Distributed Social Network in *Browsers*

Yang Ji

Michael Puckett

Problem

- Large scale personal data abuse and manipulation raises question to the cloud based architecture in social networks.
  - Sensitive personal information is sold for profit and for fun.
  - Evidence shows even encrypted data at the cloud can be mined or has hidden backdoors.

Motivation

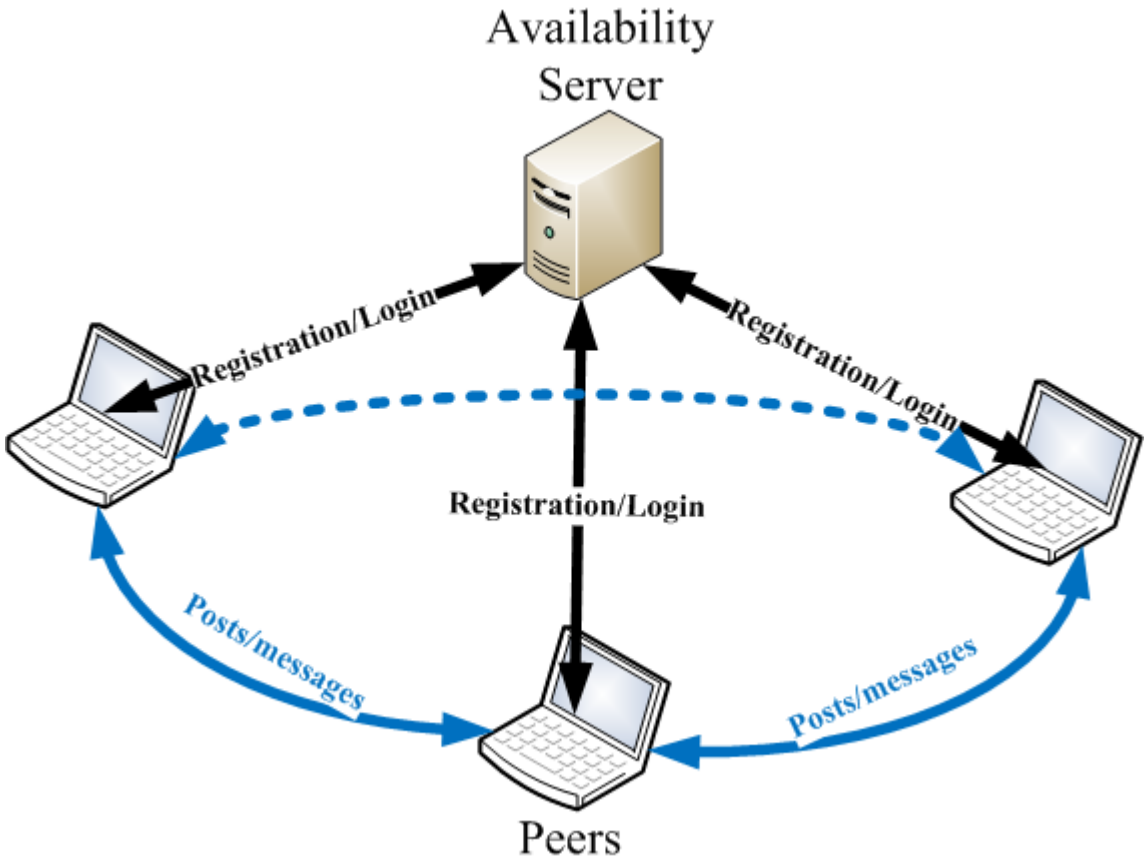
- Users desire to control their data instead of trusting the service provider.
  - Even existing sub-distributed social network like Diaspora* has regional centralized servers.
  - We deliver *a pure distributed peer-to-peer architecture*

- WebRTC enables peer-to-peer communication in browsers.
  - Great opportunity to bring DSN to browsers

Threat model

- Centralized server (AVL server) :
  - has access to all the users' public key, user name, peer id.
  - has NO access to any user's private key.
- A peer :
  - has access to any other's public key, username/peerid, anyone's encrypted post.
  - has NO access to any other's private key.
- Any sniffer:
  - has access to intercepted posts.

Design Overview

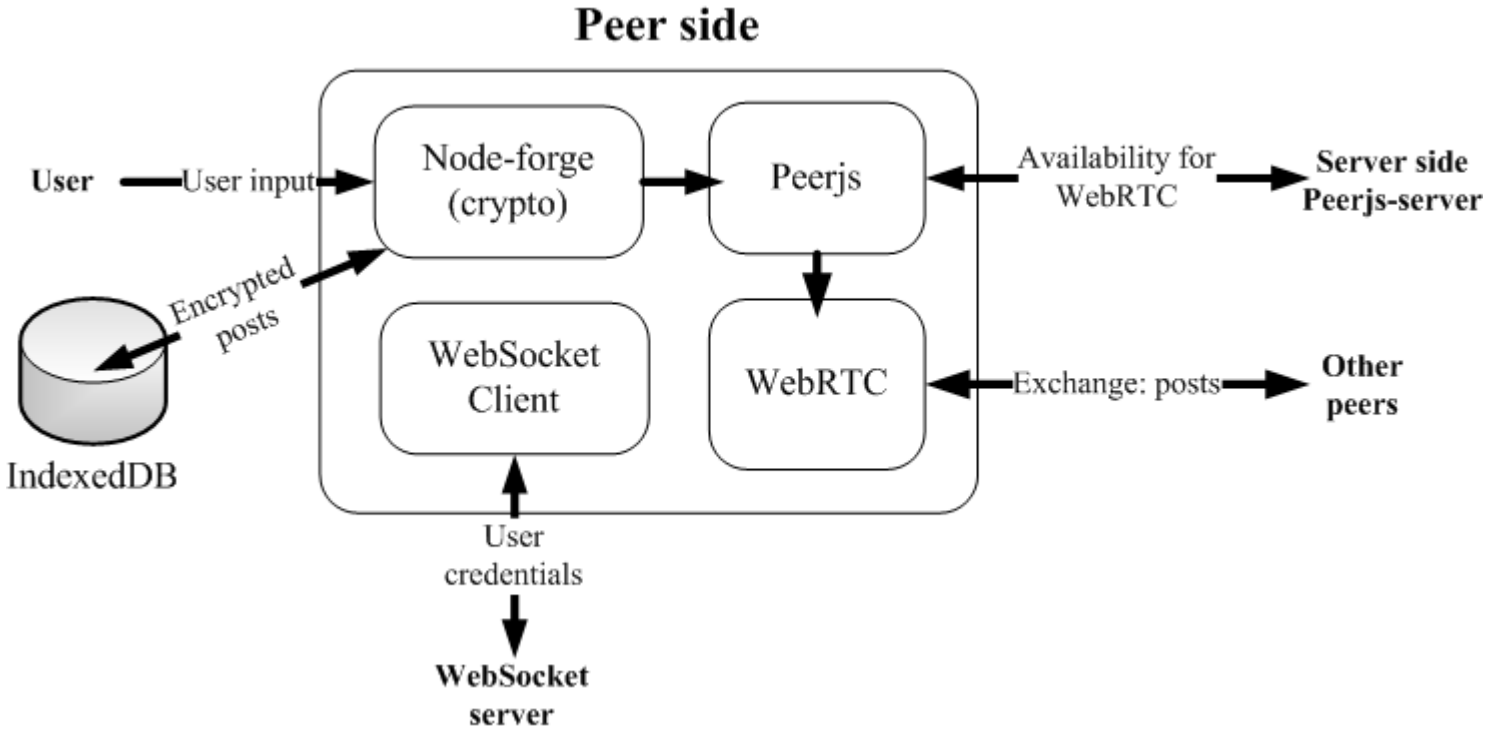


Features

- Confidentiality
  - All the posts are encrypted, only the designated receivers can decrypt.
- Privacy
  - The AVL server is isolated from the communication among peers.
- Availability
  - The missed posts during offline period are synced and restored with acceptable latency.
- Anonymity
  - Intercepted posts reveal no origin or destination username.

