *delete* function to remove the old value of the zone data.

Table 4.1. APIs of Pastry for ZFM

function	definition
join()	join to DHT network
query(key)	get the current zone data pointed out from the hash key to DHT
set(key, data)	add new data to zone data on DHT
delete(key, data)	delete the specific data from zone data on DHT

4.4.2 Zoning Layer

The zoning layer controls network access. When the game program on a user node tries to send a message, the zoning layer chooses Pastry

Table 4.2. Zoning layer API

function	definition
initialize()	to connect to the game world
step_up(zone)	to *step up* to zone owner
join(zone)	to become zone member and listen to *update* messages
update(zone, data)	to *updated* modified GSD
commit(zone, data)	to send a commit message
release(zone)	to release direct connection to the zone owner
step_down(zone)	to *step down* from a zone owner and close all connections to zone members

```
enum{
  OWNER,
  MEMBER,
  DATA,
};
struct DHT_DATA{
  unsigned int type;
  unsinged int data_length;
  char data[];
  struct DHT_DATA *next;
};
```

Fig. 4.7. Data Structure of Zone

message forwarding or the end-to-end TCP connection according to the node status and the data type of the message. On the game program side, the zoning layer represents the interface necessary to control its node status for each zone, as well as the interface of the network layer including the Pastry overlay network. Game programs access the zoning layer by using the APIs listed in Table 4.2.

Data Structure of Zone

The data structure of the zone is described in Figure 4.7. *DHT_DATA* is the basic data structure, and it constructs a one-way list. If the data type is *OWNER*, the DHT_DATA contains the identifier of the zone owner or that of a candidate for a new zone owner. The DHT_DATA is used as a part of the members' list when the data type is *MEMBER*. Each GSD is contained in the *DATA* type DHT_DATA.

```
zone data{
  OWNER{
    data_length
    "node1.example.com:8472"
  }
  MEMBER{
    data_length
    "node2.example.com:8472"
  }
  DATA{
    data_length
    binary_data
    }
  MEMBER{
    data_length
    "node3.example.com:8472"
  }
}
```

Fig. 4.8. Example of Data on Zone Data List

New DHT_DATA is added in the tail of the zone data list by the *set* function of Pastry for the ZFM. Focusing on *OWNER* type data, the zone data list is attached to the candidate list of the zone. The zone data structure is a one-way list; therefore, when a user node searches the zone owner on some zone, the zone owner is the node whose identifier is contained in the first *OWNER*-type data listed in the zone data.

Latency Optimizing Techniques

Most types of MOGs require a short response latency of less than 200 ms[238]. We designed the ZFM to optimize response latency. In addition,

we add two latency optimizing techniques onto the implementation of the zoning layer; namely, data caching, and connection caching.

The zone owner has the permission to write to the master data of GSD on its governing zone. However, updating data on a data holder node through the DHT forwarding paths causes more latency than the modifying data on the local storage of a zone owner. Therefore, by using a local cache of the zone data list on a zone owner as master data of the GSD on its zone, the local cache is able to cut the response latency caused by searching the data holder node to modify the GSD through DHT message forwarding.

Along with data caching on each zone owner, we combine connection caching to reduce the response latency on the announcing updated GSD. Each zone member of a zone is listed on the zone data list of the zone; therefore, a new zone owner can understand all current zone members. When a user node becomes a new zone owner, the new zone owner establishes an end-to-end TCP connection to each zone member, and the zone owner keeps these TCP connections until the zone owner leaves the zone. We call each TCP connection between a zone owner and a zone member a *transfer path*. When a zone owner changes the GSD on its local cache of the zone data, the zone owner announces updated GSD for each zone member directly through transfer paths. In parallel with updating zone members' GSD, the zone owner also updates GSD on the data holder node through DHT message forwarding to avoid loss of the current GSD of the zone. Hence, every node can understand new zone owner by following in sequence the latest GSD from DHT overlay network.

4.4.3 Procedures

Our zoning layer implementation provides several APIs for game programs (Table 4.2). Using these APIs, game programs should only be concerned about their own node status for each zone. On the assumption that game programs are written in an event driven model, we have designed and implemented APIs. Figure 4.9 shows pseudo codes of a zoning layer API.

Initialize function is used when a user node joins to the DHT overlay network where a MOG runs. Initially, for the user node just joining the DHT network, the user's node state is an independent state for all zones. The node tries to participant in several zones which have the GSD that the user node must read or change according to the game sequence.

When a user node wants to change the GSD of a zone, the user node tries to become the new zone owner of the zone. Next, the user node takes a *step up* procedure. First, the user node adds its identifier to the zone data list on the data holder node through the DHT forwarding path, and reads the current zone data list from the data holder node. If the user node finds the *OWNER*-type data it wrote by itself first when the user node searched the current candidate nodes, then the user node changes its node status to zone owner, reads the current zone data list again, comprehends all current zone member nodes, and establishes a transfer path to each zone member. Establishing a transfer path tells each zone member about the arrival of a new zone owner. If the user node finds an other node's identifier first when searching the candidate nodes, the user node realizes that the other node is the zone owner, then the user node removes the *OWNER*-type data it committed by itself from the zone data list.

If the zone owner already exists when an independent node wants to modify the GSD, or when an independent state node wants to just read the current GSD or to receive the updated GSD from the zone owner, the independent node changes its status to zone member by a *join* function. *Joining* steps are as follows: an independent node changes its status to zone member, writes its identifier in the tail of the zone data list as *MEMBER*-type data, and reads the latest zone data list from the DHT overlay network. Next, the new zone member checks the zone owner identifier and

```
function initialize(){
  DHT_join();
}
function step_up(zone){
  DHT_append(zone, mydata.hostdata + type::OWNER);
  data_list = DHT_get(zone);
  foreach element (data_list){
    switch(element.type){
      case MEMBER:
        regist_member(zone, element.hostdata);
        break;
      case OWNER:
        if(element.hostdata == mydata.hostdata){
          return success;
        }
        if(alive_check(element.hostdata) == alive){
          DHT_delete(zone, mydata.hostdata + type::OWNER);
          return fail;
        }else{
          DHT_delete(zone, element.hostdata + type::OWNER);
        }
        break;
    }
  }
  return fail;
}

function join(zone){
  DHT_append(zone, mydata.hostdata + type::MEMBER);
  data_list = DHT_get(zone);
  foreach element (data_list){
    switch(element.type){
      case OWNER:
        regist(zone, element.hostdata);
        break;
      case DATA:
        mydata.zone.data = element.data;
        break;
    }
  }
}
function commit(new_data.zone.data){
  send(owner.hostdata, new_data.zone.data);
}
function update (mydata.zone.data){
  foreach member (zone.member_list){
  send(member, new_data.zone.data);
  }
  DHT_delete(zone, old_data.zone.data);
  DHT_write(zone, new_data.zone.data);
}
function step_down(zone){
  DHT_delete(zone, mydata.hostdata + type::OWNER);
  foreach member (zone.member_list) {
  disconnect(member);
  }
}
function release(zone){
  DHT_delete(zone, mydata.hostdata + type::MEMBER);
  disconnect(zone::owner);
}
```

Fig. 4.9. Zoning API

tries to establish a transfer path by sending a *join message* to the zone owner. When the zone owner receives the join message, the zone owner adds the new zone member's identifier to the zone data list on its own local cache, and establishes the new transfer path.

When a zone member node wants to change some GSD value, the zone member sends a *commit* message with new GSD to the zone owner. When the zone owner receives a request for modification from some zone member or the zone owner wants to modify some GSD on its own zone, the zone owner calls an *update* function. After judging the conflict or the consistency of the GSD, the zone owner modifies the GSD on its local cache, announces new GSD to each zone member through transfer paths, and sends the new GSD to the data holder node via the DHT message forwarding for the purpose of backup.

If a zone member *leave* the zone, the zone member shuts down the transfer path and removes the *MEMBER*-type data which contains its own identifier from the data holder node through DHT forwarding path. When the transfer path has been closed by some zone member, the zone owner realizes that the zone member left from the zone, and the zone owner removes the zone member's identifier from its local zone data list.

A zone owner *steps down* to the independent state when the zone owner wants to leave from the managing zone or just wants to resign from the zone owner role. In this case, the zone owner removes its identifier from the *OWNER*-data from the zone data list on its local cache and on the data holder node respectively. Next, the zone owner closes all transfer paths, and changes its status to an independent state. All zone members realize the zone owner has just left the zone by the closed transfer path. If some zone member simply tries to *commit* after the zone owner leaves, the zone member tries to become the zone owner by calling the *step up* function.

Recovery from Failure

In this section we describe the error processing procedure that occurs when a node is suddenly isolated from the peer-to-peer network because of a network failure.

When a zone member is isolated from the network, this event of isolation looks for the "leave" action called by the zone member. The zone owner is the only node which knows that the zone member has left, so the zone owner removes the zone member's entry from the zone data list on its own local cache. If the isolated zone member comes back to the DHT overlay network, then the zone member realizes that the transfer path has been closed. In this case, the zone member must send a join message to the zone owner again.

