WebRTCを用いたグループ利用容易なP2Pネットワーク構築と効率化の検討

成田 裕司[1]　坪川 宏[1]

概要：近年，JavaScript の仕様改定や HTML5 API，それに伴うフレームワークの増加により，Web アプリケーションを選択する機会が増えている．その HTML5 API の中に WebRTC というものが存在する．WebRTC はブラウザ間 P2P 通信を可能とするコミュニケーション API の一つであり，セキュアなビデオチャット，データ通信がブラウザ間で可能となる．しかしながら，WebRTC は 1 : 1 を基本とした接続モデルであるため，会議ビデオチャットや CDN など複数ユーザ間のデータ通信においては，MCU のような制御サーバが必要となってしまう．しかし，そのようなサーバを使用した場合，セキュアな点において利点を失ってしまう．そこで，本研究では，クライアントサイドを主体に考え，シグナリングサーバとブラウザおよび JavaScript により，データチャネル上に DHT を用いたオーバーレイネットワークを構築する，これにより，セキュアな状態を維持し，ノード ID・情報の管理および大規模なノード間でのグループ内情報共有の仕組みについて報告する．

Examination and Evaluation about easy group P2P network construction using the WebRTC

YUJI NARITA[1]　HIROSHI TUBOKAWA[1]

1. はじめに

1.1 背景

近年，アプリケーション開発の選択として，Web アプリケーションを選ぶ企業および，開発者が増加している．これにより様々なサービスが Web 上で提供され，ユーザはブラウザ上で Web サービスが扱える．この流れは業務アプリケーションでも増加しており，従来の業務システムからの移行も多く見られる．この背景として，ブラウザ上で動作する JavaScript や HTML5 API およびそれに伴うライブラリやフレームワークの充実が挙げられる．JavaScript としては，ES2015 などの言語仕様の改定により，現在のアプリケーション開発のニーズにあったプログラミング言語となってきている．また，HTML5 API を用いることでブラウザ上でさまざまなコンテンツやデータを扱うことでより柔軟なアプリケーション開発が行えることも一因となっている．

[1] 東京工科大学 大学院 バイオ・情報メディア研究科
東京都八王子市片倉町 1404-1

この HTML5 API の中に WebRTC というコミュニケーション API が存在する．WebRTC はリアルタイムコミュニケーションを可能とする API であり，ブラウザ間で P2P 接続を行い，セキュアなビデオチャットや音声チャットを可能とする，主に Web 会議などに応用されており，セキュアな点などが企業などの会議システムとして注目されている．また，近年では，ビデオチャット以外にも WebRTC での通信を用いたストリーミング動画の配信（CDN）としても使用されている．

しかしながら，WebRTC は 1 : 1 を基本とした接続モデルであるため，会議ビデオチャットや CDN など複数ユーザ間のデータ通信においては，MCU のようなメディア制御サーバやユーザ間の接続を提供するためのユーザ管理サーバが必要となってしまう．また，そのようなサーバを使用した場合，セキュアな点において利点を失ってしまう問題が挙げられる．

そこで，本研究では，クライアントサイドを主体に考え，シグナリングサーバとブラウザおよび JavaScript により，データチャネル上に DHT アルゴリズムである Kademlia

を用いたオーバーレイネットワークを構築する．これにより，セキュアな状態を維持し，ユーザ間でのノード ID・ユーザ情報の管理および大規模なノード間でのグループ内情報共有のための仕組み実現する．図 1 に，本システムのネットワーク概念図を示す．図 1 に示すように，本システムでは，シグナリングサーバ以外の管理サーバを有しないことで外部でのユーザ情報の共有は行われない．