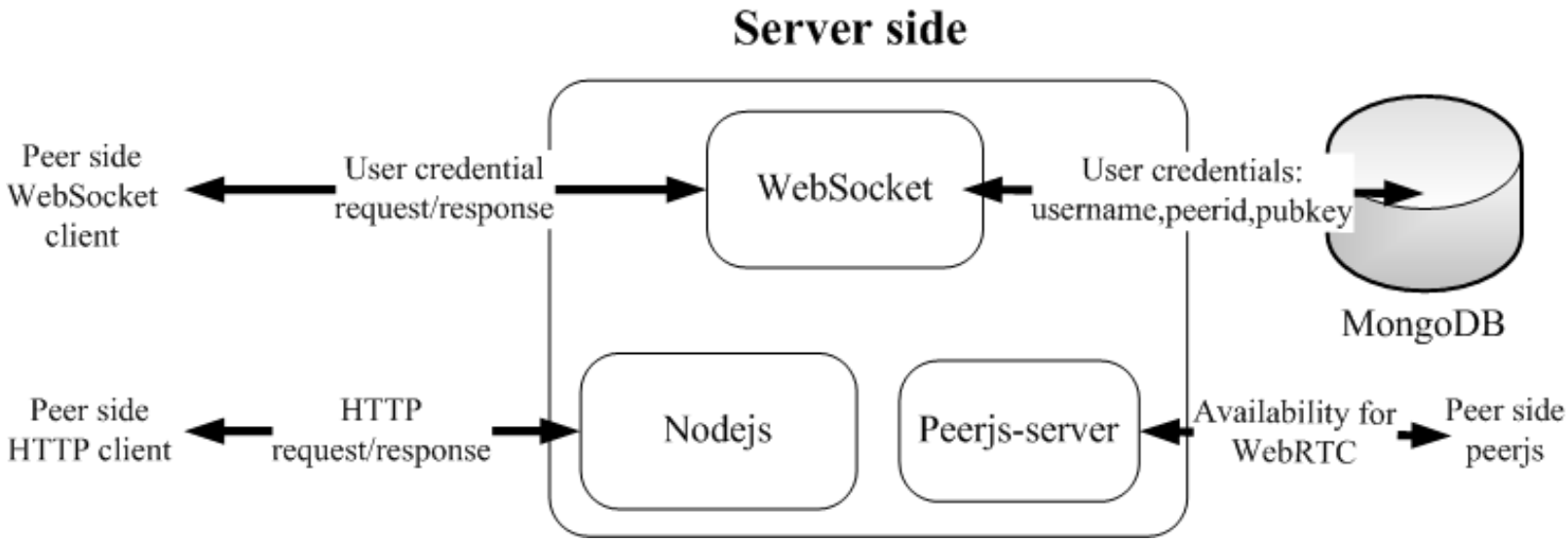
Architecture

- Peer side



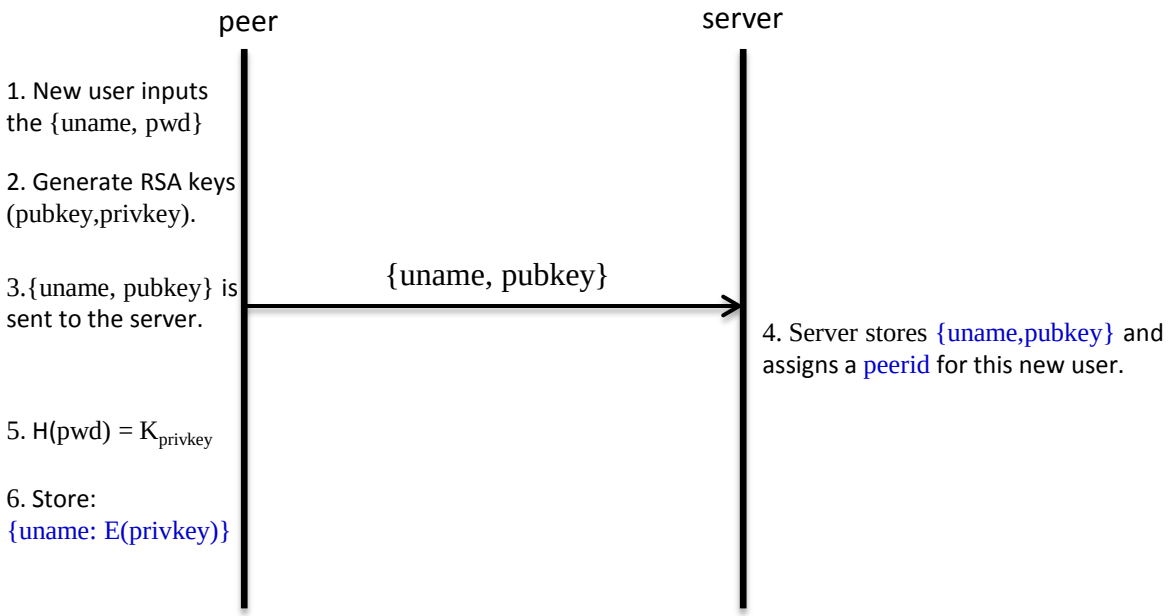
Architecture

- AVL Server side

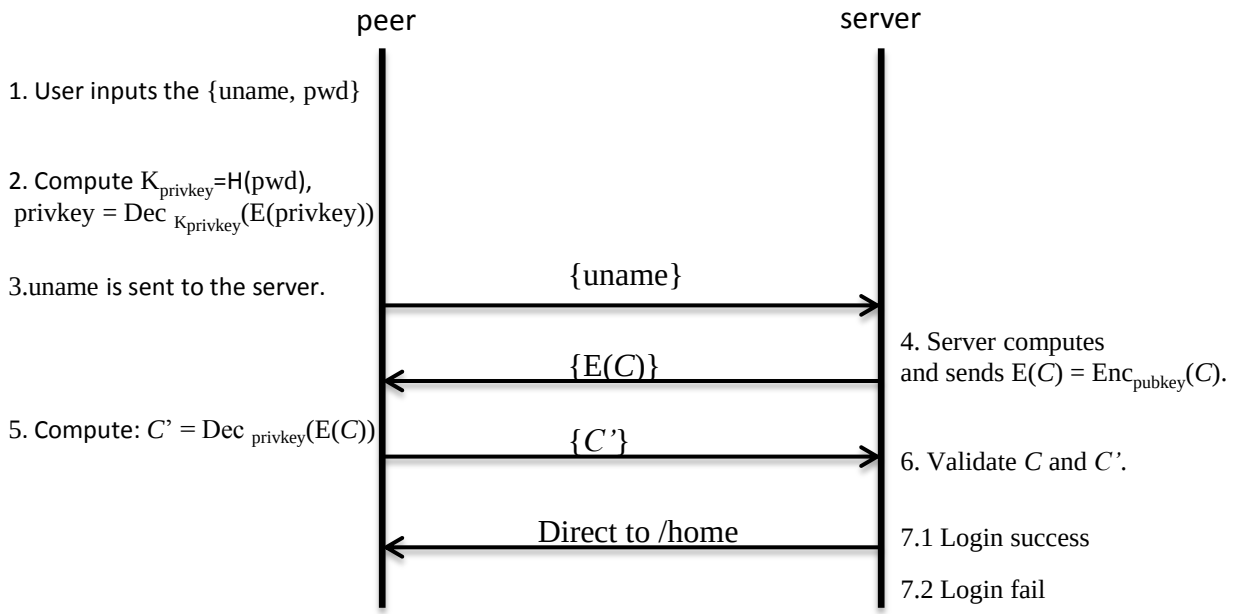


Crypto Design – user credential

- New user registration
  - A new user inputs the {uname, pwd}.
  - A pair of RSA keys (pubkey,privkey) is generated.
  - {uname, pubkey} is transmitted to the server.
  - Server stores the {uname,pubkey} and assigns a peerid.
  - pwd is hashed to become a key $K_{privkey}$ to encrypt the privkey; E(privkey) = Enc $_{K_{privkey}}$(privkey) is stored at the client side.



- User login
  - User inputs the {uname, pwd}.
  - Compute $K_{privkey}$, and decrypt privkey: privkey = $Dec_{K_{privkey}}(E(privkey))$
  - User sends login request to server: {uname}.
  - Server responses with a challenge $E(C)$ of a random num encrypted with user's pubkey: $E(C) = Enc_{pubkey}(C)$.
  - User decrypts and sends back the C with $K_{privkey}$: $C' = Dec_{privkey}(E(C))$.
  - Server validates the C' with C and directs the page to /home if good.



Crypto Design - 2

Encrypt post individually for each recipient, using their public key

```
var PostInput = React.createClass({
    handlePost: function() {
        var postText = this.refs.postText.getDOMNode().value.trim();
        if (postText !== '') {

            var timePosted = Date.now();

            connectionCache.keys.forEach(function(peerId) {
                // send the post to every currentConnection
                var conn = connectionCache.conns[peerId];

                dbService.friendStore.findByPeerId(conn.peer, function(friend) {
                    conn.send(encryptPost({
                        id: postId,
                        msgType: 'GLOBAL_POST',
                        text: postText,
                        timeStamp: timePosted,
