ブロックチェーン技術Ethereumを用いたソーシャルコミュニケーションチャネル

宝田 一希[1]　周 毅[1]　新城 靖[1]　林 致遠[1]　中村 公洋[1]

概要：ソーシャルコミュニケーションチャネルとは，SNS（Social Networking Services）において，チャットや通話等に用いる通信路である．現在広く普及している SNS は，中央サーバに依存しており，それに起因する技術的な問題，および，社会的な問題がある．これらの問題を解決するため，この論文は，ブロックチェーン技術 Ethereum を用い，中央サーバに依存しないソーシャルコミュニケーションチャネルを実現することを提案している．この論文で対象としているソーシャルコミュニケーションチャネルは，VPN（Virtual Private Network），SIP（Session Initiation Protocol），および，WebRTC（Web Real-Time Communication）である．提案方式の特徴は，コミュニケーションチャネルの接続に必要な情報の送信者はメッセージを暗号化しブロックチェーン・ネットワークにブロードキャストし，受信者は，自身が復号できるメッセージのみを処理することである．これにより，メッセージの受信者を秘匿してプライバシを保護している．この仕組みは，ブートストラップにのみ用いられ，アプリケーション・レベルの通信には高速な VPN，SIP，および，WebRTC が使われる．提案手法は，Ethereum のテストネットワークで動作している．実験により，基本的なメッセージの転送はブロックチェーンのブロック生成間隔程度で行えること，および，VPN，SIP，および，WebRTC のコミュニケーションチャネルの確立は，ブロックチェーンのオーバヘッドがなく行えることを確認している．

1. はじめに

社会やテクノロジーの発展に伴い，人々はインターネットによる利便性を享受している．例えば，テキストチャットやビデオ通話などのコミュニケーションツールや，Facebook や Twitter などの Social Networking Services（SNSs），および，Dropbox などのクラウドストレージサービスが広く普及しており，人々の生活に必要不可欠なものになっている．しかし，このようなツールは中央サーバに依存しているものが多く，特定の組織による検閲や規制に関する社会的な問題が指摘されている．加えて，中央サーバが単一障害点となり，データ損失といった技術的な問題が発生する可能性がある．これらの問題を解消するため，中央サーバなしで通信を行う分散型 SNS の研究と開発が行われている [1][2].

中央サーバなしで通信を行う場合，次のような問題を解決しなければならない．

通信相手の発見　中央サーバを用いる場合，ユーザは中央サーバを通じて通信するため，中央サーバを訪れることにより通信相手を容易に発見できる．しかし中央サーバを用いない場合，通信相手の変化する IP アドレスを追跡し続けなければならない．

通信相手の認証　中央サーバを用いる場合，ユーザ認証は中央サーバが行っている．中央サーバを用いない場合，ユーザは独自に通信相手を認証しなければならない．たとえば，ユーザは目的に応じて，WebRTC（Web Real-Time Communication）や VPN（Virtual Private Network），および SIP（Session Initiation Protocol）等の多種多様なコミュニケーションチャネルを利用する．しかし，チャネルごとに認証方法が大きく異なり，それに必要な情報の管理が煩雑になる．

本研究では，ブロックチェーン技術を用いることで，通信相手の発見と認証の問題を解決し，中央サーバに依存しないソーシャルコミュニケーションチャネルを実装し，分散型 SNS ををを実現することを目的とする．本研究で対象とするソーシャルコミュニケーションチャネルを以下に示す．

VPN　友人同士の PC の間で，LAN 用アプリケーションをインターネット越しに利用可能にする．

SIP　VPN の利用が難しい携帯端末に対応し，通話を利用可能にする．

WebRTC　Web ブラウザ間でのリアルタイム通信を可能

[1] 筑波大学
University of Tsukuba

にする．

これらのコミュニケーションチャネルにより，中央サーバに依存しない分散型 SNS を実装する．

ブロックキェーンに追加されたデータは誰でもアクセスできる．そのため，単純にブロックチェーンを用いるだけでは，プライバシが侵害される．本研究では，暗号アドレッシングを使うことでプライバシを保護する．暗号アドレッシングとは，述語アドレッシング（predicate addressing）[3] の一種であり，送信者はメッセージを暗号化して，全員が受け取れる状態でメッセージを放送する．受信者は自身が持つ鍵で暗号が解けたら受け取り，解けなかったら捨てる．暗号化通信の方法としては，1 対 1 では通常の公開鍵，1 対多の通信では属性ベース暗号を用いる．

ブロックチェーンを直接的にメッセージの送受信に用いると，ブロック生成のため時間がかかり，またコストも大きい．本研究ではブロックチェーンによる通信をブートストラップにのみ用い，アプリケーションレベルの通信には，通常の VPN，SIP，および，WebRTC を直接的に用いる．

2. 目標とするコミュニケーションチャネル

2.1 WebRTC

WebRTC は Web ブラウザの間で peer-to-peer（P2P）コミニュケーションチャネルを提供する．WebRTC の設計は，ネットワークアドレス変換（Network Address Translation: NAT）ルータを介したブラウザ間の通信に重点を置いている．これは，中央サーバの問題に対処するための有望なメカニズムである．ただし，WebRTC チャネルを確立するには，通常，Web サーバが必要である [4]．WebRTC を介して 2 つのブラウザがお互いに通信を開始する前に，短いメッセージを交換しなければならない．このプロセスはシグナリングと呼ばれ，通常，中央サーバを必要とする．

本研究の目的を達成するために，中央サーバを用いずにシグナリングを実装する必要がある．本研究では，暗号化されたメッセージをブロックチェーンネットワークにブロードキャストするスマートコントラクトを実装する．この仕組みの上位層として，WebRTC のシグナリングを実現するユニキャスト層を構築する．これについては，4 章で述べる．

2.2 VPN と SIP

VPN を利用することで，パブリックネットワークを介して，プライベートネットワークに直接接続しているかのようにデータを送受信することが可能になる．本研究では，VPN チャネルを確立するために，相手の IP アドレスを取得する必要がある．

SIP（Session Initiation Protocol）は，音声，ビデオ，メッセージングアプリケーションを含むリアルタイムセッションを開始，維持，終了するために使用されるシグナリングプロトコルである．本研究では，ユーザが自宅に SIP サーバを設置し，友人のサーバと接続する．2 台の SIP サーバが SIP 接続を確立するには，お互いに相手のサーバの IP アドレスが必要になる．

