POLYTECHNIC SCHOOL OF THE UNIVERSITY OF NANTES
COMPUTER SCIENCE DEPARTMENT

RESEARCH AND DEVELOPMENT REPORT

# SCAMPLON: A Peer Sampling Service for WebRTC

## *Gossiping with Handshakes*

**Julian TANKE &**

**November 15, 2014**

supervised by Pascal MOLLI & Brice NEDELEC

— Équipe GRIM —
LABORATOIRE D'INFORMATIQUE DE NANTES-ATLANTIQUE

coordinator: José MARTINEZ

POLYTECH NANTES

UNIVERSITÉ DE NANTES

# SCAMPLON: A Peer Sampling Service for WebRTC

## Gossiping with Handshakes

Julian TANKE &

**Abstract**

WebRTC allows web browsers to directly communicate with no server in between which enables massive, distributed applications for the web that are scalable, responsive and failure resistant. A general paradigm for information dissemination in large distributed systems is the gossip-based communication. The core of gossiping protocols is the *peer sampling service* that provide every node with random peers to exchange information with. Peer sampling services do not necessarily rely on global knowledge, instead there are many algorithms that only need local information to provide uniform random samples - for example SCAMP and CYCLON. This protocols assume that peers are directly addressable - however this is not possible in WebRTC. Instead, a handshake must be executed over a well-known signaling service whenever a new connection is established. As a central service significantly decreases the scalability we propose to carry it over into the distributed system - if this is done a node only needs to access the central service to enter the network. Nodes that are already member of the system can use other peers to function as their signaling service to establish connections to other nodes within the network - this way the protocol stays scalable.

We evaluate CYCLON and SCAMP in a handshaking environment. Furthermore, we propose SCAMPLON, a new membership protocol composite of parts of SCAMP and parts of CYCLON. SCAMPLON provides good properties and is not just applicable for WebRTC and handshaking but can also be used more generally as a generic peer sampling service.

*KEY WORDS* Membership management; peer-to-peer; epidemic/gossiping protocols; Scalability; random graphs

# Contents


**1 Introduction** **5**
- 1.1 Statement of the problem . . . 5
- 1.2 Objectives . . . 6
- 1.3 Work achieved . . . 6
- 1.4 Contribution . . . 6
- 1.5 Report organisation . . . 7

**2 Background and Motivation** **8**
- 2.1 WebRTC . . . 8
- 2.2 Massive Collaborative Editing . . . 9

**3 State of the Art** **11**
- 3.1 Gossip Protocol . . . 11
- 3.2 Peer Sampling Service . . . 12
- 3.3 SCAMP . . . 12
  - 3.3.1 Indirection . . . 13
- 3.4 CYCLON . . . 13
- 3.5 Synthesis . . . 15
- 3.6 Conclusion . . . 15

**4 Proposals** **18**
- 4.1 Combining SCAMP and CYCLON: SCAMPLON . . . 18
- 4.2 Connectivity . . . 19
- 4.3 Handshaking . . . 21

**5 Experiments and Results** **24**
- 5.1 Handshaking . . . 24

5.1.1 Convergence . . . 25
5.2 SCAMPLON . . . 29
5.2.1 Average Cluster Coefficient . . . 29
5.2.2 Average Path Length . . . 30
5.2.3 Number of Arcs . . . 30
5.3 Conclusion . . . 31

**6 Conclusion** **32**
6.1 Summary . . . 32
6.2 Outcomes . . . 33
6.3 Research directions . . . 33

**A Schedule** **39**

**B Weekly Reports** **42**

**C Self-assessment** **51**
C.1 First Half . . . 51
C.2 Second Half . . . 52

# 1 Introduction

Many efforts have been devoted to establish a data connection between browsers using the WebRTC API [1]. Services like PeerCDN [2], webtorrent [3] or Sharefest [4] demonstrate how decentralized services now can be deployed easily in web browsers. This drastically improves the user experience as they can download torrents as easy as watching a youtube video. However, current solutions do not provide a scalable broadcast. This, however, is essential for other applications such as distributed collaborative editors or distributed social networks. Traditionally, gossip protocols are used to implement broadcast in decentralized infrastructure. This protocols have been used in a various number of large scale systems serving an array of problem settings including information dissemination [DGH+87], information aggregation [JMB05] and network management [VvS03]. Peers are highly autonomous and independent, which allows the algorithms to be simple when compared to hierarchical approaches. This protocols can easily scale to millions of peers which can join and leave without significantly disrupting the service. The hearth of each gossip protocol is its fundamental distributed abstraction: the peer sampling service[JGKvS04].

1. http://w3c.github.io/webrtc-pc/
2. https://peercdn.com/
3. https://github.com/feross/webtorrent
4. https://www.sharefest.me/

## 1.1 Statement of the problem

This raises the problem of deploying gossip algorithms on WebRTC. Many gossip protocols build up upon the premise that the peers can interconnect easily with each other and, as such, constantly connect to new sets of peers[VGS05][JGKvS04]. Unfortunately, certain efforts must be taken to establish a peer-to-peer connection with WebRTC: as entities in WebRTC cannot directly address each other due to the network layout of the Internet, a handshake, consisting of an offer by one client and a responding answer by the other, must be performed. This process must be hoist by a well-known service in be-

tween, which we will call the *signaling service*. Furthermore, the handshake potentially changes the communication complexity as connecting to new peers becomes more expansive and error-prone.

## 1.2 Objectives

In this report we evaluate the cost of two well-known peer sampling services, CYCLON[VGS05] and SCAMP[GKM01], in a handshaking infrastructure [5]. We provide a technique to minimize the number of connections to the well-known signaling service, and, additionally, we propose SCAMPLON, a new algorithm that merges the advantages of CYCLON and SCAMP regarding network topology and robustness towards handshaking.

## 1.3 Work achieved

We show that CYCLON can be easily deployed in a WebRTC environment as it only communicates with other nodes in close proximity reducing the handshaking overhead. We also show that the lease-mechanism in SCAMP can be significantly disturbed under network failure [6]. We provide a WebRTC library [7] that allows nodes inside a WebRTC network to host connections between each other without the need of an external service. This minimizes the number of connections towards the signaling service as it is only needed to introduce a new node to an existing network. Furthermore, we propose an algorithm that combines the good properties of CYCLON and SCAMP and removes some of their problems. SCAMP is used to enter and remove nodes from the network, thus providing a good random starting distribution and an adaptive partial view size, but uses CYCLON to balance the graph afterwards to exploit its close-proximity exchanges.

## 1.4 Contribution

Gossip protocols that are designed for the internet have already been introduced by [GKM03] and[VJS03]. However, this protocols assume that a connection can be established with direct addressing, ignoring the limitations mentioned above. On the other side there are implementations of peer-to-peer networks on top of WebRTC [8] already available. Nonetheless they lack truly massive scalability as they rely on a single centralized signaling service to host and maintain the network. Our contribution is an algorithm, SCAMPLON, that is truly scalable in a WebRTC environment. Furthermore, the SCAMPLON-protocol is not limited for use in handshaking environments but can also be easily applied in general gossiping algorithms.

5. Direct addressing is not possible. Instead, a connection between peers must be established over a signaling service

6. See Figure 5.5

7. https://github.com/justayak/handshake

8. http://peerjs.com/

## 1.5 Report organisation

The remainder of this paper is structured as follows: Chapter 2 explains the technology in more depth and provides an example application. Chapter 3 studies the gossip protocols CYCLON and SCAMP. In Chapter 4 we will propose SCAMPLON and describe how it differs from the other two protocols. Chapter 5 first evaluates the different behaviors of SCAMP and CYCON with handshaking. We than compare SCAMPLON and CYCLON and show that our algorithm provides better overall properties under normal network conditions. Finally, the conclusion summarises the work and introduce new research issues that are worth pursuing.

# 2

# Background and Motivation

## 2.1 WebRTC

Web Real-Time Communications (WebRTC) is a client-side API to enable Real-Time Communications in web browsers [Hic]. It is an implementation of SRTP [1] with an SDP [2] control mechanism on top that can use STUN, TURN and ICE for NAT traversal.