When a zone owner has been disconnected from the network, its zone members notice that the zone owner has been isolated from the network due to an absence of heartbeat messages from the zone owner. Next, each zone member sends an owner-lost message to the game program by itself. After sending an owner-lost message, one of these zone members deletes the zone owner entry from the DHT. If the game program on some zone demands a change of the GSD, the zone member tries to "step up".

If a zone owner disappears from the network by a network accident when no zone member is listed, the entry of the zone owner remains on the DHT. In this case, a node becomes a zone member because of the remnant of the zone owner entry on the DHT, but the new zone member cannot establish the transfer path to the registered zone owner, so the zone member notices that the zone owner doesn't exist. Then, the zone member deletes the old zone owner entry from the DHT and tries to step up.

4.4.4 Sample MOG Program

To evaluate the zoning layer, we have implemented a MOG program.

Our sample MOG is "get the game flag" similar to "rally-x", but extended to a multi-player game.

Each player drives a small car and players struggle to get flags distributed on a two dimensional world map. Each player can disturb other players by a smokescreen. If a player's car hits a rock or another player's car, the player has to restart.

To divide the whole world into several zones on this MOG, we used maps and positions as features of the zones. A zone of a map image contains a map image which is 128×128 square when one square is defined as the size of a car image. On the other hand, each zone of positions contains the positions of cars, flags, and smokescreens on a specific map image are defined by a zone of the map image. These zones are distributed onto a DHT overlay network.

4.5 Evaluation

In this section, we evaluate our zoning layer implementation of the zoning layer described in section 4.4. For evaluation, we employed a sample MOG program and examined it by focusing on latency overhead as a performance metric.

4.5.1 Order

We clarify the tradeoff between the order of message forwarding hops on an application layer level and the order of messages to be sent by a node in several models. We describe the ZFM in Table 4.3, in the client-server model in Table 4.4, and in Scribe[35], which is an Application Layer Multicast (ALM) based on DHT message forwarding in Table 4.5, respectively. Scribe is employed by SimMud[165] which is another DHT overlay infrastructure model for MOGs.

Because the DHT overlay network can be accessed on all actions but *commit*, ZFM appears to be a hybrid model of a client-server model and scribe. Therefore, the ZFM appears to provide at least the same response latency as Scribe although the recovery steps of the zone owner (*step up*) may become a bottleneck point for the ZFM.

Comparing the recovery steps of the game server, the ZFM, and the client-server model requires $O(N)$ when sending messages to all clients. The ZFM also needs $O(\log N)$ hop counts to inherit the latest zone data list from the DHT overlay network.

When joining or leaving from a zone, the ZFM requires $O(\log N)$ hop counts to add or to remove its identifier from the zone. In addition, the Scribe needs $O(\log N)$ hop counts to add or to remove its entry from a multicast group.

Updating the GSD, Scribe and the ZFM requires $O(N)$ message forwarding and $O(\log N)$ hop counts. However, the Scribe needs $O(\log N)$ hop counts because the Scribe employs the DHT as the message forwarding path for all messages. On the other hand, the ZFM requires $O(\log N)$ hop counts to back up the current GSD on the DHT overlay network. If only some nodes play the zone owner role of a zone, backing up the current GSD into the DHT overlay network may not be necessary for the zone owner node. Also, updating each zone members' GSD through the direct TCP

Table 4.3. communication overhead of ZFM

	num of messages	hop counts
step up	$O(N)$	$O(\log N)$
join	$O(1)$	$O(\log N)$
step down	$O(1)$	$O(\log N)$
leave	$O(1)$	$O(\log N)$
commit	$O(1)$	$O(1)$
update	$O(N)$	$O(\log N)$

Table 4.4. communication overhead of client server model

	num of messages	hop counts
recovery	$O(N)$	$O(1)$
join	$O(1)$	$O(1)$
leave	$O(1)$	$O(1)$
request	$O(1)$	$O(1)$
update	$O(N)$	$O(1)$

Table 4.5. communication overhead of Scribe

	num of messages	hop counts
create	$O(1)$	$O(\log N)$
subscribe	$O(1)$	$O(\log N)$
unsubscribe	$O(1)$	$O(\log N)$
publish	$O(N)$	$O(\log N)$

connections is processed in parallel by backing up the GSD on the DHT overlay network. Therefore, a zone owner works with $O(N)$ message forwarding and $O(1)$ hop counts from the standpoint of zone members (game clients).

4.5.2 Response Latency Overhead

We have evaluated response latency overhead caused by our zoning layer implementation. MOGs need low latency overhead on message exchanges and on updating the latest GSD in order to provide stress-free interactions among game players. On the evaluation of response latency overhead, we examined two kinds of response latency used by the zoning layer: one on a *step up* action, and the other on *update* action. In section 4.5.1, we described how the step up action may be a bottle-neck point because the step up action requires $O(N)$ message sending and $O(\log N)$ DHT forwarding hops to inherit and to recover zone owner tasks. Also, we mentioned how update action may achieve a short response latency to zone members without concerns about back up on a DHT overlay network.

Response latency on step up action is influenced by two actions: a node becomes a zone owner and establishes direct connections to all zone members. To become a zone owner, a node has to search a data-holder node twice. The first search is needed to write its entry as an owner on the zone-data list, and, after becoming the zone owner, second search is required to fetch all zone data from the data holder-node to use as master GSD, and to comprehend all zone members. Therefore, response latency on step up is affected by the time needed to search a data holder through the DHT forwarding path. Also, response latency on step up is influenced by the number of zone members because of establishing TCP connections between a zone owner and each zone member. In section 4.5.2, we describe the relation between the number of zone members and the response latency of step up action.

On the other hand, response latency on *update* action is affected by updating a GSD on all zone members and on the data holder. We measured response latency on update action as update time, which is influenced by the number of zone members. Through experiments described in section 4.5.2, we try to clarify the relation between update time and the number of zone members on a single zone.

For each evaluation of response latency, we employ a test code which is constructed by libcookai. On the test code, a zone-owner node sends a packet with a 1024 bytes payload to each zone member node over each TCP session. Each zone member node simply receives the test packet.

The threshold of response latency which users can accept without stress is different among MOG types[10, 16, 238, 253]. Our sample game program needs the same response latency accepted by the First Person Shooter (FPS) game. For the evaluation, we set the threshold of the response latency to 200 ms which is acceptable for users on FPS[16].

Response Latency on Step Up

First, we evaluated the overhead of response latency on the *step up* procedure. In this evaluation, we used an experimental environment consisting of 7 FreeBSD PCs, 3 with 500 MHz processors and the other 4 with 850 MHz, interconnected by a 100base-TX switch. All PCs have 256 M bytes memory. To increase the number of zone members, we simulated multiple zone-member nodes by running zone-member processes on PCs.

We estimated the response latency by dividing several time ranges, for example, DHT Looking-up Time (DLT), Establishing Connections Time (ECT), and Total Stepping-up Time (TST). The relationship among these time ranges is as follows:

- DLT
  The time spent for fetching a zone-data list from the DHT overlay network.
- ECT
  The time spent for establishing each

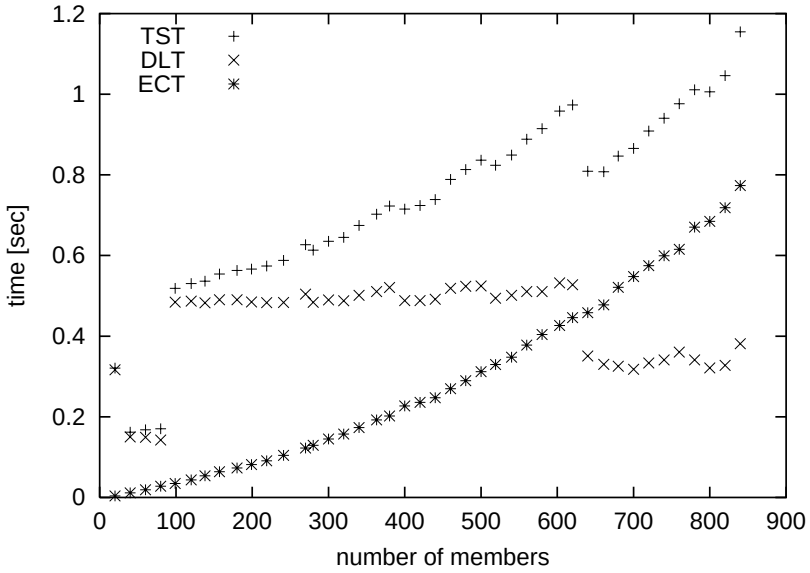

Fig. 4.10. Response Latency on Step up

connection between a zone owner and a zone member.

- TST

  The total time spent for stepping up to a zone owner.

In the test-bed environment described above, the zone owner was placed in only one PC. The scenario of this experiment was as follows: First, only zone members run; next, a new independent node steps up to the zone owner. We evaluated these response latencies while increasing the number of zone members gradually. The result of this experiment is shown in Figure 4.10.

Obviously, the topology of the Pastry forwarding path is affected on by the TST. The ECT draws the liner curve of $O(N)$, but the DLT draws an incontinuous line. When the number of zone members was 700, the TST was less than that on 600 zone members. This was because that the difference of the ECT on 600 zone members versus on 700 members was shorter than the difference of the DLT. Figure 4.10 shows that a longer Pastry forwarding path is the bottleneck point of response latency on the step up action. The maximum number of zone members which satisfied the threshold was 100 zone members; therefore, a single zone owner can deal with 100 zone members by keeping the TST at less than a 200 ms time threshold.

