社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－DPS－125 (5)
2005－EIP－ 29 (5)
2005／11／14

# 複数経路を用いたP2Pネットワーク上でのネゴシエーションと鍵共有法

高野 祐輝† 磯崎 直樹† 篠田 陽一§

北陸先端科学技術大学院大学 情報科学研究科† / 情報科学センター§

**概 要**

近年 Gnutella などの，P2P ネットワークと呼ばれる技術を用いたファイル共有ソフトが急速に広まってきており，そのユーザ数は劇的に増加してきている．また，規模追従性を考慮した P2P ネットワークの形態として，Distributed Hash Table(DHT) と呼ばれる方式が提案されてきている．我々は，P2P ネットワークでのセキュリティ向上を目指し，複数経路を用いたネゴシエーション手法を提案する．本提案方式は，従来，P2P ネットワークでは難しいとされてきた，なりすましや Man-in-the-Middle 攻撃などに強い end-to-end での鍵共有を実現する．

# Multipath Negotiation and Key Exchange Method on P2P Networks

*Yuuki Takano*† *Naoki Isozaki*† *Yoichi Shinoda*§

SCHOOL OF INFORMATION SCIENCE† / CENTER FOR INFORMATION SCIENCE§,
JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY

**Abstract**

File sharing software by using P2P network are widely used and the number of the users has been increasing. In recent years, the several algorithms of distributed hash table so called DHT were proposed. These are one of the P2P networks and address a problem of scale. In this paper, we propose multipath negotiation to improve security. Our improvement supports key exchange and protects from spoofing and man-in-the-middle attack.

## 1 Introduction

近年 Gnutella [1] などの，P2P ネットワークと呼ばれる技術を用いたファイル共有ソフトが急速に広まってきた．また，そのユーザ数は劇的に増加してきている．しかしながら，これらのソフトウェアは基本的にフラッドルーティングを用いた検索を行っているため，規模追従性において問題がある．そこでここ数年，規模追従性を考慮した P2P ネットワークの形態として，Symphony [5]，Pastry [4]，Chord [8]，Kademlia [3] などの，Distributed Hash Table(DHT) と呼ばれる方式が提案されてきている．特に，Kademlia [3] は，ファイル配布ソフトの BitTorrent [2] でも用いられており，DHT の必要性はますます高まってきていると言える．

一方，P2P ネットワークではその特性上，悪意のあるノードや不正なノードを簡単にネットワーク上に配置することが可能である．従って，セキュリティの面に関しては，従来のネットワークアプリケーション以上に注意を払う必要がある．

そこで我々は，P2P ネットワークでのセキュリティの向上を目指し，複数経路を用いたネゴシエーション手法を提案する．本提案方式は，従来，完全に分散化された P2P ネットワークでは難しいとされてきた，なりすましや Man-in-the-Middle 攻撃などに強い end-to-end での鍵共有を実現する．

本論文の構成は次のようになる，2 節では，DHT のアルゴリズムの一つである，Symphony と Chord について解説し，Portman [7] らの提案した，データ分割による安全なデータ送信法について説明する．3 節では，リンク汚染攻撃を提案し，Portman らの手法を用いても安全にデータ転送を行えない可能性があることを示す．4 節では新たに，時計回りと反時計回りルーティング，確率的ルーティングを提案する．また，提案したルーティング手法を用いることで，単純な攻撃を防ぐことが出来る事を説明する．5 節では，これらルーティングを用いた複数経路での鍵共有手法を提案し，リンク汚染攻撃に対する対抗

案も提案する．最後に 6 節で結論を述べる．

## 2 Related Works

本節では，Distributed Hash Table(DHT) のアルゴリズムの一つである Symphony [5] と Chord [8] について説明し，Portman[7] らの提案した，安全なデータ送信方法について述べる．

### 2.1 Symphony [5]

本節では，DHT におけるアルゴリズムの一つである Symphony [5] について説明する．

Symphony とは，2003 年に Manku，Bawa，Raghavan らによって提案された手法であり，その構成法には Small World [6] が適用されている．そのため，Symphony の持つリンクはランダムに張られることになる．以下，その構成法について説明する．