*These APIs should enable building applications that can be run inside a browser, requiring no extra downloads or plug-ins, that allow communication between parties using audio, video and supplementary real-time communication, without having to use intervening servers (unless needed for firewall traversal, or for providing intermediary services).*

As we are interested in sending arbitrary data between peers we will focus on the DataChannel API. Unfortunately WebRTC cannot create a connection between peers without a signaling service. That means that we need some sort of communication exchange (Handshake) prior to the direct peer-to-peer connection. Figure 2.1 demonstrates how a connection is established in WebRTC:

1. Alice needs to find out her public Internet address and her network setup (Routers, Firewall). To obtain this information, she queries a well-known STUN-Server.
2. To establish a connection with Bob, she now needs to create an offer and send it to the signaling service.
3. The signaling service selects Bob and forwards Alice's offer to him.
4. We assume, that Bob already knows his network setup (as he called a STUN-Server before). Bob accepts the request that Alice sends to him and wants to communicate. To do so he must create an answer and send it back to Alice. As Bob does not know how to connect to Alice yet, he sends the answer to the signaling service instead.

1. Secure Real-time Transport Protocol
2. Sockets Direct Protocol

5. The signaling service sends the answer back to Alice.
6. Upon receiving Bob's positive answer, Alice sends a description of how the communication will be done and how Bob can send data to her. As she does not know how to directly connect to Bob yet, she sends this information to the signaling service.
7. The signaling service forwards Alice's connection information to Bob.
8. Bob checks if he can communicate in any way that Alice suggested, and, if he can, he describes how to connect directly to him. Again, this message is send to Alice through the signaling service.
9. Alice gets the information from Bob and establishes a link to him. From now on they can communicate directly with each other.

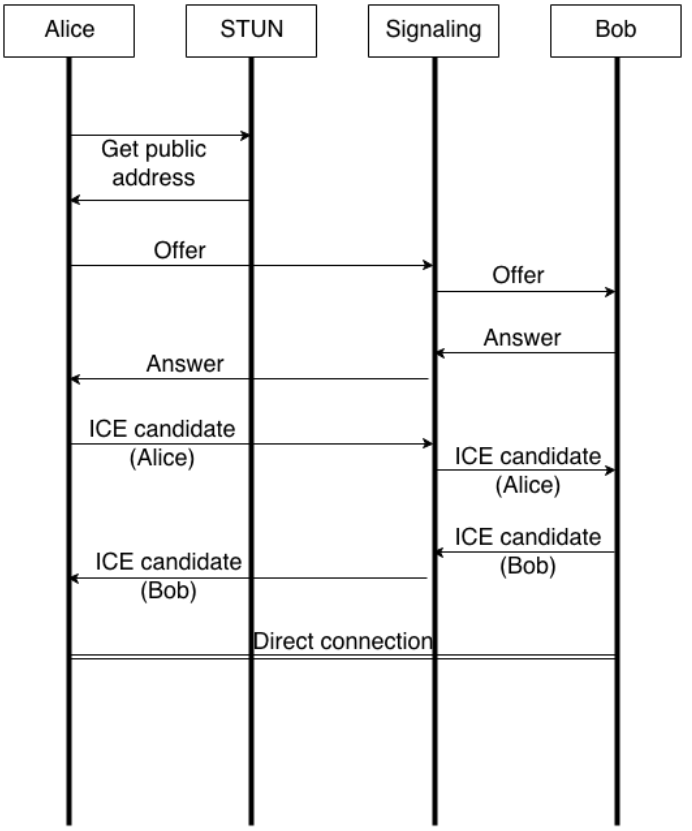



Figure 2.1: Examplary WebRTC connection

## 2.2 Massive Collaborative Editing

One application for a gossip-based WebRTC network is a massive collaborative editor [NMMD13][BN14] that runs in the web browser. Unlike current solutions such as Google Docs, Microsoft Word Online or Etherpad we want to support millions of users on one document (e.g. Google Docs only allows 50 users to work on one document). [BN14] describes how parallel editing can scale well with the number of users, implied that some sort of broadcast exists. However, no assumptions are made about how this broadcast is done. A centralized service

or a fully connected graph do not scale well with the user count. Instead, using the proposed gossip-based peer-to-peer protocol for WebRTC will make the application more scalable and robust.

# 3

# State of the Art

This chapter introduces fundamental algorithms and concepts.

## 3.1 Gossip Protocol

Gossip, also referred to as *epidemic dissemination*, is a communication protocol initially inspired by observing of how gossips spread in social networks or how diseases spread over a population. It trades reliability guarantees against scalability properties [EGH+03] and it has been applied to various large scale systems, with the purpose of information dissemination [DGH+87], information aggregation [JMB05], network management [VvS03] and many more.

The basic idea behind gossip works as follows: a network is a graph $G = (V, E)$ where $V = \{1, ..., n\}$ is a set of $n$ nodes. Let $E \subset V \times V$ denote neighboring [1] nodes and let $d_i$ denote the degree of node $i$ in $G$ where $d_i$ is constrained by the configuration parameter $c \in \mathbb{Z}_{>0}$ which is called Fanout. The Fanout defines the number of nodes that are selected when a message is broadcast from a node. A high value generates a lot of redundant messages while being more fault tolerant and reliable. $V$ and $E$ constantly change as nodes leave and join the network and nodes update there neighbors to maintain a uniform distribution.

Assuming a node $i$ wants to broadcast a message $m$ - It sends $m$ to its $d_i$ neighboring nodes. A receiving node $j$ evaluates if it already knows $m$ [Kol03] - and if not, it will repeat the previous procedure: broadcast to its $d_j$ neighboring nodes and so forth. Usually three types of epidemic dissemination of messages can be distinguished:

— **Push:** When receiving a message, a node sends it to all other nodes in its view.
— **Pull:** A node periodically queries one random other node to check for messages it has not received yet.
— **Push-Pull:** A node sends a message to another ran-

1. In this paper, nodes are considered neighbors if they are directly connected to each other

dom peer and, additionally, queries said peer for data.

When $n$ is very large, for a given node $i$ it applies that $d_i << n$. This is necessary for scalability reasons as, otherwise, the list of neighbors might exhaust a nodes memory and maintaining the list in presence of churn [2] would be too expansive. We will call $\mathcal{N}_i \equiv \{j \in V : (i,j) \in E\}$ the **partial view** of $i$. The partial view should consist of a random sample of $V$.

## 3.2 Peer Sampling Service

The peer sampling service [JGKvS04], or, random peer sampling (RPS), is a membership protocol. The purpose of RPS is to provide each peer with a *uniform sample* of $V$ in a distributed and scalable manner. It can be used in a variety of gossip-based protocols, for example in topology management [JMB09]. The peer sampling service consists of a simple API:

— **init:** Initializes the service on a given node if this has not been done before. This is implementation dependent.

— **getPeer:** Returns a peer address if the network contains more than one node. The returned address is a sample drawn from the group. This method can also be used to return multiple random peers, either by providing a paramter $n$ to the function or by simply calling the method multiple times.

2. Nodes constantly join and leave the network

There are two strategies to maintain partial views in peer sampling services [LPR] that will play an important role in our evaluation:

— **Reactive:** The partial view only changes when $V$ changes (Nodes join or leave) and remains stable otherwise. An example algorithm is SCAMP [GKM01].

— **Cyclic:** The partial view is periodically updated every $\triangle T$ time units and membership information are exchanged with neighbors. An example algorithm is CYCLON [VGS05].

## 3.3 SCAMP

The probabilistic scalable membership protocol (SCAMP)[GKM01] is a reactive peer sampling service. The size of its local views adapt automatically to a desired value of $(c+1)\log n$ where $c$ is a parameter which specifies the degree of robustness to failure([KMG03], *Theorem 1*). It maintains two separated partial views[GKM03], one for receiving gossip messages (InView $\mathcal{N}^{in}$), one for sending gossip messages (PartialView $\mathcal{N}^{out}$).

