Vol.2012-OS-123 No.3
Vol.2012-EMB-27 No.3
2012/12/5

分散型 Web ブラウザの通信機能と協調動画視聴アプリケーション

郭 飛[1,a)]　青沼 伴樹[1,b)]　新城 靖[1,c)]　佐藤 聡[1]　中井 央[1]　板野 肯三[1]

概要：近年、 Facebook や Twitter を始めとするソーシャルネットワークサービス（SNS）が広く使われるようになっている。現在の集中型 SNS には中央のサーバを用いることによるスケーラビリティの問題やプライバシの問題があるため、 SNS を分散システムとして構築する研究が数多く行われている。我々は、 Web ベースのソーシャルアプリケーションを実行する環境として、分散型 Web ブラウザを開発している。この論文では分散型 Web ブラウザの通信機能と協調動画視聴アプリケーションについて述べる。従来の分散型 Web ブラウザではインスタントメッセンジャー Skype のオーバーレイネットワークを利用し Remote Procedure Call (RPC) 機能を提供しているが、単一障害点などの問題がある。本研究ではそれらを解決するために、Extensible Messaging and Presence Protocol (XMPP) で WebSocket 風の Socket 機能 FriendSocket を提供する。FriendSocket による通信機能を利用し、グループ通信を実現する通信ミドルウェアとそれを利用して複数人で動画を同時に視聴する機能を持つ協調動画視聴アプリケーションを開発する。

キーワード：ソーシャルネットワークサービス，Web ブラウザ，ソケット，協調アプリケーション

1. はじめに

今日、多くの協調アプリケーションが利用されている。協調アプリケーションとは複数のユーザで協調作業をすることができるアプリケーションである。協調アプリケーションの例として、複数のユーザがそれぞれのパソコンで同時にドキュメントを編集したり、同時にウェブページを閲覧したりするものがあげられる。多くの SNS（Social Network Service）アプリケーションも協調アプリケーションの一種である。

協調アプリケーションの中でも Web アプリケーションとして開発されているものも多く存在する。現在広く利用されている Web アプリケーションの多くは中央のサーバを用いて実現される。これにより、潜在的なスケーラビリティが低いこと、およびセキュリティやプライバシが維持できない問題が発生することが指摘されている [1]。また、サービスが終了した時に個人のデータが取り出せなくなるといった問題が発生する。

これらの問題を解決するために、 我々は、Web ベースのソーシャルアプリケーションを実行する環境として分散型 Web ブラウザを提案している [2][3][4]。分散型 Web ブラウザは、マルチユーザのブラウザであり、ユーザの認証、ブラウザ間の通信、ブラウザ上のプロセス管理、分散ストレージ、コンタクトリストの管理などの機能を提供する。ユーザは協調作業を行いたいとき、分散型 Web ブラウザ上で協調アプリケーションを実行する。分散型 Web ブラウザ上で動作する協調アプリケーションはデータを中央のサーバに置く必要がなく、上記の中央のサーバに起因する問題を解決することができる。

本論文では分散型 Web ブラウザの通信機能の設計と実装について述べる。そして、それを利用したグループ通信を実現する通信ミドルウェアと、協調アプリケーションの例として、複数人で動画を同時に視聴する機能を持つ協調動画視聴アプリケーションを開発する。これにより、対話的な協調アプリケーションを実行した際の分散型 Web ブラウザの通信機能と性能を評価する。

2. 現在の分散型 Web ブラウザの通信機能とその問題点

現在の分散型 Web ブラウザは、通信機能として、Remote Procedure Call (RPC) に基づくものを提供している [2]。協調アプリケーションは、RPC を用いてクライアント・

[1] 筑波大学
University of Tsukuba
[a)] kaku@softlab.cs.tsukuba.ac.jp
[b)] aonuma@softlab.cs.tsukuba.ac.jp
[c)] yas@cs.tsukuba.ac.jp

サーバ・モデルに基づき JavaScript により記述される。通常の RPC とは異なり、分散型 Web ブラウザの RPC は、非同期的なインタフェースになっている。これは、JavaScrip のイベント駆動のモデルに適合させるためである。この RPC は、インスタント・メッセンジャ Skype [5] が提供している AP2AP (Application to Application) API (Application Program Interface) を利用して実装されている。協調アプリケーションは、通信相手を Skype のユーザ名で識別することができる。