このように，VPN，および，SIP を利用するにあたり本研究の目的を達成するために，中央サーバを用いずに IP アドレスを友人に広告する必要がある．本研究では，暗号化されたメッセージをブロックチェーンネットワークにブロードキャストするスマートコントラクトを実装する．この仕組みの上位層として，マルチキャスト層を実装し，IP アドレスを友人に広告する．これについては，4 章で述べる．

3. ブロックチェーン技術

スマートコントラクトとは，ブロックチェーンネットワーク上で動作するプログラムである．Ethereum[5] はスマートコントラクトをサポートする代表的なブロックチェーンである．スマートコントラクトは信頼できる第三者を介さずに信頼できるトランザクションを実行する．スマートコントラクトの実行時間はブロック生成間隔に影響される．

Ethereum のブロックチェーンでは，ノードがスマートコントラクトにトランザクションメッセージを送信すると，誰でもメッセージの内容と送信者のウォレットアドレスを知ることができる．スマートコントラクトはイベントと呼ばれるメッセージを発することができるが，これはコントラクトのストレージよりもはるかに安価である．

本研究にでは，ブロック生成間隔が大きいこと，および，ストレージコストが高いため，アプリケーションレベルの通信チャネルとしてスマートコントラクトを利用しない．代わりに，スマートコントラクトを使って接続情報を交換する．その接続情報を利用してアプリケーション・レベルのコミュニケーションチャネル，すなわち WebRTC，VPN，SIP の各接続を確立する．例えば，アプリケーションレベルの WebRTC シグナリングは，スマートコントラクトによって確立されたベースの WebRTC チャネルを介して行われる．

4. ブロックチェーン技術を用いたソーシャル・コミュニケーションチャネルの確立

4.1 プライバシの問題

単純に Ethereum のスマートコントラクトにより，WebRTC のシグナリングを実現する方法が考えられる．このスマートコントラクトの動作の概要を，**図 1** に示す．このスマートコントラクトでは，2 人の友人が事前に Ethereum のウォレットアドレスを交換しておく．このスマートコントラクトは，`offer()` と `answer()` の 2 つの関数を持つ．送信者と受信者の 2 人のブラウザが WebRTC の接続を確立をする時，送信者は Web ブラウザの WebRTC の API である

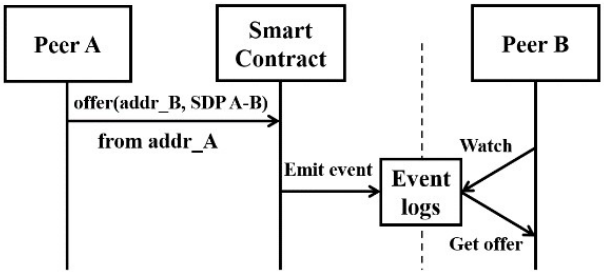

図 1 スマートコントラクトによる WebRTC シグナリング

ビデオ通話，チャット等	NFS，Web 等	移動端末間通話
アプリケーションレベル WebRTC 層	VPN	SIP
ベース WebRTC 層	IP アドレス管理層	
ユニキャスト層	マルチキャスト層	
イベントブロードキャスト層		

図 2 本研究で実装するソーシャルコミュニケーションチャネルの層構造

`RTCPeerConnection.createOffer()` を呼ぶことで，SDP (Session Description Protocol) offer メッセージを作成する．次に，送信者はスマートコントラクト関数 `offer()` を，受信者のウォレットアドレスと SDP offer メッセージの 2 つの引数と共に呼び出す．関数 `offer()` は，受信者のアドレスと SDP offer メッセージ，および送信者のアドレスを含むイベントを発行し，受信者はそのイベントログを監視する．受信者がフィルタを通じて新しい offer イベントを受信すると，そのイベントに含まれる送信者のアドレスと SDP offer メッセージを抽出する．そして，受信者は，受信した SDP offer メッセージを使用して，Web ブラウザの WebRTC の API である `RTCPeerConnection.createOffer()` を呼ぶことで，受信者は SDP answer メッセージを作成する．次に，受信者は，同じスマートコントラクトを使用して，この SDP answer メッセージを送信者に逆方向に送信する．最後に，送信者と受信者は，WebRTC 接続を確立する．

このシグナリングプロセスでは，Ethereum のブロックチェーン上のウォレットアドレスをユーザの ID として使用している．ブロックチェーンは透明性が高いため，誰でもイベントとして発せられたメッセージを見ることができる．トランザクションの送信者はガス代を支払わなければならないため，送信者を隠すことができない．中央管理者はこれらを監視することができ，ユーザのプライバシを侵害する可能性がある．

4.2 層構造によるプライバシを保護したソーシャルコミュニケーションチャネルの確立

本研究では，ユーザのプライバシを保護するために，**図 2** に示す層構造で，Ethereum ブロックチェーン上にソーシャルコミュニケーションチャネルを実装する．

```
contract Broadcast {
  event BCMessage(address indexed from,
    string content);
  function broadcast(string memory content) {
    emit BCMessage(msg.sender, content);
  }
}
```

図 3 暗号化されたメッセージをブロードキャストするスマートコントラクトの概要

交流する友人は事前に，ブロックチェーンのウォレットの公開鍵とそれから導出されるウォレットのアドレス，および，属性ベース暗号の公開鍵と属性秘密鍵を交換しているものとする．詳しくは，4.8 節で述べる．

図 2 のイベントブロードキャスト層では，送信者はスマートコントラクトのイベント送信機能を用いることで，短いメッセージをネットワーク全体にブロードキャストする．このとき，ブロードキャストされるメッセージは上位層で暗号化されているので，その内容と受信者が秘匿される．また，受信者はブロックチェーンの仕組みにより，メッセージの送信者を認証できる．

ユニキャスト層では，送信者は受信者の公開鍵を用いてメッセージを暗号化し，イベントブロードキャスト層の関数を呼び出してメッセージを送信する．受信者は相手から暗号化されたメッセージが到着した時，自身の秘密鍵によってメッセージを復号する．

ベース WebRTC 層では，ユニキャスト層の機能を用いて低速な WebRTC のシグナリングを行う．アプリケーションレベル WebRTC 層では，ベース WebRTC 層の機能を用いて高速な WebRTC のシグナリングを行う．

