D3 Web Visualization: Fading Links

Miguel Edlinger*

[1]Internship, Complexity Science Hub Vienna, Austria
*Contact: [redacted]

ABSTRACT

My goal was to create a visualization of the transactions of KOIN*s between Users of the already existing KOIN* application. This visualization should be on the web. Therefore this visualization needs a powerful JavaScript library, so-called D3 (Data Driven Documents). D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document.

Introduction

The purpose of this report is to inform the Complexity Science Hub about the outcome of my work and contribution to the visualization project that I achieved during my internship as an intern. Here, this contribution is to be understood as programming a JavaScript code which graphically simulates transactions between users of the KOIN * application (CSH) via a web interface. The main task was to create random links which symbolize KOIN* transactions between users and let them fade out when the transaction is over.

The course of the project

Preparation

To fulfil the main task, I had to learn the basics of front-end web development (CSS, JavaScript and HTML). Besides, this basic knowledge was needed in the further course of the project to be able to identify and remedy errors efficiently within the script.

Proper handling of the D3 library

Then, if not the most important step of all, was the understanding and proper handling of the D3 library (Data-Driven Documents). Because D3 is an important tool and resource to create interactive data visualization for the web and therefore necessary to achieve my task.

Splitting the main task into sub tasks

Another vital aspect to fulfil a heavy task is to split it into sub-tasks and solve them on after the other. For instance, I tried at first as a single miniature project to create a simple forced layout of a network with some nodes and randomly connected edges. After this, the arrowheads were attached to the links to show which one of the nodes the target is and which one the source. I did this until I had programmed each feature itself within a single project and was able to implement it inside of the code of the main task.

Tackling the main task

Since I had already written individual code passages in the form of individual projects, I only had to insert and restructure them in the script of the main task. But at that point, the code would only run statically. The next important step was to make a dynamical animation so you could see how the edges fade out and then release. Therefore, I had to use an additional pre-build function which executes the code after a variable time. That enabled me to embed the update-pattern for the visualization and behaviour of the animation.

Results

The result consists of a simple HTML file (***indexFade.html***) for the schematics of the page, it also consists a CSS (***stylesFade.css***) for the style of the containers and a javascript file (***TransactionSim.js***) to make the web interface interactive and to visualize the transactions.