Response Latency on Updating GSD

Next, we evaluated *update time*. To evaluate the response latency overhead of the update time, we used the experiment on *Starbed* which is a large scale network emulation test-bed environment constructed in Hokuriku IT Open laboratory[229]. On Starbed, 512 PCs are divided into five partitions and inter-connected through several switches. Each PC has Intel Pentium III 1 GHz, 512 MB main memory, two 100 Base-TX network interfaces, one of which is connected to the control network and the other to the experimental network. We ran FreeBSD 4.7 for the operating system on each PC.

For the evaluation of update time, we used 296 PCs on Starbed with a simple network topology such that each PC connected to the same layer 2 network. In this experimental environment, we did two experiments about update time. We estimated update time on each experiment. The start of the update time was defined as the time when a zone owner sends a test packet, and the end of the update time was when a zone member received the test packet. In the experiments,

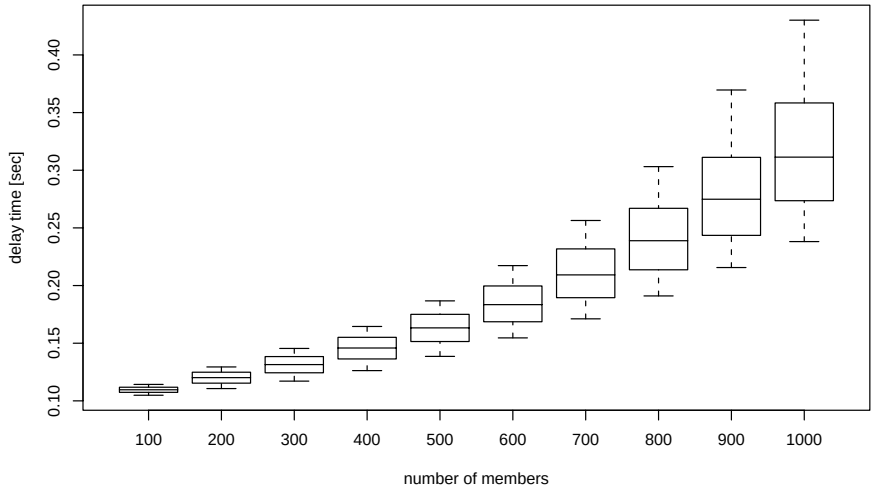

Fig. 4.11. Update time

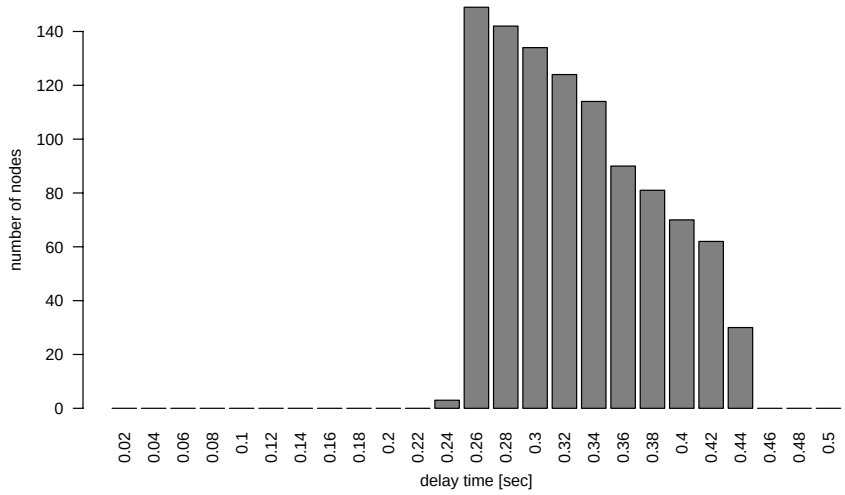

Fig. 4.12. Histogram of update time

we measured the update time on each zone member node, and drew the results using a box-whisker plot (Figure 4.11, 4.13).

First, we evaluated the effect of the number of zone members on a single zone. In this experiment, we used a PC to run only one zone owner process with a 100 ms delay caused by dummynet, Zone member processes ran on other 295 PCs uniformly.

Figure 4.11 shows the trend of distribution for the update time, and Figure 4.12 represents the distribution of the update time when the number of zone members was 1000. According to these figures, although dummynet caused a 100 ms delay, the minimum update time was 240 ms and all update times on each zone member were less than 440 ms, even when the number of zone members was 1000. The maximum number of zone members which a single zone owner can treat by satisfying the 200 ms threshold was 500 members.

In the second experiment, we measured the effect of the number of zones needed to update

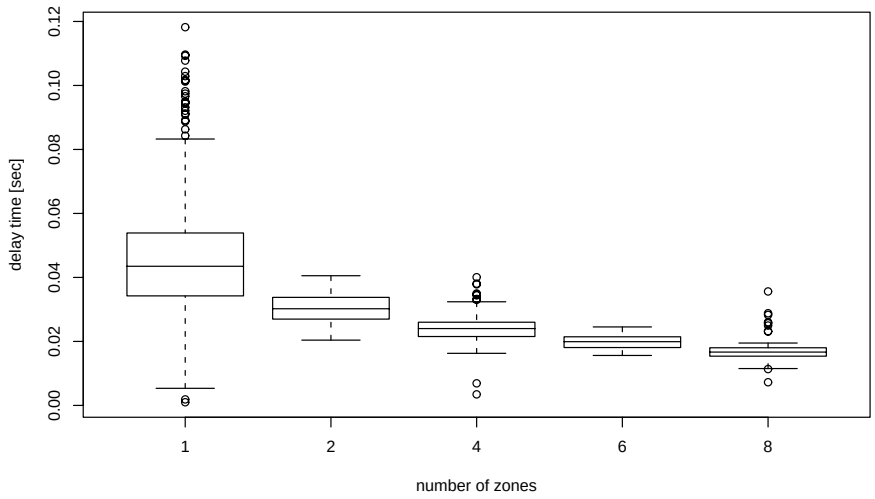

Fig. 4.13. Effect of the number of zones to the update time

time when the total number of zone members was fixed on 297 nodes. We changed the number of zones from 1 to 8, and we distributed zone members to each zone equally. Figure 4.13 shows the result of this experiment. In Figure 4.13, we divided a zone which has many members into several small size zones to enable the reduction of the response delay and to stabilize the distribution of the update time.

From the results of these experiments, we can say that a single zone owner can deal with user nodes as well as a single, not-clustered centric MOG server, and zoning can reduce the overhead on a zone owner and provide MOG the scalability necessary for at number of users of each zone.

Bandwidth Requirements

A zone owner has to update all zone members' GSD through unicast; therefore, our zoning layer implementation consumes bandwidth when a zone owner updates the GSD. Also, a single user node can become the zone owner on several zones, and if a single user node becomes the zone owner of all zones, the single node requires the same upstream bandwidth as the downstream bandwidth required for a single centric MOG server. Required bandwidth of a user node is described as follows:

about some node j ($j = 1, 2, \ldots, m$) in zone i ($i = 1, 2, \ldots, n$), node j updates GSD as the zone owner

- a_{ij}: whether or not node j is an owner on zone i, that is, a_{ij} is 0 or 1
- N_i: the number of members on the zone i
- F_i: frequency of updating GSD on zone i
- G_i: average size of updating GSD per node on zone i
- M_i: required number of messages sent by the zone owner on zone i
- B_i: required upstream bandwidth consumption on the zone owner on zone i

The total upstream bandwidth consumption on node j (B_{T_j}) is:

$$\begin{aligned} B_{T_j} &= \sum_i a_{ij} B_i \\ &= \sum_i a_{ij} N_i G_i F_i \end{aligned}$$

4.6 Other MOG models based on P2P overlay network

SimMud[165] and the PP-CA model[242] are other approaches to MOG infrastructure based on to peer-to-peer overlay network. These approaches provide audit mechanisms for GSD

consistency by a third person or by an authority.

SimMud employs Pastry[265] and Scribe[35] as base components of its architecture. In the SimMud approach, the authoritative role is given to a data holder node, which is called "coordinator". By randomly mapping data holder nodes on the DHT, SimMud prevents game players from cheating global states because the coordinator is rarely interested in the GSD stored in its local storage. Also, by preparing several replicas of a coordinator, SimMud provides fault tolerance.

Pellegrino et al. has proposed the PP-CA model, which is a peer-to-peer overlay MOG infrastructure with a central arbiter server[242]. In the PP-CA model, a central arbiter server only audits inconsistencies of the GSD and resolves the inconsistencies. Other messagings such as updating the GSD are processed by user nodes through a peer-to-peer overlay network.

Pellegrino et al. analyzed three different models: the client-server model, the peer-to-peer model, and the PP-CA model by using an open source MOG program, BZFlag[260]. The analysis shows that the PP-CA model can reduce the bandwidth requirement of the central arbiter and resolve inconsistencies of the GSD without a complex distributed agreement protocol.

Next, we try to compare ZFM, SimMud, and the PP-CA model (Table 4.6. Durability of GSD on each model is affected by the employed peer-to-peer overlay network.