この RPC に基づく通信機能を用いて、次のようなアプリケーションが開発された [2]。

- 簡単な協調ブラウジング
- 簡単なコメント共有
- ブラウザのスクリーンショットの送受信

これらのアプリケーションの開発を通じて、RPC に基づく通信の有用性は確認された。しかし、現在の分散型 Web ブラウザの通信機能には次のような問題点が残されている。

- Skype の中央のサーバが提供している認証機能に依存している。Skype の認証機能が利用できなくなると、分散型 Web ブラウザの機能も利用できなくなる[*2]。
- RPC では記述しにくいアプリケーションが存在する。たとえば、協調動画視聴では、サーバからのプッシュ型の通信やグループ通信機能が必要であるが、RPC ではうまく記述することができない。

1 つ目の問題を解決するために、本研究では、複数のインスタント・メッセージング・システム (Instant Messaging System, IMS)、または、ソーシャル・ネットワーク・サービス (Social Network Service, SNS) を利用して分散型 Web ブラウザの通信機能を実現する。詳しくは 3.1 節で述べる。

2 つ目の問題を解決するために、本研究では、RPC に加えて WebSocket[6] 風の FriendSocket 機能を提供する。詳しくは 3.2 節で述べる。

3. FriendSocket

3.1 複数の IMS や SNS の利用

本研究で実現する分散型 Web ブラウザの全体の構造を図 1 に示す。このように、通信機能は複数の IMS、または、SNS を利用して実現している。したがって、それらのうち 1 つでも利用可能であれば、本分散型 Web ブラウザを利用しつづけることができるようになる。

Skype 以外の IMS または SNS としては XMPP（Extensible Messaging and Presence Protocol）を提供しているものを利用する [7]。XMPP は Jabber 社が開発した XML ベースのインスタントメッセンジャのためのプロトコルであり、現在では MSN Messenger、Yahoo!メッセンジャー、Google Talk、Facebook などが利用している。XMPP プロトコルの利点は特定の中央サーバに依存しないことである。つまり、複数の SNS または IMS から選択した利用することができる。また、電子メールのように、自分で XMPP サーバを設置することもできる。

[*2] 例えば、2010 年 12 月、Skype のソフトウェアのバッグにによる障害が発生し、数日間使えなくなった。

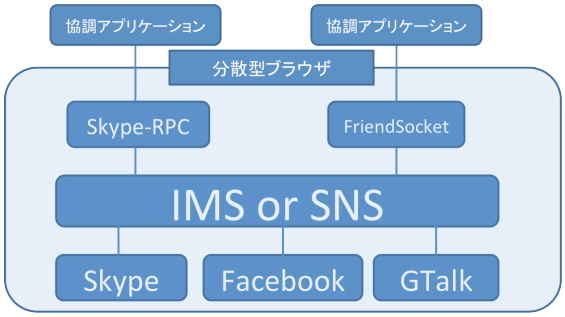

図 1 複数の SNS や IMS を利用する分散型 Web ブラウザの通信方式

```
1 var c;
2 c = friendsocketio.connect(host,
      name, password, peername,
      appname);
3 c.on("connect", function(m) {
4     do_something(m);
5 });
6 c.on("clicked", function(m) {
7     do_something(m);
8     c.close();
9 });
```

図 2 FriendSocket のクライアント側の使用例

3.2 FriendSocket

WebSocket は W3C と IEFT で標準化が進められている Web クライアントとサーバ間の通信のための API およびプロトコルである [6]。WebSocket では、クライアントからもサーバからもデータの送信することができるので、サーバからのデータのプッシュを実現することは容易になる。ただし、Web ブラウザ上の JavaScript はクライアント側の API しか利用できない。そのため、ブラウザ間の通信は必ず中央のサーバを介して行われることになる。従って、本研究の目標である中央のサーバに依存しないということは達成できない。そこで本研究は JavaScript 側にサーバ の API を提供する。本研究ではそれを FriendSocket と呼ぶ。

FriendSocket API のインターフェースは表 1、表 2 および表 3 のようになる。このインターフェースは Socket.IO を参考し設計した [8]。表 1 の connect() はクライアントからサーバに接続する関数、listen() はサーバがクライアント