Web Interface Output

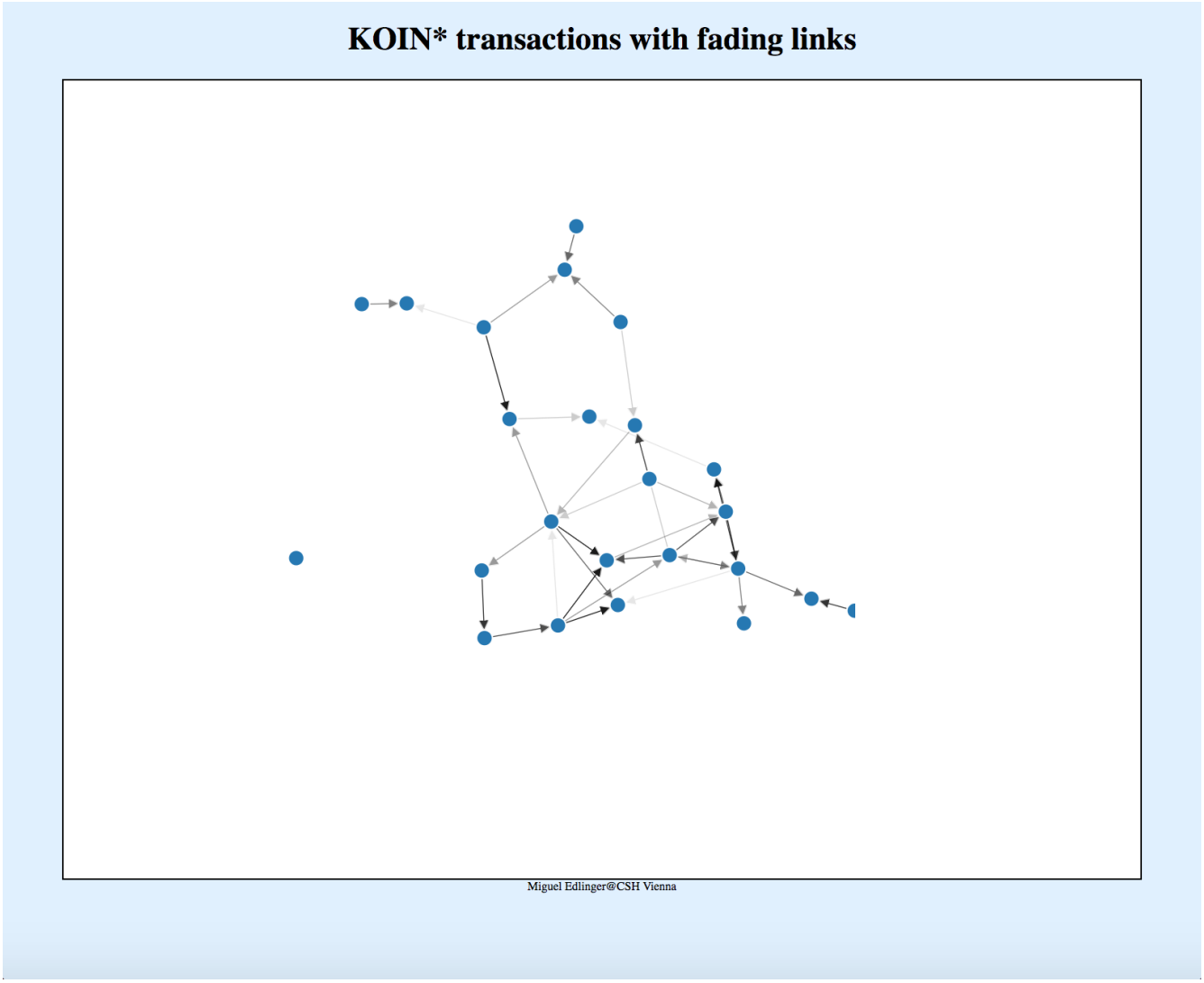

Figure 1. Screenshot of the webinterface showing the KOIN* transactions with fading links

Fig.1 shows that links have different fade as they are attached to the network at different update cycles. And links with small opacity will be removed from the network after they faded out completely.

Code

The code was written so that the number of users who transfer KOIN * to other users, as well as the time interval in which these transactions take place, can be adjusted as parameters.

HTML

indexFade.html

```
1
2  !DOCTYPE html>
3  <html>
4    <head>
5      <meta charset="utf-8">
6      <title>KOIN* transaction</title>
7      <link rel="stylesheet" type="text/css" href="stylesFade.css">
8      <script src="https://d3js.org/d3.v4.js"></script>
9      <script src="./TransactionSim.js"></script>
10   </head>
11
12 <body>
13   <div id="wrapper">
14     <div id="header">
15       <h1>KOIN* transactions with fading links </h1>
16       </div>
17     <div id="simulation"></div>
18     <div id="footer">Miguel Edlinger@CSH Vienna</div>
19   </div>
20     <script> main() </script>
21 </body>
22 </html>
```