                        receivers: receiversList
                    }, friend.pubkey));
                });
            });

```

AES encrypt post message with random key and IV. Use receiver's RSA public key to encrypt the generated key and IV.

```
function encryptPost(post, receiverPubKey) {
    var key = forge.random.getBytesSync(16);
    var iv = forge.random.getBytesSync(16);

    var cipher = forge.cipher.createCipher('AES-CBC', key);
    cipher.start({iv: iv});
    cipher.update(forge.util.createBuffer(post.text));
    cipher.finish();

    var encryptedText = cipher.output.getBytes();

    var pubKey = forge.pki.publicKeyFromPem(receiverPubKey);

    var encryptedAesKey = pubKey.encrypt(key);
    var encryptedAesIv = pubKey.encrypt(iv);

    return {
        id: post.id,
        msgType: post.msgType,
        text: encryptedText,
        timeStamp: post.timeStamp,
        receivers: post.receivers,
        aesKey: encryptedAesKey,
        aesIv: encryptedAesIv
    };
}
```

Crypto Design - 3

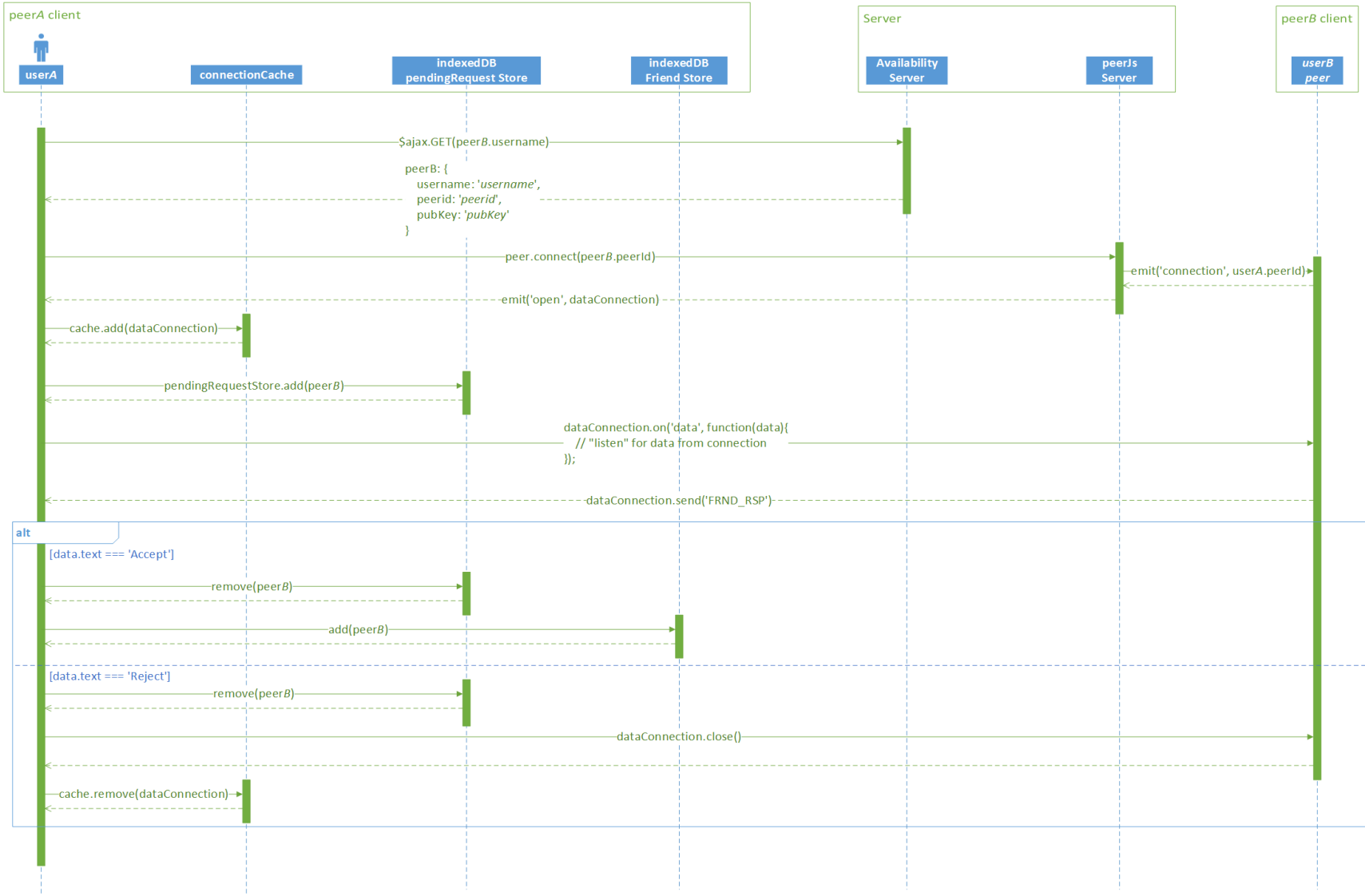
Store encrypted post in indexedDB, store decrypted post in post "cache" (for DOM rendering)

```
140  function connect(conn, callback) {
141      conn.on('open', function() {
142
143          // Receive messages
144          conn.on('data', function (data) {
145
146              var post = {
147                  author: conn.peer,
148                  postId: data.id,
149                  text: data
150              };
151
152              switch(post.text.msgType) {
153
154                  case 'GLOBAL_POST':
155
156                      post.text.latency = Date.now() - post.text.timeStamp;
157                      dbService.postStore.addPost(post);
158                      globalPosts.push(decryptPost(post));
159                      break;
160
```