Symphony では各ノードに $[0,1)$ のノード ID が割り当てられ，リング状のトポロジを形成する．また，各ノードはそれぞれ Neighbor Link と Long Distance Link と呼ばれるリンクを持つ．Neighbor Link とは，隣り合うノード同士を繋げるリンクである．Long Distance Lonk とは自ノードからプラス方向へ向けて，ある距離 $\{x|0 \leq x < 1\}$ にあるノードへ張られたリンクであり，この $x$ はランダムに選ばれる．今，Symphony に参加しているノード数を $N$ とすると，$x$ の従う確率関数 $f_{sym}(x)$ は次のようになる．

$$f_{sym}(x) = \frac{1}{x \log N} \tag{1}$$

Symohony ではこの式 (1) に従う乱数を生成し，リンク先ノードを決定する．

### 2.2 Chord [8]

次に本節では，Chord [8] について説明する．Chord も Symphony と同様，DHT を実現するアルゴリズムである．

Chord は Stoica らによって提案された手法であり，Symphony と違い各ノードは決定的なリンクを持つ．以下，そのリンクの持ち方について説明する．

Chord は $m$ ビットの整数値の ID を持ち，この ID を用いてリンク先を決定する．その決定方法は，以下の式に従う．

$$ID_i = ID_{me} + 2^i \tag{2}$$

ただしここで，$ID_i$ はリンク先ノードの ID，$ID_{me}$ は自ノードの ID であり，$i$ は $0 \leq i \leq m-1$ となる．

$ID_i$ が決定すると，$ID_i$ を ID にもつノードへとリンクを張りにいく．しかし，$ID_i$ を ID に持つノードが存在しない場合，$ID_i$ と時計回りに最も近い ID を持つノードがリンク先に選ばれる．すなわち，参加している全てのノードうち $ID_{node} - ID_i \pmod{2^m}$ が最も小さくなる ID をもつノードが，リンク先として選ばれる．ただし，$ID_{node}$ は，各ノードの ID を意味する．

Chord では，各ノードがネットワークに参加，離脱すると，それぞのノードが持つリンクは式 (2) に基づいた更新が行われる．

### 2.3 Disjoint Routing Path in Chord [7]

P2P ネットワークでは，参加ノードが中継者の役割も果たしている．従って，中継者としてデータを転送する際，そのデータを簡単に盗聴，改竄することが出来る．

Portman[7] らは，Chord 上で複数経路を用いた安全なデータ送信方法を提案している．彼らは，送信時にデータを 2 つに分割し，それぞれ別のリンクを用いてデータを転送することで安全性を高めている．このとき，送信時に違うリンクを用いて送信すると，同じ経路を通って送信される確率が低くなることを発見している．この確率は，低い時で 20% 前後となっている．

しかしながら，彼らの手法では，攻撃者は一ヶ所にノードを配置しておくだけで，20% もの確率で攻撃を成功させる事が出来る．

## 3 The Link Pollution Attack

次に本節では，攻撃対象ノードを選び，そのノードの持っているリンクを汚染する，リンク汚染攻撃を提案する．本論文ではリンク汚染攻撃が可能な例として，Chord を使った例をあげる．ここで，各ノードはそれぞれ $n$ ビットの ID を持つとする．

まず始めに，攻撃者は以下の能力を有していると仮定する．

- 攻撃対象ノードの ID を知る事が出来る．
- 自由にノード ID を設定して，複数のノードを Chord のネットワークへ参加させる事が出来る．
- あるノードの IP アドレスを，ID から知る事が出来る．
- ある IP アドレスへ，DOS 攻撃を行う事が出来る．

次に，その攻撃手法について説明する．攻撃の手順は次の通りとなる．

1. 攻撃者は攻撃対象ノードの ID を調べる．この ID を $ID_{victim}$ とする．
2. $ID_{attacker_i} = 2^i + ID_{victim} (0 \leq i \leq n-1)$ となる ID を持つノードをそれぞれ Chord のネットワークへ参加させる．
3. 2 で参加させようとした ID が，既に Chord のネットワーク上に存在した場合，その ID を持つノードの IP アドレスを調べる．
4. 3 で調べた IP アドレスへ DOS 攻撃を行い，Chord のネットワーク上から排除する．
5. 2 で参加できなかった ID を用いて，再び Chord のネットワークへ参加させる．