マルチキャスト層では，送信者は CP-ABE（Ciphertext-Policy Attribute-Based Encryption）[6] 公開鍵とポリシーでメッセージを暗号化し，イベントブロードキャスト層の関数を呼び出して送信する．受信者は，相手から暗号化されたメッセージが到着した時，CP-ABE 公開鍵と自身の CP-ABE 秘密鍵によってメッセージ復号する．

IP アドレス管理層では，IP アドレスを伝えたい友人を表現したポリシーを作成し，下位層のマルチキャスト層で友人に IP アドレスを広告する．そして，友人から受け取った受け取った IP アドレスをキャッシュとして保存する．VPN や SIP では，この IP アドレス管理層の機能を利用して，友人の IP アドレスを得て，コミュニケーションチャネルを確立する．

以下の節では，個々の層について API と実装を述べる．

4.3 ブロードキャスト層

ブロードキャスト層は，Solidity によるスマートコントラクトと，そのラッパーである JavaScript のコードから構成される．**図 3** にスマートコントラクトの概要を示す．

表 1 ブロードキャスト層の関数

関数名	説明
broadcast(encrypted)	引数として，暗号化されたメッセージ encrypted を受け取り，それをスマートコントラクトを用いてイベントを発行する．
broadcast_watch(addrs,handler)	引数として，友人のアドレスのリスト addrs とイベントハンドラ関数 handler を受け取る．スマートコントラクトを用いて，友人からのイベントを監視する．イベントが到着すると，ハンドラ関数がコールバックされる．

このスマートコントラクトは，関数 broadcast() を持つ．この関数は，メッセージである content を明示的に引数として受け取る．受け取るメッセージは，上位層で暗号化される．また，この関数は，トランザクションの送信者のアドレスを含む暗黙の引数 msg も使用する．この関数は，トランザクションの送信者のアドレスと暗号化されたメッセージから構成されるイベント BCMessage を発行する．

表 1 にブロードキャスト層の JavaScript 関数を示す．この層は，スマートコントラクトのイベントログと対話するための 2 つの関数 broadcast() と，関数 broadcast_watch() を提供する．

4.4 ユニキャスト層

ユニキャスト層では，表 2 に示す関数を提供する．関数 unicast_send() は，受信者のウォレットアドレスを受け取り，その受信者にメッセージを送信する．引数 app は，TCP/IP のポート番号のように，アプリケーションを区別するためのものである．この関数は，受信者の公開鍵でメッセージを暗号化し，イベントブロードキャスト層の関数 broadcast() を用いて送信する．関数 unicast_receive() は友人のウォレットアドレスのリストを受け取り，そのアドレスからのメッセージを受信する．メッセージが到着し，受信者の秘密鍵で復号化できると，復号化されたメッセージとともにハンドラ関数がコールバックされる．この関数は，イベントを監視するために，イベントブロードキャスト層の関数 broadcast_watch() を呼び出す．

4.5 ベース WebRTC 層

ベース WebRTC 層では，ユニキャスト層を利用して WebRTC のシグナリングを行い，友人のノードとの間でデータチャネルを次のように確立する．まず，送信者は SDP offer メッセージを作成し，関数 unicast_send() を呼び出すことで，受信者のウォレットアドレスと共に作成した offer メッセージを送信する．次に，受信者は，関数 unicast_receive() に登録されているハンドラ関数を使用して，この offer メッセージを受け取る．次に，受信者は，SDP answer メッセージを作成し，関数 unicast_send() を呼び出して，answer メッセージを送信する．最後に，送信者と受信者は，WebRTC データチャネルを確立する．

4.6 マルチキャスト層

マルチキャスト層では，表 3 に示す関数を提供する．関数 multicast_send() は，複数の友人のノードにメッセージをマルチキャストする．引数として，CP-ABE のポリシー policy，アプリケーションの識別子 app，および送信したい平文のメッセージ msg を受け取る．引数 app は，TCP/IP のポート番号のように，アプリケーションを区別するためのものである．この関数は，ポリシーと CP-ABE 公開鍵でメッセージを暗号化し，イベントブロードキャスト層の関数 broadcast() を用いて送信する．関数 multicast_receive() は友人のウォレットアドレスのリストを受け取り，そのアドレスからのメッセージを受信する．メッセージが到着し，受信者の CP-ABE 公開鍵と CP-ABE 秘密鍵で復号できると，復号されたメッセージとともにハンドラ関数がコールバックされる．この関数は，イベントを監視するために，イベントブロードキャスト層の関数 broadcast_watch() を呼び出す．

4.7 IP アドレス管理層

IP アドレス管理層は，マルチキャスト層の機能を用いて交流する友人に自身の IP アドレスを広告する．このとき，ポリシーを指定することで，ポリシーに合致する鍵を持つ特定のユーザのみが IP アドレスを復号できる．たとえば**図 4** に示すように，Alice が IP アドレスを公開し，Bob が Alice を発見する場合を考える．Alice は Bob に属性 bob が付与された秘密鍵を，Charlie に属性 charlie が付与された秘密鍵を事前に渡しておく．Alice が IP アドレスを公開するとき，ポリシーとして"bob OR david"を指定し，マルチキャスト層の機能を用いて暗号化された IP アドレスをブロードキャストする．Bob が持つ秘密鍵は Alice が指定したポリシーに対応するため，メッセージを復号して IP アドレスを得ることにより Alice を発見できる．Charlie が持つ秘密鍵はポリシーに対応しないため，メッセージを復号できず Alice を発見できない．

4.8 初期設定

各ユーザは，次の情報を自身の PC に保存する．

- Ethereum のウォレット，および，それに対応した秘密鍵，公開鍵，およびウォレットのアドレス（公開鍵から派生）．

表 2　ユニキャスト層の関数

関数名	説明
unicast_send(addr,app,message)	引数としてアドレス addr，アプリケーション識別子 app，メッセージ message を受け取る．受信者の公開鍵でメッセージを暗号化し，暗号化したメッセージを broadcast() を利用して送信する．
unicast_receive(addrs,handler)	引数として，友人のアドレスのリスト addrs とイベントハンドラ関数 handler を受け取る．関数 broadcast_watch() を呼び出して，友人からのメッセージを監視する．メッセージが到着し，それが受信者の秘密鍵で復号できると，ハンドラ関数がコールバックされる．