表 1 FriendSocket API

関数	説明
`frinedsocketio.connect(h,n,pw,pn,an)`	サーバに接続する準備を行い、friendsocket オブジェクトを生成する。h は接続する host、n はユーザ名、pw はパースワード、pn はサーバのユーザ名、an は appname である。
`frinedsocketio.listen(h,n,pw,an)`	クライアントからの接続を待ち、friendsocketserver オブジェクトを生成する。h は接続する host、n はユーザ名、pw はパースワード、pn はサーバのユーザ名、an はアプリケーションの名前である。

表 2 friendsocket オブジェクトのメンバ関数

メンバ関数	説明
`on(e, f)`	イベントを追加する。e はイベント名の文字列、f はイベントが発生した時のコールバック関数である。コールバック関数の引数は受信したデータの文字列である。
`emit(e, d)`	相手に送信する、e は相手が定義したイベント名、d は送信したいデータの文字列である。
`getpeername()`	接続する相手のユーザ名を取得する。
`close()`	接続を切断する。

```
1  var friend_list = new List();
2
3  var s;
4  s = friendsocketio.listen(host, name
       , password, appname);
5  var list = s.contactlist();
6  s.on("connection", function(friend)
       {
7      if(list.contains(friend.
           getpeername()))
8          friend_list.push(friend);
9      else
10         friend.close();
11 });
12
13 function sendMsg() {
14     var friend, i, len;
15     for (i = 0, len = friend_list.
           length; i < len; i++) {
16         friend = friend_list[i];
17         friend.emit("clicked", "
               Button1");
18     }
19 }
20
21 var el = document.getElementById("
       Button1");
22 el.addEventListener("Click", sendMsg
       , false);
```

図 3 FriendSocket のサーバ側の使用例

からの接続を待つする関数である。これらの関数を使い、開発者は Web ブラウザ上で動作するクライアントとサーバの両方を記述できる。表 2 はクライアントおよびサーバで接続が確立され、通信に用いる FriendSocket のインターフェースを示している。on(e, f) はそのイベント e が発生した時にユーザが定義したコールバック関数 f を呼ぶように登録する関数である。emit(e, d) は接続する相手にデータ d を送信する関数である。このデータはイベント e に登録されたコールバック関数 f で受信することができる。close() は接続を切断する関数である。表 3 ではサーバ側のインターフェースを示している。サーバ側では接続が確立した時、コールバック関数が呼ばれる。コールバック関数の引数として接続する相手の friendsocket オブジェクトを渡す。このオブジェクトの利用方法は表 2 にしたものと同じである。contactlist() はコンタクトリストの一覧を返す関数である。

FirendSocket を分散型 Web ブラウザから使うためのクライアント側の API の使用例を図 2 に示す。利用可能な関数は、通常の WebSocket の 1 つの実現である Socket.IO と同じであるが、接続方法が異なる。一般の Web ブラウザの WebSocket ではクライアントは URL を指定して接続を行う。分散型 Web ブラウザの場合、図 2 の 2 行目のように、クライアントでは、IMS や SNS のユーザ名を利用し接続先の指定することができる。また、メッセージの受信側はメッセーシ送信者を認証することができる。分散型 Web ブラウザは、この認証処理を IMS や SNS のユーザ情報を利用して行う。

FriendSocket を分散型 Web ブラウザから使うためのサーバ側の API の使用例を図 3 に示す。Socket.IO ではブラウザでクライアントしか使えないが、FriendSocket ではブラウザでクライアントとサーバを両方使うことができる。このプログラムはマウスでクリックすると、クライアントへメッセージを送信する（１３～１９行目）。

サーバ側の分散型 Web ブラウザは、IMS あるいは SNS にログインし、コンタクトリストを取得し、クライアント

表 3　friendsocketserver オブジェクトのメンバ関数

メンバ関数	説明
`on("connection", f)`	サーバに接続した時、コールバック関数 `f` を呼び出す。コールバック関数の引数は friendsocket オブジェクトである。
`emit(e, d)`	相手に送信する、`e` は相手が定義したイベント名、`d` は送信したいデータの文字列である。
`contactlist()`	ユーザのコンタクトリストの一覧を配列として返す。