このように行うことで，ある攻撃対象ノードの持っているリンク先は全て，攻撃者の用意したノードとなる．

図 1 は $n = 3$ のときに，$ID_{victim} = 1$ の攻撃対象ノードを攻撃した場合である．まず始めに，攻撃者は $ID_{attacker_i} = 2^i + ID_{victim} (0 \leq i \leq n-1)$ を計算し，その ID を持つノードを参加させる．この場合，攻撃者は $ID_{attaker_0} = 2, ID_{attacker_1} = 3, ID_{attacker_2} = 5$ の ID を持つノードを Chord のネットワーク上へ参加させる．

このとき，$ID_{attacker_1} = 3$ の ID を持つノードは，既にネットワーク上に存在しているため，攻撃者はこの ID を持つノードを参加させることが出来ない．従って，攻撃者はノードを参加させる前に，既に存在しているノードの IP アドレスを調べ，その IP アドレスに対して DOS 攻撃を行いネットワーク上から排除し，再び攻撃ノードの設置を試みる．このようにして，結果的に攻撃者は攻撃対象ノードの持つリンク全てを汚染させる事が出来る．

リンクが全て汚染されると，Portman らのようにデータを分割して送信しても，結果的に分割したデータが全て攻撃者へと転送されるため，安全にデータを送信することは不可能になる．

この攻撃が成立する理由は，次の 2 つであると考えられる．

- リンク先が ID により決定的に決まる．
- 新規ノードは，既存のノードの持っているリンクを更新する．

Chord ではこの 2 つのどちらの条件も満たしているため，容易にリンクを汚染することが可能だと考えられる．一方，Symphony の場合，Neighbor Link は決定的に決まり，新規ノードによりアップデートされるが，Long Distance Link はランダムに決定され，新規ノードによりアップデートされない．従って，Symphony は Chord よりもリンク汚染攻撃に対して耐性があると考えられる．

## 4 Enhanced Routing

本節では，時計回り，反時計回り方向のルーティングと，確率的ルーティングを提案し，その詳細を説明する．

### 4.1 Clockwise and Anticlockwise Routing

本節では，時計回りと反時計回りの 2 種類のルーティングを提案する．

各ノードは Chord と同じように，$n$ ビットの ID を持ち，リング状のトポロジを形成する．メッセージの最終到達ノードは，Symphony と同じように，宛先 ID とノード ID の最短距離が最も短いノードとなる．つまり，宛先 ID とノード ID の距離は次のアルゴリズム 1 で表せ，全てのノード中この値が最も小さくなるノードが最終到達ノードとなる．

**アルゴリズム 1.** 宛先 *ID* とノード *ID* の距離

*DstID* : 宛先 *ID*
*NodeID* : ノード *ID*

```
abs_bidirect(DstID, NodeID)
{
    abs1 = NodeID - DstID;
    abs2 = DstID - NodeID;

    return min(abs1, abs2);
}
```

ここで，時計回り方向，反時計回り方向にルーティングする際に用いる距離をそれぞれアルゴリズム 2, 3 に示す．

**アルゴリズム 2.** 時計回り方向の距離

*DstID* : 宛先 *ID*
*NodeID* : ノード *ID*

```
abs_clock(DstID, NodeID)
{
    return DstID - NodeID;
}
```

**アルゴリズム 3.** 反時計回り方向の距離

*DstID* : 宛先 *ID*
*NodeID* : ノード *ID*

```
abs_anticlock(DstID, NodeID)
{
    return NodeID - DstID;
}
```