表 3　マルチキャスト層の関数

関数名	説明
multicast_send(policy,app,msg)	引数としてポリシー policy，アプリケーション識別子 app，メッセージ message を受け取る．ポリシーと受信者の公開鍵でメッセージを暗号化し，暗号化したメッセージを broadcast() を利用してネットワークにマルチキャストする．
multicast_receive(addrs,handler)	引数として，友人のアドレスのリスト addrs とイベントハンドラ関数 handler を受け取る．関数 broadcast_watch() を呼び出して，友人からのメッセージを監視する．メッセージが到着し，それが受信者の公開鍵と秘密鍵で復号できると，ハンドラ関数がコールバックされる．

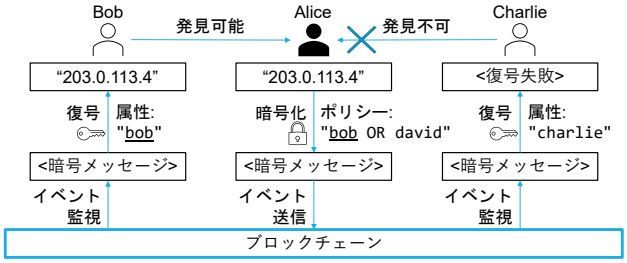

図 4　ブロックチェーン技術を用いた通信相手の発見

- 属性ベース暗号のマスター秘密鍵，および，公開鍵．本研究では，属性ベース暗号の一つである CP-ABE（Ciphertext-Policy Attribute-Based Encryption）[6] を用いる．

ユーザ同士が交流を開始する時，次のような情報を携帯端末を利用して対面で交換する．

- ウォレットのアドレス（ID）
- ウォレットの公開鍵
- CP-ABE の公開鍵
- CP-ABE のマスタ秘密鍵から生成した交流相手の秘密鍵
- チャネル固有の認証に用いる情報

このように認証情報を対面で交換することで，認証情報が交流したい相手のものであると担保できる．各ユーザは，このような静的な情報をそれぞれの PC 上で動作するデータベースに保存する．データベースでは，ID や表示名からこれらの情報を引き出せるようにする．データベースには，IP アドレスのように動的に変化するものは保存せず，別途管理する．

5. ソーシャルコミュニケーションチャネルの実装

本章では，ブロックチェーン技術を用いた 3 つのソーシャルコミュニケーションチャネルの実装について述べる．

本研究では，様々なアプリケーションが利用する多種多様なコミュニケーションチャネルを利用可能にする．本研究では，その中でも IP アドレスがあれば動作するものをサポートする．具体的には，次のようなコミュニケーションチャネルとアプリケーションを利用可能にする．

- VPN: NFS 等の LAN 用のアプリケーション，Web 等のイントラネットのアプリケーション．
- SIP: 通話やチャット等，携帯端末でも利用可能なアプリケーション．

これらのコミュニケーションチャネルを確立する時に必要となる IP アドレスは，4.7 節で述べた IP アドレス管理層より得られたものを用いる．

既存のソフトウェアを利用し，VPN や SIP 等の個々のコミュニケーションチャネルを確立させるためには，ユーザは複雑な設定を行わなければならない．本研究では，設定を自動化することでユーザの手間を無くす．具体的には，変化する友人の IP アドレスに追随し，各チャネルの設定を自動的に更新する．

VPN と SIP に加えて本研究ではブラウザ間のコミュニケーションチャネル WebRPC を利用可能にする．WebRTC では，複雑なシグナリングを必要とするが，これは 4.4 節で述べたユニキャスト層の機能を用いて実現する．

5.1 ソーシャル VPN

ソーシャル VPN とは，友人の PC との間でインターネット越しに安全な通信路を提供する VPN である．これにより，友人との交流を既存の LAN 用アプリケーションを用いて容易かつ安全に行うことが可能になる．

4.7 節で述べた IP アドレス管理層が提供する IP アドレスは通信相手のものであると保証される．しかしながら，既存の VPN ソフトウェアにおいては IP アドレスだけでは通信相手を認証するためには不十分である．

本研究では，strongSwan[*1]を用いてソーシャル VPN を実装している．通信相手の発見は，4.7 節で述べた方法で行う．通信相手の認証は，ブロックチェーンのウォレットアドレス，および strongSwan で用いる公開鍵暗号系に基づく証明書を 2 つ組み合わせて実装する．

本研究では，ユーザ名に基づくドメインを名を解決できる DNS サーバを各 PC で実行する．この DNS サーバは友人の名前を含むドメイン名（たとえば alice.socialvpn）の解決が要求されると，ドメイン名からユーザ名を抽出し，データベースからユーザ名に対応するウォレットアドレスを取得する．そして，ウォレットアドレスをもとに VPN セッションを特定し，接続先の IP アドレスを取得する．最後に，取得した IP アドレスを名前解決の結果として返す．

5.2 ソーシャル SIP

ソーシャル VPN では，ユーザは各自の PC で VPN サーバや DNS サーバを立ち上げる必要がある．したがって，サーバ機能を有することが難しい携帯端末では利用が困難である．携帯端末におけるコミュニケーションツールとしては，音声通話やビデオ通話に対する要求が高い．これらのツールの中で，SIP を利用するものが数多く存在する．そこで本研究では，携帯端末における友人間のコミュニケーションチャネルを実装するにあたり，SIP をサポートすることにした．この機能を本研究では，**ソーシャル SIP** と呼ぶことにする．

本研究では，SIP サーバである Asterisk[*2]を利用し，ソーシャル SIP の実装する．各ユーザは，自宅の PC で Asterisk サーバ，およびユーザ情報を管理するためのデータベースを実行する．各ユーザは，携帯端末で SIP のクライアントを実行し，自宅のサーバに接続して利用する．自宅のサーバの発見，および，友人の発見に 4.7 節で述べた IP アドレス管理層の機能を用いる．

自宅の PC で動作する Asterisk には，次の設定を行う．

- 自分の携帯端末がログインするためのユーザ名とパスワード．
- 自分の携帯端末の内線番号．ランダムに生成する．

ユーザが交流を開始する時，以下のチャネル固有のデータを交換し，自宅のサーバのデータベースに保存する．

- 自身のサーバにおける友人を識別するための内線番号．ランダムに生成する．
- 自宅のサーバと友人のサーバ間における認証に用いるパスワード

本研究では，4.7 節で述べた方法で通信相手の発見，すなわち，友人の IP アドレスの変更を常に追跡する．IP アドレスが変更されるたびに Asterisk の設定ファイルを生成する．**図 5** および**図 6** に生成されたファイルの例を示す．

