平成21年度　　卒業研究論文

匿名通信方式Bifrostにおける匿名性調整機能と汎用アプリケーション対応の実現

指導教官
齋藤 彰一 准教授

名古屋工業大学　情報工学科
平成18年度入学　18115036番

小川 栄司

目 次

第 1 章　はじめに **1**

第 2 章　関連研究 **2**

2.1 Onion Routing 2

2.2 Chord 4

2.2.1 ノード管理 4

2.2.2 検索 5

2.2.3 ノードの参加処理と離脱処理 6

2.3 Bifrost 7

2.3.1 全体構成 7

2.3.2 ノード管理層 8

2.3.3 匿名通信路層 9

2.3.4 可用性と拡張性 12

2.3.5 問題点 12

第 3 章　提案手法 **14**

3.1 受信エリアの設定方法 14

3.1.1 ノードへの ID 割り当て方法 14

3.1.2 参加ノード数の推測 15

3.1.3 受信エリアの設定 16

3.2 SOCKS プロキシ化 18

3.2.1 SOCKS 18

3.2.2 TUN/TAP 19

3.2.3 SOCKS と TUN/TAP の比較 20

第 4 章 実装と動作 **21**

4.1 ID へのレベル定義 21

4.2 SOCKS プロキシ化 22

4.3 動作検証 23

第 5 章 まとめ **26**

謝辞 **27**

参考文献 **28**

第1章

はじめに

近年，インターネットの普及に伴い，医療や行政に関わる機密情報をインターネットを介してやりとりする機会が増えている．機密情報のやりとりには通信内容自体の守秘と送受信者の身元を秘匿する通信の匿名性が必要不可欠である．今日，暗号化技術により通信内容の守秘は実現されている．しかし，IP アドレスと時間から個人の特定が可能であるため，通信の匿名性は完全には実現されていない．よって，これを実現する手法が必要である．一般に，通信における匿名性 [1] とは下記の 3 つを指す．

- 送信者匿名性
- 受信者匿名性
- 送信者と受信者の関係による匿名性

上記の 3 つを実現する代表的な手法として，Onion Routing[2][3] という匿名通信方式が提案されている．しかし，Onion Routing にはいくつかの問題点が存在する．これらの問題点を解決した手法として，本研究室では Bifrost[4] という匿名通信方式の研究と開発を行っている．本研究では，ユーザが Bifrost を利用することを考慮し，匿名性と通信速度の調整機能に加えて汎用アプリケーションにおける通信を匿名化する機構を備えた Bifrost の実装を提案する．

以下，2 章では本研究の基盤となっている関連研究について述べ，3 章では提案手法について述べる．続いて，4 章では実装について述べ，5 章でまとめを述べる．

第2章

関連研究

本章では，多段中継と多重暗号化による匿名通信を行う Onion Routing，分散ハッシュテーブル (DHT) を実現するアルゴリズムの 1 種である Chord[5]，提案手法の基盤となる手法である Bifrost について述べる．

2.1 Onion Routing

図 2.1 に Onion Routing の全体構成の図を示す．Onion Routing は送信者，受信者，複数の中継ノードから構成される．送信者は予めなんらかの方法で受信者と各中継ノードの各々のみが復号可能な暗号化鍵を入手し，受信者との通信内容を多重暗号化することで，通信に用いるメッセージを生成する．表 2.1 に記号を定義する．S から R に至るまでに m 個の中継ノードを経由するメッセージ M_{Nn} は式 2.1 で得られる．

$$M_{Nn} = Key_{Nn}(A_{Nn+1} \parallel Key_{Nn+1}(A_{Nn+2} \parallel \cdots$$
$$Key_{Nm-1}(A_{Nm} \parallel Key_{Nm}(A_R \parallel Key_R(Data)))\cdots)) \tag{2.1}$$

S は M_{N1} を生成し，A_{N1} へ M_{N1} を送信する．M_{Nn} を受信した各中継ノード N_n は自身が持つ復号鍵にて M_{Nn} を復号し，A_{Nn+1} と M_{Nn+1} を得る．その後，N_n は A_{Nn+1} へ M_{Nn+1} を送信する．最終的に，R が受信した M_{Nm+1} を復号し，$Data$ を得る．

このように通信を行うことで，送信者以外はどのノードが送信者または受信者であるかを判別することができない．よって，匿名通信が実現される．しかし，中継ノー

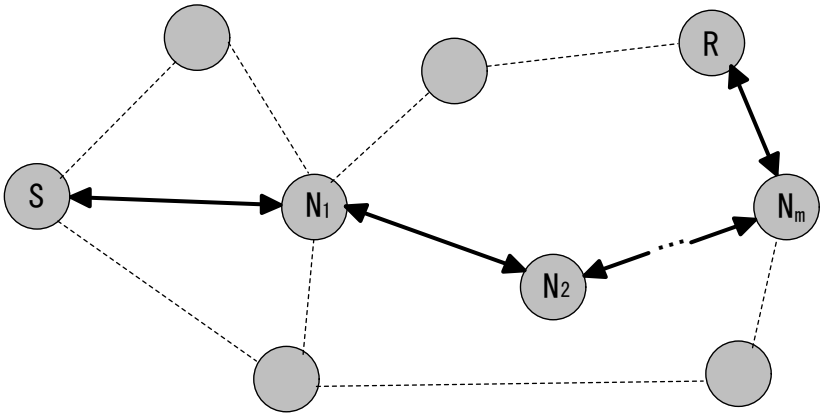

図 2.1: Onion Routing

表 2.1: 2.1 節における記号表

記号	意味
S	送信者 (ノード)
R, N_{n+1}(n は中継ノードの数)	受信者 (ノード)
N_n	n 番目の中継ノード
$Data$	S と R の通信内容
M_X	ノード X が受信するメッセージ
A_X	ノード X の宛先
Key_X	ノード X のみが復号可能な暗号化鍵
$Key_X(A)$	A を Key_X にて暗号化
$A \parallel B$	A と B の結合

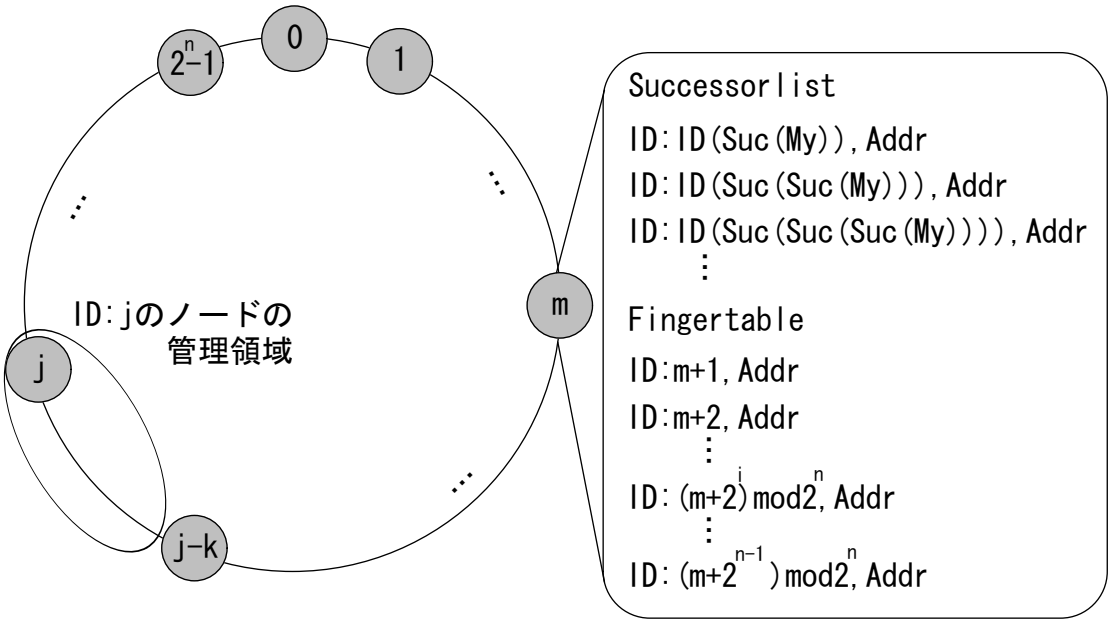

図 2.2: Chord によるノード管理

ドが 1 つでも故障すると，送受信者は通信を継続することができなくなる．そのため，Onion Routing の可用性は低い．また，Onion Routing の拡張性はネットワークの形態に依存する．