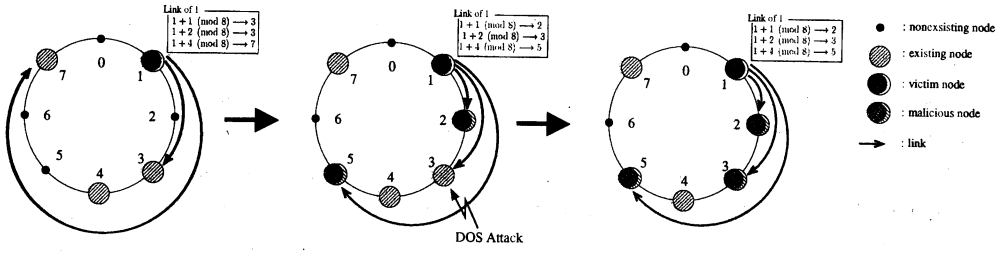


図 1: Links Polluted

次に，指向性のあるルーティングについて説明する．この指向性のあるルーティングとは，時計回り方向，反時計回り方向と，つねに一方向に無かってしかデータが転送されないルーティング方法である．その概念的なアルゴリズムを次のアルゴリズム 4 に示す．

**アルゴリズム 4.** 指向性ルーティング

*link* : リンクの配列
*link[i].id* : リンク $i$ の指すノードの *ID*
*DstID* : 宛先 *ID*
*MyID* : 自ノード *ID*
*Direc* : ルーティング方向

```
forward(DstID, Direc)
{
again:
    if (Direc = CLOCKWISE)
        abs = abs_clock;
    else if (Direc = ANTICLOCK)
        abs = abs_anticlock;
    else if (Direc = BIDIRECTION)
        abs = abs_bidirect;

    for (i = 0; i < #Link; i++) {
        dist = abs(DstID, link[i].id);
        if (dist < min_dist) {
            min_dist = dist;
            next_hop = link[i];
        }
    }

    dist = abs(DstID, MyID);
    if (dist < min_dist) {
        if (Direc != BIDIRECTION)
            Direc = BIDIRECTION;
            goto again;
        }
        message is mine;
    } else {
        next_hop.forward(DstID, Direc);
    }
}
```

このアルゴリズム 4 では，時計回り，反時計回り方向にデータを転送していく．もし，宛先 ID と自分の ID 間の距離が時計回り，または反時計回り方向で最短であった場合，距離の求め方をアルゴリズム 1 で表したものとし，もう一度検索を行う．

アルゴリズム 4 のように転送すると，時計回り，

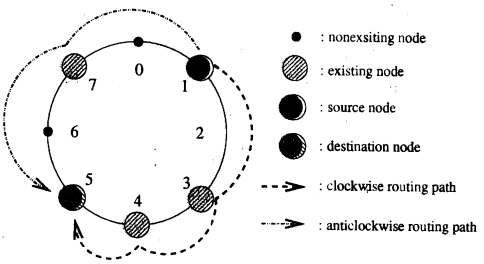


図 2: Clockwise and Anticlockwise Routing

反時計回りの経路は，送信ノードと受信ノードを除くとまったく違うものとなる．つまり，$Path_{clock}$ を時計回り方向へ転送した時の経路，$Path_{anti}$ を反時計回り方向へ転送した時の経路とすると，

$$Path_{clock} \cap Path_{anti} = \{Src, Dst\} \quad (3)$$

となる．ここで，$Src$ は送信ノード，$Dst$ は受信ノードとなる．

図 2 は，時計回り方向と反時計回り方向へルーティングを行った際に辿る経路の概念図を表している．この図の通り，時計回りと反時計回りでは，まったく違う経路を辿る事が分かる．Portman らの手法では，一ヶ所にノードを配置するだけで，20% 程度もの確率で攻撃を行うことが出来たが，本手法を用いると，一ヶ所にノードを配置するだけでは攻撃を行うことが不可能になる．

### 4.2 Probabilistic Routing

次に，本節では確率的ルーティングを用いたルーティング手法を提案する．DHT を用いると経路が決定的に決まるため，攻撃ノードを設置するポイントが知られてしまう．そのため，なるべく経路を特定されないよう，確率的に経路を決定する．

確率的ルーティングでは，次のアルゴリズム 5 を用いて次ホップを決定する．

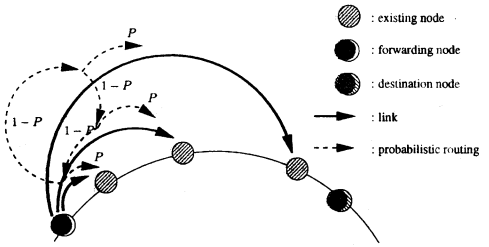


図 3: Probabilistic Routing

**アルゴリズム 5.** 確率的ルーティング

*link* : リンクの配列
*link[i].id* : リンク $i$ の指すノードの *ID*
*drand48()* : $[0, 1)$ の乱数生成関数

```
next_hop(link)
{
    i = 0;
    while (drand48() < 1 - P) {
        i = (i + 1) % #Link;
    }
    return link[i];
}
```