MOG, ZFM, and PP-CA models are superior to SimMud, because SimMud is a customized model only for a Massive online RPG and MOG, ZFM, and PP-CA models are more adaptable.

Table 4.6. Comparing Three Models

	ZFM	SimMud	PP-CA
durability	○	○	○
adaptability	○	×	○
scalability	○	○	—
response latency	○	×	△
bandwidth	×	○	△
cheat proofing	×	△	○
incentive to serve	○	×	○

SimMud has scalability ensured by a simulation. ZFM also has scalability; however, we have not evaluated ZFM in a simulation with a size as large as that of SimMud. PP-CA has been evaluated only in a small LAN environment; therefore, we cannot discuss the scalability of the PP-CA model.

ZFM is a more latency-optimized approach than SimMud because SimMud uses Pastry[265] as the DHT, and Scribe[35] as the message exchange method based on an application-layer multicast (ALM). While ALM reduces the bandwidth consumption of the coordinator, it incurs network delay by crossing several hops on both the DHT and the ALM. In our zoning layer approach, each node exchanges messages directly; therefore, ZFM can achieve a shorter response latency than the ALM, except for the initial rendezvous by the DHT. But ZFM consumes more bandwidth than Scribe because ZFM updates the GSD through unicast connections.

The PP-CA is a hybrid model of the peer-to-peer and client-server. Of course, the bandwidth consumption of the central arbiter server is low. However, in user nodes, the response latency and bandwidth consumption are affected by the data transfer protocol among user nodes.

The ZFM distributes arbiter servers if every zone owner works with fair play. If we assume that malicious users join in the ZFM as malicious zone owners, then the ZFM has the drawback of cheating or unfairness. Although SimMud equips cheat proofing through a third person, SimMud's cheat proofing can be overwhelmed by overriding numerous malicious nodes. The central arbiter server on the PP-CA is the authority or certificate server of the game; therefore, the PP-CA is tolerant to cheating or unfairness.

The SimMud third person check employs a coordinator who is not interested in the GSD of the managing zone. The game server tasks highly consume the resources of a user node; for this reason,

an incentive is needed for the user nodes to process the game server tasks. However, a third person check of SimMud employs a coordinator who is randomly selected and may be not be interested in the GSD of the managing zone; hence, no incentive or interest for the randomly selected third person exists. If a user is not interested in a particular zone, the user cannot grasp what is needed by most other users on the zone correctly. If several MOGs run on the same DHT overlay network of the SimMud, this forces user nodes to act as the game server for several MOGs. In such a situation, no merit exists on the user's machine. Dealing with several MOGs on the same DHT overlay network is difficult for SimMud.

On the other hand, in both the ZFM and PP-CA, the data transfer and judgment are processed by nodes interested in the same game world. In other words, an incentive to serve GSD in both models exists.

4.7 Related Work

The scalable data dissemination problem has been addressed in the application-level multicast literatures[35, 334], where large receiver groups are of particular concern. In contrast, our work focuses on zone-local data dissemination with low latency.

A large number of small groups can be supported in small-group multicast protocols[314]; our work can exploit such infrastructure support, for the purpose of efficient data dissemination from the zone owner.

While we have looked only at the application-layer a topology in this paper, topology-aware overlay[255] will further reduce the latency of intra-zone communication by optimizing the network-layer topology.

The API described in this paper resembles the CAST interface which is part of the common API effort[49]. However, the underlying semantics have notable differences: zone-local serializability, and the presence of multiple roles. Typical any-source multicast protocols are not serializable, in the sense that one particular receiver cannot ensure the same order of packet arrival as other receivers. In the MOG context, we believe that the serializability is of particular importance.

4.8 Future Work

We evaluated the ZFM by focusing on response latency, and showed the scalability of the ZFM. However, our experimental environment was conveyed on a local subnet; therefore, we have to measure the scalability of the ZFM on Internet size topology. Also, we have not had subjective evaluations, so we plan to have several subjective evaluations using our sample game program shown in section 4.4.4.

We have constructed the ZFM on the DHT in order to achieve data consistency on the assumption that the employed DHT has strong durability. However, our implementation of Pastry is not durable. When the data holder node leaves the DHT overlay network, no other node can refer to the GSD backed up on the disappeared data holder node until the data holder node returns again to the DHT network. In future work, we need consider the durability of GSD; a method such as OceanStore[170], which uses the DHT overlay network as its largest storage, would be useful.

In our model, increasing the number of zone members increases the CPU and bandwidth overhead on the zone owner. To solve this problem, increasing the number of zone owners in a zone actively is necessary, in order to distribute a zone owner's tasks. In this case, achieving consistency of the global state is difficult; we should consider a the synchronization of data between multiple zone owners in a specific zone. We may have to consider workaround to reduce the overhead of a zone owner with numerous zone members.

Basically, for our design policy of the ZFM, we assume that the protection for cheating or unfairness on the ZFM is managed by participant users. In section 4.6, we discussed how the third-person check employed by SimMud[165] is

not suited for the ZFM. Reputation techniques on a peer-to-peer overlay network[158] meets the ZFM requirements, because such a reputation system is based on interests.

However, most peer-to-peer overlay network have congenial defects that apply to malicious user nodes, such as catastrophe by a betrayer, hijacking by numerous malicious nodes, or undermining a chain of vouchers from forged multiple identities[63, 158]. A central arbiter server may resolve these threats by providing a consistency check and certificates of the users. Therefore, the hybrid model of the ZFM and PP-CA can be constructed. Such a hybrid model can present short response latency and scalability with consistency and authentication.

4.9 Conclusion

In this chapter, we have proposed the Zoned Federation Model, which adapts MOGs to peer-to-peer overlay networks. In this model, the whole game world is divided into several zones; each zone is maintained by a federation of nodes: an owner and one or more members. The zone owner plays two critical roles. First, it provides zone-local serializability of state changes by aggregating modifications from all members, and by sending state-change notifications to all members. Second, the ZFM ensures the consistency of changes committed by other member nodes. The DHT harnesses this zoning layer by providing rendezvous capability and by working as a backup storage medium for zone data.

We have applied this model to our prototypical MOG implementation, with which we have evaluated latency and scalability. Our experimental results show the relation between latency of update time and the number of zone members on a single zone, and represents the effectiveness of distributing the functions of a centric authoritative node to several zone owners. Moreover, we have compared other models with ours according to the number of messages and the order of hop counts, and we have described the upstream bandwidth consumption of a zone owner node.

On our zoning layer implementation, the whole game world can be divided into several zones with no restrictions. Therefore, by considering the appropriate number of zones and the permissible number of zone members on a single zone according to our experimental results, we showed how game creators can design scalable MOGs on peer-to-peer environment with the short response latency required by each type of MOG.

第 5 章 WOT for WAT: Spinning the Web of Trust for Peer-to-Peer Barter Relationships

abstract[1]

Peer-to-peer complementary currencies can be powerful tools for promoting collaborations and building relationships on the Internet.

i-WAT[272] is a proposed such currency based on the WAT System[324], a polycentric complementary currency using *WAT tickets* as its medium of exchange. Participants spontaneously issue and circulate the tickets as needed, whose values are backed up by chains of trust. *i*-WAT implements the tickets electronically by exchanges of messages signed in OpenPGP[31].

This chapter clarifies the trust model of *i*-WAT, and investigates how it is related with that of PGP[297]. To implement the model by dynamically building an appropriate web of trust (WOT), the author claims that it would suffice if the behaviors of participants satisfy the following three properties:

1. *mutual signing by knowing*, or any two mutual acquaintances sign the public keys of each other,
2. *mutual signing by participation*, or the drawer and a user of an *i*-WAT ticket sign the public

1 This chapter is an extended version of the paper with the same title, which is to appear in the IEICE TRANSACTIONS on Communication in April 2005.

keys of each other, and

3. *mutual full trust by participation*, or the drawer and a user of an *i*-WAT ticket fully trust each other, and a recipient fully trusts the corresponding user of a ticket, in the context of PGP public key signing.

Likelihood of satisfaction of these properties is supported by the (dis)incentives imposed by the semantics of *i*-WAT.

A reference implementation of *i*-WAT has been developed in the form of an XMPP[270, 271] instant messaging client. We are beginning to put the currency system into practical use.

5.1 Introduction

5.1.1 Peer-to-peer complementary currencies and their potential impacts on the Internet

Distributed autonomous (or peer-to-peer) systems, such as an overlay network of people over the Internet, require coordination among participants to achieve their goals. Since each participant may behave selfishly to maximize their benefit, *incentive-compatibility*[97], roughly restated as the goal of the system being accomplished by collection of selfish behaviors, becomes important. Because relationships among participants in such a system necessitate fair exchanges of resources, the medium of exchange must take an important role.

Money is a well-known medium of exchange, but its scarcity has caused a lot of problems. *Complementary currencies*, or alternative forms of monetary medium, have been proposed and tested to achieve an autonomous, sustainable local economy even in short of money. There have been succeeding cases, such as experiments in Wörgl in 1932 (stamp money[275]), in Comox Valley in 1983 (Local Exchange Trading System[276]) and in Ithaca since 1991 (Ithaca HOURs[109]).