[*1] https://www.strongswan.org/
[*2] https://www.asterisk.org/

```
[sv2_in]
exten => 1101,1,Dial(SIP/3001)
[phones]
exten => 2001,1,Dial(SIP/sv2/1102)
```

図 5 自動生成した Asterisk の設定ファイル extensions.conf の重要な部分

```
[general]
allowguest=no
register => sv1:passwd1@203.0.113.4/sv2
[sv2]
secret=passwd2
context=sv2_in
[3001]
context=phones
secret=pass3
```

図 6 自動生成した Asterisk の設定ファイル sip.conf の重要な部分

図 5 における内線番号は以下のようになる．

- 1101: 友人のサーバが自身の携帯端末を識別するための内線番号
- 3001: 自身の携帯端末に割り当てる内線番号
- 2001: 自身のサーバにおいて友人を識別するための内線番号
- 1102: 自身のサーバが友人の携帯端末を識別するための内線番号

このうち，1101 は，1102 はユーザが交流を開始するときに交換する内線番号である．

図 6 で，203.0.113.4 は友人の Asterisk サーバの IP アドレスである．この設定ファイルにおける register はサーバ間でパスワードによる認証を行うことを意味する．すなわちパスワード認証を行っていないサーバは未知のサーバとして通信を拒否する．この設定による認証は，サーバ 1 台につき 1 ユーザと限定することでチャネルレベルのユーザ認証が行われる．

各ユーザは携帯端末で SIP のツール，たとえば，Zoiper[*3]を実行し，それぞれ自身のログイン用のユーザ名とパスワードを使って，それぞれの自宅の SIP サーバにログインする．携帯端末の SIP ツールで通信を行いたい場合，自身のサーバにおける友人を識別するための内線番号を指定して，発信の操作を行う．すると，以下のように SIP の通信を行えるようになる．

(1) 発信側の SIP サーバは，自身のサーバにおける友人を識別するための内線番号を用いて，着信側の SIP サーバに接続要求を送る．
(2) 着信側の SIP サーバは，接続要求を受け付け，携帯端末を呼び出す．

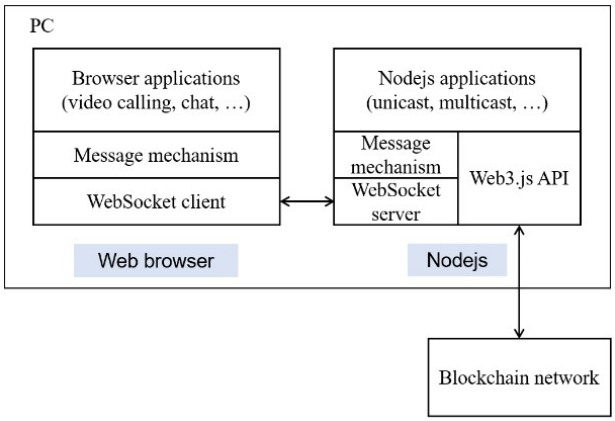

図 7 ブラウザと Node.js サーバの通信メカニズム

5.3 ソーシャル WebRTC

ソーシャル WebRTC とは，2 人の友人がそれぞれ Web ブラウザを実行し，それらのブラウザの間で確立される WebRTC のコミュニケーションチャネルである．そのアプリケーションとしては，ビデオ通話やテキストチャットがある．WebRTC を利用したアプリケーションは，広く普及しているが，それらは中央サーバを利用して WebRTC のシグナリングを行っている．本研究では，中央サーバを使わずにソーシャル WebRTC を実現する．

WebRTC のシグナリングは，4.4 節で述べたユニキャスト層の機能を利用しても実現できる．しかしながら，これは低速であり，一般的な利用に適さない．たとえば，ユーザがビデオ通話を行いたいと思って，呼び出しボタンを押したとする．そこからユニキャスト層の機能でメッセージを送受信すると，完了するまで最低でも Ethereum のブロック生成時間の 2 倍（約 30 秒）も必要になる．

本研究では，アプリケーションレベルの WebRTC のシグナリングを，ユニキャスト層の機能ではなく，ベース WebRTC 層の機能を利用して行う．ベース WebRTC 層は，Web ブラウザは起動時にコンタクスリストにある友人の Web ブラウザとの間で WebRTC データチャネルを接続する．このデータチャネルは，ビデオや音声ではなく，汎用のデータを送受信する機能を提供する．

アプリケーションレベルの WebRTC は，ベース WebRTC 層が提供するデータチャネルを利用してシグナリングを行う．例えば，ビデオ通話アプリケーションが実行されると，アプリケーションレベル WebRTC 層では，ベース WebRTC 層のデータチャネルを使って SDP メッセージを送受信する．

Web ブラウザで Ethereum の API を直接利用することは容易ではない．そこで本研究では，ブラウザを実行しているノードで Node.js サーバを実行し，そのサーバで Ethereum ブロックチェーンと通信し，Web ブラウザはその Node.js サーバの機能を利用する．

*3 `https://www.zoiper.com/`

本研究では，JavaScript の関数を，Web ブラウザで実行される関数と，Node.js で実行される関数に分割して実装する．**図 7** に示すように，ブラウザと Node.js サーバは同一の PC 上で動作する．Node.js サーバはブロックチェーンネットワークと通信するための Web3 ライブラリ?が含まれている．ブラウザと Node.js サーバはそれぞれが起動するときに WebSockert チャネルが確立される．WebSocket 層の上に，ブラウザとサーバの両方に Message mechanism を実装する．本研究では，Remote Procedure Calls（RPCs）におけるスタブの概念を利用する．Message mechanism では，以下に示す関数を提供する．なお，関数名における”bnm”は，”browser-nodejs messaging”を表す．

`bnm_setup_node(port, handler)` Node.js 側で WebSocket の接続を作成し，もう一方のノードからイベントを受信するハンドラ関数を登録する．

`bnm_setup_browser(url, handler)` ブラウザ側で WebSocket の接続を作成し，もう一方のノードからイベントを受信するハンドラ関数を登録する．