ただし，*link* は以下のように整列されているとする．

abs(link[0].id, DstID) < abs(link[1].id, DstID) < $\cdots$
< abs(link[n − 1].id, DstID) < abs(MyID, DstID)

図 3 はアルゴリズム 5 を用いて，経路が決定される様子を示している．転送するノードは乱数を生成し，確率 $P$ で宛先ノードに最も近いリンクを選択する．最も近いリンクが選ばれ無かった場合，さらにもう一度乱数を生成し，確率 $P$ で次に近いリンクを選択する．このように確率的ルーティングでは，再帰的にリンクを選択していく．

## 5 Multipath Negotiation

本節では，複数経路を用いたネゴシエーション手法と，送信リンクの決定方法を提案する．また，リンク汚染攻撃に対する対策案についても提案する．

### 5.1 Negotiation Algorithm

本節では複数経路を用いたネゴシエーション手法を提案する．基本的な概念は図 4 で示す通りである．まず，鍵を秘密分散法を用いて複数個に分割し，複数の経路を通るように送信する．具体的なアルゴリズムは，次の 6 に示す通りである．

**アルゴリズム 6.** 複数経路ネゴシエーション

*Src* : 送信ノード
*Dst* : 受信ノード
$p$ : 素数
$g$ : 生成元

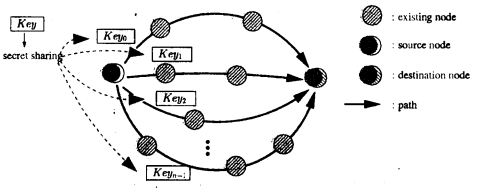


図 4: Multipath Negotiation

1. *Src* は乱数 $r_0$ を生成し，$Key_0 = g^{r_0} \pmod p$ を計算．
2. *Src* は $Key_0 = \sum_{i=0}^{n-1} ks_i \pmod p$ となる乱数 $ks_0, \cdots, ks_{n-1}$ を生成．
3. *Src* は $n$ 本の経路を用いて $ks_i (0 \le i < n)$ を別々に *Dst* へと送信．
4. *Dst* は $ks_0, \cdots, ks_{n-1}$ を受信．
5. *Dst* は乱数 $r_1$ を生成し，$Key_1 = g^{r_1} \pmod p$ を計算．
6. *Dst* は同様に $Key_1 = \sum_{i=0}^{n-1} kd_i \pmod p$ となる乱数 $kd_0, \cdots, kd_{n-1}$ を生成．
7. 同様に *Dst* は $kd_i (0 \le i < n)$ を $n$ 本の経路を用いて *Src* へと返信．
8. *Dst* は $Key_0 = \sum_{i=0}^{n-1} ks_i \pmod p$ を計算．
9. *Dst* は $Key = Key_0^{r_1} = g^{r_0 \cdot r_1} \pmod p$ より，共有鍵を得る．
10. 同様に *Src* は *Dst* より $kd_0, \cdots, kd_{n-1}$ を受信し，$Key_1 = \sum_{i=0}^{n-1} kd_i \pmod p$ を計算．
11. *Src* も $Key = Key_1^{r_0} = g^{r_0 \cdot r_1} \pmod p$ を計算し共有鍵を得る．

アルゴリズム 6 では，Diffie-Helman 鍵共有に用いる初期鍵を送信者，受信者側で秘密分散法を用いて分割し，その分割した鍵を複数の経路を用いて転送している．秘密分散法に満場一致法を用いているため，攻撃者は複数に分割した鍵を全て集めなければ攻撃することが出来ない．

つまり，この複数経路を用いたネゴシエーションが安全に行われるために重要な事は，次の 2 点であると言える．

- 経路の特定が難しい．
- リンクが汚染され難い．

### 5.2 Protection for Link Pollution Attack

次に本説では，3 節で述べたリンク汚染攻撃に対する対策案を提案する．本提案は，主に Symphony を用いた対策案である．何故ならば，Chord のリンクは全て決定的に決まるため，リンク汚染攻撃を防ぐことができないためである．