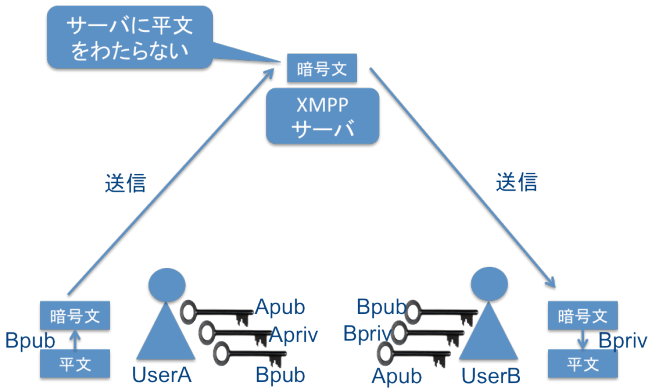

図 4　XMPP クライアント間の暗号化

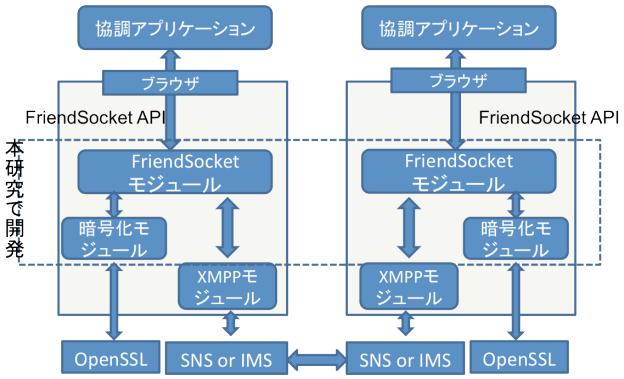

図 5　実装

からの接続要求を待つ。

サーバ側では、クライアント側から接続要求が到着すると、分散型 Web ブラウザは、コンタクトリストを調べる（図 3 の 6 ～ 10 行目）。クライアントのユーザ名がコンタクトリストにない場合には、接続を拒否する。コンタクトにあれば、分散型 Web ブラウザは、接続を許可する。サーバ側のアプリケーションは、コールバック関数の中で getpeername() という関数を呼ぶことでクライアントの IMS または SNS のユーザ名を得ることができる（図 3 の 7 行目）。FriendSocket の通信機能としては Skype と XMPP を考えている。本論文では XMPP の実装について述べる。

3.3　FriendSocket における XMPP クライアント間通信の暗号化

現在、FriendSocket を XMPP を用いて実現している。XMPP のクライアントとサーバが通信する時、暗号通信の方法が標準化されている。しかし、クライアントとクライアントの間の暗号通信は決められていない。FriendSocket によるブラウザ間の通信は XMPP サーバを経由する。そのため、ブラウザとブラウザの間でデータを送受信する場合、データが全て XMPP サーバに傍受される。もしその XMPP サーバが悪意のあるサーバであれば、ユーザが送信したデータが盗まれてしまう。

そこで本研究ではブラウザとブラウザの間の通信を暗号化する。この様子を図 4 に示す。暗号化方式としてはインスタントメッセンジャ Gajim [*3] と同様に公開鍵暗号方式を利用する。そのため、ユーザは PGP（Pretty Good Privacy）と同様に鍵を管理する。ユーザは分散型 Web ブラウザを利用する前にお互いに公開鍵を交換する。そして、データを送信する時、クライアント側は受信者の公開鍵でデータを暗号化しサーバに送信する。サーバ側は鍵を持ってないので、復号化することができない。サーバは暗号文をもう一つのクライアントに送信し、クライアントは自分の秘密鍵で暗号文を復号化する。

[*3] http://gajim.org/

3.4　Firefox における FriendSocket の実装

本研究では、分散型 Web ブラウザを Firefox ブラウザをベースとして開発している。OS は Linux を用いている。

本研究で FriendSocket を Friefox の Addon として実装する。図 5 のように、FriendSocket の Addon は JavaScript で記述している。Addon では XPCOM（Cross Platform Component Object Model）を通じて他のモジュールの機能を利用できる。