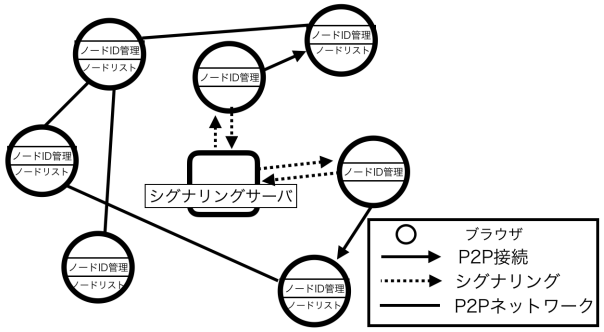

図 1 ネットワーク概念図

1.2 本論文の構成

2 節では関連技術について述べ，本論文で使用する技術である WebRTC，DHT について述べる．3，4 節で手法・実装について説明する．5 節では，評価前の事前調査を述べた後，本システムの大規模ノードでの評価・結果について示す．6 節では，まとめと今後の課題・展望について述べる．

2. 関連技術

2.1 DHT アルゴリズム

DHT（Distributed Hash Table）アルゴリズム [1] とは，P2P 内のノードやコンテンツにハッシュ値を与えることで，P2P ネットワーク上をハッシュテーブル空間としてネットワーク全体を管理する技術である．Pure-P2P には，サーバが存在しないため，効率の良いノード検索手段が重要となる．そこで DHT（Distributed Hash Table）を用いることで，ネットワーク負荷を抑え，ネットワーク間のノード検索を高速化を実現する，代表的なものに Chord，Pastry，Kademlia などが存在する．

2.2 Kademlia

本研究で使用する DHT アルゴリズムである Kademlia[2] の概要について簡単に述べる．Kadmelia の特徴として，ノードが頻繁に出入りする状況を想定して，設計されている．Kademlia は各ノードに ID として 160 ビットのハッシュ値を付与し，このハッシュ値を用いて，2 分木上のハッシュ空間を形成する．各ノードは自身 ID を元に各ハッシュ ID を排他的論理和（XOR）の距離を元に 2 分木上のルーティングテーブルを構築し，ノード・コンテンツの管理を行う．ルーティングテーブルには XOR の距離ごとに k 個のノード情報が格納される k-bucket が存在する．表 2，図 2 はハッシュサイズを 3 ビットした場合のルーティングテーブルおよびノードツリーを示したものである．

ルーティングテーブルの更新はメッセージ受信時に行われ，新しいノードは k-bukcet の末尾に追加される．k-bucket が k に達している場合に，k-bucket の先頭ノードの生存を確認し，存在しない場合，ノード情報の削除を行い，新しいノードが末尾に追加する．このため，ノード離脱時の処理は行わない．この時，k の値とは k-bucket のリスト長である．

表 1 2 の場合のルーティングテーブル

i	ノード 101 からの距離 (XOR 値)	k-bucket の内容例
0	1	100
1	2 ～ 3	110，111
2	4 ～ 7	001，011

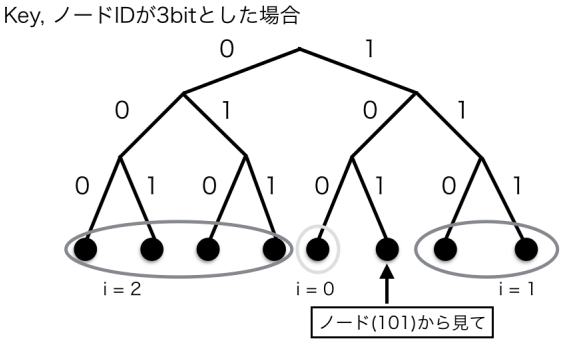

図 2 2 分木ハッシュ空間

Kademlia のメッセージは 4 種類あり，それぞれ非同期的に処理を行う．以下に 4 種類のメッセージについて説明する．

Ping 対象ノードの生存確認を行う．

FindNode(key) key に XOR 距離の近いノード k 個に問い合わせを行い，問い合わせ中，常に送信候補ノードを受信データにより，更新・洗練を行う．問い合わせを受けたノードは該当する k-bucket を返す．k-bucket 内が k に達していない場合，周辺の k-bucket からノード情報を集め，k 個のノード情報を返す．

FindValue(key) FindNode と動作は同じであるが，問い合わせ先のノードが key に対応する value を保持していた場合，value を返す．