Use current user's RSA private key to decrypt the AES key and IV, then use the decrypted key and IV to decrypt post message.

```
60   function decryptPost(post) {
61       var privKey = forge.pki.privateKeyFromPem(currentUser.privateKey);
62       var aesKey = privKey.decrypt(post.text.aesKey);
63       var aesIv = privKey.decrypt(post.text.aesIv);
64
65       var decipher = forge.cipher.createDecipher('AES-CBC', aesKey);
66       decipher.start({iv: aesIv});
67       decipher.update(forge.util.createBuffer(post.text.text));
68       decipher.finish();
69
70       var decryptedText = decipher.output.toString();
71
72       return {
73           author: post.author,
74           text: {
75               msgType: post.text.msgType,
76               text: decryptedText,
77               timeStamp: post.text.timeStamp
78           }
79       }
80   }
```

Friendship management



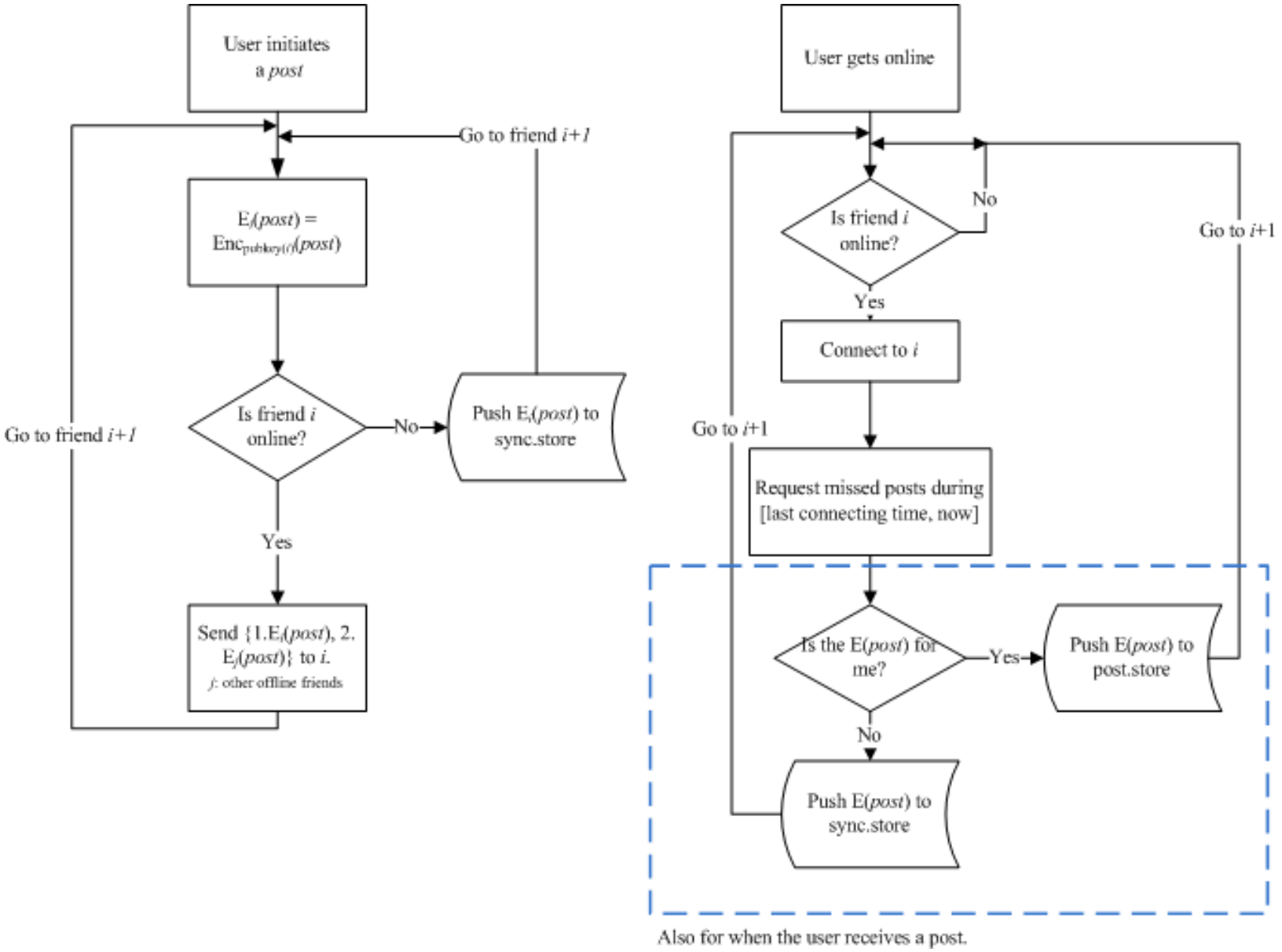
Synchronization

- Why do we need it
  - Users stay online sparsely. But she still gets to receive the missed posts during offline period.
  - Without a centralized server, synchronization becomes challenging.
  - Leverage other peers to sync the posts.
- Design goal
  - Minimize the latency between an initial posting and the arriving at a receiver.

Flooding Sync - 1

- Basic idea:
  - Try best to propagate posts among peers (even not friends); the user can get their supposed posts, or help propagate the sync posts.
- Confidentiality
  - It is kept as only the receiver can decrypt the post with her private key.

Flowchart


User initiates a *post*
Go to friend *i+1*
$E_i(post) = Enc_{pubkey(i)}(post)$
Is friend *i* online?
No
Push $E_i(post)$ to sync.store
Go to friend *i+1*
Yes
Send {1.$E_i(post)$, 2. $E_j(post)$} to *i*.
j: other offline friends
User gets online
No
Is friend *i* online?
Go to *i*+1
Yes
Connect to *i*
Go to *i*+1
Request missed posts during [last connecting time, now]
Is the E(*post*) for me?
Yes
Push E(*post*) to post.store
No
Push E(*post*) to sync.store
Also for when the user receives a post.