協調アプリケーションは JavaScript で書かれている。その JavaScript で記述された協調アプリケーションは Addon で提供する FriendSocket API を呼び出して利用する。XMPP の通信は xmpp4mox という XPCOM を利用し、実装している。暗号化は OpenSSL ライブラリを利用し、実装している。言語としては C 言語を利用している。現在までに、暗号化なしで通信をおこなう FriendSocket の基本機能が動作している。現在、暗号化の部分を実装している。

4.　グループ通信を実現する通信ミドルウェア

協調アプリケーションの通信にはグループ通信が適して

表 4 通信ミドルウェアの関数

関数	説明
`joinGroup(n)`	`joinGroup` 関数はメンバをグループ `n` に加入させ、メッセージを送受信できるようにする。グループへの接続に成功すると `group` オブジェクトを返す。
`leaveGroup(g)`	`leaveGroup` 関数はメンバをグループから離脱させる。 `g` は離脱したいグループの `group` オブジェクトである。

表 5 group オブジェクトのメンバ関数

メンバ関数	説明
`subscribe(t, cb)`	`subscribe` 関数は種類 `t` のメッセージを受信したときに呼ばれるコールバック関数 `cb` を登録する。コールバック関数の第 1 引数には message オブジェクトが渡される。
`publish(t, c)`	`publish` 関数は種類 `t` のメッセージを送信する。 `c` はメッセージに含まれる任意のオブジェクトであり、 ネットワーク上では JSON 形式の文字列にエンコードされ送受信される。

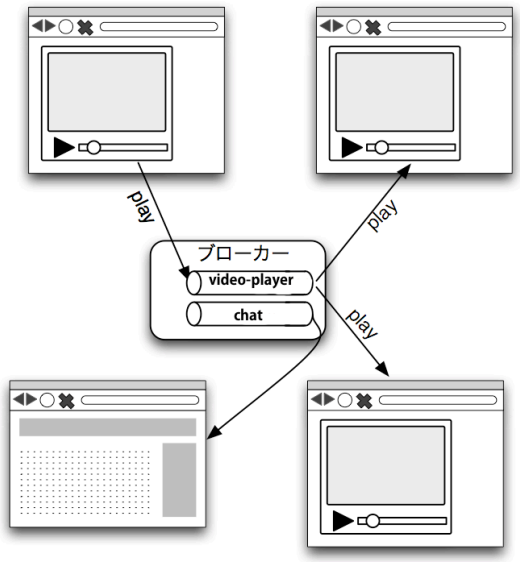

図 6 ブローカとメンバの関係図

いることがあるが、3 章で述べた FriendSocket のグループ通信機能では不十分なことがある。たとえば、 FriendSocket のサーバとして動作しているメンバが途中で離脱したとき適切な処理を行う機能や、送受信されたメッセージの時間的順序を保証する機能は備わっていない。分散アプリケーションが必要とするこれらの機能をアプリケーションが別個に実装するのは非効率的である。

そこで本研究では、分散型 Web ブラウザの FriendSocket を基盤として高機能なグループ通信を実現する通信ミドルウェアを開発し、アプリケーション開発の負担を軽減する。本ミドルウェアでは、 Publish/Subscribe モデルに基づくメッセージング機構を導入する。その JavaScript からの API を**表 4**、**表 5** および**表 6** に示す。

本ミドルウェアの Publish/Subscribe モデルでは、メッセージを送信または受信するメンバと、メッセージを中継するブローカからなる（**図 6**）。メッセージを送信するメンバは、**表 4** の `joinGroup` 関数でグループに参加し、表 6 の `publish` メンバ関数でメッセージを送信する。メッセージはブローカに送られ、ブローカによってタイムスタンプとシーケンス番号が付加されて受信側のメンバに送信される。受信側で受け取るメッセージを表す message オブジェクトのプロパティを**表 6** に示す。このオブジェクトの timestamp プロパティにブローカが付加したタイムスタンプが格納されている。 sequence プロパティにはメッセージを送信したメンバが付加したシーケンス番号が格納されている。メッセージを受信するメンバは、受信したいメッセージの種類とコールバック関数を**表 5** の `subscribe` メンバ関数で登録する。メンバにメッセージが到着すると、message オブジェクトを引数にして、登録されたコールバック関数が呼び出される。