Store(key, value) 対象ノードに key : value のペアを保持させる．

Kademlia のネットワーク参加を行う方法として，何らかの方法で得たノード情報からノードに接続し，接続ノードに自身の ID を key として，FindNode を行う．

また，FindNode, FindValue, Store を行う際には α 個のノードに対して，同時に問い合わせを非同期的に行う．この α の値は非同期並列数の数である．

2.3 WebRTC（Web Real-Time Communication）

2.3.1 概要

WebRTC とは，W3C，IETF により，標準化が行われいるリアルタイムコミュニケーション API であり，プラグインなしでブラウザ間のビデオチャット，音声チャット，ファイル共有を可能とする API である．プロトコルには UDP が採用されており，音声や映像，データの送受信を行う際に大量のデータを高速・低遅延での通信を可能とする．通信は標準で暗号化が施されており，データグラム向けの TLS である DTLS が採用されている．WebRTC API は複数の API からなり，WebRTC のピア間接続に関する API とブラウザ同士が直接通信を行うため，NAT 超えを実現する仕組みも組み込まれている．また，マイクや Web カメラなどを扱う Media Stream に関する API からなる．以下に主要な API について述べる．

RTCPeerConnectioin WebRTC オブジェクトを生成するメイン API であり，このオブジェクトを通して P2P 通信が行われる．

RTPSessionDescription P2P 通信のセッションパラメータを保持したオブジェクトであり，このオブジェクトをピア間で共有して所持することで接続完了とする．

RTCIceCandidate ICE[4] による接続経路を決定するためのオブジェクトであり，ICE サーバより IP アドレスやポート番号を入手する．

navigator.getUserMedia ブラウザからカメラやマイクなどへのアクセスを可能とする．

2.3.2 通信方式

WebRTC は 2 つの通信方式が存在し，getUserMedia から得られた Stream データを扱う Media Channel とどのようなデータでも扱うこともできる Data Channel がある．Media Channel は，リアルタイムに映像・音声を配信するためのプロトコルとして，RTP/RTCP を採用している．DataChannel はプロトコルとして，TCP と UDP それぞれ長所を持つような SCTP を採用しており，データの種類に制限はなく，さまざまなデータの使用が可能であるため，本研究ではこのプロトコルを使用し，ネットワークの構築を行う．2 つの通信方式とは別に SDP というセッション用プロトコルが存在し，SDP プロトコルにより，コネクション成立後，上記 2 つのプロトコルで P2P 通信が行われる．

2.3.3 P2P 接続

WebRTC はブラウザ間 P2P 接続を可能とするものであるが，ブラウザのみでの接続は行えず，シグナリングを行う必要がある．これは相手ピアとのマッチングを行うための仕組みであるが，WebRTC ではこの手法については定義されておらず，開発者が自由に設計を行って良いが，本研究ではクライアント主体でネットワークを構築することを想定しているため，現状では標準的な設計で行う．そのため，最低限の機能として，接続相手の IP アドレスの解決・接続の可否・通信方式およびそれに伴う情報の共有のための仲介機能を実現している．

WebRTC は NAT 超えを行うために，ICE[4] という仕組みを使用している．ICE サーバは，STUN サーバと TURN サーバから提供される．STUN サーバはピアの接続を受けるための候補として IP アドレス，ポートをピアに通知する．その後，受けったピアは．IP アドレスとポートの組み合わせを相手ノードと共有を行い，それぞれの組み合わせから UDP ホールパッチングを行い続け，疎通できるまで行う．TURN サーバは P2P 接続が行えないネットワーク環境であると判断された場合にその通信を中継する役割を担う．シグナリングでのピア接続と ICE（STUN サーバのみ）での処理手順を図 3,4 を用いて説明する．

(1) B へ接続要求
(2) A からの接続要求の通知
(3) A へ返答および接続の許可
(4) B からの返答受け取り
(5) ICE および STUN を使用し，お互いの ICE 情報の交換・UDP ホールパッチング
(6) P2P 接続の完了