`bnm_send_event(app, event)` イベント `event` を引数に受け取り，もう一方のノードに送信する．

現在，本研究における分散型 SNS では，ユニキャスト層でプログラムを分割している．Node.js サーバでは，関数 `unicast_send()` および関数 `unicast_receive()` を実装している．Web ブラウザでは，これらの関数のスタブを実装して，Node.js サーバと通信する．ブラウザで関数 `unicast_send()` および関数 `unicast_receive()` を呼び出すと，関数 `bnm_send_event()` が呼び出される．関数 `bnm_send_event()` の引数には，関数名と引数を指定する．Node.js サーバでは，Web ブラウザからのイベントは，関数名と引数に従って処理される．

5.4 ユーザ情報交換プログラムの設計

4.8 節で述べたように，交流を開始する友人は，ウォレットアドレス，証明書，公開鍵をしなければならない．そこで本研究では，携帯端末で動作するアプリケーションを用いてこの交換を容易にする．具体的には，**図 8** に示すように JSON 形式のユーザ情報交換フォーマットを定め，携帯端末を用いてユーザ情報を対面で交換する．ユーザ情報は次の要素から構成される．

- id: ウォレットアドレス
- unicast-key: ウォレットの公開鍵
- date: 交換する日時
- vcard: vCard に準拠した送信者を識別するための情報
- cpabe: CP-ABE の公開鍵および秘密鍵
- channel: 各チャネルの種類名および固有の認証情報

ユーザ情報を交換する際，unicast-key，cpabe，および channel は認証情報を含むため，相手毎に変化する．たとえば，図 8 の例では SIP のための認証情報として内線番

```
{
  "id": "0x8e55B2613bF64ed4920D0345...",
  "date": "2021-01-01T06:00:00Z",
  "vcard": {
    "fn": "Tarou Suzuki",
    ...
  },
  "unicast-key": {
    "public": "03be57cd393ac38b8a3e..."
  },
  "cpabe": {
    "public": "AAABZ3R5cGUgYQpxIDg3...",
    "secret": "AAAAgHjvO/OV3k8BIl/s..."
  },
  "channel": [
    {
      "type": "sip",
      "auth": {
        "number": "1101",
        "secret": "passwd"
      }
    },
    ...
  ]
}
```

図 8 交流開始時に携帯端末で交換するユーザ情報の例

号と認証用のパスワードが含まれている．この携帯端末で動作するユーザ情報交換プログラムは，このような変化する情報を生成し，交流相手に送信する．そして，交流相手から受け取った情報と生成した情報を自宅のサーバに保存する．

6. 評価

6.1 機能評価

本研究の目的は，中央サーバ無しでソーシャル・コミュニケーション・チャネル実現することである．本研究で対象とするソーシャルコミュニケーションチャネルは，VPN，SIP，および WebRTC である．これらのチャネルにおいて，中央サーバを用いずに，通信相手の発見と通信相手の認証を実現する必要がある．

VPN と SIP で通信相手の発見とは，通信相手の IP アドレスを知ることである．各ユーザは自分の IP アドレスをマルチキャスト層で CP-ABE により暗号化し，ブロックチェーンネットワーク上にブロードキャストしている．それを受信した側は，メッセージを復号し，IP アドレスをキャッシュする．以上のことから，中央サーバ無しで通信したい相手の IP アドレスを知ることができたと言える．WebRTC で通信相手の発見とは，通信相手を指定して SDP メッセージを送り届けることである．SDP メッセージをユニキャスト層で受信者のウォレットの公開鍵により暗号化し，ブロックチェーンネットワークを介して送信している．各ユーザはそれを受信した側は，自身のウォレットの秘密鍵でメッセージを復号し，SDP メッセージを受け取る．このように，中央サーバ無しで通信したい相手を発見することができたと言える．

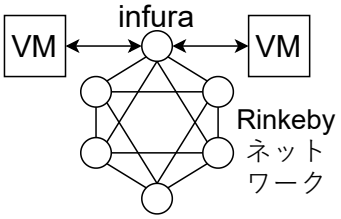

図 9 実験環境の概要

本研究では中央サーバを用いずに通信相手を認証する必要がある．ブロックチェーンに送信されるメッセージ (イベント) では，発信者は認証されている．メッセージの受信者は，自分が復号できるメッセージしか受信できない．これに加えて，VPN と SIP では，チャネル固有の認証に用いる情報として，それぞれ証明書とパスワードを利用して認証している．よって，中央サーバを用いずに通信相手を認証することが実現された．

VPN，SIP，および WebRTC のいずれにおいても中央サーバ無しで通信相手の発見と認証が実現できており，本研究の目的が達成された．

6.2 プライバシの評価

ブロックチェーン上のデータは誰でも監視できる．本研究では，ユニキャスト層およびマルチキャスト層により，ブロックチェーン上のデータはすべて暗号化されている．この時，トランザクションを実行するため送信者を隠すことはできないが，受信者はわからない．必要ならダミーのメッセージを送信することもできる．したがって，メッセージの内容と共に，誰が誰にメッセージを送信したかは秘匿され，その意味でプライバシは保護される．

6.3 性能

提案手法は，Ethereum のテストネットワークで動作している．この節では，その実験結果を示す．

ある調査 [7] では，典型的な Faccebook のユーザは，155 名と交流していると報告している．別の調査 [8] では，典型的な Twitter のユーザは，707 名のフォロワーを持っていると報告している．本研究では，分散型 SNS として，1 人のユーザが最大 1000 名の友人を交流すると想定する．この節の実験では，友人の数を 1000 名まで増やした時の性能を測定する．

ソーシャル VPN，および，ソーシャル SIP においては，コミュニケーションチャネルを確立する時に友人の IP アドレスが必要になる．これは，4.7 節で述べた IP アドレス管理層の働きにより，ローカルにキャッシュされている．したがって，コミュニケーションチャネルを確立するために要する時間は，通常の VPN，および，SIP と同一であ

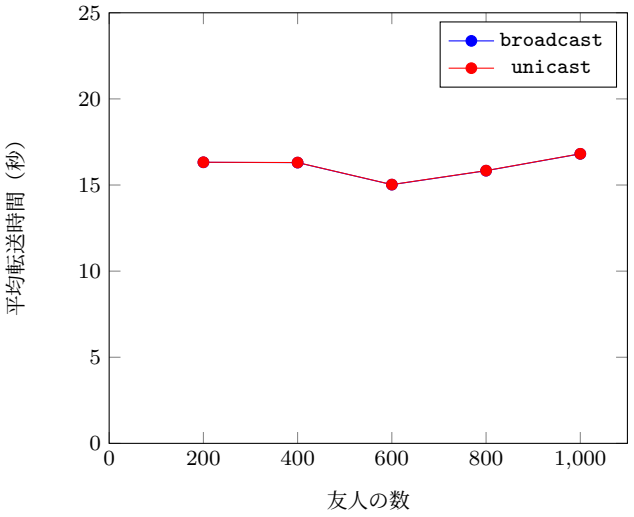

図 10　友人の数とメッセージの転送時間の関係

り，ブロックチェーンを利用したことによるオーバヘッドは生じない．したがって，この節では，イベントブロードキャスト層におけるメッセージの送受信に要する時間を測定する．また，WebRTC のシグナリングに要する時間も測定する．