If a new node $k$ wants to join the network $G(V,E)$ it needs to have one member $i \in V$ in $\mathcal{N}_k^{out}$ and send a subscription request to $i$ [3]. On receiving a subscription request, $i$ forwards the new peer to all members of its own $\mathcal{N}_i^{out}$, and, additionally, creates $c$ copies of the new subscription and forwards them to randomly chosen

3. See 1

nodes in $\mathcal{N}_i^{out}$ [4]. On receiving a forwarded subscription for node $k$, a receiving node $j$ integrates $k$ into its $\mathcal{N}_j^{out}$ with probability $p$ which depends on the size of $\mathcal{N}_j^{out}$ ($p_i = \frac{1}{1+|\mathcal{N}_i^{out}|}$). If $j$ does not integrate $k$, it forwards it to another randomly chosen peer from its $\mathcal{N}_j^{out}$. When a node $i$ decides to keep the subscription of a node $j$, it integrates $j$ into its $\mathcal{N}_i^{out}$ and notifies $j$ to put $i$ into its $InView_j$. On leaving the network, the leaving node $j$ sends a *replace request* containing a peer from its $\mathcal{N}_j^{out}$ to $|\mathcal{N}_j^{in}| - c$ peers in its $\mathcal{N}_j^{in}$ and a *deleting request* to the remaining ones. When a node $i$ receives the *replace request* from $j$, it will replace $j$ for the sent peer in its $\mathcal{N}_i^{out}$.

A subscription has a finite lifetime called *lease* [5]. When expired, the subscribed node will be removed from all PartialViews. It is a nodes responsibility to resubscribe once its subscription expires. This mechanic ensures that failed nodes will not remain in the network. Furthermore, the lease mechanism rebalances the network making it more robust. Indeed, when a peer joins the system, it only has one node in its partial view, which makes it vulnerable to failures. However, when peers resubscribe, it adds the link to its inView and therefore, to others partial view. Hence, even the new peers get subscription requests.

Pseudocode: [GKM01], Section 2.1

4. See 2
5. The lease is implementation dependent: it could be set individually or it could be set globally

### 3.3.1 Indirection

In SCAMP, good scaling behavior is ensured when new nodes select their entry point uniformly random among the existing members [6]. However, in reality this is seldom the case. Instead it can be expected that few nodes will be designated entrances into the network. To ensure a random distribution under this circumstances SCAMP introduces *indirection* where the initial entrance node forwards the subscribers request to a node that is approximately chosen at random among the existing nodes. This is done by a random walk [7] of length $2 * (c + 1) \log n$ [8] which is the expected diameter of the graph.

## 3.4 CYCLON

CYCLON [VGS05] is a cyclic peer sampling service. Each peer maintains a partial view of size $c$ and repeatedly exchanges its view with a neighbor - this operation is called a shuffle. The protocol works as follows [9]:

Every $\triangle T$ time unit a node $i$ executes the shuffle function:

1. Increase by one the age of all neighbors.
2. Select neigbor $j$ with the highest age among all neighbors, and $l - 1$ [10] other random neighbors.

6. [GKM01], 3.1
7. the exact algorithm is out of the scope of this report
8. constant defined in the paper [GKM01]
9. Enhanced Shuffling, described in [VGS05]
10. $1 \leq l \leq c$

**Algorithm 1** Subscription management in SCAMP

**procedure** ONSUBSCRIBE($subscriber$)
    **for** each $peer$ in $\mathcal{N}^{out}$ **do**
        send **forwardedSubscription** with $subscriber$ to $peer$
    **end for**
    **for** each $peer$ in sample($\mathcal{N}^{out}$, $c$) **do**
        send **forwardedSubscription** with $subscriber$ to $peer$
    **end for**
**end procedure**

**Algorithm 2** Handling of a forwarded subscription in SCAMP

**procedure** ONFORWARDEDSUBSCRIPTION($subscriber$)
    $keep \leftarrow$ randomChoiceBetween0and1()
    $keep \leftarrow \lfloor(|\mathcal{N}^{out}| + 1) * keep\rfloor$
    **if** $keep \equiv 0$ and $subscriber \notin \mathcal{N}^{out}$ **then**

        // potentially very expensive as the handshake must be traced over
        // possibly many hops
        $\mathcal{N}^{out} \leftarrow \mathcal{N}^{out} \cup \{subscriber\}$

    **else**
        $peer \leftarrow$ sample($\mathcal{N}^{out}$, 1)
        send **forwardSubscription** with $subscriber$ to $peer$
    **end if**
**end procedure**

3. Replace $j$'s entry with a new enty of age 0 and with $i$'s address.
4. Send the updated subset to peer $j$
5. Receive from $j$ a subset of no more than $l$ of its own entries.
6. Discard entries pointing at $j$ and entries already contained in $i$'s partial view.
7. Update $i$'s partial view to include all remaining entries, by firstly using empty slots (if any), and secondly replacing entries among the ones sent to $j$

On receiving the shuffle request from $i$ at $j$, $j$ sends back a random subset of $l$ of its neighbors and updates its own partial view with the sent data but it does not increase the age of the items in the partial view. When a node $i$ does not get a response from a node $j$ after it sent a shuffle request it will discard node $j$ from its partial view as it seems to be failed. Because of this, failed nodes will be removed from the network in bounded time. As in SCAMP, if a node wants to join the network, it needs to know another node that is already part of it.

## 3.5 Synthesis

The main goal for using gossip based algorithms with WebRTC is to allow the Network to scale to vast sizes. To fulfill this properties the protocol needs to relay (mostly) on local information and make use of a partial view that adapts automatically to the full network size to guaranty efficient information dissemination. Additionally, as the system needs to employ handshaking due to WebRTC, the protocol must reduce the number of messages needed to establish a connection between two nodes. This is important as each additional node in between reduces the probability of successfully establish a connection. This restriction implies that nodes for exchanging information should be chosen from the temporary close neighborhood. Table 3.5 shows that the two introduced algorithms each provide one feature while lacking the other.

| properties | SCAMP | CYCLON |
|---|---|---|
| adaptive size | Yes | No |
| close proximity to neighbors | No | Yes |
| divergence speed | slow | fast |

Furthermore, we want the algorithm to diverge all its partial view sizes to the optimal value as fast as possible. Here, CYCLON is clearly better as its $\triangle T$ can be much smaller than the lease time in SCAMP allows [11].

## 3.6 Conclusion

All mentioned algorithms have certain problems and limitations when handshaking is added. In CYCLON the value of $\triangle T$ must be higher than the time the peers need to connect. This could be implemented by setting $CONNECTION-TIMEOUT < \triangle T$. This will increase the time a news will take to be fully disseminated

11. if the lease time is too small, too many peers would rejoin in the same moment thus disconnecting the graph

**Algorithm 3** Active thread in CYCLON

1: **while** termination condition not reached **do**
2: wait $\triangle T$
3: $partialView \leftarrow$ increaseAge($partialView$)
4: $peer \leftarrow$ max(orderByAge($partialView$))
5: $rand \leftarrow$ sample($partialView \backslash \{peer\}$ , $l-1$) $\cup$[addr:$ownAddress$, age:0]
6: send $rand$ to $peer$
7: $otherRand \leftarrow$ receive from $peer$
8: $otherRand \leftarrow otherRand \backslash \{partialView \cup \{[addr : ownAddress, age : *]\}\}$
9: $partialView \leftarrow partialView \backslash rand$
10:
11: // this part is very expensive with handshake
12: $partialView \leftarrow partialView \cup otherRand$
13:
14: **while** $|partialView| < c$ **do**
15: // shift removes the first element of an list
16: $partialView \leftarrow shift(orderByAge(rand))$
17: **end while**
18: **end while**

**Algorithm 4** Passive thread in CYCLON

```
1:  procedure OnExchange(peer, otherRand)
2:      rand ← sample(partialView, l)
3:      send rand to peer
4:      otherRand ← otherRand\partialView
5:      partialView ← partialView\rand
6:
7:      // this part is very expensive with handshake
8:      partialView ← partialView ∪ otherRand
9:
10:     while |partialView| < c do
11:         // shift removes the first element of an list
12:         partialView ← shift(orderByAge(rand))
13:     end while
14: end procedure
```

into the network by constant time. We show that, other than that, the protocol remains relatively stable and performs reliable as the added overhead is only constant.

SCAMP, on the other hand, must employ more complex engineering as its connecting contract is much more complicated. The added complexity for SCAMP is not constant but rather depending on the network size. This results in SCAMP being more fragile which is even increased by its lease-mechanism which can be shown in our experiments (5.4).