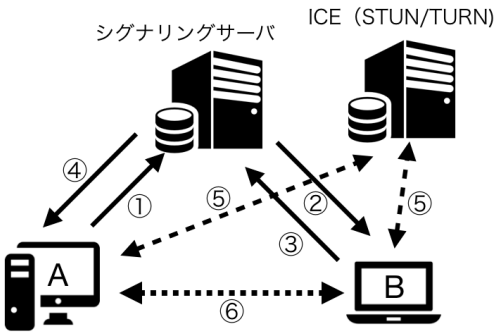

図 3 シグナリング接続概要

3. 手法・実装

3.1 システム概要

本システムは JavaScript と WebRTC および WebRTC のラッパーライブラリである PeerJS を用いて，DHT アルゴリズムの一つである Kademlia を実装している．Kademlia を選択した理由として，JavaScript および WebRTC に対して，多くのネットワーク制御を行わない，また，ネットワーク離脱時の処理も行うことがない．そのため，ルーティングテーブルの更新を通常のメッセージ送受信の中で行うことにより，ルーティングテーブルの更新を行う．こ

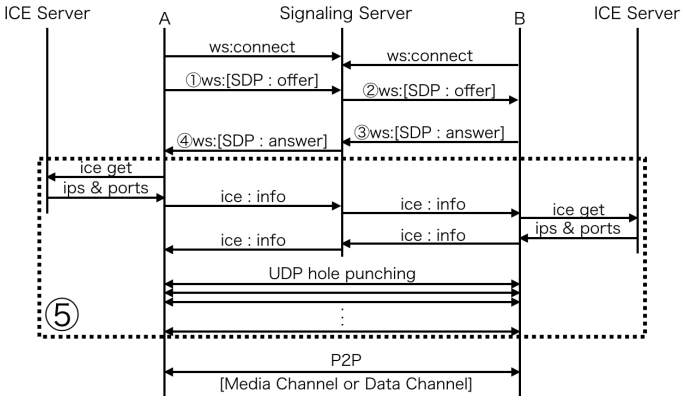

図 4　シグナリング接続での通信の流れ

のことからネットワーク制御による処理を削減できると考えた，また，XOR による ID 管理が明確であるため，本システムでの WebRTC ネットワーク上で各 ID の管理手法としてのルール化として適用している．

本システムでの Kademlia の HashSize および ID の長さは 160bit，k-bucket のリスト長である k の長さを 5，非同期並列数 α の値を 3 として，実装を行った．k については，本システムでの検証を行う規模を考え，値を低い数値にしている．

本論文では，WebRTC および JavaScript を用いた Kademlia でのネットワーク構築および制御の検討が主のため，最低限の Kademlia の実装を行っており，それに伴う効率的なデータ保持や Churn 耐性等は現状では考慮できていない．本システムで，Kademlia の機能として実装したものは，2 節で説明を行っている XOR を用いた ID 管理の機構およびルーティングテーブルに関する部分と 4 種類のメッセージング機能であり，Kademlia を動作させる最低限の機能を実装している．また，実装にあたって，JavaScript はシングルスレッドであり，ブラウザ上での実装となるため，ブラウザでの UI 操作が Kademlia の処理により，阻害されないことが重要である．そのため，JavaScript の仕様にあった Kadmelia の実装が必要不可欠である．

3.2　イベント駆動

JavaScript はシングルスレッドであり，効率的に処理を行うために非同期処理を行うことで効率よく処理を行っている．しかし，for や while を行った際には，スレッドを専有することになってしまい，他の処理が停止していまう．またイベント駆動型プログラミングは、イベントや状態の変化によって決定されるアプリケーションのフロー制御，JavaScript はシングルスレッドの対策として，イベント駆動型プログラミングが行える．ブラウザなどの onClick イベントなどがこれに当たる．