Many of the outcomes are short-lived, however, because most of the existing complementary currencies are dependent on the qualities of their administrations. It would thus benefit the autonomy and sustainability of economy if we could design an administration-free complementary currency; if we want to make a peer-to-peer world, money too needs to be peer-to-peer.

If such peer-to-peer complementary currencies are applied to the Internet, it would benefit many areas including multicast cost sharing, inter-domain routing, web caching, file sharing, distributed task allocation, and other application-layer overlay networks. Freedom to pursue these possibilities depends on whether we can have a free economic medium or not.

5.1.2 Contribution of this chapter

i-WAT[272] is a proposed such currency based on the WAT System[324], a polycentric complementary currency using *WAT tickets* as its medium of exchange. A WAT ticket is like a bill of exchange, but without a specified redemption date or place. *i*-WAT implements the tickets electronically by exchanges of messages signed in OpenPGP[31].

This chapter clarifies the trust model of *i*-WAT, and investigates how it is related with that of PGP[297]. In particular, this chapter deduces three properties satisfying which the participants can dynamically extend their webs of trust to accommodate trades using *i*-WAT. This can be reflected to the designs of software tools which implement the *i*-WAT protocol.

5.2 Background

5.2.1 Digital signature

Digital signature is an essential technology for designing a dependable economic medium, which can provide a proof of debits or credits.

Throughout this chapter, let us use notations from[29] for formalization, with additional abstractions built upon them to fit our purposes.

Suppose Alice (A) is associated with a public/secret key pair denoted as $\langle K_A, K_A^{-1} \rangle$. To simplify the arguments to follow, let us assume that each user has exactly one key pair associated with them.

A digital signature has two objectives:

1. To prove that Alice once admitted a message m.
2. To prove that m has not been altered since then.

These can be realized by encrypting m with Alice's secret key K_A^{-1}, obtaining $\{m\}_{K_A^{-1}}$ which is only decrypted with her public key K_A. Since K_A^{-1} is a secret known only to Alice, those who could decrypt $\{m\}_{K_A^{-1}}$ can infer that it must have been encrypted by Alice. They can also be certain that m has not been altered since Alice made $\{m\}_{K_A^{-1}}$ if the result of decryption equals m.

Usually, for efficiency reasons, instead of encrypting m itself, a digital signature is made by applying a secure hash function H to m, then encrypting the hash value with the secret key. H must be carefully chosen so that it is computationally infeasible to obtain m' where $m' \neq m$ such that $H(m) = H(m')$.

Definition 1 (digital signature) *Let us write* $A \overset{signs}{\rightarrow} m$ *if and only if A presents both a plain-text message* m *and its encrypted form* $\{H(m)\}_{K_A^{-1}}$. *The latter is called a* signature *on the former.*

The signature can be verified by Bob if he has a copy of Alice's public key K_A. To verify the signature, he calculates $H(m)$ from m, decrypts $\{H(m)\}_{K_A^{-1}}$ with K_A, and compares the two resulted values.

One question is how Bob can be sure that his copy of Alice's public key is genuine.

Definition 2 (validating relation) $x \overset{v}{\rightarrow} y$ *if* x *possesses a copy of* y*'s public key* K_y*, and infers that the copy is genuine.*

Let us also write $x \overset{v}{\leftrightarrow} y$ *iff* $x \overset{v}{\rightarrow} y \wedge y \overset{v}{\rightarrow} x$ (mutually validating relation).

A trust model around validity of public keys is a specific definition of *validating relation* $\overset{v}{\rightarrow}$ in the system in concern. Typically, validity of a public key is supported by a *certificate*, or a signature on the key. For example, if Bob (B) sees Cameron (C) such that $B \overset{v}{\rightarrow} C \wedge C \overset{signs}{\rightarrow} K_A$, then $B \overset{v}{\rightarrow} A$ assuming that C's certificate is trustworthy. This relation is recursive, so that someone needs to self-certify at some point.

A public key infrastructure uses a tree of *certificate authorities*, or issuers of certificates, whose public keys are validated by the parent nodes, rooted by a self-certifying authority.

5.2.2 Web of trust

In a *web of trust*, however, responsibility for validating public keys is delegated to people one trusts, without necessitating certificate authorities. It is a network of people signing the public keys of others.

Signing relation $\overset{s}{\rightarrow}$ states that one certifies that its copy of someone's public key is genuine.

Definition 3 (signing relation) $\overset{s}{\rightarrow}$ *is defined as follows:*

1. $x \overset{s}{\rightarrow} x$
2. $x \overset{s}{\rightarrow} y$ *if* $x \overset{signs}{\rightarrow} K_y$

Let us also write $x \overset{s}{\leftrightarrow} y$ *iff* $x \overset{s}{\rightarrow} y \wedge y \overset{s}{\rightarrow} x$ (mutually signing relation).

Definition 4 (signing-apart relation) $\overset{s[n]}{\rightarrow}$ *is defined as follows:*

1. $x \overset{s[0]}{\rightarrow} x$
2. $x \overset{s[1]}{\rightarrow} y$ *if* $x \overset{s}{\rightarrow} y \wedge x \neq y$.
3. $x \overset{s[a+b]}{\rightarrow} z$ *if there exists* y *such that* $x \overset{s[a]}{\rightarrow} y \wedge y \overset{s[b]}{\rightarrow} z$.

Let us also write $A \overset{s}{\rightarrow} B \overset{s[n]}{\rightarrow} C$ in place of $A \overset{s[n+1]}{\rightarrow} C$ if $A \overset{s}{\rightarrow} B \wedge B \overset{s[n]}{\rightarrow} C$ (expansion of signing-apart relation) in order to clarify who stands in between the chain of signing relations.

Definition 5 (web of trust) *A web of trust for* x *is a set of all* y *such that* $x \overset{s[n]}{\rightarrow} y$ *where* $n \geq 0$.

A specific validation relation needs to be defined over a web of trust. PGP (Pretty Good Privacy) is an example of a cryptographic technology which defines such a relation. Let us use GnuPG[296] as our choice of implementation of OpenPGP[31] standard.

5.2.3 PGP trust model

Let us further define that $\mathcal{T}_x$ is the set of users x considers fully trustable, and $\mathcal{T'}_x$ is the set of users x considers marginally trustable.

In the context of PGP public key signing, *fully trustable* means that one considers that the owner of a public key has an excellent understanding of key signing, and his or her signature on a key would be as good as their own, and *marginally trustable* means that one considers that the owner of a public key understands the implications of key signing and properly validates keys before signing them[297].

The PGP trust model is a definition of *validating relation* $\overset{v}{\rightarrow}$ over a web of trust.

Definition 6 (PGP trust model) $x \overset{v}{\rightarrow} y$ *if*

1. *sufficient number of valid key owners sign y's public key, i.e.*
   (a) $x \overset{s}{\rightarrow} y$, *or*
   (b) *there exist at least f instances of z such that* $z \in \mathcal{T}_x$, $x \overset{v}{\rightarrow} z \wedge z \overset{s}{\rightarrow} y$, *or*
   (c) *there exist at least m instances of z such that* $z \in \mathcal{T'}_x$, $x \overset{v}{\rightarrow} z \wedge z \overset{s}{\rightarrow} y$; *and*
2. $x \overset{s[n]}{\rightarrow} y$ *where* $n \leq h$,

where f, m and h are the required number of fully trusted key owners, required number of marginally trusted key owners, and number of maximum steps in the path in the web of trust tracing x back from y, respectively.

Let us define the marginally validating relation ($\overset{v}{\rightarrow}$) as follows (although this is not used in the design of our currency):

Definition 7 (weak PGP trust model) $x(\overset{v}{\rightarrow})y$ *if*

1. *insufficient number of valid key owners sign y's public key, i.e.*
   (a) *there exist at least one but less than f instances of z such that* $z \in \mathcal{T}_x$, $x \overset{v}{\rightarrow} z \wedge z \overset{s}{\rightarrow} y$, *or*
   (b) *there exist at least one but less than m instances of z such that* $z \in \mathcal{T'}_x$, $x \overset{v}{\rightarrow} z \wedge z \overset{s}{\rightarrow} y$; *and*
2. $x \overset{s[n]}{\rightarrow} y$ *where* $n \leq h$

By default, GnuPG defines $f = 1$, $m = 3$ and $h = 5$.

5.2.4 The WAT System

Overview

The WAT System[324] is a complementary currency designed by Mr. Eiichi Morino, the founder of Gesell Research Society Japan[105]. A *WAT ticket*, a physical sheet of paper resembling a bill of exchange, is used as the medium of exchange in the system.