実験環境を仮想計算機を使って構築した．その概要を，**図 9** に示す．また，実験に用いたソフトウェアバージョンを以下に示す．

- ホスト OS Ubuntu 20.04，CPU core i7-5820K，メモリ 32GB
- Node.js 14.17.5
- Web ブラウザ Firefox 89.0
- 仮想マシンモニタ VirtualBox 6.1.16，ゲスト OS Ubuntu 20.04，CPU 1 コア割当て，メモリ 4GB

各仮想計算機では，5.3 節で述べたように，Web ブラウザと Node.js サーバが動作している．Node.js では，Web3 ライブラリを用いて，Ethereum のホスティング・サービス infura[*4] を経由して Ethereum のテストネットワーク Rinkeby[*5] へ接続している．Rinkeby のブロックの生成間隔は，約 15 秒である．仮想計算機は，学内ネットワーク，および，SINET を通じてインターネットに接続されている

6.3.1　基本性能

次の層について，メッセージを送信してから受信するまでの実行時間を測定した．

- イベントブロードキャスト層
- ユニキャスト層

友人が最大 1000 名いることを想定して，メッセージの送信に必要な時間を測定した．受信側では，4.3 節，および，4.4 節の機能を利用して，友人の数だけ Ethereum のアドレスを指定してメッセージを監視するようにした．その状態で，あるノードからメッセージを送信して，別のノードでメッセージを受信するまでの時間を測定した．実験は，10 回繰り返し，その平均時間を求めた．

結果を**図 10** に示す．横軸は，友人の数であり，縦軸は，メッセージの転送時間である．メッセージの転送には，イベントブロードキャスト層でもユニキャスト層でも約 17 秒要している．これは，Ethereum のテストネットワーク Rinkby のブロック生成時間 15 秒より少し長い．イベントブロードキャスト層とユニキャスト層の転送時間がほとんど重なっている．その差は 5 ミリ秒しかなかった．このことから，暗号化のオーバヘッドはブロックの生成間隔に比べて無視できることがわかる．また，友人の数が 1000 名まで増えてたとしても，メッセージの転送に要する時間には影響がないことがわかった．

6.3.2　WebRTC のシグナリング実行時間

この実験環境で，WebRTC のシグナリング実行時間を計測した．その結果を，表 4 に示す．

ベース WebRTC 層でのシグナリング実行時間は，約 26 秒である．ベース WebRTC 層でのシグナリングは Web ブラウザの起動時に一度だけ行うだけなので，アプリケーションの実行時間にには影響しない．アプリケーションレベル WebRTC 層でのシグナリング実行時間は，約 2.7 秒であり，ビデオ通話などのアプリケーションを実用的な時間で利用できる．

6.4　ガス代

スマートコントラクトの関数呼び出しは，トランザクションによりブロックチェーン上のデータを変更するものと，単にデータを参照するだけで，トランザクションを伴わないものに分類される．トランザクションを伴わないもの，すなわち，ブロックチェーン上のデータをアクセスするだけの関数の呼び出しは，その遅延を待つ事なくローカルでも実行可能である．ただし，ブロックチェーン上のデータを全て自ら保持するか，自分が保持していないデータを他のノードから取得する必要がある．トランザクションを伴う関数呼び出しは，最低でもブロックが承認されるための時間（13 秒程度）の遅延が生じる．

実際には，トランザクションの実行時間は Ethereum における手数料に相当するガス価格（gas price）に影響される．ガス価格は，トランザクションを行わせたい利用者が設定するもので，高く設定すると，Ethereum ノード（マイナー）が得る手数料が上がるので，トランザクションと

表 4　WebRTC のシグナリング実行時間

実行するレイヤ	実行時間（秒）
ベース WebRTC 層	25.87
アプリケーションレベル WebRTC 層	2.66

[*4] https://infura.io/
[*5] https://www.rinkeby.io/

して早く処理される．低く設定すると，トランザクションとしてなかなか実行されない．2021 年 11 月現在，ガスの価格は高騰しており，130 gwei 支払うと 30 秒以内に処理されるが，115 gwei では 5 分かかるとしている [*6]．なお，1gwei=10^{-9}ETH で計算される．

イベントブロードキャスト層で，トランザクションを必要とする操作は表 1 のうち，`broadcast()` 関数だけである．`broadcast_watch()` はデータを取得するだけなので，トランザクションの手数料は不要である．表 5 に `broadcast()` の実行に必要はガス数を示す．ガス数は，Ethereum の geth における estimateGas 関数により得られた値である．これは，EVM (Ethereum Virtual Machine) の命令実行に応じたコストの和である．表 5 の日本円は，ガス価格を 115 gwei，1ETH を 520000 円 として日本円に換算したものである[*7]．このように，`broadcast()` 関数の実行は，現在のガス価格では日常的な利用には適していない領域にある．

Ethereum におけるガス価格の上昇は，本研究におけるソーシャルコミュニケーションチャネルだけでなく，Ethereum 上で動作する全ての分散アプリケーション (Distributed Applications, DApps) に共通の問題である．今後この問題を解決するために，Ethereum のスケーラビリティを改善する技術が開発されることが期待されている [9][10][11]．あるいは，本研究のソーシャルコミュニケーションチャネルが利用するブロックチェーン技術を，Ethereum とは違う分散アプリケーションに適したものに移行する必要がある．

7. 関連研究

Klukovicr らの研究 [12] は，携帯端末の利用も考慮したプライバシを保護する分散型 SNS を提案している．これはモバイルデバイスからサーバを経由した通信と，端末同士の直接通信をサポートしている．この研究では外部のクラウドストレージサービスを必要としているが，本研究では必要としない．