Kadmela 上での処理で複数のノードへの通信を行う際には for や while を使用し，受信データの処理を逐次更新して行かなければならない．そのためノードへの接続や受信をループ内で非同期的に行った場合，受信タイミングでの受信データおよびノード情報などを処理することが難しくなる．その問題を解決する場合，コールバックや ES2015 から仕様が可能となった Promise や Generator などを用いて処理が意図したにフローで処理が行わるように実装を行う．しかし，ループ内での受信待ちや接続待ち等を行うことは JavaScript 全体の処理を止めることにある．そのため，イベント駆動型での実装が不可欠となる．WebSocket や WebRTC はイベントハンドラでの受信方式を採用しており，Kademlia の実装でもこの方式を適切に取り扱い，ループ処理内に組み込む必要がある．このため，図 5 のような形で設計を行った．

この図 5 のような処理は FindNode および FindValue を想定している．そこで本システムにおける FindNode の処理の流れを述べる．FindNode の初回呼び出し時に設定の初期化および初回送信を行い，それ以降はイベントハンドラから受信データが渡され次第，ループ処理として終了判定まで処理を行い続ける．以上の処理をフローチャートとして図 6 に示す．

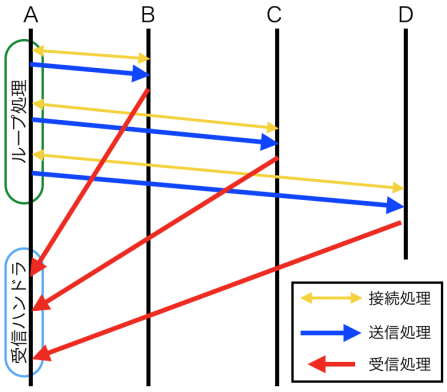

図 5　送受信時のデータフロー

3.3　処理・通信の識別

イベントハンドラでの実装だけでは複数のノードとのセッション管理が複雑になってしまう．そのため，JavaScript のインスタンスを利用し，各メッセージングおよびコネクションごとにインスタンスオブジェクトとして，管理を行う．管理を行う際には Kademlia の動作にはすべて key が与えられるため，key を鍵として，key:value の形で保持・管理を行う．これにより，イベントハンドラ受信時のデータ識別が可能となり，対応したオブジェクトを呼び出すことで処理をスイッチさせている．

3.4　コネクション数の削減

WebRTC での接続には多くの手順がかかることがわか

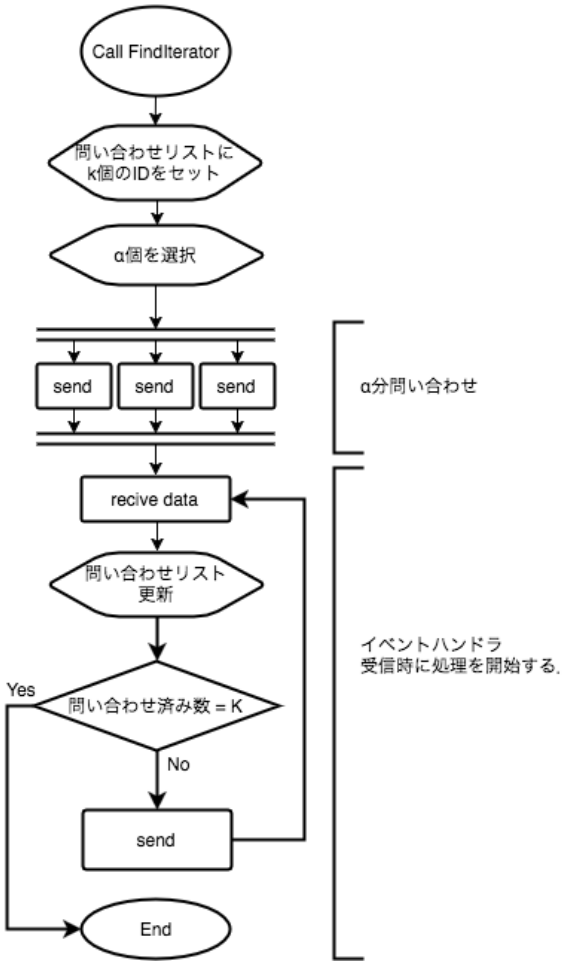

図 6 FindNode

る．短期間に複数回同じノードへの接続行為を行うことは非効率である．そのため，一定時間コネクションを保持し，コネクションが不要であればコネクションを閉じ，コネクションオブジェクトの削除を行う機構を設けた．接続維持上限数は k と同じ数値で設定している．新しい接続イベントを受けた際，接続上限に達していた場合，通信ステータスを参照し，切断が可能となっているものをすべて破棄する．また，切断時には一定時間後に切断され，意図しないデータの未受信を避けている．この機能に関しては，ネットワークの規模や各ノードの稼働量により調整が必要である．