# 4

# Proposals

Our research question states: *What is the cheapest handshake gossip algorithm?* We consider an algorithm to be cheaper than another, if, on average, it employs less handshakes and as few open connections as possible. This also counts the number of peers in an *intermediate handshake*. We consider a handshake to be *intermediate* if a peer $i$ cannot directly hoist a connection between two nodes $a$ and $b$ but instead needs to apply $m, m < 0$ intermediate nodes to establish a connection. This is necessary due to the requirement to reduce the connections to a centralized signaling server as much as possible and, instead, use the decentralized peer-to-peer network to carry out the signaling service, as soon as a node is member of the system. Additionally, we want our partial view to stay adaptive towards the network size.

## 4.1 Combining SCAMP and CYCLON: SCAMPLON

We propose to combine SCAMP and CYCLON into one protocol to exploit both its advantages and to remove both its issues. SCAMP will be used to introduce new peers into the network and to remove them if they wish to leave. Instead of SCAMP's lease mechanism CYCLON's shuffle protocol will take over and balance out the partial views. To allow this, the aging process from CYCLON is applied. As the protocol cannot rely on constant values for $c$ and $l$ from CYCLON anymore those need to be calculated on the fly: $l = \left\lceil \frac{|partialView|}{2} \right\rceil$, SCAMP guarantees that $|partialView| \geq 1$. The partial view size $c$ might change whenever a shuffle is performed as the algorithm will adapt its size towards the neighbor it exchanges data with. It is defined by the size of the partial view subtracted by the size of the subset sent to the partner peer plus the size of the subset received from the partner peer. In the long run, this will result in all peers

having similar sized partial views.

Suppose that node $a$ wants to exchange its partial view with node $b$ and that the graph is created with the previously described SCAMP algorithm. The resulting partial view at $a$ must be of size $c_a = |partialView_a| - l_a + l_b$ while the partial view of $b$ must be of size $c_b = |partialView_b| - l_b + l_a$ after the shuffling process. Nodes with an even initial partial view size will have $\left\lceil \frac{|a|+|b|}{2} \right\rceil$ peers in the resulting view while nodes with an odd sized initial partial view will have a new view with length $\left\lfloor \frac{|a|+|b|}{2} \right\rfloor$.

A remarkable difference regarding CYCLON is that it is explicitly allowed to have the same node multiple time in ones partial view. On the other hand, it must be ensured that, after each shuffle, the partial view has exactly the size as described above. This is important to keep the number of arcs in the graph constant to the value created by SCAMP. A shuffle executes the following steps:

1. Increase by one the age of all neighbors.
2. Select neighbor $b$ with the highest age among all neighbors, and $l_a - 1$ other random neighbors.
3. Replace $b$'s entry with a new enty of age 0 and with $a$'s address.
4. Send the updated subset to peer $b$
5. Receive from $b$ a subset of $l_b$ random peers from its partial view.
6. Exchange entries pointing at $a$ with nodes that were previously send to $b$ (ordered by age). If not enough nodes are available to exchange all arcs pointing to $a$ create new entires that point to $b$. It is essential that the size of this set does not change.
7. Remove $b$ and the $l_a - 1$ sent peers from $a$'s partial view.
8. Join together the remaining partial view and add the updated received list sent from $b$. If the partial view was odd at the beginning it must be of size $\left\lceil \frac{|a|+|b|}{2} \right\rceil$ now, otherwise $\left\lfloor \frac{|a|+|b|}{2} \right\rfloor$.

The basic algorithm to connect a node to the network is described in the previous chapter about SCAMP in Algorithm 1 and Algorithm 2. The algorithms for shuffling are described in Algorithm 5 and Algorithm 6.

## 4.2 Connectivity

As the protocol is based on CYCLON it also inherits CYCLONS connectivity properties [1] that states that, given a fail-free environment, the connectivity of the network is guaranteed. That means that no node becomes disconnected as a result of a shuffling operation. We show that even with the variable partial view size the graph stays connected when subject to an shuffle:

1. [VGS05], 3.1.

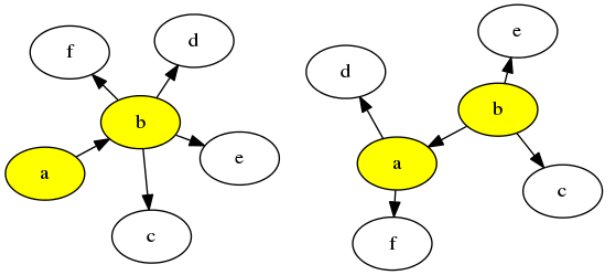



Figure 4.1: Shuffle from $a$ to $b$. The left graph represents before and the right graph represents after the shuffle. The link is inverted and $b$ transfers some of its peers to $a$.

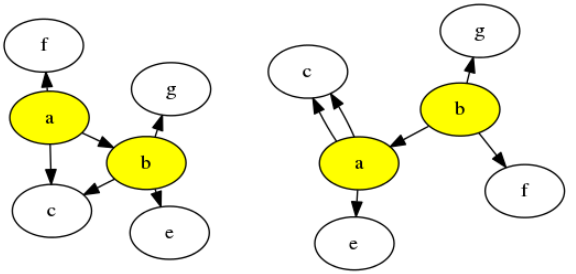



Figure 4.2: Shuffle from $a$ to $b$. The left graph represents before and the right graph represents after the shuffle. The link is inverted and $b$ and $a$ exchanged random peers. $a$ must keep alive all the links, even though they point to the same node.

When two nodes shuffle, the connecting arc is inverted. Additionally, the bigger[2] peer will transfer some of its node to the smaller one to seek for an size equilibrium.

When two subsets $A$ and $B$ are connected by at least one link, the link will be passed around inside the subsets keeping the subsets connected or will simply invert when the linked nodes shuffle thus keeping the graph connected.

We can also show that the property of holding multiple links to the same peer, as described in figure 4.2, will not affect the overall network in the long run.

The graph stays connected as no link is destroyed. The double connection will be resolved eventually as $c$'s entries in $a$'s partial view age over time which will result in $a$ exchanging partial views with $c$. As $c$ cannot hold links to itself, $a$ will be supplied by a fresh set of peers different from $c$.

When two nodes reference each other we call this a loop (A loop is described in figure 4.3). We must show that they will ultimately be resolved as they yield undesirable properties regarding the clustering coefficient which badly impacts the resilience of the graph towards network failure.

Fortunately, loops can be broken both from within the loop as well as from outside. In the internal break, one of the loop peers $p_{loop}$ has other non-looping members in its partial view. In deterministic ( $\log n$ shuffles) time, this non-looping node will be selected for shuffling. This

2. the node with more elements in its partial view

potentially removes some or all looping arcs from $p_{loop}$. Alternatively, the outside link can be passed inside the loop and potentially replace the pointers that create the loop.

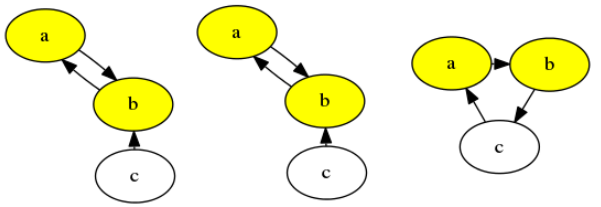



Figure 4.3: Shuffle from $a$ to $b$. An external node breaks the loop.

When at least one member of the loop is inside a partial view from another peer inside the graph, the loop can be broken from outside, as described in figure 4.3. The external link potentially replaces one looping link thus either reducing or removing the loop.

## 4.3 Handshaking

As we want to use SCAMPLON in a WebRTC environment we must evaluate its behavior when applied to handshaking. As our tests show, SCAMP‘s lease mechanism breaks [3] when handshaking is involved and a fail-free network cannot be guaranteed. On the other hand, CYCLON just adds a constant overhead on top of its complexity but otherwise stays resistant towards network failure letting it perform similar to its non-handshaking implementation. When facing a potentially erroneous network it is advised to use a $c$ value $c > 0$ for the introduction part of SCAMPLON to ensure that the number of arcs is around $(c+1)n \log n$. After the hurdle of introduction is solved, handshaking will play only a minor role as the CYCLON‘ic part of the protocol will take over which is more adapted towards handshaking as it only makes use of neighbors in close proximity.