Twister [13] は，Twitter をモデルとした分散型マイクロブログソーシャルネットワークである．この研究では，ユーザ登録にブロックチェーンを利用し，ピアとコンテンツのルーティングとインデックス作成に分散ハッシュテーブル（DHT）を利用する．SAND [14] は，ソーシャル VPN を利用して信頼できる友人同士のピアツーピア接続を確立し，VPN 接続を通じてソーシャルメディアのコンテンツを配信する．本研究では，ブロックチェーン技術を用いて，WebRTC，VPN，SIP のピアツーピア・チャネルを確立する．

Blockstack [15] は，ブロックチェーン技術で保護されたネーミングとストレージを提供する．ユーザはデータに対して人間にとって意味のある名前を安全に関連付けることができる．名前を登録した特定の秘密鍵の所有者だけが，名前と値のペアを書き込んだり更新したりすることができる．本研究では，暗号アドレッシングを使用してユーザのプライバシを保護している．

BCOSN [16] は，ブロックチェーンベースの分散型 SNS である．この研究では，CP-ABE を用いることで，暗号化とアクセス制御を実装している．また，ブロックチェーン上の高価なストレージにユーザデータを保持する．本研究では，ブロックチェーンの安価なイベント通知機能を用いている．

表 5 `broadcast()` の実行に必要なトランザクションコスト

メッセージ	使用ガス数	gwei	日本円
空文字列	23507	2703305	1406
SDP メッセージ（双方向）	75842	8721830	4535
IP アドレス	49484	5690660	2959

[*6] https://etherscan.io/gastracker
[*7] https://etherscan.io/chart/tx

8. おわりに

本研究では，ブロックチェーン技術を用いて 3 つのソーシャルコミュニケーションチャネル，VPN (Virtual Private Network)，SIP（Session Initiation Protocol），および，WebRTC（Web Real-Time Communication） を実装している．その特徴は，送信者はメッセージを暗号化しブロックチェーン・ネットワークにブロードキャストし，受信者は，自身が復号できるメッセージのみを処理することで，メッセージの受信者を秘匿してプライバシを保護している点にある．この仕組みは，ブートストラップにのみ用いられ，アプリケーション・レベルの通信には高速な VPN，SIP，および，WebRTC が使われる．SIP では，友人の IP アドレスの変化に応じて自宅で動作しているサーバ Asterisk の設定ファイルが自働的に更新される．WebRTC では，PC 上で Web ブラウザと共に Node.js を動作させ，ブロックチェーン・ネットワークとの通信を実装している．

提案手法は，Ethereum のテストネットワークで動作している．友人の数が 1000 名程度の場合，基本的なメッセージの転送はブロックチェーンのブロック生成間隔程度で行える．アプリケーション・レベルの WebRTC のシグナリングは，ブート時に作成されたデータチャネルを用いてなされるので，2.7 秒程度で完了する．変化する友人の IP アドレスは，ローカルにキャッシュされるので，それを利用した VPN および SIP のセッションの確立にはブロックチェーンを利用したことによる追加の時間はかからない．これらの実験結果により，提案手法には有用性があると言える．

提案手法では，SNS の交流開始時に，Ethereum のウォ

レットのアドレスや，属性ベース暗号の属性秘密鍵，VPNの公開鍵等，様々な情報を交換する必要がある．現在，この情報交換を簡単に行えるようにするため，携帯端末で動作するアプリケーションを開発している．今後の課題は，このアプリケーションを完成させ，コンタクスリストの管理と連携させることである．また，Ethereum 以外の，低コストでスマートコントラクトが実行できるブロックチェーン技術を利用して提案手法を実装したいと考えている．

参考文献

[1] Datta, A., Buchegger, S., Vu, L.-H., Strufe, T. and Rzadca, K.: Decentralized Online Social Networks, *Handbook of Social Network Technologies and Applications*, pp. 349–378 (2010).

[2] Schwittmann, L., Wander, M., Boelmann, C. and Weis, T.: Privacy Preservation in Decentralized Online Social Networks, *IEEE Internet Computing*, Vol. 18, No. 2, pp. 16–23 (2014).

[3] Tanenbaum, A. S.: *Distributed Operating Systems*, Prentice Hall (1994).

[4] Sredojev, B., Samardzija, D. and Posarac, D.: WebRTC technology overview and signaling solution design and implementation, *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1006–1009 (2015).

[5] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger.

[6] Bethencourt, J., Sahai, A. and Waters, B.: Ciphertext-Policy Attribute-Based Encryption, *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 321–334 (2007).

[7] Agency, O.: Facebook by the Numbers: Stats Demographics & Fun Facts, (online), available from ⟨`https://www.omnicoreagency.com/facebook-statistics/`⟩ (accessed 2021-11-06).

[8] KickFactory: The Average Twitter User Now has 707 Followers, (online), available from ⟨`https://www.omnicoreagency.com/facebook-statistics/`⟩ (accessed 2021-11-06).

[9] Ethereum Foundation: The Eth2 upgrades, Ethereum Foundation (online), available from ⟨`https://ethereum.org/en/eth2/`⟩ (accessed 2021-11-06).

[10] Ethereum Foundation: Ethereum Proof-of-Stake Consensus Specifications, GitHub (online), available from ⟨`https://github.com/ethereum/consensus-specs`⟩ (accessed 2021-11-06).

[11] Ethereum Foundation: Scaling, Ethereum Foundation (online), available from ⟨`https://ethereum.org/en/developers/docs/scaling/`⟩ (accessed 2021-11-06).

[12] Klukovich, E., Erdin, E. and Gunes, M. H.: POSN: A privacy preserving decentralized social network app for mobile devices, *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 1426–1429 (2016).

[13] Freitas, M.: Twister: The Development of a Peer-to-Peer Microblogging Platform, *International Journal of Parallel, Emergent and Distributed Systems*, Vol. 31, No. 1, p. 20–33 (2016).

[14] Ding, D., Conti, M. and Figueiredo, R.: SAND: Social-aware, network-failure resilient, and decentralized microblogging system, *Future Generation Computer Systems*, Vol. 93, pp. 637–650 (2019).

[15] Ali, M., Nelson, J., Shea, R. and Freedman, M. J.: Blockstack: A Global Naming and Storage System Secured by Blockchains, USENIX Annual Technical Conference, pp. 181–194 (2016).

[16] Jiang, L. and Zhang, X.: BCOSN: A Blockchain-Based Decentralized Online Social Network, *IEEE Transactions on Computational Social Systems*, Vol. 6, No. 6, pp. 1454–1466 (2019).