3.5 実装

実装は JavaScript の新仕様である ES2015 を用いて，複数クラスでの設計・実装を行った．イベント駆動を軸にした実装となっているため，通信部分および WebRTC 部と Kademlia 部はよりシンプルな構造となっている．また，コネクション・メッセージングの動作ごとにインスタンス生成を行い，効率よく処理を分割している．送受信後の処理は非同期処理内でイベントハンドラでのループ処理とインスタンス空間ごとの処理分割を行なっており，JavaScript のライフサイクルを阻害されない．

今回実装した Kademlia はフレームワークとして設計されている．そのため，どの Web サイトにでも Kademlia を実装することが可能である．図 7 にフレームワークの使用方法を示す．

```
<html>
<head>
<script src="./peerjs.js"></script>
<script src="./kad.js"></script>
</head>
<body>
    // 以下のスクリプトと操作画面を紐付け
</body>
<script>
    const kad = new KNode();
    kad.on('open', (id) => {
        // get my id
    });
    kad.on('FV', (key, value) => {
        // receive get data
    });
    kad.join(key);
    kad.set(key, value);
    kad.get(key);
</script>
</html>
```

図 7 Kademlia フレームワーク

4. 評価

今回，実装を行った Kademlia の評価を行う．JavaScript および WebRTC で実装を行った Kademlia が大規模ノードで安定した動作が可能であることを確認する．

検証環境を以下に挙げる．

- MacOSX マシン
- Google Chrome Latest
- Web サーバ (nginx)
- シグナリングサーバ
- Selenium Web Driver

シグナリングサーバは，ローカル環境内に構築している．基本的な接続前のピア間通信のシグナリング機能と検証用のデバッグ機能を実装している．また ICE サーバは Google にて公開されている STUN サーバ [*1] を用いている．今回の検証では TURN サーバは必要がないため除外した．また，1 つのマシンには複数のブラウザ（ノード）を起動させることとする．

4.1 検証方法

検証方法として，Selenum Web Driver を使用し，ブラウザを自動操作を行う．以下の動作流れを示す．

(1) 初期ノードを生成する．

(2) 2 ノード目で初期ノードから任意の key:value の Store

[*1] stun.l.google.com:19302

を行う．

(3) 目的ノード数 N まで Join を行い，ネットワーク参加ノードを増加させていく．

(4) N まで達した後，N ノードが 2 ノード目に Store した value の検索を行う．

(5) ランダムに選んだノードから任意の key:value の Store を行う．

(6) (5) で Store したデータを保持していないノードからランダムに選び，(5) のデータ検索を行う．

具体的な評価として，(4) 時点でネットワーク参加後のルーティングテーブルの生成完了時間，問い合わせ回数および検索時間と検索問い合わせ数，(5) における，検索時間と検索問い合わせ数の比較を行う．

4.2 外部シグナリングサーバ利用時との比較

前回 [5]，実装していた Kadmelia では，PaaS のシグナリングサーバを使用して，小規模なもので動作検証を行った際の結果を図 8，図 9，図 10 に示す． この時，20 ノードから 30 ノードでの数値が急激に上がっていることがわかる．この原因の調査を行った．PaaS でのシグナリングサーバから観測可能なシグナリングサーバの構築を行い，サーバの負荷の確認を行った．この時，シグナリングサーバの負荷は見られなかった．そこでブラウザでのタブ数の調査を行い，1 つのマシンに複数のブラウザを立ち上げた状態で計測を行った．結果，計測マシン内で Websocket の