2.2 Chord

本節では，DHT のアルゴリズムの 1 種である Chord の本研究における基盤手法に用いられている部分について述べる．

2.2.1 ノード管理

図 2.2 に Chord によるノード管理の図を示す．Chord は $0 \leq NodeID \leq 2^n - 1 (n \geq 0)$ から成る環状の ID 空間を持ち，各ノードはその中で固有の ID を持つ．各ノードは自身の ID を減少させる方向で次に存在しているノードを Predecessor とし，自身の ID と Predecessor の ID の間の ID 空間を管理する．また，各ノードは自身の ID を増

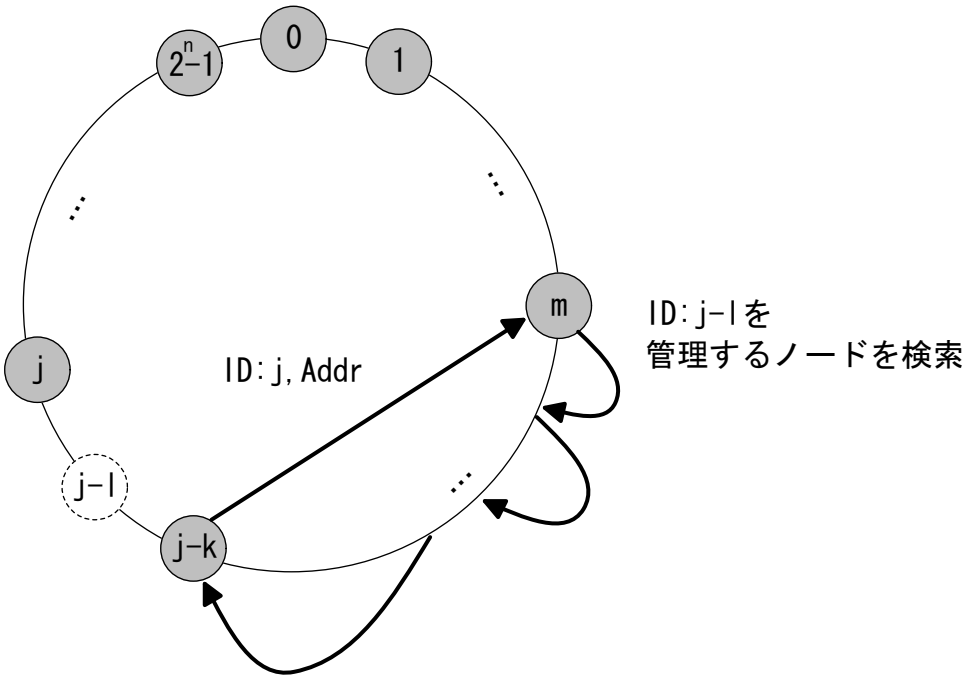

図 2.3: 検索

加させる方向で次に存在しているノードを Successor とし，その ID と宛先を保持する．ここで，$Suc(X)$ をノード X の Successor とすると，各ノードは $Suc(My)$ に加えて $Suc(Suc(My))$, $Suc(Suc(Suc(My)))\cdots$ といったように複数のノードの ID と宛先を Successorlist として保持する．これは，ノードの離脱によるネットワークの破綻を防ぐためである．ノードの離脱については 2.2.3 項にて述べる．$ID(X)$ をノード X の ID とすると，各ノードは $Z = (ID(My) + 2^i) \bmod 2^n (0 \leq i \leq n-1)$ の ID を管理するノードの ID と宛先を Fingertable として保持する．これは，2.2.2 項にて述べる検索の効率を高めるためである．

2.2.2 検索

検索とは，あるノードが指定した ID を管理するノードの ID と宛先を取得することである．図 2.3 に検索の図を示す．検索を行うノードを含めた各ノードは自身が保持する Successorlist および Fingertable を参照し，検索対象の ID 以下の最大の ID のノー

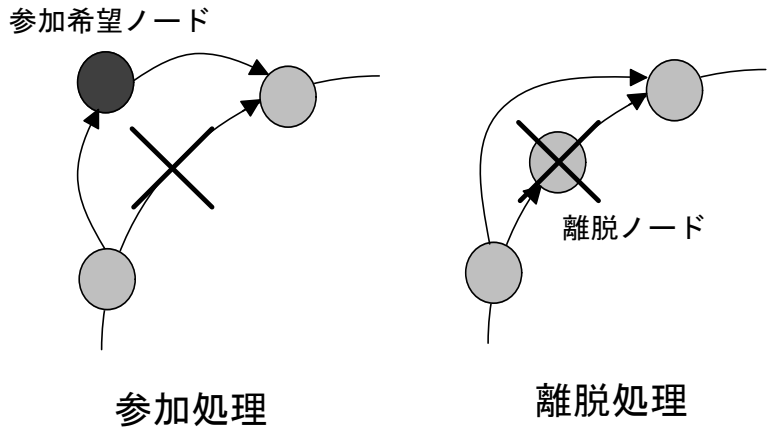

図 2.4: ノードの参加処理と離脱処理

ドに順次問い合わせを行う．検索対象の ID を管理するノードの ID と宛先を保持するノードが検索を行うノードに検索対象の ID を管理するノードの ID と宛先を通知することで，検索は実現される．

2.2.3 ノードの参加処理と離脱処理

図 2.4 にノードの参加処理と離脱処理の図を示す．参加処理は，あるノードがネットワークへの参加を希望する際に行われる．参加を希望するノードはホスト情報などから一意な ID を生成し，自身の Successor となるノードの ID と宛先を保持する．参加を希望するノードの Predecessor となるノードは，参加を希望するノードを自身の Successor としてその ID と宛先を保持する．以上をもって，参加処理は実現される．

離脱処理は，各ノードが Successor に対して定期的に生存確認を行い，その離脱が検知された際に行われる．離脱を検知したノードは離脱したノードの Successor すなわち 2.2.1 項における $Suc(Suc(My))$ を自身の Successor として ID と宛先を保持する．Suc(Suc(My)) も同時に離脱した場合，離脱を検知したノードは Suc(Suc(Suc(My))) を自身の Successor として ID と宛先を保持する．このように，各ノードは Successorlist において複数のノードの ID と宛先を保持することで，ネットワークの破綻を防ぐことができる．以上をもって，離脱処理は実現される．