JavaScript

TransactionSim.js

```
1  /*
2  @author: Miguel Edlinger
3  @date: 16.08.2018
4  @description:
5  */
6  function main() {
7
8      /* Creating the containers */
9      var width = 800,
10         height = 800;
11
12     /* container for the simulation */
13     var svg = d3.select("#simulation").append("svg")
14         .attr("width", width)
15         .attr("height", height),
16         color = d3.scaleOrdinal(d3.schemeCategory10);
17
18     /* setting of the canvas dimensions  */
19
20     d3.select("#wrapper")
21         .style("width", 25 + width * 4 / 3 + "px");
22
23
24     /* ***************Simulation of KOIN* transactions****************** */
25
26     // drives the simulation process and controls the time of each update
27     setInterval(simulate, 3000); // simulation updates after 3s
28
29     // The nodes object contains a list of nodes (users) simulation
30     var nodes = [
31         { name: "User1" },
32         { name: "User2" },
```

```
        { name: "User3" },
        { name: "User4" },
        { name: "User5" },
        { name: "User6" },
        { name: "User7" },
        { name: "User8" },
        { name: "User9" },
        { name: "User10" },
        { name: "User11" },
        { name: "User12" },
        { name: "User13" },
        { name: "User14" },
        { name: "User15" },
        { name: "User16" },
        { name: "User17" },
        { name: "User18" },
        { name: "User19" },
        { name: "User20" },
        { name: "User21" },
        { name: "User22" },
        { name: "User23" },
        { name: "user24" }
    ];
    // Links symbolise transactions between Users (nodes)
    var links = [];

    // defining force attributes for the simulation
    var simulation = d3.forceSimulation(nodes)
        .force("charge", d3.forceManyBody().strength(-500))
        .force("link", d3.forceLink(links).distance(50))
        .force("x", d3.forceX())
        .force("y", d3.forceY())
        .alphaTarget(1)

    // groups the link and node svg elements
    var g = svg.append("g").attr("transform", "translate(" + width / 1.5 + "," + height /2 + ")"),
        link = g.append("g").attr("stroke", "#000").attr("opacity", 1).selectAll(".link"),
        node = g.append("g").attr("stroke", "#fff").attr("stroke-width", 1.5).selectAll(".node");

    // build the arrow.
    g.append("svg:defs").selectAll("marker")
        .data(["end"])
        .enter().append("svg:marker")
        .attr("id", String)
        .attr("viewBox", "0 -5 10 10")
        .attr("refX", 18) //  15
        .attr("refY", 0) // -1.5
        .attr("markerWidth", 10)
        .attr("markerHeight", 10)
        .attr("orient", "auto")
        .append("svg:path")
        .attr("d", "M0,-5L10,0L0,5")
        .style("stroke", "#ccc");

    // this function creates edges, stores them into the links array,
    // decreases the lifespan of edges and kicks them out if their lifespan is over
    function updateEdges() {

        // this variable is used to control the amount of transaction per cycle
        var amountOfTransactions =
        Math.floor(Math.random() * nodes.length); // chose here the amount of transaction per cycle

        for (let x = 0; x < amountOfTransactions; x++) {
            //calc random source & target index
            var sourceIndex = Math.floor(Math.random() * nodes.length);
            var targetIndex = Math.floor(Math.random() * nodes.length);

            //compare until they do not match, in order to prevent self-targeting nodes
            while (sourceIndex == targetIndex) {
                targetIndex = Math.floor(Math.random() * nodes.length);
```

```
            }

            var source = nodes[sourceIndex];
            var target = nodes[targetIndex];

            // creates a link with target, source and a lifespan
            links.push({
                source: source, target: target, lifeSpan: 100
            });
            // logging the links to console for debugging purpose
            console.log(links);
        }

        // when lifespan of a link is over it gets removed
        for (let y = 0; y < links.length; y++) {
            if(links[y].lifeSpan) {
                links[y].lifeSpan = links[y].lifeSpan - 10;
                if (links[y].lifeSpan === 0) {
                    links.splice(y, 1);
                }
            }
        }

         /* here gets the restart function called in order to apply general
          update pattern to the created svg elements. */
        restart();
    }

    function restart() {

        // Apply the general update pattern to the nodes.
        node = node.data(nodes, function(d) { return d.id;});
        node.exit().remove();
        node = node.enter().append("circle")
            .attr("fill", function(d) {
                return color(d.id); })
            .attr("r", 8).merge(node);

        // Apply the general update pattern to the links.
        link = link.data(links, function(d) { return d.source.id + "-" + d.target.id; });
        link.exit().remove();
        link = link.enter().append("line").merge(link);

        // decreases the opacity of links in order to apply a "fading"-effect
        link.attr("opacity", function (d) {
            console.log("opacity: " + d.lifeSpan * 0.01);
            return d.lifeSpan * 0.01;
        });

        // attaches the arrowheads to the edges
        link.attr("marker-end", "url(#end)");

        // Update and restart the simulation.
        simulation.nodes(nodes);

        simulation.force("link").links(links);
        simulation.alpha(1).restart();
    }

    // updates the current position of the svg elements each cycle
    function ticked() {
        node.attr("cx", function(d) { return d.x; })
            .attr("cy", function(d) { return d.y; });

        link.attr("x1", function(d) { return d.source.x; })
            .attr("y1", function(d) { return d.source.y; })
            .attr("x2", function(d) { return d.target.x; })
            .attr("y2", function(d) { return d.target.y; });
    }

```

```
    // ensures the simulation dynamic and calls every other function inside of this script
    function simulate() {
        updateEdges();
        simulation.on("tick", ticked);
    }
    /* ******************Simulation of KOIN* transactions****************** */
}
```

CSS

styleFade.css

```
body {
    margin:0;
    padding:0;
    color: #000;
    background: #e0f0ff;
  }
  #header h1{
    text-align: center;
    /* font-family: 'Josefin Sans', sans-serif; */
  }


  #wrapper {
    margin: 0 auto;
  }

  #simulation {
    background-color: white;
    border: 2px solid black;
  }

  #footer {
    clear: both;
    text-align: center;
    font-size: 12px;
  }
```

References

1. Scott Murray *Interactive Data Visualization for the Web*. An introduction to designing with d3
2. Data-Driven Documents,
   https://d3js.org