Post-specific sync (W.I.P.) - 1

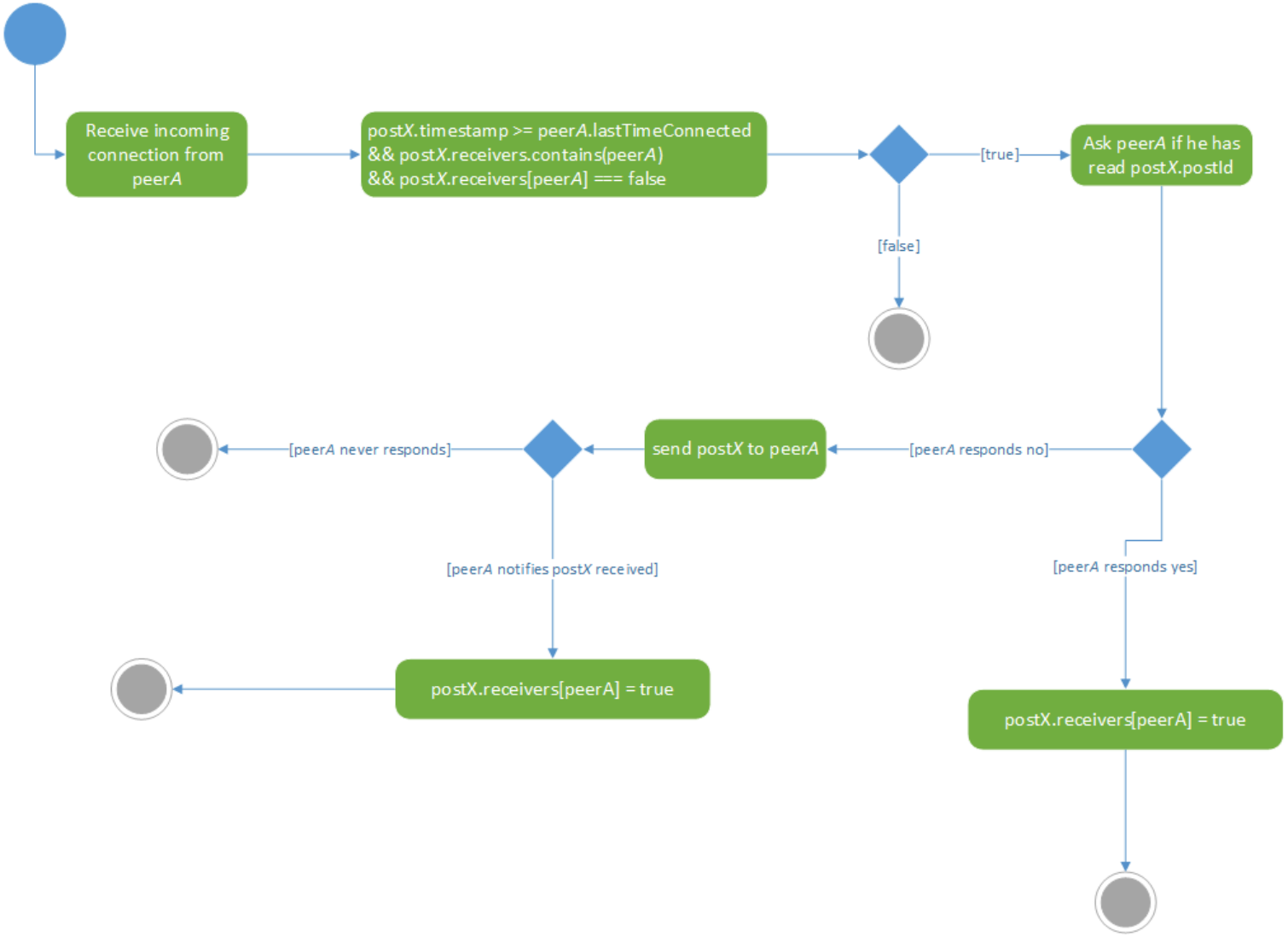
- Each post has a list of all intended recipients, so that every peer who receives can participate in the distributed syncing.
- Advantages:
  - High availability of posts (assuming many friend connections in common)
  - Storage efficiency because a user only stores posts intended for them.
- Disadvantages:
  - A lot of sync communication happening asynchronously which may lead to collisions/duplicates.

<u>Proposed post representation:</u>

```
post: {
    postId: 'uniquePostId',
    author: 'senderPeerId',
    text: {
        text: 'encryptedMessage',
        timestamp: 1234567890
    },
    receivers:  {
        'receiverAPeerId': false,
        'receiverBPeerId': false,
        'receiverCPeerId': true
    }
}
```

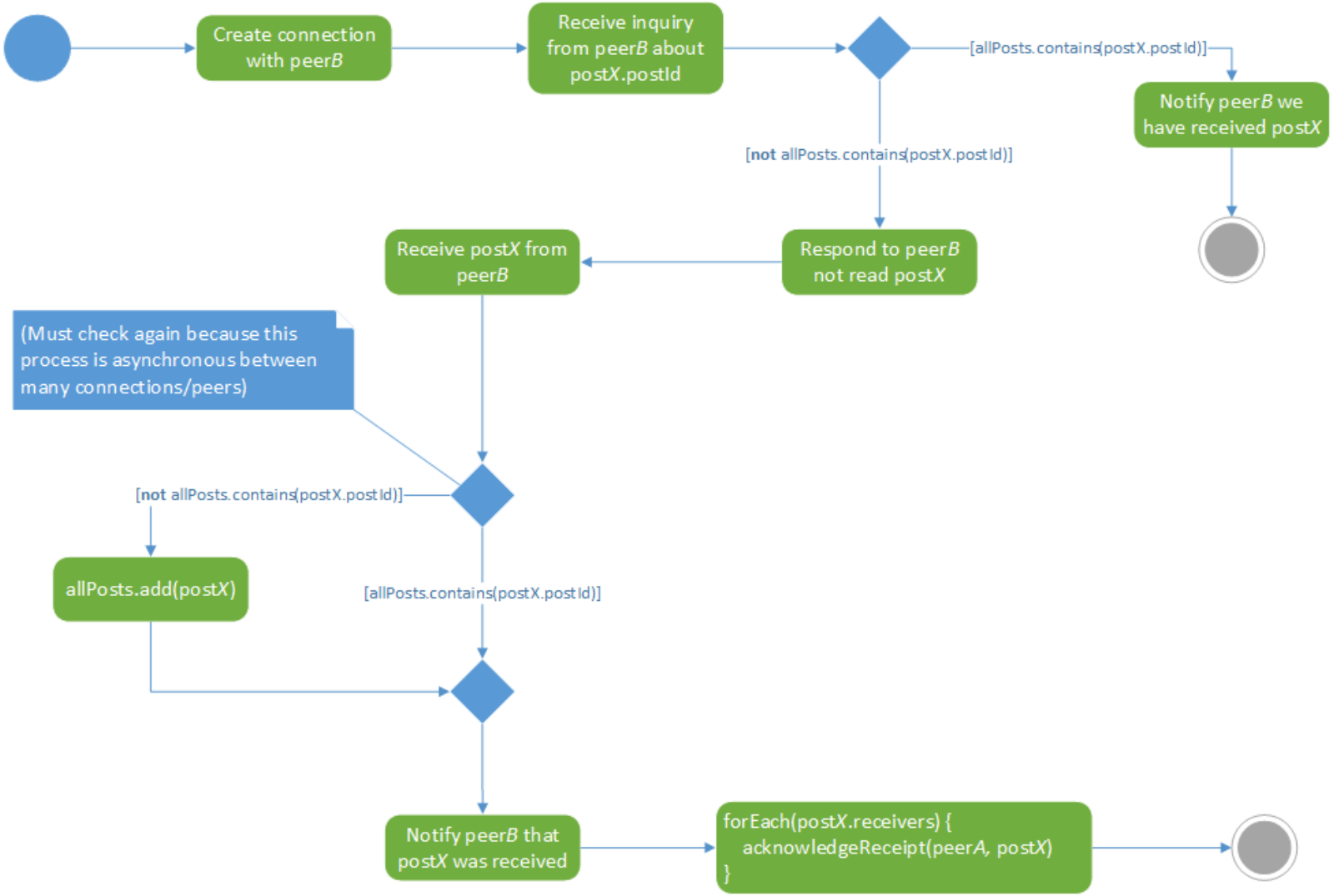
Post-specific sync (W.I.P.) - 2

Receive connection:



Post-specific sync (W.I.P.) - 3

Login and initiate connection:



Snapshots

deface

Welcome to Distributed Social Network

User Info

User Name: gossip
Public Key: -----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1KPI6OIq8C4VRhJea5CT 1tAacIdil
al6dbmysTrBQYdb4nG2Auyr7+jqERoBgjLHeWMCulFamE+at8RU18u063gMrrjwT LPbxWz412Ubfj4xQOQNI3pRMfhB/MTmcKK4r/0iRpZ
7wLnWG6IGr2jOUilejxpr4wTZFgnh8QN/VTHihFTdfo+j651yPUzZn7bGQ7ZDBsI fAMvtodMRbXPXZHjYSh/YXAntQJ2OBLy607tVYjk8pO
Status: off
Peer ID: c9lq5p5yhv4vaemi
Public IP: 10.10.10.11
Local IP:
Last Logoff:

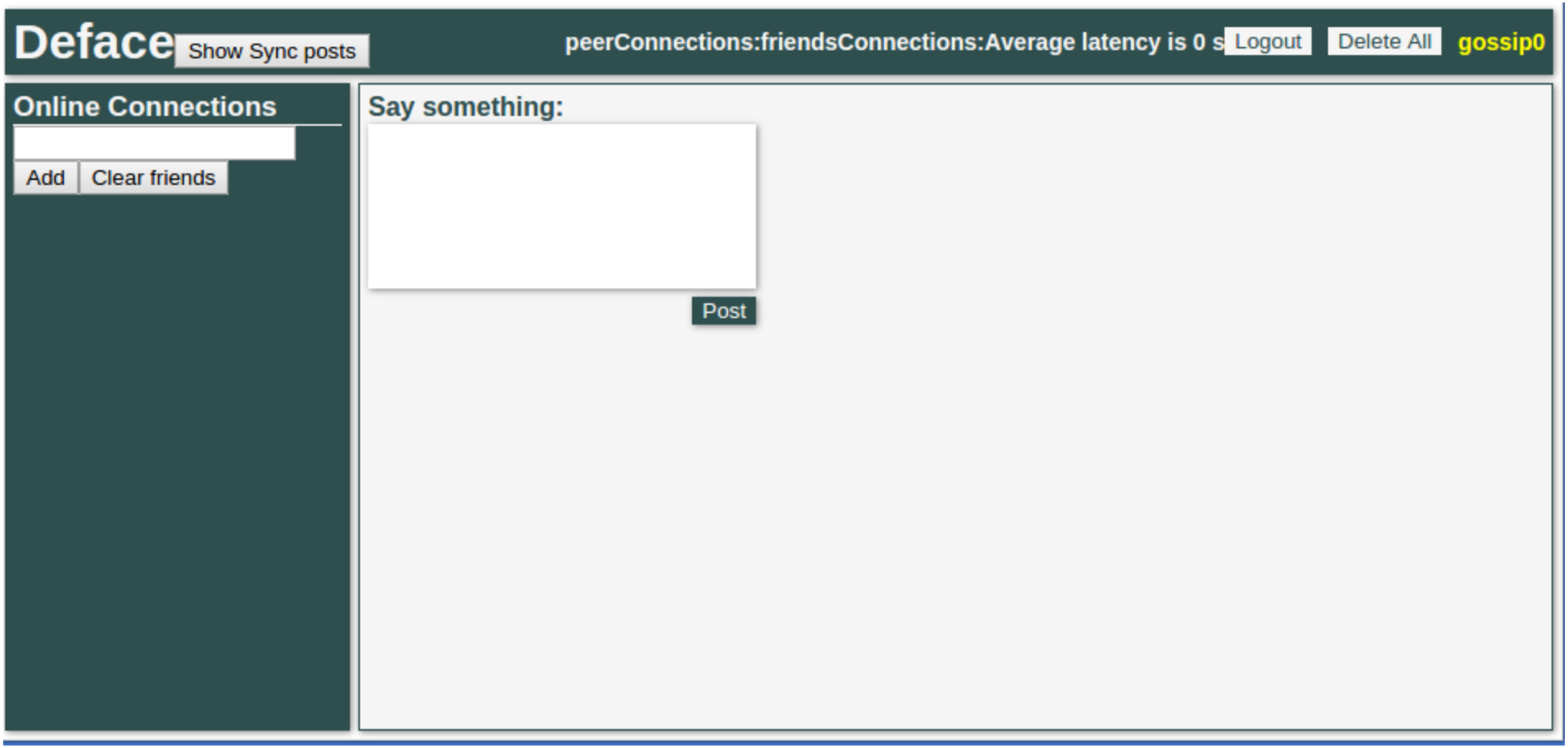
User List

UserName	Public key
gossip	-----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1KPI6OIq8C4VRhJea5CT 1tAacIdiFOgk50PbqJ5IvQUaCfOaZ2Rht9GUUJjwrvaM/xuJDuODfChEhiaWXXVy al6dbmysTrBQYdb4nG2Auyr7+jqERoBgjLHeWMCulFamE+at8RU18u063gMrrjwT LPbxWz412Ubfj4xQOQNI3pRMfhB/MTmcKK4r/0iRpZ/CCmTcEgV1uK3clHnGJyBu 7wLnWG6IGr2jOUilejxpr4wTZFgnh8QN/VTHihFTdfo+j651yPUzZn7bGQ7ZDBsI fAMvtodMRbXPXZHjYSh/YXAntQJ2OBLy607tVYjk8pOq+4tWBexAIFnGbc9rBXEM xQIDAQAB -----END PUBLIC KEY-----

Add User

Username | Password

Register


Deface
Show Sync posts
peerConnections:friendsConnections:Average latency is 0 s
Logout
Delete All
gossip0
Online Connections
Add
Clear friends
Say something:
Post

Availability Evaluation (flooding sync)

- From the angle of a user:
  - Mean of posts arriving latency - L
    - Time difference between the spot when the post is sent from sender, and when the post arrives at the receiver.
- From the angle of a post:
  - Post propagation rate – $P(\%)$
    - The period of time it takes for a post to reach % percentage of receivers.