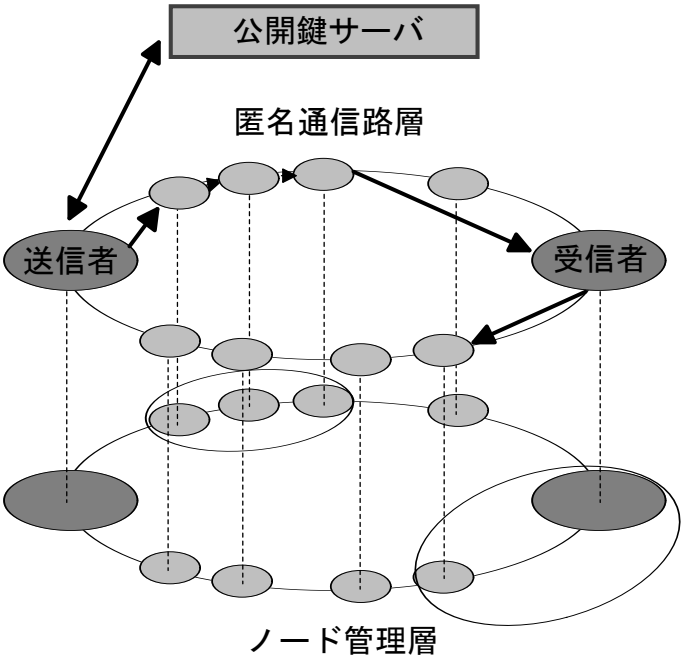

図 2.5: Bifrost の全体構成

2.3 Bifrost

本節では，本研究の基盤手法である Bifrost について述べる．Bifrost は Onion Routing の可用性の低さと拡張性の不定さを Chord を用いることで解決した手法である．

2.3.1 全体構成

図 2.5 に Bifrost の全体構成の図を示す．Bifrost は Chord によるノード管理，ノードの参加処理，ノードの離脱処理に加えて通信時の経路設定を行うノード管理層，多段中継と多重暗号化によって匿名通信を行う匿名通信路層，各ノードの公開鍵を管理する公開鍵サーバから構成されている．

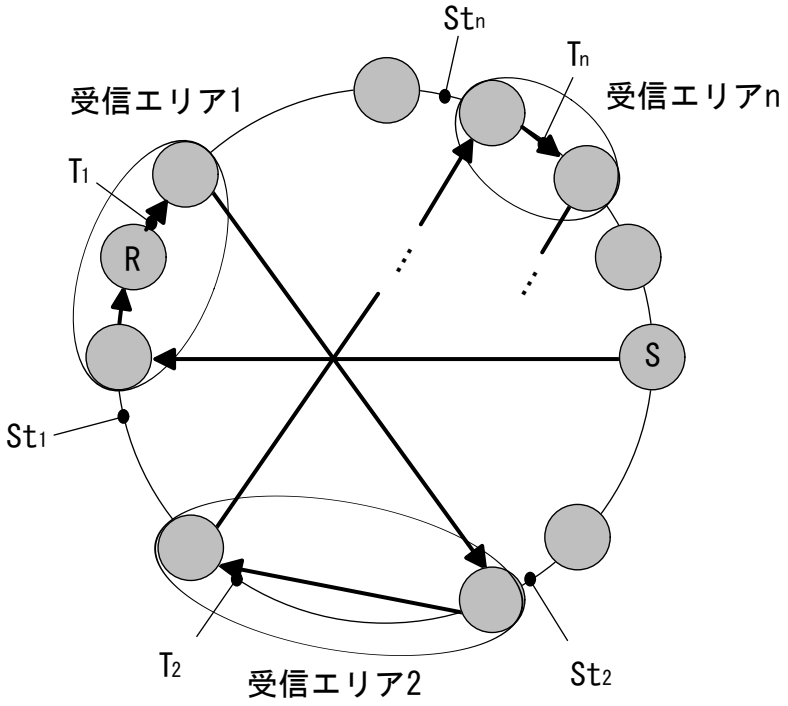

図 2.6: 経路設定

2.3.2 ノード管理層

ノード管理層は Chord によるノード管理，ノードの参加処理，ノードの離脱処理に加えて，匿名通信の経路設定を行う層である．図 2.6 に経路設定の図を示す．経路設定は，送信者が匿名通信を行う際に行われる．送信者は始点 ID と終点 ID を設定する．この時，始点 ID から終点 ID までの ID 空間とその ID 空間における ID を管理するノード群を受信エリアと呼ぶ．送信者は複数の始点 ID と終点 ID を設定する．その際，受信者はいずれかの受信エリアに属している必要がある．送信者が送信するメッセージはすべての受信エリアを送信者が設定した順番で経由し，各受信エリア内では始点 ID を管理するノードから終点 ID を管理するノードまでを順に経由する．

表 2.2: 2.3 節における記号表

記号	意味
$S, N(T_0)$	送信者 (ノード)
R	受信者 (ノード)
N_n	n 番目の受信エリアに属するノード
St_n	n 番目の受信エリアの始点 ID
T_n	n 番目の受信エリアの終点 ID
$PKey_X$	ノード X の公開鍵
$CKey_{X,Y}$	ノード X, Y の共通鍵
SID_X	ノード X が生成したセッション ID
$M1_X$	ノード X が受信するメッセージ (1 度目)
H_X	$M1_X$ に含まれるヘッダ部
$D1_X$	$M1_X$ に含まれるデータ部
$M2_X$	ノード X が受信するメッセージ (2 度目以降)
$D2_X$	$M2_X$ に含まれるデータ部
$Data$	S と R の通信内容
$RData$	任意のデータ (送信者)，返信用経路構築のためのデータ (受信者)
TH	経路の終点を表すデータ
$N(I)$	$ID : I$ を管理するノード
$RKey_X(A), CKey_{X,Y}(A)$	A を $RKey_X$，$CKey_{X,Y}$ にて暗号化
$A \parallel B$	A と B の結合

2.3.3 匿名通信路層

匿名通信路層は，送信者がノード管理層で設定した経路に従い，多重暗号化と多段中継を用いて匿名通信を実現する層である．Bifrost における通信は 1 度目と 2 度目以降でわけられる．1 度目の通信では，送信者は各ノードの公開鍵を用いて 2 度目以降の通信を高速に行うために共通鍵の配布を行う．これを、経路の構築を行うという．2 度目以降の通信では，送信者は 1 度目の通信において配布した共通鍵を用いて受信者と通信内容のやりとりを行う．

表 2.3: メッセージ受信時の処理 (1 度目)

1 $M1_{Nn}$ から $SID_{N(Tn-1)}$, H_{Nn}, $D1_{Nn}$ を取得
2 自身の持つ秘密鍵で H_{Nn} の復号を試行
2.1 H_{Nn} の復号に成功した場合
・$SID_{N(Tn-1)}$ に関連付けて，自身が $N(T_n)$ だと記憶
2.1.1 復号した H_{Nn} が TH でない場合
・復号した H_{Nn} から $CKey_{S,N(Tn)}$，St_{n+1}，H_{Nn+1} を取得
・新たなセッション ID $SID_{N(Tn)}$ を生成
・$D1_{Nn}$ を $CKey_{S,N(Tn)}$ で復号し，$D1_{Nn+1}$ を取得
・$SID_{N(Tn)}$, H_{Nn+1}, $D1_{Nn+1}$ を用いて $M1_{Nn+1}$ を生成
・St_{n+1} を管理するノードを検索し，$N(St_{n+1})$ へ $M1_{Nn+1}$ を送信
・$SID_{N(Tn-1)}$ に関連付けて，$CKey_{S,N(Tn)}$, St_{n+1}, $SID_{N(Tn)}$ を記憶
2.2 H_{Nn} の復号に失敗した場合
・$M1_{Nn}$ を Successor へ送信
3 自身の持つ秘密鍵で $D1_{Nn}$ の復号を試行
3.1 $D1_{Nn}$ の復号に成功した場合
・$SID_{N(Tn-1)}$ に関連付けて，自身が R だと記憶
・復号した $D1_{Nn}$ から $CKey_{S,R}$ と $RData$ を取得
・$SID_{N(Tn-1)}$ に関連付けて，$CKey_{S,R}$ を記憶