本ミドルウェアでは、ブローカでメッセージにタイムスタンプを付加するが、受信側でメッセージの到着順を保証しない。すなわち、受信側ではタイムスタンプが新しいメッセージが先に到着し、タイムスタンプの古いメッセージが後から到着することがある。メッセージの到着順を保証しない理由は、第 1 に、FriendSocket の実装で用いる Skype の通信路がメッセージの到着順を保証していないからである。第 2 の理由は、アプリケーションによってはメッセージの到着順が重要ではないからである。なお、メッセージの到着順が必要なアプリケーションは、シーケンス番号を用いて実現することができる。

ブローカは、信頼できるユーザが実行している分散型 Web ブラウザ上で動作する。ブローカを動作させる分散型 Web ブラウザは通信ミドルウェアが決定する。そのため、ミドルウェアを利用するアプリケーションからブローカを直接制御する必要はない。

通信ミドルウェアについては、現在までにグループ参加・離脱機能とメッセージの送受信機能を実装した。現在

表 6 message オブジェクトのプロパティ

プロパティ	説明
sender	sender プロパティはメッセージを送信したユーザのユーザ名を表す文字列である。
content	content プロパティはメッセージを送信する際 publish 関数に渡されたオブジェクトである。
timestamp	timestamp プロパティはメッセージが送信された時刻をミリ秒単位の UNIX 時刻で表した整数である。
sequence	sequence プロパティはメッセージに付加されるシーケンス番号である。

表 7 MediaPlayer オブジェクトのメンバ関数

メンバ関数	説明
play(t)	play 関数は動画の再生を開始する。 t は再生位置を表す実数（秒単位）。
onPlayButtonClick(cb)	onPlayButtonClick 関数は動画プレイヤの再生ボタンがクリックされたときに呼ばれるコールバック関数 cb を登録する。
getCurrentTime()	getCurrentTime 関数は現在の再生位置を秒単位の実数で返す。

図 7 協調動画視聴アプリケーションの動作画面

ブローカの耐障害性を実現している。

5. 協調動画視聴アプリケーション

本研究では、4 章で述べた通信ミドルウェアを利用する対話的アプリケーションとして協調動画視聴アプリケーションを開発した。

図 7 は協調動画視聴アプリケーションのプレイヤー画面のスクリーンショットである。このアプリケーションは、協調視聴に参加している複数の分散型 Web ブラウザ上で動画を同時に再生する機能を持つ。ユーザはこのアプリケーションを一般的な Web ページ埋め込み型動画プレイヤと類似のインターフェースで操作する。アプリケーションの最上部には動画が表示され、その上にはマウスで絵を描くことができる。また、他のメンバが描いた絵が重ねて表示される。動画の下には再生ボタンと停止ボタン、シークバーが並び、動画を操作することができる。続いて自分を含む参加メンバが投稿したコメントを一覧するリストがある。リストの下部には新しいコメントを投稿するテキストエリアと送信ボタンが配置されている。

本動画協調視聴アプリケーションで動画を再生・停止する操作やコメントを投稿する操作などをアクションと呼ぶ。アクションは通信ミドルウェアの機能を使い他の分散型 Web ブラウザ上の動画プレイヤにも配信される。これによって、複数人が再生操作を同期して同じ動画を視聴することが可能となる。

本協調動画視聴アプリケーションは次のようなアクションを配信する。

- play アクション：動画の再生を開始する
- pause アクション：動画の再生を停止する
- seek アクション：動画の再生位置を変更する
- comment アクション：コメント欄にコメントを書き込む
- draw アクション：動画に重ねられた描画領域に画像を表示する

協調動画視聴アプリケーションでは、表**表 7** に示した MediaPlayer オブジェクトを通じて動画プレイヤを操作できる。この関数と通信ミドルウェアの API を組み合わせて play アクションの配信を実装した主要部分を**図 8** に示す。

協調動画視聴アプリケーションは必ず 'player-group' グループに接続し、アクションの配信に関しては 'media-player' という種類のメッセージを送信する。また、送信するメッセージはオブジェクトであり、必ず action プロパティを持っている。 action プロパティはアクションの種類を表す文字列である。 さらにメッセージはアクションの種類に応じて固有のプロパティを持つ（**表 8**）。