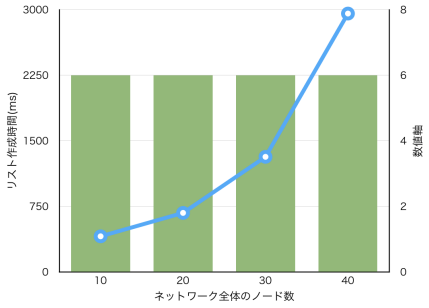

図 8　ルーティングテーブル作成時間　改善前

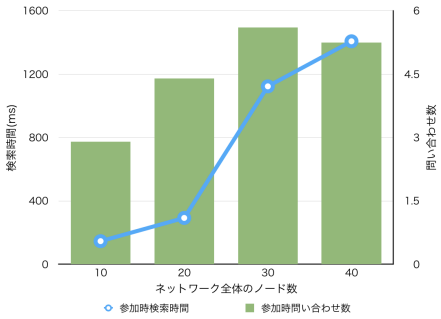

図 9　参加時データ検索　改善前

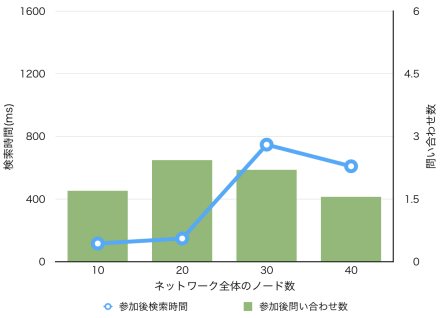

図 10　新データ Store からのデータ検索　改善前

接続台数が 20 セッション以上から，急激に速度が遅くなることがわかった．また，この計測を行った Kadmelia ではループ処理が最適化されていなかったため，ノード数の上昇により，動作時間がかかってしまうことがわかった．

そのため，今回，実装した Kadmelia での検証した結果を図 11，図 12，図 13 に示す．

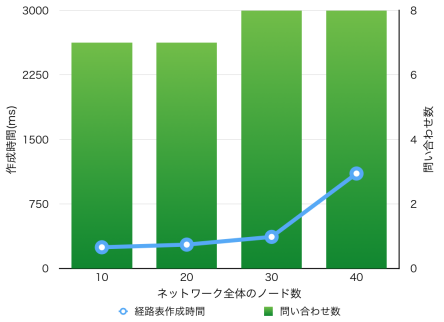

図 11　ルーティングテーブル作成時間　改善後

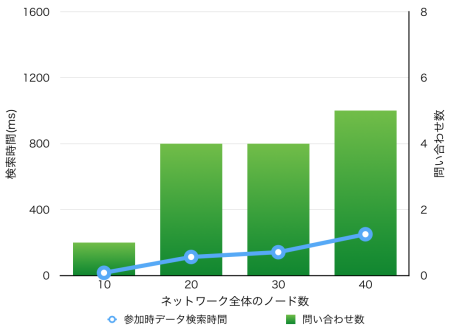

図 12　参加時データ検索　改善後

図 11，図 12，図 13 より，今回実装した Kademlia ではコネクション再利用機能や処理の最適化により，30 ノードでの急激な上昇は見られなくなったが，40 ノードではやはり急激な上昇が見られた．