1 度目の通信

1 度目の通信において，送信者は設定した経路に従い，公開鍵サーバから各受信エリアの終点 ID を管理するノードの公開鍵を入手し，1 度目の通信に用いるメッセージの生成を行う．1 度目の通信に用いられるメッセージはセッション ID，ヘッダ部，データ部から生成される．セッション ID は経路ごとに一意なものであり，2 度目以降の通信の際の各ノードの挙動を保持するのに利用される．表 2.2 に記号を定義する．H_{Nn} と $D1_{Nn}$ は受信エリア数を m で受信者の存在する受信エリアを l 番目の受信エリアとすると，それぞれ式 2.2 と 2.3 で得られる．

$$H_{Nn} = PKey_{N(Tn)}(CKey_{S,N(Tn)} \parallel St_{n+1} \parallel PKey_{N(Tn+1)}(CKey_{S,N(Tn+1)} \parallel St_{n+2} \parallel \cdots PKey_{N(Tm-1)}(CKey_{S,N(Tm-1)} \parallel St_m \parallel PKey_{N(Tm)}(TH))\cdots)) \quad (2.2)$$

$$D1_{Nn} = CKey_{S,N(Tn)}(CKey_{S,N(Tn+1)}(\cdots CKey_{S,N(Tl-1)}(PKey_R(CKey_{S,R} \parallel RData))\cdots)) \quad (2.3)$$

表 2.4: メッセージ受信時の処理 (2 度目以降)

1 $M2_{Nn}$ から $SID_{N(Tn-1)}$ と $D2_{Nn}$ を取得
2 1 度目の通信で $SID_{N(Tn-1)}$ に関連付けられた情報を取得
3 自身が $N(T_n)$ かチェック
3.1 自身が $N(T_n)$ である場合
3.1.1 St_{n+1} が記憶されている場合
・$D2_{Nn}$ を $CKey_{S,N(Tn)}$ で復号し，$D2_{Nn+1}$ を取得
・$SID_{N(Tn)}$ と $D2_{Nn+1}$ から $M2_{Nn+1}$ を生成
・St_{Tn+1} を管理するノードを検索し，$N(St_{Tn+1})$ へ $M2_{Nn+1}$ を送信
3.2 自身が $N(T_n)$ でない場合
・$M2_{Nn}$ を Successor へ送信
4 自身が R かチェック
4.1 自身が R である場合
・$CKey_{S,R}$ で $D2_{Nn}$ を復号し，$Data$ を取得

$D1_{Nn}$ に含まれる $RData$ は受信者が送信者への返信用経路の構築に用いるためのデータ部，ヘッダ部とヘッダ部に含まれる共通鍵，最初の受信エリアの始点 ID から成る．これらは送信者によって生成される．返信用経路の構築に用いられるデータ部の送信者が復号する部分には，送信者が任意のデータを含めることができる．さらに，$M1_{Nn}$ は式 2.4 で得られる．

$$M1_{Nn} = SID_{N(Tn-1)} \parallel H_{Nn} \parallel D1_{Nn} \tag{2.4}$$

S は $M1_{N1}$ を生成すると，St_1 を管理するノードを検索し，$N(St_1)$ へ $M1_{N1}$ を送信する．その後，$M1_{Nn}$ を受信した Nn は表 2.3 に示す処理を行う．

2 度目以降の通信

2 度目以降の通信に用いられるデータ部 $D2_{Nn}$ は式 2.5 で得られる．

$$D2_{Nn} = CKey_{S,N(Tn)}(CKey_{S,N(Tn+1)}(\cdots CKey_{S,N(Tl-1)}(CKey_{S,R}(Data))\cdots)) \tag{2.5}$$

また，メッセージ $M2_{Nn}$ は式 2.6 で得られる．

$$M2_{Nn} = SID_{N(Tn-1)} \parallel D2_{Nn} \tag{2.6}$$

S は $M2_{N1}$ を生成すると，St_1 を管理するノードを検索し，$N(St_1)$ へ $M2_{Nn}$ を送信する．その後，$M2_{Nn}$ を受信した Nn は表 2.4 に示す処理を行う．

2.3.4 可用性と拡張性

Bifrost では，各受信エリアの終端ノード $N(T_n)$ は経路の維持に必要な $CKey_{S,N(Tn)}$ を分割して，自身の Successor と Predecessor に預ける．$N(T_n)$ の Predecessor が $N(T_n)$ の離脱を検知し，Chord に従いネットワークの修復を行う際に，$N(T_n)$ の Predecessor が預かっている分割した $CKey_{S,N(Tn)}$ を $N(T_n)$ の Successor に引き渡す．$N(T_n)$ の Successor は $CKey_{S,N(Tn)}$ を統合し，新たな $N(T_n)$ となる．これにより，送受信者は通信を継続することができる．また，$S, R, N(T_n)$ 以外のノードが離脱および参加しても，送受信者は通信を継続することができる．これは，Chord に従って修復されるネットワークと，$S, R, N(T_n)$ 以外のノードは受信したメッセージを自身の Successor に送信するのみという，2.3.3 項にて述べた Bifrost の各ノードの動作によるものである．このことから，Bifrost の可用性は Onion Routing に比べて高いと言える．また，Chord は拡張性の高い DHT を実現するアルゴリズムであり，これを用いてノードを管理する Bifrost も拡張性が高いと言える．

2.3.5 問題点

既存の Bifrost には 2 つの問題点が存在する．1 つ目の問題点は送信者が送受信者間の経由ノード数と経路全体のノード数を調整できないことである．送受信者間の経由ノード数は通信速度に，経路全体のノード数は匿名性に影響をおよぼす．そのため，ユーザは匿名性や通信速度を調整することができない．これらのノード数を調整するためには，受信エリアを詳細に設定する必要がある．しかし，既存の Bifrost には始点 ID と終点 ID の設定方法が示されていない．単純な手法としては，始点 ID と終点 ID を無作為または Chord の検索を複数回用いて設定する手法が考えられる．前者は，送信

者が送受信者間の経由ノード数と経路全体のノード数を調整することができない．後者は，送信者が $Suc(N(St_n))$, $Suc(Suc(N(St_n)))$，… とノードを順次検索することで，受信エリアごとのノード数，経路全体のノード数，さらには送受信者間の経由ノード数までも正確に調整できる．しかし，検索に伴う通信コストが受信エリアの数と受信エリア内のノード数に比例して大きくなる．また，連続したノードの検索を行う必要があるため，検索の際に問い合わせるノードに受信エリアの情報を与えることになる．これにより，通信の匿名性が低下する．2 つ目の問題点はアプリケーションにおける通信内容を Bifrost に受け渡す機構が存在しないことである．ユーザが通信を行うために用いるアプリケーションはさまざまであるため，あらゆるアプリケーションにおける通信内容を Bifrost へ受け渡す機構が必要不可欠である．

第3章

提案手法

本章では 2.3.5 項で述べた既存の Bifrost の問題点を解決する手法を提案する．1 つ目の問題点の解決のために受信エリアの設定方法を提案する．2 つ目の問題点の解決のために，Bifrost の SOCKS[6][7] プロキシ化を提案する．

3.1 受信エリアの設定方法

本節で提案する受信エリアの設定方法を用いると, 送信者は受信エリアを詳細に設定することができる．加えて従来の受信エリア数，受信者が属する受信エリアの順番の設定を用いることで，送信者は送受信者間の経由ノード数と経路全体のノード数を調整することができる．提案する受信エリアの設定方法はノードへの ID 割り当て方法，参加ノード数の推測，受信エリアの設定の 3 つの手順から成る．