3. See Figure 5.5

**Algorithm 5** Active thread in SCAMPLON

**while** Termination condition not reached **do**
  wait $\triangle T$
  $l \leftarrow \max\{\left\lceil \frac{|partialView|}{2} \right\rceil, 1\}$
  $p \leftarrow$ the peer of this node
  $partialView \leftarrow$ increaseAge($partialView$)
  $q \leftarrow$ max(orderByAge($partialView$))
  $sample \leftarrow$ sample($partialView \backslash \{q\}$,$l-1$) $\cup$\{node: $p$, age:0\}
  send $sample$ to $q$
  $sample$‘ $\leftarrow$ receive from $q$
  $partialView \leftarrow partialView \backslash sample$
  **if** $p \in sample$‘ **then**
    $sent \leftarrow sample \backslash \{$\{node: $p$ age:$*$\}, \{node: $q$, age: $*$\}\}
    $sizeBefore \leftarrow |sample$‘$|$
    $sample$‘ $\leftarrow sample$‘$\backslash$\{node: $p$ age:$*$\} // remove all possible links to our node
    $removeCount \leftarrow sizeBefore - |sample$‘$|$
    **for** $i \rightarrow removeCount$ **do**
      **if** $|sent| > 0$ **then**
        $sample$‘ $\leftarrow sample$‘$\cup$ pop(orderByAge($sent$)) // fill up with the youngest items
      **else**
        $sample$‘ $\leftarrow sample$‘ $\cup$ \{node:$q$ age: 0\} // create new links to the oldest node
      **end if**
    **end for**
  **end if**
  $partialView \leftarrow partialView \cup sample$‘
**end while**

**Algorithm 6** Passive thread in SCAMPLON

1: **procedure** ONEXCHANGE($p$, $q$, $sample$‘) // $p$ = local peer, $q$ = sending peer, $sample$‘ = sample of $q$‘s partial view
2: $l \leftarrow \max\{\left\lceil \frac{|partialView|}{2} \right\rceil, 1\}$
3: $sample \leftarrow$ sample($partialView$, $l$)
4: send $sample$ to $peer$
5: $partialView \leftarrow partialView \backslash sample$
6: **if** $p \in sample$‘ **then**
7: $sent \leftarrow sample \backslash \{\{$node: $p$ age:$*\}, \{$node: $q$, age: $*\}\}$
8: $sizeBefore \leftarrow |sample‘|$
9: $sample‘ \leftarrow sample‘ \backslash \{$node: $p$ age:$*\}$ // remove all possible links to our node
10: $removeCount \leftarrow sizeBefore - |sample‘|$
11: **for** $i \rightarrow removeCount$ **do**
12: **if** $|sent| > 0$ **then**
13: $sample‘ \leftarrow sample‘ \cup$ pop(orderByAge($sent$)) // fill up with the youngest items
14: **else**
15: $sample‘ \leftarrow sample‘ \cup \{$node:$q$ age: $0\}$ // create new links to the other node
16: **end if**
17: **end for**
18: **end if**
19: $partialView \leftarrow partialView \cup sample‘$
20: **end procedure**

# 5 Experiments and Results

All simulations were conducted using PeerSim Simulator [1]. Existing implementations of CYCLON and SCAMP could be used and adapted for handshaking-based simulation.

The first part of the experiment revolves around handshaking with CYCLON and SCAMP while the second part of the experiments compares the new algorithm SCAMPLON with CYCLON.

We define a *cycle* to be the time period during which all nodes executed one gossiping function, which are defined more precisely in Algorithm 1, 2, 3, 4, 5 and 6. From this follows that it takes two cycles in CYCLON to complete a shuffle - the first cycle handles the shuffle initialization (3), the second cycle handles the shuffle response (4). SCAMP on the other hand needs a minimum of three cycles to complete a subscription: the first cycle handles indirection, the second one the subscription (1) and the last one handles the multiple forwarded subscriptions (2). However, SCAMP's subscription can easily consume more than three cycles as both recursive functions, indirection as well as forward, greatly depend on the network size and thus may include more hops and, as such, more cycles.

Peers communicate between each other with messages. We also test the behavior of the algorithms against a specific failing quota of these messages meaning that a certain percentage of them will not reach their target peer. With this experiments we hope to get some insight into the failure resistance of the different protocols.

## 5.1 Handshaking

In this section we will compare SCAMP and CYCLON with their handshaking counterparts. We want to evaluate how handshaking influences the algorithms properties. Furthermore, we want to find out what error rates can be tolerated by the handshaking implementations. The experiments with CYCLON are done with $c = 20$ and $l = 8$ while the experiments with SCAMP are done with

1. http://peersim.sourceforge.net/

$c = 2$. All simulations where run with 1000 nodes. The network is initialized as a line, that means: $n_0$ is connected to $n_1$ is connected to $n_2$ and so on.

### 5.1.1 Convergence

We want to evaluate how fast SCAMP and CYCLON converge to their ideal properties, given that both algorithms are initialized in non-optimal ways.

We use two metrics to determine if a network is converged or not: the *clustering coefficient* and the *average path length*. In the following we describe the experiments in more detail:

#### Clustering Coefficient

The *clustering coefficient* is a measure for a graphs tendency to contain cliques as well as for its transitivity. We calculate the average clustering coefficient by summing together all local clustering coefficients which then gets divided by the number of nodes $n$.

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$$

The local clustering coefficient $C_i$ defines how close a node and its immediate neighbors are to be a clique.

In general, it is not desired for a peer-to-peer network to have a high average clustering coefficient as it weakens the connectivity of the cluster to the rest of the graph which increases the chance of partitioning. Another disadvantage of a highly clustered network is that information dissemination will be more redundant.

Figure 5.1 shows the average clustering coefficient with CYCLON, run over 2500 cycles. The x-axis shows the average cluster coefficient while the y-axis shows the number of cycles. Both implementations, with and without handshaking, behave very similar when no network errors occur. It should be noted that the handshaking algorithm converges towards a slightly higher limit than the non-handshaking version. When network errors are introduced, the handshaking variant degrades faster than the non-handshaking version.

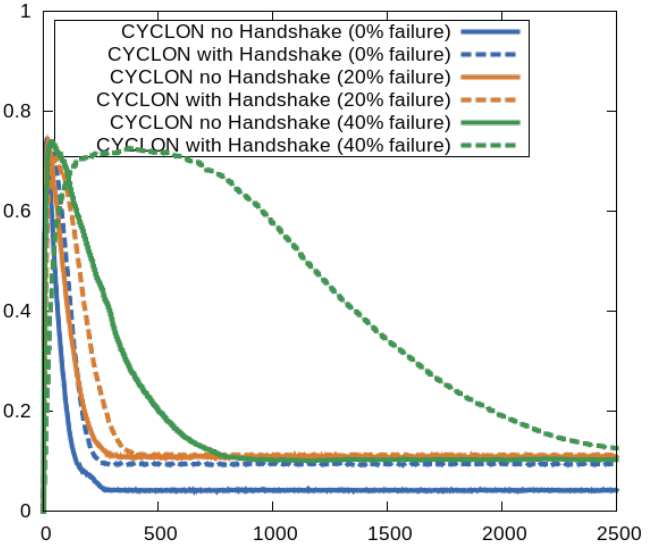



Figure 5.1: average clustering of the CYCLON protocol. The x-axis denotes the number of cycles while the y-axis denotes the cluster coefficient.

Figure 5.2 show the average clustering coefficient with SCAMP, run over 100000 cycles (note that the x-axis is scaled by $(*100)$). The first thing to note is that SCAMP needs notably more cycles to converge than CYCLON. While CYCLON converges after roughly 200 cycles, SCAMP fully converges only after around 20000 cycles.

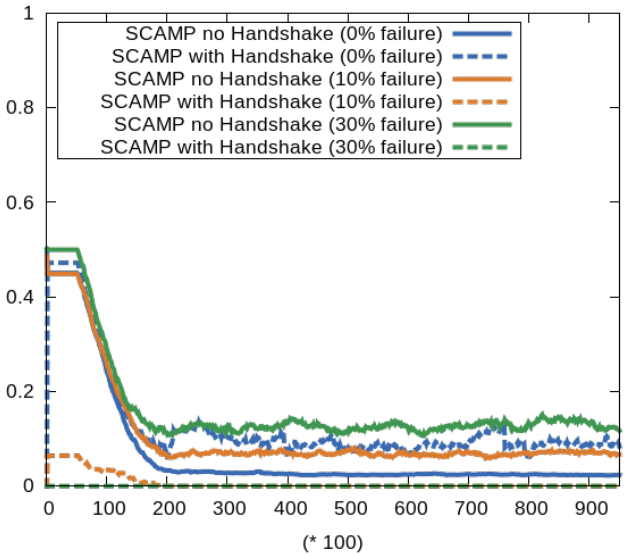



Figure 5.2: average clustering of the SCAMP protocol. The x-axis denotes the number of cycles while the y-axis denotes the cluster coefficient.