また，ルーティングテーブル作成時間に着目し，前回との比較を行った．結果は図 14 となり，最適化されたことがわかる．

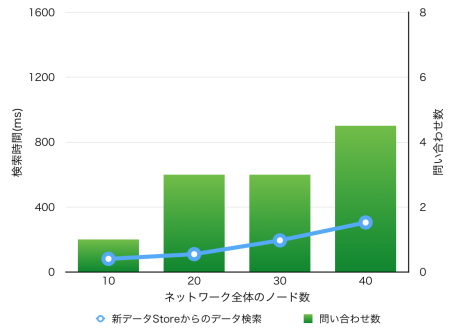

図 13　新データ Store からのデータ検索　改善後

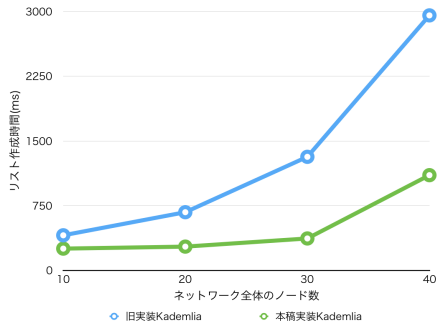

図 14　新旧 Kademlia の比較

4.3　複数ノードでの Kademlia 動作検証

検証マシンを 10 台用意し，それぞれのマシンに 10 ノードまでアクセスさせることにより，100 ノードまでの動作検証を行った．また，今回，実装した Kademlia では各マシン 20 ノードであっても動作は安定すると考えられるが，より安定したデータ計測を行うため各 10 ノードとして計測を行った．検証の結果，表 2 の結果となった．

表 2　100 ノードでの検証結果

	計測時間 (ms)	問い合わせ数
経路表作成時間	369	10
(4) での検索時間	186	5
(5) での検索時間	205	7

表 2 より，100 ノードでのネットワークの構築・基本的なメッセージング動作を行った場合でも，動作の遅延などは見られず，正しく動作していると言える．しかし，(4), (5) の問い合わせ回数が Kademlia 問い合わせ数とされている $O(\log N)$ よりも多く，計測時に問い合わせ数が多くかかってしまう場合があり，平均値の上昇につながってしまった．そのため，ID 比較部分やルーティングテーブルのソート動作を十分確認を行う必要があるとかん考えられる．

100 ノードの検証結果との比較として，問い合わせ数あたりの計測時間の比較を行った．ここでは 1 マシン 10 ノードでの計測データと 10 マシン 1 ノードでの計測データの比較を表 3 に示す．

表 3 より，1 ノードあたりの処理時間に大きな変化はなく，1 マシン 1 ノードでの動作を行った場合，ノード数が 100 以上の場合でも，同様な動作が期待できる．また，この数値を基にさまざまなネットワーク規模での処理時間の目安になると考えている．

表 3　問い合わせ数あたりの計測時間

マシン×ノード数	10 × 10	1 × 10	10 × 1
経路表作成時間 (ms)	37	35	36
(4) での検索時間 (ms)	37	31	40
(5) での検索時間 (ms)	29	30	32

5.　まとめ・今後の課題

今回，WebRTC を用いた P2P ネットワークの構築時の実装および 100 ノードでの検証を行った結果，基本的な動作に問題がないことが確認できた．また，少数ノードとの問い合わせあたりの処理時間を比較し，ノード数が増加した場合でも，変化がないことが確認できた．これにより，Kademlia を用いて，シグナリングサーバでのユーザ管理を必要とせず，ノード間でユーザ管理機構が提供できる．今後の課題として，現在，実装した Kademlia は Churn 耐性やデータ検索．保持の効率的な仕組みを取り入れていない点，本格的なネットワークとして動作を行った状態での検証が必要と考えている．また，現状，サーバサイドとしてシグナリングサーバを用いているが，サーバサイドでのノード同士の関連性を隠蔽するような仕組みなどの取り込み，より柔軟なネットワーク構築を目指す．

参考文献

[1] 安倍 洋丈：DHT：分散ハッシュテーブル： Pure Peer-to-Peer システムのための基盤技術：コンピュータソフトウェア 23(1)，1-14, 2006-01-26, 一般社団法人日本ソフトウェア科学会

[2] P. MAYMOUNKOV：”Kademlia: A Peer-to-peer Information System Based on the XOR Metric”：Peer-to-Peer Systems，Proc. the 1st International Workshop on Peer-to-Peer Systems，IPTPS’02，53-65，2002-03

[3] WebRTC: WebRTC, https://webrtc.org

[4] IETF: Interactive Connectivity Establishment (ICE), A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, http://tools.ietf.org/html/rfc5245

[5] 成田 裕司，坪川 宏：WebRTC を用いた P2P ネットワークの構築と 効率化についての検討と評価：情報処理学会第 78 回全国大会 (3)，213-214, 2016-03-12, 情報処理学会