3.1.1 ノードへの ID 割り当て方法

既存の Bifrost において，ノードへ割り当てる ID はランダムに決定される．提案手法においては受信エリアの設定のために，ノードへの新たな ID 割り当て方法を提案する．まず，ID 空間において同レベルの ID が等間隔に配置されるように ID にレベルを定義する．図 3.1 に ID へのレベル定義の具体例を示す．ID の最大値を MAX_ID，レベル 0 のノードの数を $N = 2^n (n = 1, 2, \cdots)$ とすると，レベル X の ID ID_X は式 3.1 で

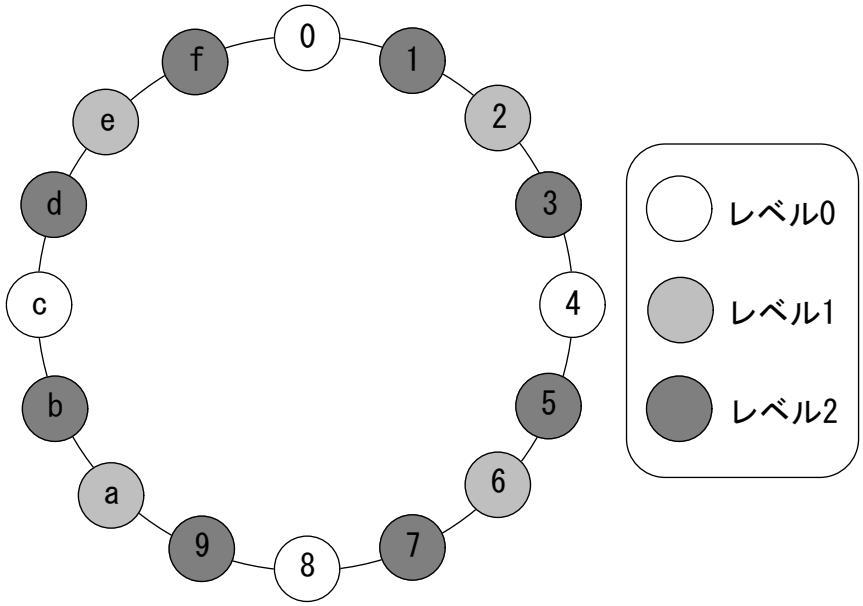

図 3.1: ID へのレベル定義の具体例 (N=4)

定義される．

$$\begin{cases} ID_X = \frac{MAX\_ID+1}{N} \times n \quad (n = 1, 2, \cdots, N-1)(X = 0) \\ ID_X = \frac{MAX\_ID+1}{N \times 2^X} \times n \quad (n = 1, 3, \cdots, N \times 2^X - 1)(X \geq 1) \end{cases} \tag{3.1}$$

ノードがネットワークに参加する際には，最もレベルの小さい未割り当ての ID の中から無作為に割り当てる．この割り当て方法に従い ID を割り当てることで，ID 空間におけるノードの配置をほぼ等間隔にすることができる．

3.1.2 参加ノード数の推測

参加ノード数の推測は，各ノードが自身の Successorlist と Fingertable および Predecessor を適宜参照することによって行う．自身が既知のノードの ID のレベルの最大値を MAX_LV とする．3.1.1 項で述べた ID の割り当て方法により，MAX_LV 未満のレベルの ID はすべて割り当て済みである．また，MAX_LV のレベルの ID はすべて割り当て済みか割り当て途中である．さらに，$MAX\_LV + 1$ のレベルの ID はすべて未割り当てか割り当て途中である．図 3.1 に示す例の場合，各ノードは自身の Successor

と Predecessor を参照するものとすると，参加ノードの推測数 p は式 3.2 で得られる．

$$\begin{cases} 1 \leq p < N & (MAX\_LV = 0) \\ N^{MAX\_LV-1} \leq p < N^{MAX\_LV+1} & (MAX\_LV \geq 1) \end{cases} \tag{3.2}$$

この例においては，実際の参加ノード数と推測数の差 (誤差) の最大値が最小となるように推測数に $p = N^{MAX\_LV}$ を用いたとしても，実際の参加ノード数は $\frac{p}{2}$〜$2p$ となり，受信エリアに含まれるノード数を調整する際に大きな影響を及ぼす．よって，この差を小さくするレベルの定義方法が必要である．また，可能な限り多くのノードが，現在割り当て途中の ID のレベルを知ることができる必要がある．

3.1.3 受信エリアの設定

受信エリアの設定は受信者の存在の有無でわけられる．

受信者が存在しない受信エリアの設定

図 3.2 に受信者が存在しない受信エリアの設定の図を示す．受信エリア内のノード数を d，始点 ID を St，3.1.2 項で述べた方法で求めた参加ノード数の推測値を p とする．受信者が存在しない受信エリアの設定の場合は，無作為に St を決定するものとし，終点 ID T は式 3.3 で算出される．

$$T = (St + \frac{MAX\_ID + 1}{p} \times (d - 1)) \bmod (MAX\_ID + 1) \tag{3.3}$$

受信者が存在する受信エリアの設定

図 3.3 に受信者が存在する受信エリアの設定の図を示す．3.1.3 項における記号に加えて，受信者の ID を r，受信者の受信エリア内における順番を n とすると，St と T はそれぞれ式 3.4 と 3.5 で算出される．

$$St = (r + MAX\_ID + 1 - \frac{MAX\_ID + 1}{p} \times (n - 1)) \bmod (MAX\_ID + 1) \tag{3.4}$$