図 8 において、 メッセージを送信するメンバはまず 'player-group' グループに接続する。そして協調動画視

表 8　アクションとプロパティの詳細

アクション	固有プロパティ	説明
play	time	play アクションは動画の再生開始を表す。 time プロパティは動画の再生開始位置を表す実数（秒単位）。
pause	time	pause アクションは動画の停止を表す。 time プロパティは動画の停止位置を表す実数（秒単位）。
seek	time, prev_time	seek アクションは動画のシークを表す。 time プロパティはシーク先の位置を表す実数（秒単位）、 prev_time はシーク前の位置を表す実数（秒単位）。
comment	text	comment アクションはコメント欄にテキストが書き込まれたことを表す。 text はコメントの文字列。
draw	image	draw アクションは描画領域に絵が書き込まれたことを表す。 image は画像データをエンコードした文字列（data URI 形式）。

```
1  // Publisher side
2  var group = joinGroup('player-group'
       );
3  MediaPlayer.onPlayButtonClick(
       function() {
4      group.publish(
5          'media-player',
6          {
7              action: 'play',
8              time: MediaPlayer.
                   getCurrentTime(),
9          }
10     );
11 });
12
13 // Subscriber side
14 var group = joinGroup('player-group'
       );
15 group.subscribe('media-player',
       function(m) {
16     var c = m.content;
17     if (c.action == 'play') {
18         MediaPlayer.play(c.time);
19     }
20 });
```

図 8　動画プレイヤにおける通信ミドルウェアの API を利用した play アクションの配信

聴アプリケーションにコールバック関数を登録する。このコールバック関数は動画プレイヤの再生ボタンが押されたときに実行される。コールバック関数は 'media-player' メッセージを送信する。このときメッセージの action プロパティには play アクションを表す 'play' をセットする。さらに、 play アクションに固有の time プロパティには動画の再生位置をセットする。

メッセージを受信するメンバは、送信側と同じく 'player-group' グループに接続する。そして 'media-player' メッセージを受信するためのコールバック関数を登録する。このコールバック関数は、受信したメッセージの action プロパティが 'play' のとき、 time プロパティの値の位置から動画の再生を開始する。

ネットワークの状態によっては、2 つのメッセージを受信するとき、先に送信されたメッセージを後に受信することがある。たとえば、メンバ A が動画の再生を開始し、その後でメンバ B が動画の再生を停止したが、メンバ C にはメンバ A からのメッセージが後に届くことが考えられる。すべてのメッセージを無条件に受理してしまうと、あるメンバは動画が再生されているが、他のメンバでは再生が停止しているといった状態になりうる。

この問題に対して、 play、 pause、 seek アクションについては、それぞれのアクションについて最新のタイムスタンプを持つメッセージを記録しておき、それより古いタイムスタンプのメッセージを受信しても無視することで整合性を保つ。 comment アクションについては、メッセージの到着順は重要ではないため、すべてのメッセージを受理する。 draw アクションについては、送信したメンバごとに分けて、最新のメッセージを 1 つずつ記録し古いメッセージを無視する。画像を表示するときはメンバごとの最新のメッセージに含まれる画像を合成し、描画領域の下に重ねる。

6. 関連研究

Opera Unite [9] は、Opera Web ブラウザ上で Web サーバを稼動させる技術である 。 例えば、Opera Unite のファイルサーバ機能を使うと、ローカルマシンのファイルを全インターネットユーザまたは限られた Opera Unite ユーザに対して公開することができる。また、JavaScript で記述したアプリケーションをブラウザ上の HTTP サーバで動作させ、ファイルサーバの場合と同様に公開先を決めて公開できる。Opera Unite の中央サーバは、ブラウザ上のサーバをインターネットからアクセス可能にするとともに、他のユーザがブラウザ上のサーバにアクセスする際のユーザ認証も行う。一方、本研究で利用する分散型 Web ブラ

ウザは、Skype を利用する場合起動時に外部の認証サーバに接続することを除き、ブラウザ間の通信で中央サーバを必要としない。XMPP を利用する場合でも末端のブラウザ間で暗号通信がなされるので中央のサーバに通信内容が傍受されることがない。

