foglet Documentation

Release 4.0.6

Arnaud Grall, Brice Nedelec, Thomas Minier

Jun 18, 2018

Contents

CHAPTER 1

Getting started

To demonstrate all key concepts of a Foglet, let's build a Hello world aplication. it will be a simple Foglet that broadcast the message *hello world!* to all connected browsers.

All code written during this tutorial can be found on this repository.

1.1 Setting up the project

First, setup a new npm project

```
mkdir foglet-hello-world
cd foglet-hello-world
npm init
```

Next, install the core library and the development tools for foglet apps

```
npm install --save foglet-core
npm install --save-dev foglet-scripts
```

Edit your `package.json` file to add the following fields:

```
"scripts: {
  "start": "foglet-scripts start"
}
```

Now, create the following files

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
```

(continues on next page)

(continued from previous page)

```
    <title>Foglet Hello world</title>
  </head>

  <body>
    <button id="send-message">Hello World!</button>
    <!-- foglet-core bundle -->
    <script src="node_modules/foglet-core/dist/foglet.bundle.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

app.js

```
'use strict'

console.log('hello world')
```

To test your installation, open `index.html` in a browser, you should see **hello world!** in the console.

1.2 Let's build the real app now!

Now that's your project is ready, let's create our Hello World Foglet. Here is the complete code to put in **index.js**.

NB: Notice that we use a `require` style syntax here to import dependencies, as foglet-core bundle is built using Browserify

```
'use strict'

const Foglet = require('foglet').Foglet

const app = new Foglet({
  verbose: true, // activate logs. Put false to disable them in production!
  rps: {
    type: 'spray-wrtc',
    options: {
      protocol: 'foglet-hello-world', // name of the protocol run by your app
      webrtc:   { // WebRTC options
        trickle: true, // enable trickle (divide offers in multiple small offers sent␣
↪by pieces)
        iceServers : [] // iceServers, we lkeave it empty for now
      },
      timeout: 2 * 60 * 1000, // WebRTC connections timeout
      delta: 10 * 1000, // spray-wrtc shuffle interval
      signaling: { //
        address: 'http://localhost:3000/',
        room: 'foglet-hello-world-room' // room to join
      }
    }
  }
})

// connect to the signaling server
app.share()

// connect our app to the fog
```

(continues on next page)

(continued from previous page)

```
app.connection()
.then(() => {
  console.log('application connected!')

  // listen for incoming broadcast
  app.onBroadcast((id, msg) => {
    console.log('I have received a message from peer', id, ':', msg)
  })

  // send our message each time we hit the button
  const btn = document.getElementById("send-message")
  btn.addEventListener("click", () => {
    app.sendBroadcast('hello World!')
  }, false)
})
.catch(console.error) // catch connection errors
```

Now, run `npm start` to start the signaling server, and then open **index.html** in two tabs, to create two distinct peers.

Open the console, wait for connections to be done, and then click those damn buttons! You should see messages popping in each tab!

1.3 Setting up a signaling server

A signaling server acts as a forwarding server in order to connect all new peers on the specified room. You can access an implenmentation at https://github.com/RAN3D/foglet-signaling-server

However, if you juste need a signaling server out of the box, the foglet build tools contains one that can be run with `foglet-scripts start`.

CHAPTER 2

Basic communication

Basically, a foglet offers five communication primitives: `unicast`, `multicast`, `broadcast`, `streaming unicast` and `streaming broadcast`. They can be used to exchange messages between peers in the network.

We now review these four primitives to demonstrate their use.

2.1 Unicast communication

A peer can use the *unicast* primitive to send a message to one of its direct neighbour. However, it can't use it to send a message to a peer that is not one of its neighbours!

```
// Get the ID of my first neighbour
const id = my_foglet.getNeighbours[0]

my_foglet.sendUnicast(id, 'Hi neighbour! Do you want to party tonight?')
```

A foglet can listen for incoming unicast messages using the `onUnicast` method, that registers a callback executed for each unicast message received by the foglet.

This callback is called with two parameters:

- `id` the ID of the peer who sent the message
- `message` the message received

Notice that `id` can be used to contact the peer using `sendUnicast`.

```
my_foglet.onUnicast((id, message) => {
  console.log(`Unicast message received from ${id}: ${message}`);

  // anwser to our neighbour
  my_foglet.sendUnicast(id, 'Sure. Can I bring a salad?')
})
```

2.2 Multicast communication

In real-world applications, one may want to send a message to several neighbours. Instead of repeateadly send a unicast messages to each peer individually, the `multicast` primitive allow a foglet to send a unicast message to a set of neighbours. These messages are received by others peers as *regular unicast messages*.

```
// Get the IDs of all neighbours
const ids = my_foglet.getNeighbours

my_foglet.sendMulticast(ids, 'Everyone, free salad at my place tonight!')
```

2.3 Broadcast communication

Where the `unicast` and `multicast` primitives allow to contact neighbours, the `broadcast` primitive allow a peer to send a message to **all peers in the network**. This broadcast is done using a flooding algorithm and implements a **causal broadcast**, which guarantee the following properties:

- *Validity:* if a peer received a message `m` at least once, then `m` has been diffused at least once by another peer.
- *Uniformity:* if a peer received a message `m`, then all peers will receive `m`.
- *FIFO reception:* if a peer broadcast a message `m` and next another message `m'`, then no peer will receive `m'` before `m`.
- *Causal reception:* if a peer receive a message `m` and next broadcast a message `m'`, then no peer will receive `m'` before `m`.

```
my_foglet.sendBroadcast('Can I borrow some salt from someone?')
```

Like `unicast` messages, a foglet can listen for incoming broadcast messages using the `onUnicast` method, that registers a callback executed for each broadcast message received by the foglet.

This callback is called with two parameters:

- `id` the ID of the peer who sent the message
- `message` the message received

```
my_foglet.onBroadcast((id, message) => {
  console.log(`Broadcast message received from ${id}: ${message}`);
})
```

Warning: contrary to unicast messages, a broadcast message can be recevied from any peer in the network. Thus, the `id` can be used to conctact the emitter (using `sendUnicast`) at the condition that the emitter is a neighbour of the receiver. Otherwise, the message will not be sent.

2.4 Streaming unicast and broadcast

CHAPTER 3

ICE servers

Work in prgress

A complete and usefull documentation is available here

In short: when your are in a local network, you don't need to use ICE. But if you want to contact a peer on the other side of the world, you may have to pass through firewalls and all sort of things. ICE servers are here to resolve this.

CHAPTER 4

Key concepts

`foglet-core` allows to build fog computing applications, *i.e.*, applications that runs in a *fog of browsers*. Such application is called a **Foglet**.

A Foglet connect browsers through a Random Peer Sampling (RPS) overlay network [1]. Such a network approximates a random graph where each data consumer is connected to a fixed number of neighbors. It is resilient to churn, to failures and communication with neighbors is a zero-hop.

In the context of browsers, basic communications rely on WebRTC to establish a data-channel between browsers and SPRAY [2] to enable RPS on WebRTC. Each browser maintains a set of neighbors K called a view that is a random subset of the whole network. To keep its view random, a data consumer renews it periodically by shuffling its view with the view of a random neighbor.

As a Foglet rely on WebRTC for communication, it requires **a signaling server** to disocver new peers and **ICE servers** to connect browsers, throught NAT for example. These points will be discussed in details later.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search