SCAMP relies on its lease mechanism to rebalance the graph where a node has a certain lifespan and, after the lifespan is over, it gets removed from other peers view, removes itself from the network and reintroduces itself through a node in its partial view.

SCAMP with handshaking converges slightly slower and to a higher value than the non-handshaking version when all messages are delivered. However, when errors are introduced, the handshaking version yields lower values than its non-handshaking version. When the error rate is set to 30% SCAMP‘s handshaking version does not deliver any useful values anymore.

There are several explanations for the results: as handshaking needs more hops, heck has more sources of error, it is expected that those versions perform worse than the non-handshaking protocols meaning that they take longer to converge to optimal properties and that their optimal properties are not as good as those generated with non-handshaking protocols. CYCLON‘s results perfectly follow these expectations even when network failure is present. It can be stated that handshaking CYCLON behaves robust and very similar to its non-handshaking counterpart. Furthermore, it converges to its ideal state relatively fast as few cycles are needed to fill up a peers partial view. When the partial views are filled up CYCLON can be considered random thus providing properties of random graphs. SCAMP‘s non-handshaking version provides good results even when network errors are present. Furthermore, it converges to a smaller average clustering coefficient than CYCLON, which is preferred. This can be explained by the smaller number of peers in its partial views: as there are fewer interconnections there are also fewer redundant connections thus there are less

cliques. In contrast, SCAMP converges very slowly (after around 20000 cycles) when compared to CYCLON as its lifespan must be selected carefully to avoid large chunks of the network to leave in parallel. In the simulation we obtained this behavior by assigning random values over a range that is bigger than the number of peers. Handshaking with SCAMP, on the other hand, does not does not adapt to message loss. Even with low failure rates the graph quickly degrades until it is not connected anymore. Another observation is that the handshaking SCAMP becomes unreliable when network errors are introduced. The reason for this is that, with every lease and join on average more arcs are destroyed than created. This slowly degrades the graph until it gets disconnected. The graph is completely degenerated and does not connect nodes anymore (see: Figure 5.2) after a certain time.

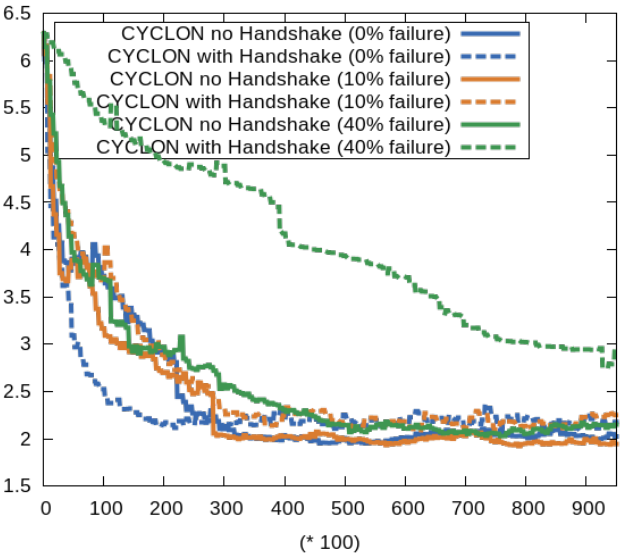



Figure 5.3: average path length of the CYCLON protocol.

### Average Path Length

The *average path length* is the average of the shortest path length between nodes in the graph. It counts the hops to reach a node from a given source. We want the average path length to be small so that the whole network can be reached as fast as possible.

Figure 5.3 shows the average path length with CYCLON, run over 1000 cycles. The y-axis denotes the path length while the x-axis denotes the cycle.

Similar to the clustering coefficient, the average path length diverges slower when handshaking is involved. When the network error rate increases, the handshaking version becomes significantly wider than the non-handshaking one.

Figure 5.4 shows the average path length with SCAMP, run over 100000 cycles. Similar to the average cluster coefficient the handshake version degrades quickly when network failures are introduced.

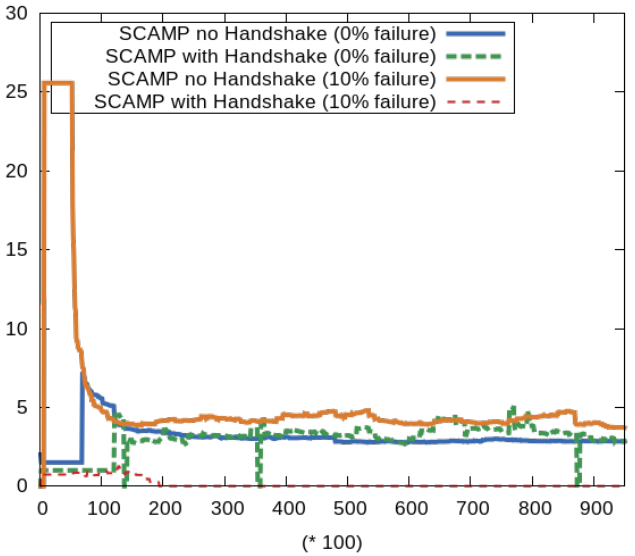



Figure 5.4: average path length of the SCAMP protocol. The y-axis denotes the path length in nodes and the x-axis denotes the number of cycles multiplied by 100.

The non-handshaking version handles increasing error rates stabilizing at similar values. Without network failure, Handshaking SCAMP stabilizes at a similar length as the non-handshaking version - a notable difference are little gaps that sometimes occur that are not present in the non-handshaking variant.

Figure 5.5 shows the connectivity of the SCAMP protocol. The y-axis denotes the connectivity (ranging from 0 to 1) while the x-axis denotes the number of cycles multiplied by 100. SCAMP without handshaking almost always has a high connectivity level - it sometimes drops for a while but recovers shortly after. When errors are introduced the path length quickly degrades to 0 suggesting a disconnected graph.

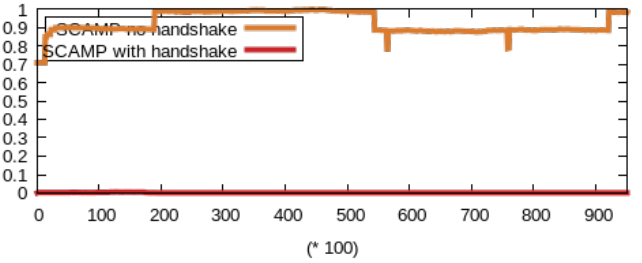



Figure 5.5: average connectivity of the SCAMP protocol (10% failure). The x-axis denotes the number of cycles multiplied by 100 while the y-axis denotes the reaching quota of the network.

The results mirror those from the *average clustering coefficient*: CYCLON seems to perform very similar with or without handshaking. Only a significant error rate degrades the handshaking version clearly. SCAMP on the other hand quickly degrades: the reasons for this stays the same: as the lease and rejoin need to to perform many hops, this also represents many potential sources of error. Furthermore, as messages travel along the hops they get delayed, thus making nodes waiting for messages to arrive longer than in the non-handshaking version - as we are only evaluating 10 random nodes [2] out of the network

2. due to performance reasons

to calculate the average path length this explains the gaps in Figure 5.4.

## 5.2 SCAMPLON

SCAMPLON, being a cyclic peer sampling service, naturally shares more similarities and properties with CYCLON than with SCAMP. Because of that we compare those two protocols. The configuration value for SCAMPLON is $c$, which has the same purpose as the $c$ in SCAMP. In this simulation, $c = 3$. Handshaking will not be used. As previous experiments have shown, it only adds constant complexity to CYCLON, and, as SCAMPLON shares the same philosophy, this can be transfered to it as well.