A lifecycle of a WAT ticket involves three stages of trade (illustrated in Figure 5.1):

1. Issuing—the birth of a WAT ticket
   A *drawer* issues a WAT ticket by writing on an empty form the name of the provider (*lender*) of the goods or service, the amount of debt[2], the present date, and the drawer's signature. The drawer gives the ticket to the

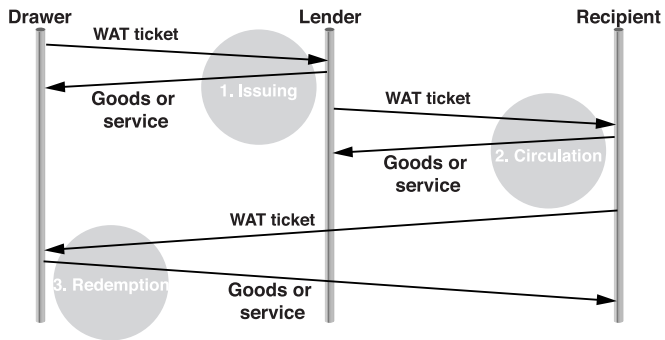

Fig. 5.1. Three stages of trading with a WAT ticket

2 Typically in the unit kWh, which represents cost of producing electricity from natural energy sources.

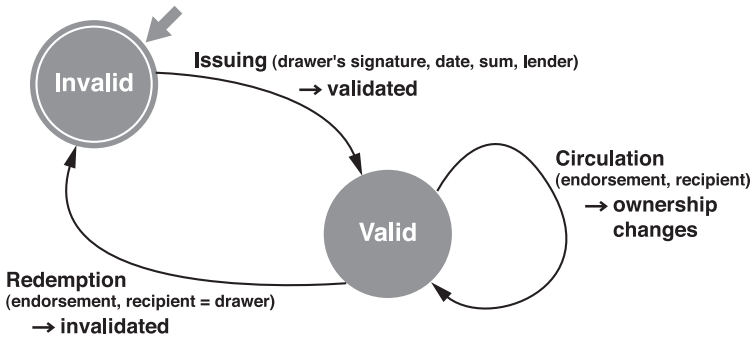

Fig. 5.2. State machine of a WAT ticket

lender, and in return obtains the goods or service.

2. Circulation—ordinary exchange
   The person to whom the WAT ticket was given can become a *user*, and use it for another trading. To do so, the user writes the name of the recipient, as well as their own, on the reverse side of the ticket. The recipient will become a new user, repeating which the WAT ticket circulates among people.
3. Redemption—the return of the WAT ticket
   The WAT ticket is invalidated when it returns, as a result of a trade, to the drawer.

Figure 5.2 shows the state machine of a WAT ticket.

Distinctive features

Autonomy Anyone can spontaneously become a member of the WAT System with a sheet of paper if they follow the above protocol.

Compatibility A WAT ticket is compatible with any other WAT tickets in the world, so that the currency system is operable globally, as long as the drawer can be credited.

Extensibility The protocol illustrated in Figure 5.1 and 5.2 defines *the WAT Core*, the essence of the WAT System. An *extended part* can be defined for a new currency based on the WAT System, stating, for example, the region, group and duration in which the tickets are usable, as well as the unit in which the debit is quantified.

Security In case the drawer fails to meet their promise on the ticket, the lender assumes the responsibility for the debit. If the lender fails, the next user takes over. The responsibility follows the chain of endorsements. The longer the chain is, the more firmly backed up the ticket is. Therefore the length of the chain of endorsements represents the extent of trust the ticket has gained.

5.3 *i*-WAT: the Internet WAT System

5.3.1 Overview

i-WAT is a translation of the WAT Core onto the Internet. In *i*-WAT, messages signed in OpenPGP are used to implement transfers of an electronically represented WAT ticket. The exchanged messages are called *i-WAT messages*, and the ticket represented by the messages is called an *i-WAT ticket*.

An *i*-WAT ticket contains the identification number, amount of debt and public key user IDs of the drawer, users and recipients. Endorsements are realized by nesting PGP signatures as illustrated in Figure 5.3.

Table 5.1 shows the types of *i*-WAT messages. All *i*-WAT messages are signed by the senders, and are formatted in the canonical form[19] of XML[21] with nested signatures. The messages cause state transfers of an *i*-WAT ticket as illustrated in Figure 5.4.

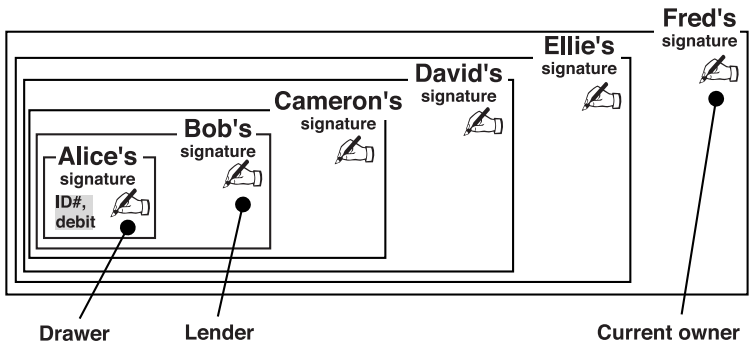

Fig. 5.3. Signature chain in an *i*-WAT ticket

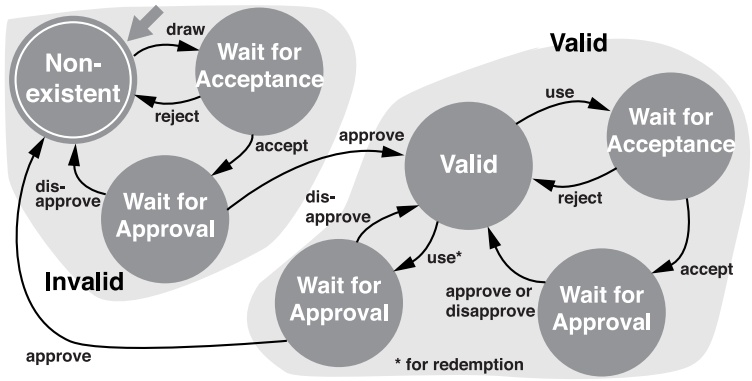

Fig. 5.4. State machine of an *i*-WAT ticket

Table 5.1. *i*-WAT messages

message	sender	receiver	function
<draw/>	drawer	recipient (lender)	draws an *i*-WAT ticket.
<use/>	user	recipient	uses an *i*-WAT ticket.
<accept/>	recipient	drawer and user	confirms readiness to accept the *i*-WAT ticket once it is validated.
<reject/>	recipient	drawer or user*	rejects an *i*-WAT ticket.
<approve/>	drawer	user and recipient	validates an *i*-WAT ticket, and approves the transaction.
<disapprove/>	drawer	user and recipient	denies an *i*-WAT transaction.

* depending on whether the ticket has just been issued or in circulation, respectively.

5.3.2 Changes from the WAT System

Changes in the state machine

Upon translating the WAT Core onto the digital communication domain, the author has made the following changes from the state machine of a WAT ticket:

1. Trades need to be asynchronously performed. Intermediate states, such as waiting for acceptance or approval, are introduced.
2. Double-spending needs to be prohibited. The drawer is made responsible for guaranteeing that the circulating ticket is not a fraud. This means that every trade has to be approved by the drawer of the involved ticket.

Justification of the design

The author regards these changes as necessary modifications to design a dependable economic medium. But they, especially the latter one, may introduce bottlenecks in the system.

This issue has not been quantitatively analyzed yet. However, by a casual analysis, the author believes that the design is justifiable because anyone can spontaneously become a drawer as long as they are trusted, and the relation between their trust and the processing load is incentive-compatible:

1. If everyone is trusted equally in a circle of friends, the load should be evenly distributed among participants.
2. Otherwise, the load should be distributed in an incentive-compatible way:
   (a) If one is late to respond (thus avoids to comply with the imposed load), or tends to fail to answer requests for redemption, they will lose trust from others. Then it will become more difficult for them to have their tickets accepted for trades in the future.
   (b) If one is quick to respond, and accepts requests for redemption with certainty, they will gain more trust from others. Since their tickets become easier to use, they may attract more load. But it will become easier for them to draw tickets at will in the future, and initiate trades spontaneously to obtain goods or services.

5.3.3 Protocol

Issuing—the birth of an *i*-WAT ticket

1. The drawer sends a <draw/> message which contains the public key user IDs of the drawer and lender, identification number and amount of debt. This message becomes the original *i*-WAT ticket after the protocol is completed.
2. The lender sends back the content of the message as an <accept/> message.
3. The drawer sends an <approve/> message to the lender.

Circulation—ordinary exchange

1. The user adds to the *i*-WAT ticket the public key user ID of the recipient, and sends it to the recipient as a <use/> message. This message becomes a valid *i*-WAT ticket after the protocol is completed.
2. The recipient forwards the content of the message to the drawer and user as an <accept/> message.
3. The drawer verifies the ticket, and sends an <approve/> message to the user and recipient.

Redemption—the return of the *i*-WAT ticket

1. The user sends a <use/> message to the recipient, who equals the drawer.
2. The drawer verifies the ticket, and invalidates it as the debit is now redeemed. The drawer sends an <approve/> message to the user.