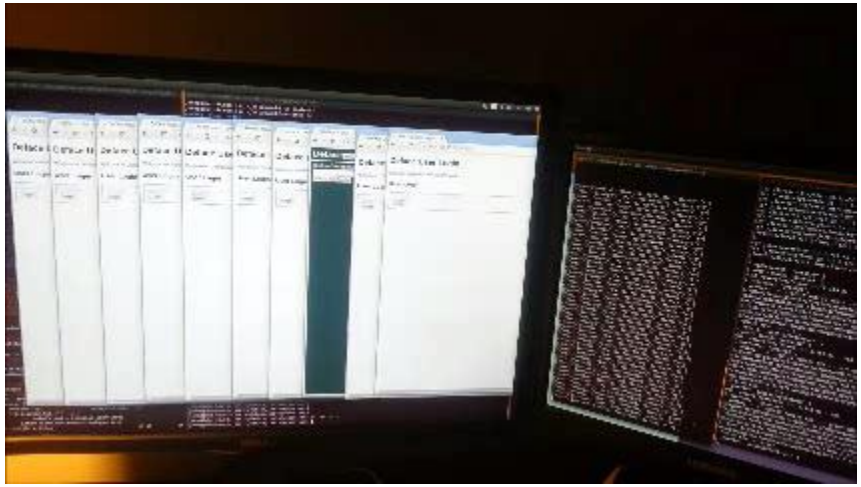
Evaluation - 2

- Tool: webdriver-js
  - A full head test automation library
- Environment:
  - Hardware: desktop /w 8G memory
  - OS: Ubuntu 12.04
  - Browser: Chrome, Firefox

Evaluation - 3

- Parameter vector
  - Size of the users pool - n
  - Friends ratio - f
  - Online period of time - T_{on}
  - Offline period of time - T_{off}
  - Starting time phase – Phase
    - focus or sparse
- Procedure
  - Compute and assign every user's friends - nf
    - Every user does registration
    - The friends map is assigned to every user.
  - Launch n webdriver instances
    - A user logs in with a random phase of delay(within 10s or T_{off}), sends a post, stays online for T_{on} and logs out.
    - Then waits for approx T_{off}, and logs in again.

Emulating…



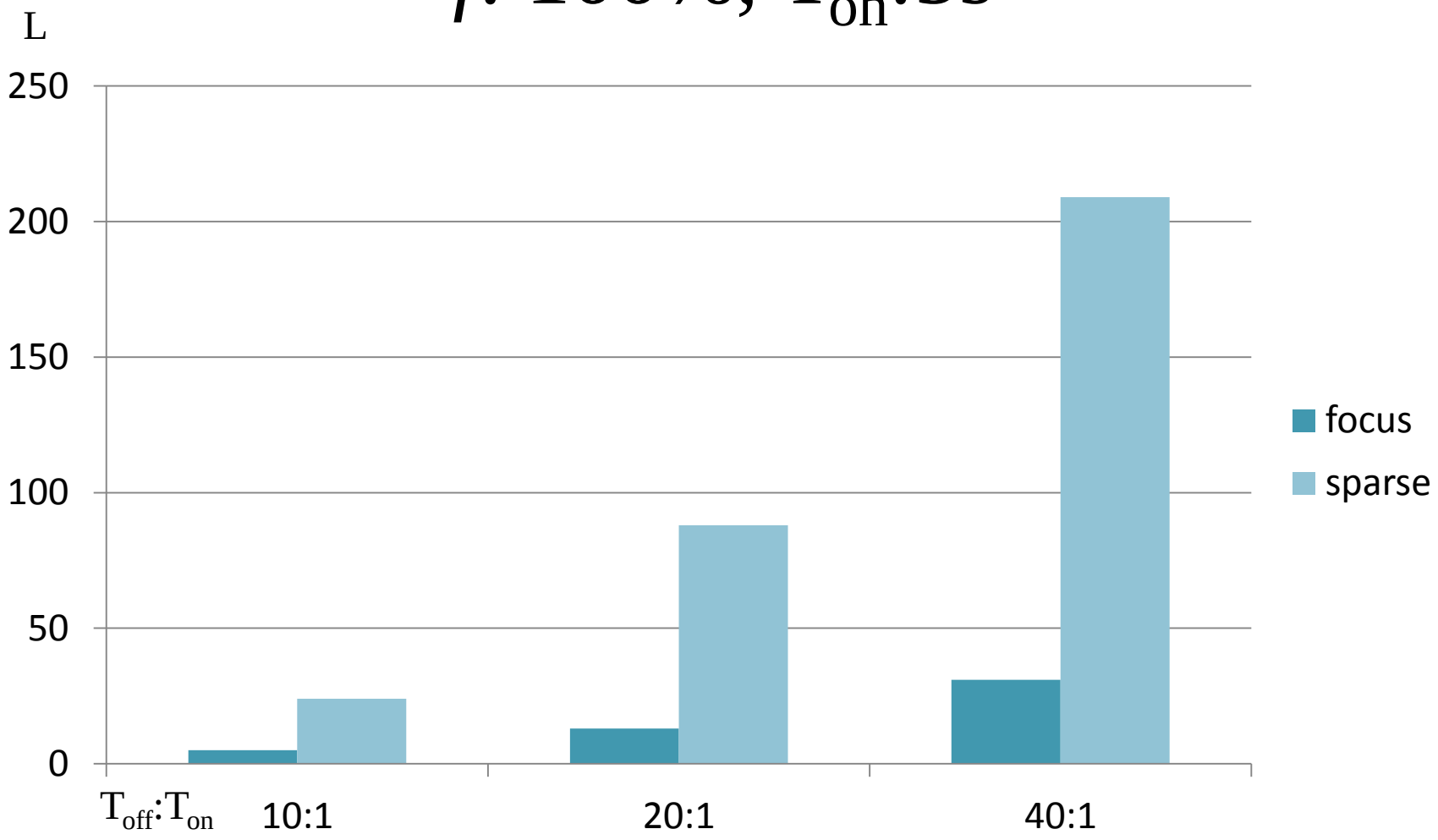
Evaluation - 4

n	*f*	T_{on}	T_{off}	Phase	L	P(30%)	P(60%)	P(90%)
2	100%	10s	5s	focus	5s	5s	5s	5s
10	100%	6s	60s	focus	5s	2s	4s	5s
10	100%	6s	60s	sparse	24s	11s	21s	26s
10	50%	6s	60s	focus	7s	3s	4s	6s
10	50%	6s	60s	sparse	43s	19s	36s	48s
10	30%	6s	60s	focus	6s	3s	6s	11s
10	30%	6s	60s	sparse	56s	23s	39s	53s
10	30%	5s	100s	sparse	139s	46s	87s	112s
10	30%	5s	200s	sparse	253s	99s	191s	244s