多くの SNSs や OSNs が提案され、開発された。その中では分散 OSNs(Distributed OSNs)[1] と呼ばれるものがある。分散 OSNs は DHT(distributed hash table) などの P2P(peer-to-peer) 技術を利用し、中央サーバを使わずに、サービスを提供するシステムを目指している。これに対して、分散型 Web ブラウザの目標は、中央のサーバを使わない協調アプリケーションの実行環境を提供することである。この目標を達成するために、本研究では既存の SNS や IMS を利用する。SNS や IMS を利用することで、分散型 Web ブラウザの実装が簡単になっている。

PeerSoN は、 分散ハッシュテーブル（DHT）で分散 SNS を構築する研究である [10]。 PeerSoN では現在までに DHT を利用したノードのルックアップ機能を実装している。ノード間の通信は、送信者のユーザ ID などをキーとしてメッセージをハッシュテーブルに格納することで行われる。 PeerSoN はメッセージの暗号化機能を提供しないが、本研究ではメッセージは分散型 Web ブラウザにより暗号化される。

Collaborative browsing and search (COBS) は、分散型 SNS のアプリケーションとして協調ブラウジングとアノテーション共有を実現する研究である [11]。COBS は Web ブラウザのフロントエンドと DHT のバックエンドからなる。ただし現在の実装では協調ブラウジングの実現のために中央の XMPP サーバを用いている。 COBS と比較して本研究には 2 つの特徴がある。1 つ目は、既存のインスタントメッセンジャネットワークと API を利用するために実装が比較的単純になる点である。もう 1 つは、グループ通信機能をミドルウェアが協調アプリケーションから利用可能になっている点である。

7. おわりに

この論文では、分散型 Web ブラウザの通信機能の設計について述べた。その特徴は、 複数の IMS や SNS を利用することにより、可用性を高めていること、および、 ブラウザ側で FriendSocket API が利用可能になっている点にある。FriendSocket API では、WebSocket API と似ているが、ブラウザ上でサーバ機能が利用できる点が異なる。

また、FriendSocket を利用し Pub/Sub モデルに基づきグループ通信機能を提供する通信ミドルウェアとを実現した。そして、通信ミドルウェアを利用する協調動画視聴アプリケーションについて述べた。今後はアプリケーションの未完成の部分を完成させ、分散型 Web ブラウザの機能性と性能の評価を行う。

参考文献

[1] Datta, A., Buchegger, S., Vu, L., Strufe, T. and Rzadca, K.: Decentralized Online Social Network, *Handbook of Social Network Technologies and Applications*, pp. 349–378 (2010).
[2] Shinjo, Y., Guo, F., Kaneko, N., Matsuyama, T., Taniuchi, T. and Sato, A.: A Distributed Web Browser asa Platform for Running Collaborative Applications, *CollaborateCom 2011* (2011).
[3] 青沼伴樹, 郭 飛, 新城 靖, 佐藤 聡, 中井 央, 板野肯三：分散型ブラウザにおける協調動画視聴アプリケーションの開発，情報処理学会コンピュータシステムシンポジウムポスターセッション (2011).
[4] 郭 飛, 青沼伴樹, 新城 靖, 佐藤 聡, 中井 央, 板野肯三：分散型ブラウザにおける通信機能の設計，情報処理学会コンピュータシステムシンポジウムポスターセッション，情報処理学会 (2011).
[5] Baset, S. A. and Schulzrinne, H. G.: An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, *25th IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1–11 (2006).
[6] Hickson, I. (ed.): The WebSocket API, http://www.w3.org/TR/2011/WD-websockets-20110419/ (2011).
[7] Saint-Andre, P. (ed.): *Extensible Messaging and Presence Protocol (XMPP): Core*, http://www.ietf.org/rfc/rfc3920.txt (2004).
[8] Rauch, G.: Socket.IO, http://socket.io/ (2012).
[9] Tmmerholt, H. S. and Davis, D.: Skype Public API, *Opera Unite developer's primer* (2009).
[10] Buchegger, S., Schiöberg, D., Vu, L.-H. and Datta, A.: PeerSoN: P2P Social Networking: Early Experiences and Insights, *ACM EuroSys Workshop on Social Network Systems (SNS 09)* (2009).
[11] von der Weth, C. and Datta, A.: COBS: Realizing Decentralized Infrastructure for Collaborative Browsing and Search, *IEEE Advanced Information Networking and Applications (AINA 2011)* (2011).