$$T = (r + \frac{MAX\_ID + 1}{p} \times (d - n)) \bmod (MAX\_ID + 1) \tag{3.5}$$

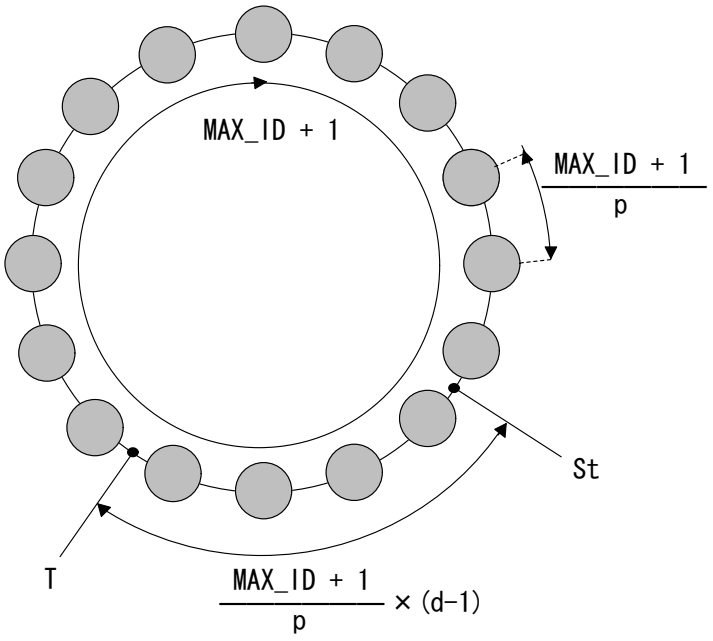

図 3.2: 受信者が存在しない受信エリアの設定

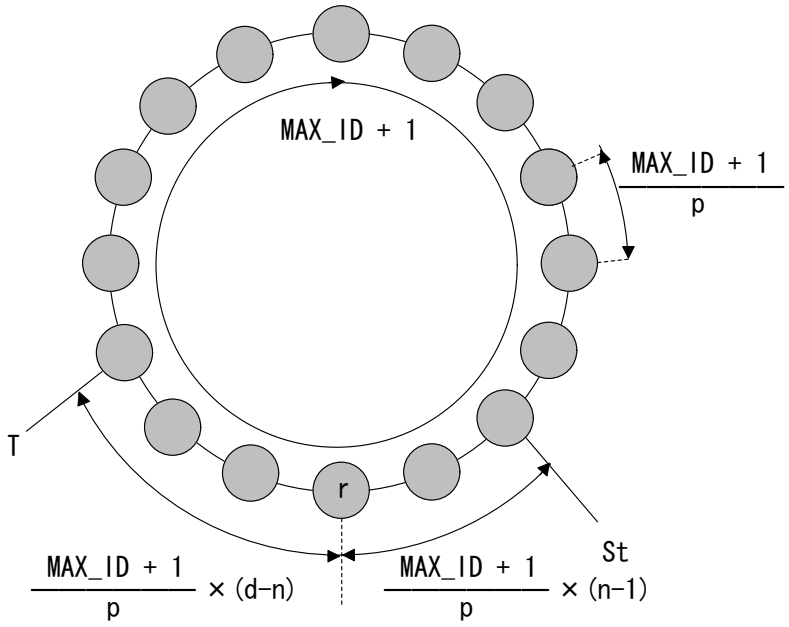

図 3.3: 受信者が存在する受信エリアの設定

3.2 SOCKSプロキシ化

本節では，任意のアプリケーションの通信内容をBifrostに受け渡す機構としてBifrostのSOCKSプロキシ化を提案する．3.2.1項ではSOCKSについて述べ，3.2.2項ではTUN/TAP[8]という他の機構について述べる．その後，3.2.3項にて両者の比較について述べる．

3.2.1 SOCKS

SOCKSはファイアウォールを安全に透過利用するためのプロトコルであり，SOCKSプロキシはアプリケーションの通信を中継する．BifrostをSOCKSプロキシ化することで，ユーザは中継するアプリケーションにおける通信を匿名化することができる．また，通信を行うホストはBifrostのネットワーク外に存在してもよい．これにより、汎用性の高い匿名通信をユーザに提供することができる．しかし，SOCKSを利用するにはアプリケーションごとにSOCKSに対応している必要がある．これに対しては，tsocksやfreecapといったSOCKSに対応していないアプリケーションをSOCKSに対応させるアプリケーションが存在する．これらとSOCKSプロキシ化したBifrostを併用することで，ユーザは任意のアプリケーションにおける通信を容易に匿名化することができる．

図3.4にSOCKSプロキシを介した通信の流れの図を示す．SOCKSプロキシはクライアントから認証要求を受信する．SOCKSはファイアウォールを透過利用するためのプロトコルなので，SOCKSプロキシは安全性の確保のためクライアントと認証交渉を行う．認証交渉については実装に依存するため，ここでは説明を割愛する．続いて，SOCKSプロキシはクライアントへ認証結果を送信する．クライアントの受信した認証結果が成功を示すと，クライアントはリクエストをSOCKSプロキシへ送信する．SOCKSプロキシは受信したリクエストに従って，ホストとの接続を行う，ソケットを確保してホストからの接続を待つ，などのリクエスト処理を行う．SOCKSプロキシはリクエストが完了すると，クライアントへリクエスト結果を送信する．クライアントが受信したリクエスト結果が成功を示すと，クライアントとホストはSOCKSプ

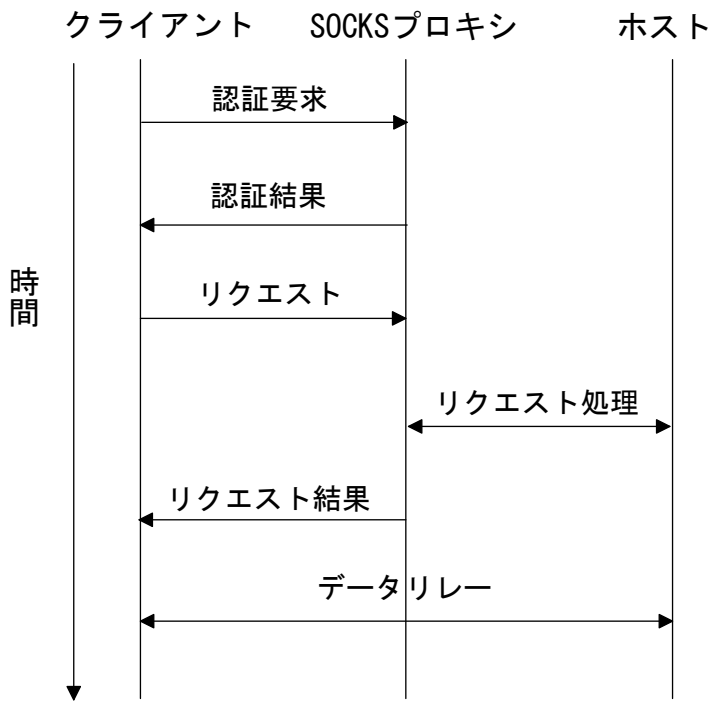

図 3.4: SOCKS プロキシを介した通信の流れ

ロキシを介して通信内容のやりとりを行う．

3.2.2 TUN/TAP

SOCKS を用いる以外に，TUN/TAP を用いた通信の匿名化機構についても考察する．TUN/TAP はプロトコルスタックにおけるネットワーク層またはトランスポート層をシミュレートし，それぞれにおいてヘッダ付きパケットをユーザアプリケーションへ出力することができるデバイスドライバである．逆に，TUN/TAP はユーザアプリケーションからプロトコルスタックへパケットを入力することもできる．TUN/TAP から出力されたパケットを Bifrost を介してやりとりすることで，ユーザはパケット単位でアプリケーションにおける通信を匿名化することができる．TUN/TAP を用いる場合，ホストは Bifrost のネットワーク内に存在している必要がある．

表 3.1: SOCKS と TUN/TAP を用いた匿名化機構の相異点

	SOCKS	TUN/TAP
対応 OS	すべての OS	UNIX 系のみ
ルート権限	不要	必要
リレー単位	通信内容	パケット
Bifrost の外部との通信	可	不可

3.2.3 SOCKS と TUN/TAP の比較

表 3.1 に SOCKS と TUN/TAP を用いた匿名化機構の相違点を示す．TUN/TAP は UNIX 系 OS 専用のデバイスドライバであり，その利用の際にはルート権限が必要となる．データの中継単位については，SOCKS が通信内容を直接中継するのに対し，TUN/TAP ではパケットを中継する．よって，TUN/TAP ではパケットのヘッダの分だけデータ量が増大する．また，通信においてクライアントやホストが Bifrost のネットワーク内に存在するとは限らない．これらの違いから，より一般的な匿名通信を実現するためには SOCKS を用いることが望ましい．