表 1: Neighbor Link の出現する確率 (単位 : %)

| #Node | #Long Distance Link | | | |
|---|---|---|---|---|
| | 5 | 10 | 20 | 40 |
| 1000 | 80.31 | 76.14 | 73.27 | 70.09 |
| 10000 | 84.39 | 80.05 | 75.09 | 73.06 |
| 100000 | 86.52 | 82.04 | 77.80 | 74.75 |

2.1 節で述べたように，Symphony は決定的にリンク先が決まる Neighbor Link と，ランダムにリンク先が決定される Long Distance Link を持つ．従って，Neighbor Link はリンク汚染攻撃に弱く，Long Distance Link はリンク汚染攻撃に強いといえる．

4.2 節の確率的ルーティングを用いた Symphony において，送信ノードと宛先をランダムに選んだ時，経路上に最終到達ノードの Neighbor Link が出現する確率をシミュレーションにより算出した．表 1 はその結果を表している．なお，この経路は 4.1 節で説明した，時計回り方向のルーティングを用いて決定されたものである．

ここで，5.1 節で述べた方式を用いてネゴシエーションを行う方法について述べる．5.1 節では，$n$ 個に分割した鍵を送信するため，送信ノードは始めに $n$ 個のリンクを選択する必要がある．そこで我々は，分割した鍵のうち $n/2$ 個を時計回り方向のルーティングを用いて送信し，残りの $n/2$ 個を反時計回り方向のルーティングを用いて送信する方式を用いる．このとき，送信ノードはそれぞれの分割した鍵を，違うリンクを用いて送信する．すると，$n$ 個に分割したデータが全て，最終到達ノードの Neighbor Link を通らない確率は次のように表される．

$$p_{suc} = 1 - pr^n \tag{4}$$

ただしここで，$pr$ を表 1 で示した，Neighbor の出現する確率とする．また，時計回りと反時計回り方向で，経路上に Neighbor Link が出現する確率は同じであると仮定する．

表 2 は，$n$ 個に分割した鍵が全て，最終到達ノードの Neighbor Link を通る確率を示している．Neighbor Link が汚染された場合，鍵の分割数を多くすると，より安全にネゴシエーションを行えることが分かる．

表 2: 全ての経路で Neighbor Link の出現する確率 (単位 : %)

| #Node | #Key | #Long Distance Link | | | |
|---|---|---|---|---|---|
| | | 5 | 10 | 20 | 40 |
| 1000 | 10 | 11.15 | 6.55 | 4.46 | 2.86 |
| | 12 | 7.19 | 3.80 | 2.39 | 1.41 |
| 10000 | 10 | 18.32 | 10.80 | 5.70 | 4.33 |
| | 12 | 13.05 | 6.92 | 3.21 | 2.31 |
| 100000 | 10 | 23.51 | 13.81 | 8.12 | 5.45 |
| | 12 | 17.60 | 9.30 | 4.91 | 3.04 |

## 6 Conclusion

P2P ネットワークの安全性向上を目指し，時計回りと反時計回りルーティング，確率的ルーティングを提案した．さらに，提案したルーティング手法を用いて，複数経路を用いたネゴシエーション手法を提案し，リンク汚染攻撃に対する対策法も提案した．また，提案手法を用いると，安全にネゴシエーションを行えることをシミュレーションによって確認した．

## 参考文献


[1] The gnutella protocol specification v0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[2] The official bittorrent home page. http://www.bittorrent.com/.

[3] Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors. *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*. Springer, 2002.

[4] Rachid Guerraoui, editor. *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, volume 2218 of *Lecture Notes in Computer Science*. Springer, 2001.

[5] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed hashing in a small world. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[6] S. Milgram. The small world problem. In *Psychology Today*, page 67(1), 1967.

[7] Marius Portmann, Sebastien Ardon, and Aruna Seneviratne. Mitigating routing misbehav. In *SAINT Workshops*, pages 541–545, 2004.

[8] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.