### 5.2.1 Average Cluster Coefficient

Figure 5.6 compares the average clustering coefficient from SCAMPLON and CYCLON over the run of 400 cycles. We can observer that SCAMPLON delivers a significantly better clustering coefficient than CYCLON when the failure rate is low but when the failure rate exceeds a certain turning point[3] CYCLON starts to yield better results. The reason for this is that SCAMPLON's partial view has an almost ideal size of $\log n$ while CYCLON's partial view with a value of $c = 20$ exceed this optimum by far thus resulting in a higher interconnection which in turn results in a higher clustering degree. However, when the failure rate increases, CYCLON outperforms SCAMPLON as it has more links that it can use as backup when others fail.

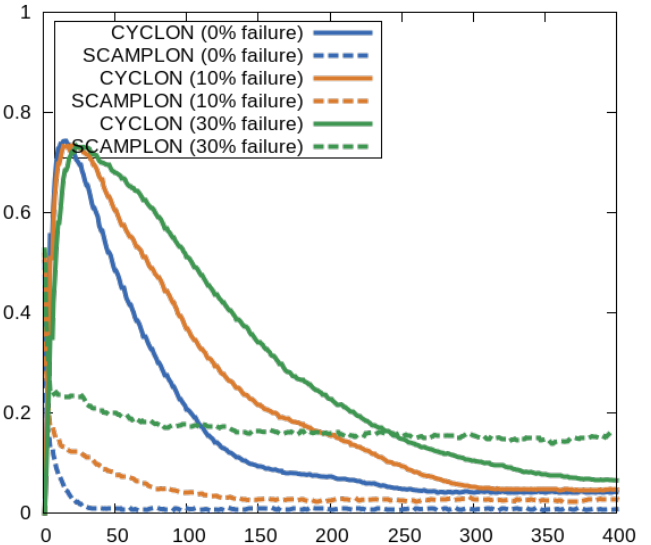



Figure 5.6: average cluster coefficient of the SCAMPLON and the CYCLON protocol. The y-axis denotes the clustering coefficient while the x-axis denotes the number of cycles.

Another positive property of SCAMPLON is the fast convergence rate[4] which is magnitudes lower than at CYCLON's. This can be explained by two properties:

3. In our experiments this was around 23% failure rate

4. here after around 25 cycles

first: smaller partial views saturate faster, and second: SCAMP‘s subscription protocol is used for SCAMPLON which results in a better initial distribution of the network thus giving it a head start for the convergence rate. Figure 5.6 underpins this theory as SCAMPLON has a very low maximum value when compared to CYCLON.

### 5.2.2 Average Path Length

Figure 5.7 compares the average path length between SCAMPLON and CYCLON. As we observed previously SCAMPLON converges faster than CYCLON due to the smaller partial views and the better initial distribution thanks to SCAMP‘s initialization.

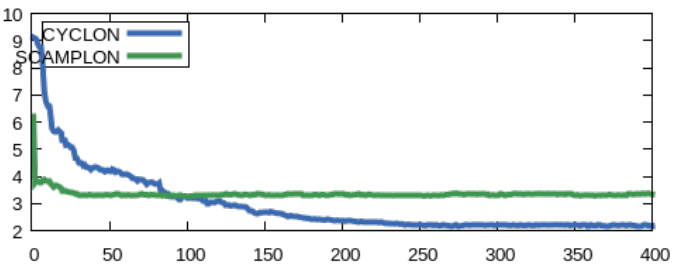



Figure 5.7: average path length of the SCAMPLON and the CYCLON protocol. The y-axis denotes the number of hops while the the x-axis denotes the number of cycles.

However, in the long run CYCLON delivers a better average path length. The reason for this is that CYCLON maintains numerous links more than SCAMPLON and thus is better interconnected with other nodes which in turn results in a lower diameter of the graph.

### 5.2.3 Number of Arcs

Figure 5.8 compares the number of arcs between SCAMPLON and CYCLON. The y-axis denotes the number of total edges in the graph while the x-axis denotes the time in cycles. The result yields no surprise as CYCLON has a bigger partial view and thus is interconnected better and thus has a higher count of arcs. CYCLON‘s arc count converges to $|E_{CYCLON}| \approx c_{CYCLON} * n$. SCAMPLON on the other hand has a constant number of arcs that are established to $|E_{SCAMPLON}| \approx (c_{SCAMPLON} + 1)n \log n$ by the initial SCAMP-protocol. The shuffle function cannot and must not change this value. A smaller number of arcs has the advantage of reducing the numbers of redundant messages sent in the network thus improving the used bandwidth. On the other hand, a high count of arcs can result in shorter paths and thus in faster message delivery. It also improves the resilience against failures.

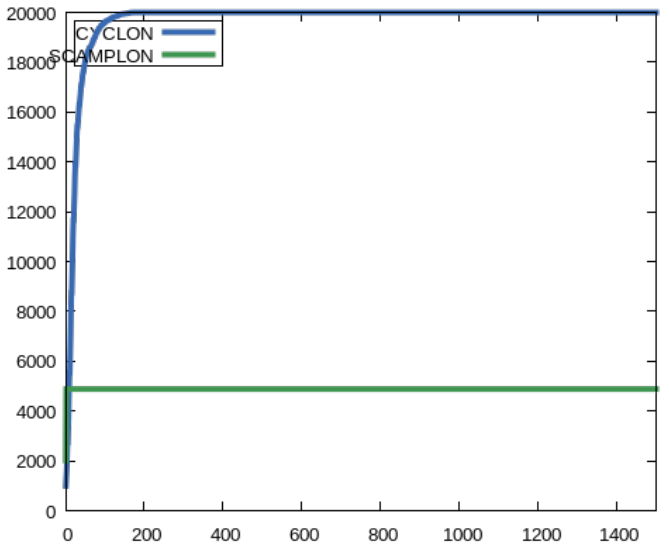



Figure 5.8: Number of arcs in SCAMPLON and CYCLON. The x-axis denotes the time passed in cycles while the y-axis denotes the arc count.

## 5.3 Conclusion

We see that our proposed algorithm delivers desirable results which often even exceeds the qualities of CYCLON when the failure rate is low. Beyond that, SCAMPLON's adaptive nature gives him additional advantages when compared to traditional cyclic protocols with fixed partial view sizes. The adaptiveness helps the protocol to be able to virtually always reach the whole network but also to keep the traffic and redundancy low - features that traditional cyclic protocols lack.

However, further research must be conducted to evaluate SCAMPLON's behavior under churn [5] to observe the interplay of SCAMP's subscription protocol with SCAMPLON's cyclic algorithm. Additionally, a handshake version must be investigated to validate that the non-handshaking properties can be transfered as it was assumed in this report. Last, the protocol should be implemented in WebRTC to evaluate it under real network conditions. The code for the experiments can be found on github [6].

5. Nodes dynamically leaving and joining the network
6. https://github.com/justayak/peersim-1.0.5

# 6 Conclusion

The experiments with SCAMPLON provide good result showing that it is a potential candidate for our WebRTC peer sampling service. Another good candidate is CYCLON as it can easily be adapted for handshaking with which it performs reliable and robust even with network failures.

## 6.1 Summary

Our study can be divided in three parts: First, we wanted to find an efficient and scalable peer sampling service for WebRTC. As we want to avoid the use of central services as much as possible and instead use local knowledge whenever possible we had to enable the distributed network to fulfill tasks usually handled by a central server. This task is the handshaking that is mandatory to connect two WebRTC clients with each other. We implemented handshaking in WebRTC [1] and concluded that this task can successfully be handled in a distributed manner.

Secondly, we compared two well known peer sampling services, SCAMP and CYCLON, regarding their properties when adapted to handshaking mechanics. We concluded that CYCLON can handle handshaking fairly well even when in erroneous networks. SCAMP on the other hand quickly degenerates and thus is not a good choice when handshaking should be executed by the protocol. However, we found SCAMP‘s adaptive view size very desirable as it significantly reduces the messaging overhead when used for information dissemination.

This led to the third step: the creating of SCAMPLON, a cyclic peer sampling service that tries to combine the good properties of SCAMP‘s adaptive partial view size, CYCLON‘s robustness regarding handshaking and CYCLON‘s fast convergence rate. Our experiments show that SCAMPLON satisfies the desired properties.

1. https://github.com/justayak/handshake

## 6.2 Outcomes

The outcome of our research is the new cyclic peer sampling algorithm SCAMPLON. It combines the two protocols SCAMP and CYCLON, is simple and, most of the time, it makes use of peers with close proximity making it a good candidate for a handshaking WebRTC environment.