第4章

実装と動作

4.1 IDへのレベル定義

本実装でのIDへのレベル定義は，3.1.1項の方法に加えて各レベル内でさらにID群を分割する．各レベルのID数を少なくすることで，既参加ノード数に対する配置途中のレベルのノード数を少なくし，推測誤差の低減を図る．具体的にはS個のノードの情報を保持するSuccessorlistを利用し，3.1.1項におけるレベル0以外の各レベルのID数をそれぞれ$d = 2^n \leq \lceil \frac{S}{2} \rceil (n = 0, 1, \cdots)$分割する．この時，各レベルのIDが割り当て終わった際に全てのノードがそのレベルのIDを持つノードのいずれかを知ることができるようにする．式4.1は分割したレベル定義の式である．ノードの参加時には，$ID_{X,Y}$で表わされるX値の最も小さい未割り当てのIDの中で，最もY値の小さいIDから無作為に割り当てる．これにより，3.1.1項のレベル定義に比べて推測誤差の最大値を$\frac{1}{d}$程度にすることができる．

$$
\begin{cases}
ID_0 = \frac{MAX\_ID+1}{N} \times n & (n = 1, 2, \cdots, N-1) \\
ID_{X,1} = \frac{MAX\_ID+1}{N \times 2^X} \times n & (n = 1, 1+2d, \cdots, N \times 2^X - 2d + 1)(X \geq 1) \\
ID_{X,2} = \frac{MAX\_ID+1}{N \times 2^X} \times n & (n = 1+d, 1+3d, \cdots, N \times 2^X - d + 1)(X \geq 1) \\
ID_{X,3} = \frac{MAX\_ID+1}{N \times 2^X} \times n & (n = 3, 3+2d, \cdots, N \times 2^X - 2d + 3)(X \geq 1) \\
ID_{X,4} = \frac{MAX\_ID+1}{N \times 2^X} \times n & (n = 3+d, 3+3d, \cdots, N \times 2^X - d + 3)(X \geq 1) \\
\qquad\qquad \vdots & \\
ID_{X,d-1} = \frac{MAX\_ID+1}{N \times 2^X} \times n & (n = 1, 3, \cdots, N \times 2^X - d - 1)(X \geq 1) \\
ID_{X,d} = \frac{MAX\_ID+1}{N \times 2^X} \times n & (n = 1, 3, \cdots, N \times 2^X - 1)(X \geq 1)
\end{cases}
\tag{4.1}
$$

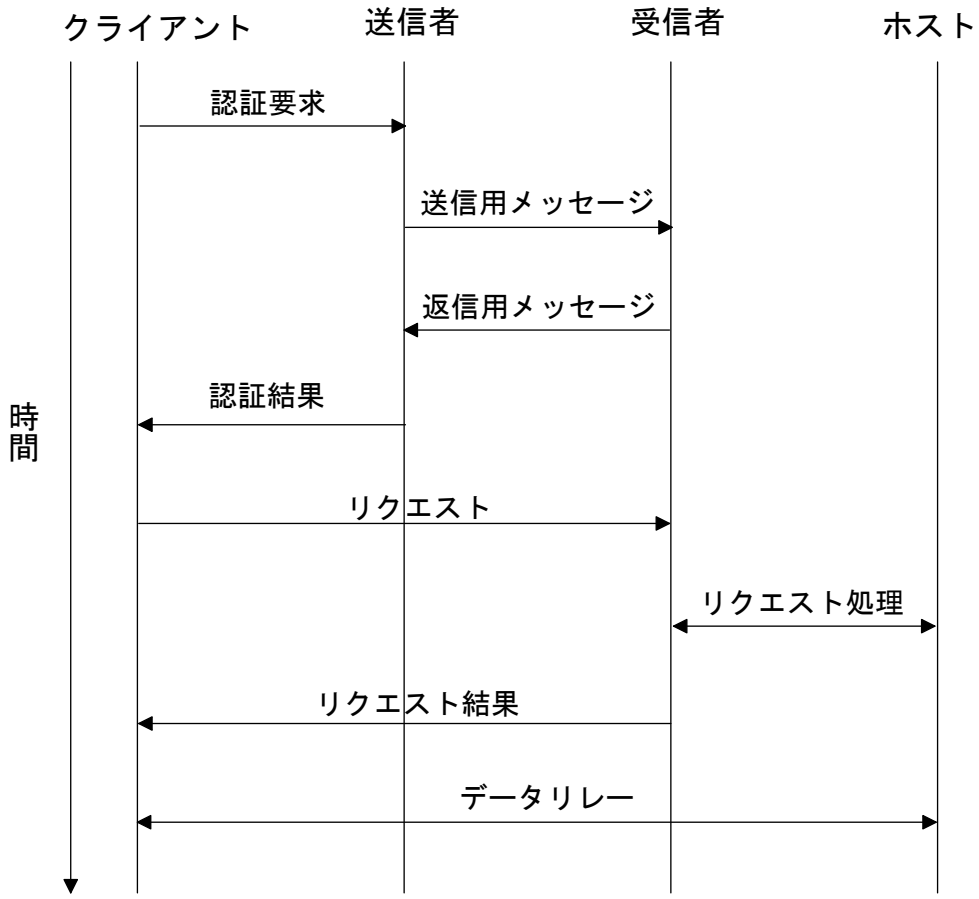

図 4.1: 匿名通信

4.2 SOCKSプロキシ化

本実装では既存の Bifrost の構成に ID 発行サーバを加え，4.1 節で述べた手法に従ってノードへ ID を割り当てる．ユーザは受信エリア数，受信者の存在する受信エリアの順番，受信エリアにおける受信者の順番，各受信エリアに含まれるノード数を設定したノードを送信者として用いることで，匿名性と通信速度を調整する．図 4.1 に匿名通信の一連の動作の図を示す．各ノードはクライアントからの認証要求の受信を待つ．本実装においての認証には経路の構築の成否を用いる．認証要求を受信したノードは Bifrost における送信者となり，3.1 節で述べた手法を用いて送信用と返信用の経路を設定する．次に，送信者となるノードは送信用と返信用の経路の設定を用いて，1 度目の

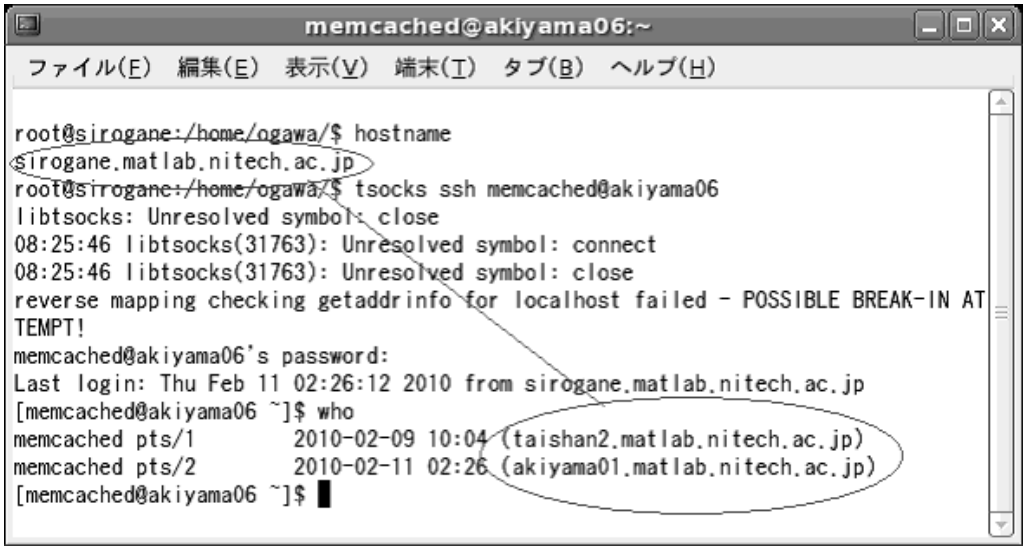

図 4.2: SOCKS プロキシ機構の動作検証

通信におけるメッセージを生成する．送信者が生成したメッセージを送信し，受信者は受信したメッセージから返信用のメッセージを生成する．受信者が返信用メッセージを送信し，送信者が返信用メッセージを受信したことをもって，送信用と返信用の経路の構築は完了する．この時，送信者はクライアントへ認証結果を送信する．これ以降の送信者と受信者の通信は 2 度目以降の通信となり，返信用の経路は受信者から送信者への送信用の経路として利用される．続いて，送信者はクライアントからリクエストを受信する．送信者は受信したリクエストを受信者へ送信する．リクエストを受信した受信者はリクエストに従ってリクエスト処理を行い，その成否に応じてリクエスト結果を生成し，送信者へ送信する．送信者は受信したリクエスト結果をクライアントへ送信する．クライアントが受信したリクエスト結果がリクエスト処理の成功を示すと，クライアントとホストは構築された経路を用いて通信内容のやりとりを開始する．