5.4 *i*-WAT and the PGP trust model

5.4.1 *i*-WAT trust model

Let us define that $t(x)$ is an i-WAT ticket t drawn by x, $\mathcal{U}_{t(x)}$ is the set of users throughout the lifecycle (up to redemption) of $t(x)$, and $y \overset{t(x)}{\rightarrow} z$ denotes that y gives $t(x)$ to z as a result or promise

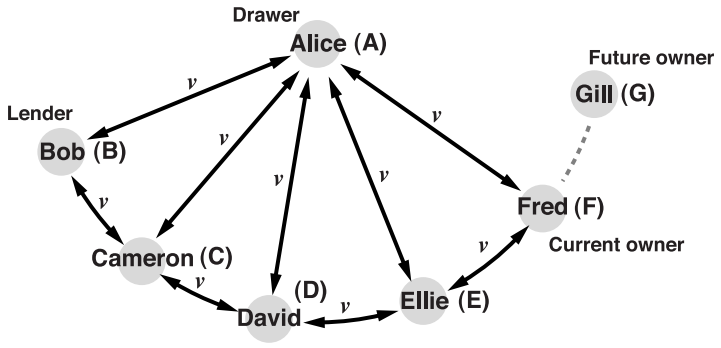

Fig. 5.5. *i*-WAT trust model

of a trade.

The i-WAT trust model is a definition of how *mutually validating relation* $\overset{v}{\leftrightarrow}$ must hold over a network of participants.

Definition 8 (i-WAT trust model) *for every* $t(x)$,

1. *for all* y *such that* $y \in \mathcal{U}_{t(x)}$, $x \overset{v}{\leftrightarrow} y$
2. *for all* y, z *such that* $\{y, z\} \subseteq \mathcal{U}_{t(x)}$, $y \overset{v}{\leftrightarrow} z$ *if* $y \overset{t(x)}{\rightarrow} z$

Figure 5.5 illustrates the model by an example. This model is naturally induced from the necessity for the participants to validate i-WAT messages.

5.4.2 Spinning the web of trust—preconditions

If the PGP trust model over the network of participants does not readily support the above model, the model needs to be implemented by dynamically building an appropriate web of trust. In order to do so, the author claims that it suffices (but not necessitates) if the behaviors of the participants satisfy the following properties.

Property 1 (mutual signing by knowing) *for every* x *and* y,

- $x \overset{s}{\leftrightarrow} y$ *if* x *knows*[3] y

Intuitively, this states that any two mutual acquaintances sign the public keys of each other.

Property 2 (mutual signing by participation) *for every* $t(x)$,

- *for all* y *such that* $y \in \mathcal{U}_{t(x)}$, $x \overset{s}{\leftrightarrow} y$

Intuitively, this states that the drawer and a user sign the public keys of each other.

Property 3 (mutual full trust by participation) *for every* $t(x)$,

1. *for all* y *such that* $y \in \mathcal{U}_{t(x)}$, $x \in \mathcal{T}_y \wedge y \in \mathcal{T}_x$
2. *for all* y, z *such that* $\{y, z\} \subseteq \mathcal{U}_{t(x)}$, $y \in \mathcal{T}_z$ *if* $y \overset{t(x)}{\rightarrow} z$

Intuitively, this states that the drawer and a user are confident about each other that their correspondents have an excellent understanding of key signing, and a recipient is confident about the corresponding user that they have such an excellent understanding. They need to reflect such views in their PGP trust databases.

Let us assume that GnuPG's default values are used for variables f, m and h.

5.4.3 Spinning the web of trust—case studies

Let us justify the above claim by case studies. Throughout the studies, the network of participants in Figure 5.5 is used as an example. It is assumed that no external source of information is available.

Our goal is to show that the i-WAT trust model is satisfied in every stage of trades starting from likely initial states, i.e., a joining party knows someone in the network of participants, if the properties explained in section 5.4.2 are satisfied by the participants.

The statement that follows each claim is both a casual proof and a procedure to achieve the goal.

Issuing

The goal is to form the initial network of participants between Alice and Bob.

Claim 1 $A \overset{v}{\leftrightarrow} B$ *results if Alice knows Bob.*

In case Alice knows Bob

1. By *mutual signing by knowing*,

$$A \overset{s}{\leftrightarrow} B$$

2. By the definition 1a of *PGP trust model*,

$$A \overset{v}{\leftrightarrow} B$$

Circulation

The goal is to let Gill join the existing network of participants.

Claim 2 $G \overset{v}{\leftrightarrow} F \wedge G \overset{v}{\leftrightarrow} A$ *results if Gill knows either Fred or Alice, or someone (in* $\mathcal{T}_G$*) or some people (in* $\mathcal{T'}_G$*) in the network of participants.*

3 In the context of this chapter, *knows* relation is defined to be symmetrical, i.e., y knows x if x knows y.

In case Gill knows Fred

1. By *mutual signing by knowing* and *mutual signing by participation*,

$$G \overset{s}{\leftrightarrow} F \wedge F \overset{s}{\leftrightarrow} A$$

2. By *expansion of signing-apart relation*,

$$G \overset{s}{\leftrightarrow} F \wedge G \overset{s}{\leftrightarrow} F \overset{s}{\leftrightarrow} A$$

3. By the definition 1a of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} F \wedge G \overset{s}{\leftrightarrow} F \overset{s}{\leftrightarrow} A$$

4. $F \in \mathcal{T}_A$ and $F \in \mathcal{T}_G$ by the properties 1 and 2 of *mutual full trust by participation*, respectively. Also, the path length between Gill and Alice is shorter than h. Therefore, by the definition 1b of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} F \wedge G \overset{v}{\leftrightarrow} A$$

In case Gill knows Alice

1. By *mutual signing by knowing* and *mutual signing by participation*,

$$G \overset{s}{\leftrightarrow} A \wedge A \overset{s}{\leftrightarrow} F$$

2. By *expansion of signing-apart relation*,

$$G \overset{s}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} A \overset{s}{\leftrightarrow} F$$

3. By the definition 1a of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} A \overset{s}{\leftrightarrow} F$$

4. $A \in \mathcal{T}_G$ and $A \in \mathcal{T}_F$ by the property 1 of *mutual full trust by participation*. Also, the path length between Gill and Fred is shorter than h. Therefore, by the definition 1b of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} A \wedge G \overset{v}{\leftrightarrow} F$$

In case Gill knows neither Alice nor Fred

The goal can still be met if

1. there is one user x such that $x \in \mathcal{U}_{t(A)}$ who knows Gill and appeared earlier than Fred, and $x \in \mathcal{T}_G$, or
2. there are three users x, y, z such that $\{x, y, z\} \subset \mathcal{U}_{t(A)}$ who all know Gill and appeared earlier than Fred, and $\{x, y, z\} \subseteq \mathcal{T'}_G$.

The proofs for the above two cases are similar; they both involve first establishing $G \overset{v}{\leftrightarrow} A$ by way of someone or some people in the middle, only that the latter is more complex.

Suppose Gill knows Cameron, David, Ellie, and marginally trust them.

1. By *mutual signing by knowing* and *mutual signing by participation*,

$$G \overset{s}{\leftrightarrow} C \wedge C \overset{s}{\leftrightarrow} A \wedge A \overset{s}{\leftrightarrow} F$$

2. By *expansion of signing-apart relation*, and by the definition 1a of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} C \wedge C \overset{s}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} C \overset{s}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} C \overset{s}{\leftrightarrow} A \overset{s}{\leftrightarrow} F$$

3. The above also holds if we replace C with D or E. It is given that $\{C, D, E\} \subseteq \mathcal{T'}_G$. Also, the path length between Gill and Alice is shorter than h. Therefore, by the definition 1c of *PGP trust model*,

$$G \overset{v}{\rightarrow} A \wedge C \overset{s}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} C \overset{s}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} C \overset{s}{\leftrightarrow} A \overset{s}{\leftrightarrow} F$$

4. $C \overset{v}{\leftrightarrow} A$ by the definition 1a of *PGP trust model*. $C \in \mathcal{T}_A$ by the property 1 of *mutual full trust by participation*. Therefore, by the definition 1b of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} C \overset{s}{\leftrightarrow} A \overset{s}{\leftrightarrow} F$$

5. Now that Gill and Alice mutually validate their public keys, they can establish $G \overset{s}{\leftrightarrow} A$ by *mutual signing by participation*.

$$G \overset{v}{\leftrightarrow} A \wedge G \overset{s}{\leftrightarrow} A \overset{s}{\leftrightarrow} F$$

6. $A \in \mathcal{T}_G$ and $A \in \mathcal{T}_F$ by the property 1 of *mutual full trust by participation*. Also, the path length between Gill and Fred is shorter than h. Therefore, by the definition 1b of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} A \wedge G \overset{v}{\leftrightarrow} F$$

Redemption

The goal is to complete the lifecycle of the ticket in concern. This needs to be done without further expanding the existing network of participants.

Claim 3 *$G \overset{v}{\leftrightarrow} A$ results without extending the web of trust any further.*

1. By *mutual signing by participation*,

$$G \overset{s}{\leftrightarrow} A$$

2. By the definition 1a of *PGP trust model*,

$$G \overset{v}{\leftrightarrow} A$$

5.4.4 Justification of the preconditions

Let us casually explain how the preconditional properties are supported by the natural behaviors of people. The formal proof that the design of *i*-WAT is incentive-compatible is left out for a future work.

Mutual signing by knowing

If two parties know each other (well enough), it should be possible to safely exchange the fingerprints[4] of their public keys. Therefore this is only a question of the communication cost.

Mutual signing by participation

Because it becomes easier for other people to join the circle of friends around an *i*-WAT ticket if this property is met, both the drawer and user have incentives to sign each other's public keys after properly validating them.