*All the tests are run for 20min

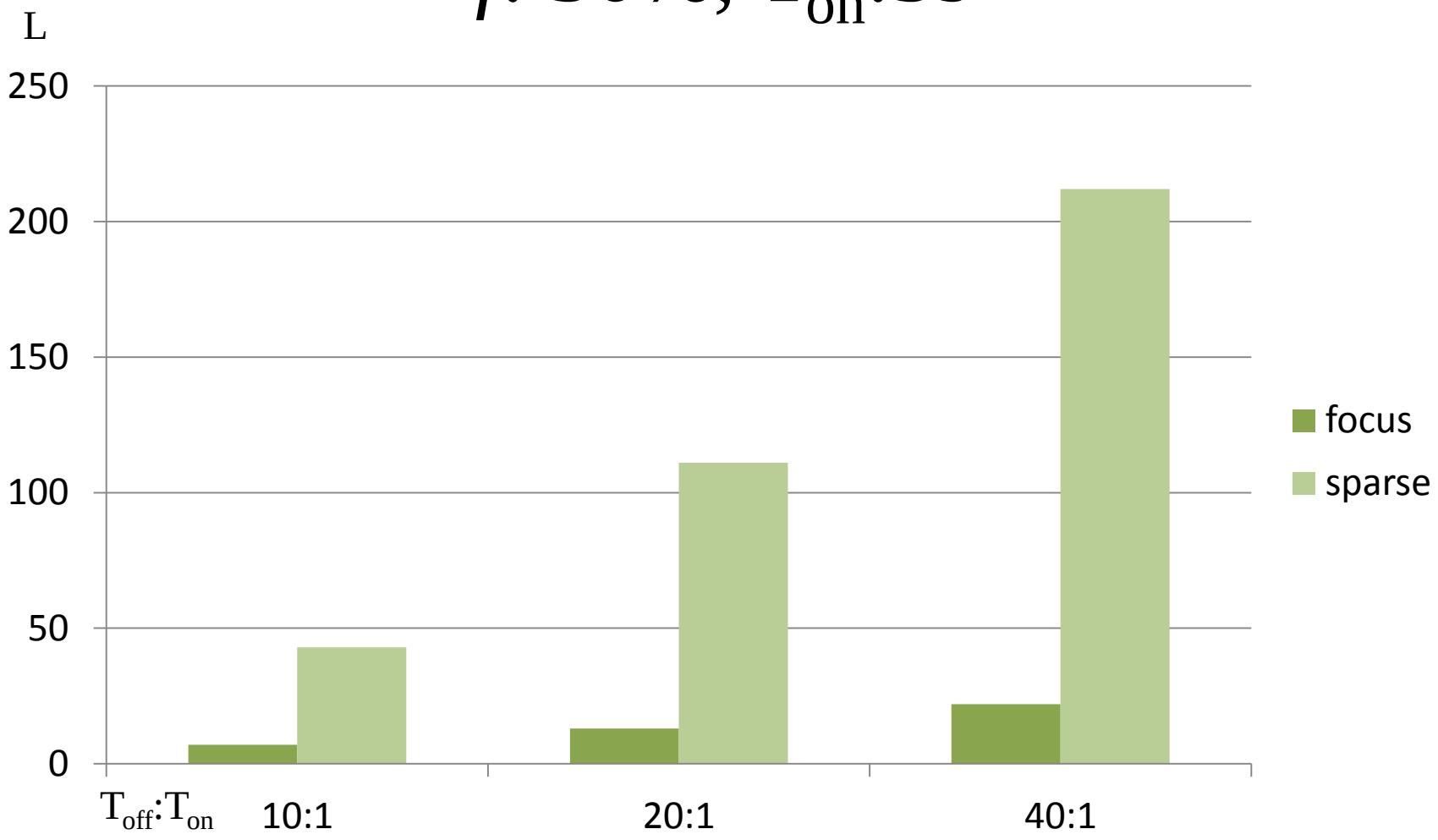
Latency /w different online patterns

f: 100%, T_{on}:5s



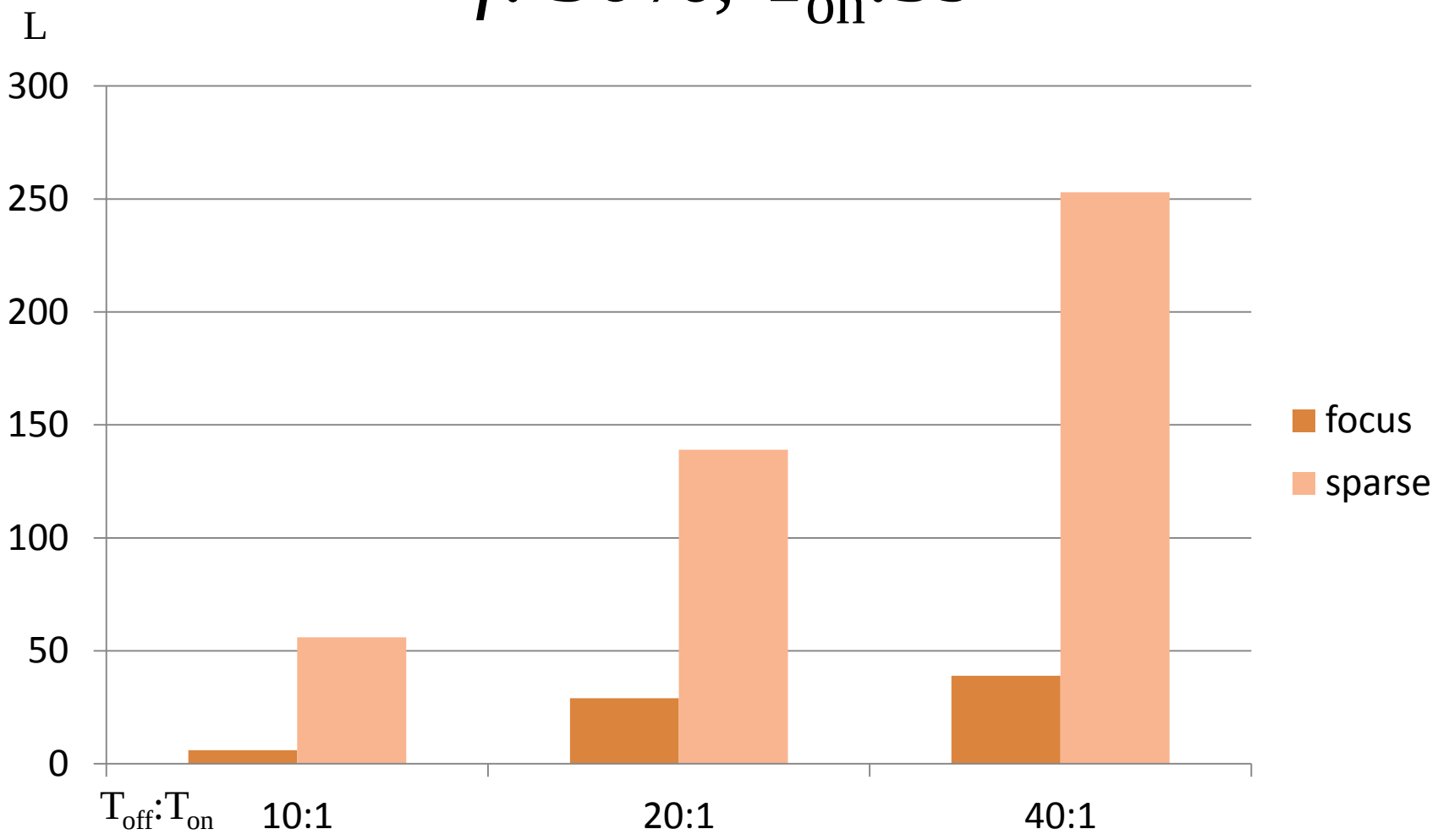
Latency /w different online patterns

f: 50%, T_{on}:5s



Latency /w different online patterns

f: 30%, T_{on}:5s



Discussion

- Based on the evaluation result:
  - The flooding sync performance is acceptable in most emulated cases, note that the perceptible latency is $L - T_{off}$.
    - Even Facebook syncs posts to user when she is online.
  - The performance of sync depends on the time when friends "***meet***" online.
- Limitation
  - Due to the same origin policy, all the credential and data at the peer side are not portable.
- Future work
  - Crypto process may involve message verifications.
  - Scale the emulation.
  - Implement and evaluate the post-specific sync method.
  - To further improve the availability, more assumptions will be considered.
    - E.g., we may require at least one user must be online at all times inspired by Internet Relay Chat (IRC).