表 4.1: 検証用端末

ネットワークを構築する端末		クライアント端末	
OS	:CentOS 5.4	OS	:CentOS 5.2
CPU	:Core i5 2.67Ghz	CPU	:Core2Duo 2.13Ghz
Memory	:4.0GB	Memory	:2.0GB

4.3 動作検証

SOCKS プロキシ機構の動作検証

表 4.1 に示すネットワークを構築する端末 4 台に Bifrost のプロセスを各 4,4,4,3 つずつ起動し，合計 15 ノードにてネットワークを構築する．その後，クライアント端末にて Bifrost を起動し，構築したネットワークへ参加させる．クライアント端末において ssh に tsocks を用いて SOCKS に対応させ，SOCKS プロキシとしてクライアント端末内の Bifrost を指定した結果，Bifrost のネットワークを介してホストとの匿名通信を行うことができた．図 4.2 に動作検証時のスクリーンショットを示す．クライアント端末 sirogane は ssh を用いて接続を行っているが，Bifrost のネットワークを経由した結果，接続先の端末で who コマンドを用いても sirogane のホスト名は表示されない．

経由ノード数調整機能の動作検証

ネットワークを構築する端末 7 台とクライアント端末に 1 つずつプロセスを起動し，合計 8 ノードでネットワークを構築する．1024byte のデータをサーバプログラムと送受信し合うクライアントプログラムを tsocks を用いて SOCKS に対応させ，Bifrost を介してサーバプログラムを起動しているホストと接続し，ラウンドトリップタイム (RTT) を計測する．この時，受信エリア数は 3 で受信エリアに属するノード数は各 8 ノードとし，経路全体のノード数は 24 ノードとする．受信者ノードの順番 (送受信者間の経由ノード数=受信者の順番-1) の設定を 3,6,9,12,15,18,21,24 ノード目と変化させたところ，図 4.3 に示す結果が得られた．受信者の順番の設定に比例して RTT が増加していることから，設定した受信者の順番に従って実際の受信者の順番を調整できていることが確認できる．6-9 と 15-18 の区間において他の区間より RTT の増加が大きい理由

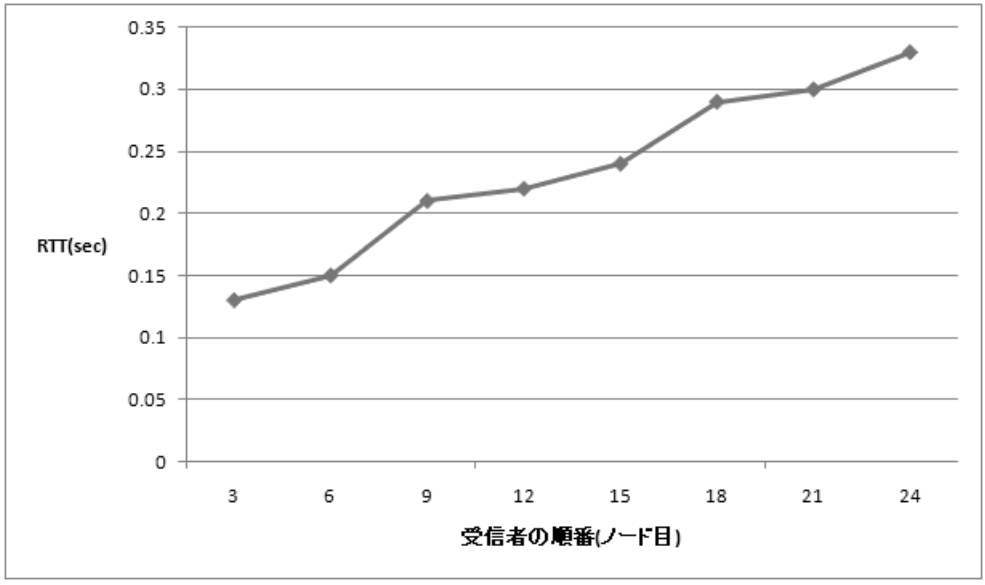

図 4.3: 経由ノード調整機能の動作検証

は，メッセージが受信エリアの終端に達し復号処理が行われるためである．

第5章

まとめ

本研究では，匿名通信方式の 1 つである Bifrost の拡張実装を提案した．Bifrost は匿名通信方式の 1 つである Onion Routing の可用性の低さと拡張性の不定さを分散ハッシュテーブルのアルゴリズムの 1 つである Chord を用いて解決した手法である．本論文で提案した ID の割り当て方法と受信エリアの設定方法により，ユーザは Bifrost を介して匿名通信を行う際に匿名性と通信速度を調整することができる．Bifrost の SOCKS プロキシ化により，ユーザは任意のアプリケーションを SOCKS に対応させるアプリケーションと併用することで，任意のアプリケーションにおける通信を容易に匿名化することができる．実装については，ID 発行サーバによるノードへの ID 割り当てを行うものとし，推測誤差の低減が可能な ID へのレベル定義方法を実装した．また，Bifrost を SOCKS プロキシ化した．動作検証は，複数のノードによるネットワークを構築し，任意のアプリケーションを SOCKS に対応させるアプリケーションと併用して，ssh などの広く利用されるアプリケーションの通信を匿名化できることを確認した．また，送受信者間の経由ノード数の設定を変更し RTT を計測することで，送受信者間の経由ノード数を調整する機能についても動作を確認した．

謝辞

本研究のために多大な御尽力を頂き，日頃から熱心な御指導を賜った名古屋工業大学の齋藤彰一准教授に深く感謝致します.

また, 本研究の際に多くの助言と協力をして頂いた齋藤研究室ならびに松尾・津邑研究室の皆様に深く感謝致します。

参考文献

[1] Pfitzmann, A. and Waidner, M. : Networks without user obserbility, Eurocrypt ’85, LNCS 219, pp. 245–253 (1986).

[2] Goldschang, D. , Reed, M. and Syverson, P. : Onion routing for anonymous and private internet connections, Comm. ACM, Vol. 42, No. 2, pp. 39–41 (1999).

[3] Reed, M. G., Syverson, P. F. and Goldschlag, D. M: Anonymous connections and Onion routing, IEEE Journal on Specific Areas in Communications, Vol. 16, No. 4, pp. 482–494 (1998).

[4] Masaki Kondo, Shoichi Saito, Kiyohisa Ishiguro, Hiroyuki Tanaka and Hiroshi Matsuo, ”Bifrost: A Novel Anonymous Communication System with DHT”, Second International Workshop on Reliability, Availability, and Security, pp. 324-329 (2009.12).

[5] Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Bal akrishnan, H. : Chord: A Scalable Peer–To–Peer Lookup Ser vice for Internet Applications, Proc. 2001 ACM SIGCOMM Conference, pp. 149–160 (2001).

[6] SOCKS: A protocol for TCP proxy across firewalls(NEC)
http://ftp.icm.edu.pl/packages/socks/socks4/SOCKS4.protocol

[7] RFC 1928 - SOCKS Protocol Version 5
http://tools.ietf.org/html/rfc1928

[8] Universal TUN/TAP device driver
http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt

[9] 首藤一幸, 田中良夫, 関口智嗣. オーバーレイ構築ツールキット Overlay Weaver. 情報処理学会論文誌:コンピューティングシステム, Vol. 47, No. ACS15 (2006).