Mutual full trust by participation

The participants are motivated to fully trust their correspondents in the context of public key signing by the same incentives as the above. Also, they are disincentivized to be negligent of the precautions for signing public keys, in order to protect themselves from possible attacks by impostors.

5.4.5 Possible attacks

Overview

Attacks may be possible by compromising the preconditional properties. When *mutual signing by knowing* or *participation* is compromised, it would result in a failure to build an appropriate web of trust. This is fail-safe, so that no harm will result for other members of the system. This is not the case when *mutual full trust by participation* is compromised. The system should be strong enough to prevent mishaps from resulting when participants show untrustworthy behaviors with respect to signing public keys.

The author claims that a web of trust to support the *i*-WAT trust model protects itself from such a threat.

Case studies

Table 5.2 shows the cases of possible attacks to exploit the first three people appearing in the network of participants in Figure 5.5.

An impostor forges Alice's public key The impostor receives goods or service from Bob in

Table 5.2. Cases of possible attacks

#	signing	compromised	forged key	forger	prevented by
1	$B \overset{s}{\rightarrow} A$	$B \in \mathcal{T}_C$	$\langle K_A, K_A^{-1} \rangle$	an impostor	Bob
2	$B \overset{s}{\rightarrow} A$	$B \in \mathcal{T}_C$	$\langle K_A, K_A^{-1} \rangle$	Bob (Alice is imaginary)	Bob, Cameron
3	$A \overset{s}{\rightarrow} B$	$A \in \mathcal{T}_C$	$\langle K_B, K_B^{-1} \rangle$	an impostor	Alice
4	$A \overset{s}{\rightarrow} B$	$A \in \mathcal{T}_C$	$\langle K_B, K_B^{-1} \rangle$	Alice (Bob is imaginary)	Cameron
5	$B \overset{s}{\rightarrow} C$	$B \in \mathcal{T}_A$	$\langle K_C, K_C^{-1} \rangle$	an impostor or Bob	Bob
6	$A \overset{s}{\rightarrow} C$	$A \in \mathcal{T}_B$	$\langle K_C, K_C^{-1} \rangle$	an impostor or Alice	Alice

4　A fingerprint is a hash value of a key so that the key's identity can be checked with small cost.

return of an empty promise. Bob is incentivized to obtain the fingerprint of Alice's public key directly from her and compare it with that of the forged public key. Otherwise, the debit will become his responsibility.

Bob creates Alice, and forges her public key Bob tries to make an *i*-WAT ticket with an empty promise, and use it against Cameron. However, Bob is disincentivized to create Alice in the first place, because he must take the responsibility of the debit once people discover that Alice is not able to repay. Or he could escape, so that Cameron is incentivized to keep in touch with Bob to make him more traceable.

An impostor forges Bob's public key The impostor receives an *i*-WAT ticket Alice issues without giving her anything in return. Although she can always disapprove further trades with the ticket, she will lose her trust because people can infer that she was a careless signer, making it more difficult for her to participate in future trades. Alice is incentivized to be careful.

Alice creates Bob, and forges his public key Issuing is always the hardest part. Alice tries to make her *i*-WAT tickets easier to circulate by skipping the step. If, for any reason, she fails to meet her promise on the ticket, Cameron must take over the responsibility as Bob does not exist. This can easily be considered an attack to Cameron. Therefore, Cameron is incentivized to keep in touch with Alice to make her more traceable in case she escapes.

An impostor or Bob forges Cameron's public key Someone pretends to be or creates Cameron, and receives the *i*-WAT ticket from Bob giving nothing in return. Bob is incentivized to be careful, and disincentivized to create Cameron because whatever the imaginary friend causes, people would first suspect Bob.

An impostor or Alice forges Cameron's public key Someone pretends to be or creates Cameron, and receives the *i*-WAT ticket from Bob giving nothing in return. Alice is incentivized to be careful because she is the first to be blamed, and disincentivized to create Cameron because when someone disappears with a valid ticket, it means Alice does not have to repay, so that people would consider that she has a motive.

5.5 Deployment

5.5.1 Overview

i-WAT allows the underlying carrier of messages to be existing e-mail or instant messaging systems. As a reference implementation, the author has developed an *i*-WAT plug-in and the hosting XMPP (Extensible Messaging and Presence Protocol)[270, 271] client called *wija*. The software is available from `http://www.media-art-online.org/wija/` (the *i*-WAT plug-in is bundled with all platform-specific packages).

i-WAT and a public key exchange mechanism to support the system have been implemented as extensions to XMPP instant messaging protocol.

The reference implementation has already been in use by the WAT System communities. It has been used, for example, to exchange goods, such as books, with services, such as working hours for developing an open source software, namely *wija* itself.

5.5.2 Support for the preconditional properties

Our software lets users exchange their public keys directly (by way of XMPP servers) without consulting a public key server. From a user's point of view, this is performed by choosing a correspondent from a buddy list, and selecting either importing or exporting their keys. When imported, a window pops up with the fingerprint of the public key, asking the user whether to sign the key or not. This is expected to enhance the ease for *mutual signing by knowing*.

Our software currently does not directly support *mutual signing by participation*. However, the current design of the software uses the buddy

list to locate the owner of a public key, so that new participants will be required to add the drawer in their buddy list if they have not already, to which the drawer would respond by adding them back. Then the above mechanism for key exchange can be used.

Our software currently does not have a support for *mutual full trust by participation*.

5.6 Future work

Our development team will add a user interface to *wija* to support *mutual signing* and *mutual full trust by participation*. We will experiment further how we can reduce the communication cost so that people can easily satisfy the three preconditional properties. At the same time, as we put *i*-WAT into practical use, we will see if these properties are actually useful for building up the circle of friends around an *i*-WAT ticket.

5.7 Related work

5.7.1 Magic Money

Magic Money[329] is an example of message-based currencies on the Internet based on PGP signatures. It was designed and implemented by an anonymous programmer known as Pr0duct Cypher in the early 1990s. Although there were a few enthusiasts, the use of Magic Money did not spread widely for several reasons:

1. It utilized Chaum's blind signature protocol[38] which was patented at the time. Since Magic Money was distributed as a free, open source software, its existence itself was unlawful.
2. It required presence of a server, which had to be maintained by someone.
3. It pursued untraceability while there was nothing to back up the values of the digital coins. The system was regarded as untrustworthy.

The author regards Magic Money as an important experience of deploying a complementary currency on the Internet, and has tried to do the opposites: 1) chosen GnuPG as the implementation of OpenPGP which does not use patented technologies, 2) chosen not to rely on servers (we use XMPP servers as routers), and 3) chosen to give up anonymity to some extent (public key IDs and signing relations are made known) to build up trust instead.

5.7.2 Geek Credit

Geek Credit[167] is an example closer to *i*-WAT. It defines *Geek Credit policy*, which is similar to the *i*-WAT state machine, but the problem of double-spending is handled differently. Geek Credit detects double-spending at redemption, so that each trading does not need to be consulted with the drawer.

While this simplifies the protocol, the risk of attacks is higher for Geek Credit than for *i*-WAT. Having not to consult the drawer also makes the trust model of Geek Credit simpler, but it means that there is no implicit support for building the web of trust dynamically other than joining the circle of friends by knowing the current owner of the ticket. Since the drawer does not have a way to check the usage of their tickets, there is no way to enforce the imposed restrictions by an extended part if there is one.

5.7.3 Ripple

Ripple[103] is another example of a decentralized currency system. It finds a chain of credit connections between parties to make payments. If A is trusted by B and B is trusted by C, and if A wants to make a payment to C, then A pays to B so that B pays to C.

This may work if everyone in the found chain is present on the Internet. The author does not see that as a big problem; *i*-WAT has a similar assumption of the drawers being present on the Internet most of the time.

The behaviors of the two currency systems would look similar in that if A and C do not yet know each other, the system depends on the intermediate person B to secure the trustworthiness between A and C. But *i*-WAT does so by

checking the signatures of B on the public keys of A and C, so that it does not require the presence of B when A and C want to make a transaction. Therefore i-WAT should be both more efficient and more tolerant of failures.

Currently, there is no working implementation of Ripple.

5.8 Conclusions

Peer-to-peer complementary currencies can be powerful tools for promoting collaborations and building relationships on the Internet. i-WAT is a proposed such currency based on the WAT System, a polycentric complementary currency using WAT tickets whose values are supported by chains of trust.

This chapter clarified the i-WAT trust model. To implement the model by dynamically building an appropriate web of trust, the author showed that it would suffice if the behaviors of participants satisfy the following three properties:

1. *mutual signing by knowing*
2. *mutual signing by participation*
3. *mutual full trust by participation*

Likelihood of satisfaction of these properties is supported by the (dis)incentives imposed by the semantics of i-WAT.

The author has developed an XMPP client called *wija* in order to put i-WAT into practical use. The author's team has been experimenting on user interfaces for exchanging public keys, so that participants of i-WAT can satisfy the above properties with little or no subjective communication cost.

Acknowledgment

The author would like to thank Mr. Eiichi Morino and other members of Gesell Research Society Japan for valuable advices and discussions on the content of this chapter, as well as feedback on *wija* software.