## 6.3 Research directions

Further research must be devoted to SCAMPLON‘s behavior under churn. Additionally, we must evaluate how the failure of full nodes affect the network as there is no lease mechanism to repair the overall structure. So far, the unsubscription was not tested which is an important part of the protocol. And, least, the protocol must be implemented in WebRTC to see how it behaves under network conditions.

# Bibliography


[Abo] Feross Aboukhadijeh. webtorrent. https://github.com/feross/webtorrent. Accessed: 2014-11-27.

[BN14] Achour Mostefaoui Brice Nedelec, Pascal Molli. A scalable sequence encoding for massive collaboratice editing. 2014. 9

[DGH+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, pages 1–12, New York, NY, USA, 1987. ACM. 5, 11

[EGH+03] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, November 2003. 11

[GKM01] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, NGC '01, pages 44–55, London, UK, UK, 2001. Springer-Verlag. 6, 12, 13

[GKM03] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, February 2003. 6, 12

[Hic] Ian Hickson. Webrtc 1.0: Real-time communication between browsers. http://w3c.github.io/webrtc-pc/. Accessed: 2014-11-22. 8

[JGKvS04] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '04, pages 79–98, New York, NY, USA, 2004. Springer-Verlag New York, Inc. 5, 12

[JH] Abi Raja John Hiesey, Feross Aboukhadijeh. Peercdn. https://peercdn.com/. Accessed: 2014-11-27.

[JMB05] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM*

*Trans. Comput. Syst.*, 23(3):219–252, August 2005. 5, 11

[JMB09] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *Comput. Netw.*, 53(13):2321–2339, August 2009. 12

[JS02] MÃ¡rk Jelasity and Maarten Van Steen. Large-scale newscast computing on the internet. Technical report, 2002.

[KMG03] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):248–258, March 2003. 12

[Kol03] Boris Koldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *In Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS '03*, pages 76–85. IEEE, 2003. 11

[LPR] JoÃ£o LeitÃ£o, JosÃ© Pereira, and LuÃs Rodrigues. Gossip-based broadcast. 12

[MJ] Alberto Montresor and Mark Jelasity. Peersim: A scalable p2p simulator.

[NMMD13] Brice Nédelec, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils. LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing. In *13th ACM Symposium on Document Engineering (DocEng)*, pages 37–46, Florence, Italy, September 2013. 10 pages. 9

[Pee] Peer5. Sharefest. https://www.sharefest.me/. Accessed: 2014-11-27.

[VGS05] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13:2005, 2005. 5, 6, 12, 13, 19

[VJS03] Spyros Voulgaris, Márk Jelasity, and Maarten Van Steen. A robust and scalable peer-to-peer gossiping protocol. In *In 2nd Int'l Workshop Agents and Peer-toPeer Computing, LNCS 2872*, pages 47–58. Springer, 2003. 6

[VvS03] Spyros Voulgaris and Maarten van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In Marcus Brunner and Alexander Keller, editors, *DSOM*, volume 2867 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2003. 5, 11

# List of Figures


2.1 Examplary WebRTC connection . . . . . 9

4.1 Shuffle from $a$ to $b$. The left graph represents before and the right graph represents after the shuffle. The link is inverted and $b$ transfers some of its peers to $a$. . . . . 20
4.2 Shuffle from $a$ to $b$. The left graph represents before and the right graph represents after the shuffle. The link is inverted and $b$ and $a$ exchanged random peers. $a$ must keep alive all the links, even though they point to the same node. . . . . 20
4.3 Shuffle from $a$ to $b$. An external node breaks the loop. . . . . 21

5.1 average clustering of the CYCLON protocol. The x-axis denotes the number of cycles while the y-axis denotes the cluster coefficient. . . . . 25
5.2 average clustering of the SCAMP protocol. The x-axis denotes the number of cycles while the y-axis denotes the cluster coefficient. . . . . 26
5.3 average path length of the CYCLON protocol. . . . . 27
5.4 average path length of the SCAMP protocol. The y-axis denotes the path length in nodes and the x-axis denotes the number of cycles multiplied by 100. . . . . 28
5.5 average connectivity of the SCAMP protocol (10% failure). The x-axis denotes the number of cycles multiplied by 100 while the y-axis denotes the reaching quota of the network. . . . . 28
5.6 average cluster coefficient of the SCAMPLON and the CYCLON protocol. The y-axis denotes the clustering coefficient while the x-axis denotes the number of cycles. . . . . 29
5.7 average path length of the SCAMPLON and the CYCLON protocol. The y-axis denotes the number of hops while the the x-axis denotes the number of cycles. . . . . 30
5.8 Number of arcs in SCAMPLON and CYCLON. The x-axis denotes the time passed in cycles while the y-axis denotes the arc count. . . . . 31

A.1 Drafting . . . . . 40
A.2 Planning . . . . . 41

# List of Tables

B.1 Advance of the project with respect to the planned minimal theoretical time (respectively, a high involvement) . . . . . 50

# List of Algorithms


1 Subscription management in SCAMP . . 14
2 Handling of a forwarded subscription in SCAMP . . . . . 14
3 Active thread in CYCLON . . . . . 16
4 Passive thread in CYCLON . . . . . 17
5 Active thread in SCAMPLON . . . . . 22
6 Passive thread in SCAMPLON . . . . . 23

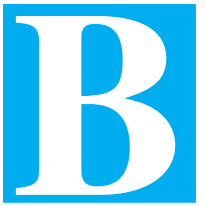
# Schedule

Figure A.1 shows the draft schedule establish *a priori*...

Figure A.2 introduce the planning that has been build week after week during the course of the work.

Figure A.1: Drafting

Figure A.2: Planning



# Weekly Reports

**Weekly Report #1**
**From September 29, 2014 to October 04, 2014**

Working time of Julian TANKE: 6 h 00 m

Working time of  : 0 h 00 m

**Work done.**

- Studying paper *LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing*
- Studying paper T-Man: Gossip-based Overlay Topology Management

**Next week planning.**

- Researches to be conducted

**Weekly Report #2**
**From October 06, 2014 to October 10, 2014**

Working time of Julian TANKE: 8 h 30 m

Working time of  : 0 h 00 m

**Work done.**

- Paper: The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations
- Paper: Gossip-based broadcast protocols
- Technical evaluation: WebRTC
- Technical evaluation: PeerJS

**Work not yet done.**

**Communication with the client.**

**Next week planning.**

- Researches to be conducted
- Testimplementations with WebRTC

**Weekly Report #3**
**From October 13, 2014 to October 18, 2014**

Working time of Julian TANKE: 17 h 00 m
Working time of : 0 h 0 m
**Work done.**
— Testing PeerJs -> Implementation of Chat prototype
— Papers: Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail
**Work not yet done.**
**Communication with the client.**
**Next week planning.**
— prototype for Peer-Samping-Service
— prototype for T-Man

**Weekly Report #4**
**From October 20, 204 to October 25, 2014**

Working time of Julian TANKE: 22 h 30 m
Working time of : 0 h 00 m
**Work done.**
— Implementing prototype for Peer-Samping-Service (RSP) based on PeerJS
— Paper: A Robust and Scalable Peer-to-Peer Gossiping Protocol
— Paper: SCAMP: Peer-to-peer lightweight membership service for large-scale group communication
— Buffer Management in Probabilistic Peer-to-Peer Communication Protocols
**Work not yet done.**
— Prototype for T-Man: Implementation of RSP took longer than expected.
**Communication with the client.**
**Next week planning.**
— prototype for T-Man

**Weekly Report #5**
**From October 27, 2014 to 01 November, 2014**

Working time of Julian TANKE: 6 h 00 m
Working time of : 0 h 00 m
**Work done.**
— Implementation of T-Man with PeerJS (needs RPS)
**Work not yet done.**
— still some problems with RPS (create new unit tests)
**Communication with the client.**
**Next week planning.**
— Researches to be conducted

— testing prototype for T-Man

---

**Weekly Report #6**
**From November 03, 2014 to November 06, 2014**

---

Working time of Julian TANKE: 14 h 00 m
Working time of : 00 h 00 m
**Work done.**
— Paper: CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays
— Test: Implementation of T-Man with PeerJS (needs RPS)
**Work not yet done.**
**Communication with the client.**
**Next week planning.**
— Researches to be conducted

---

**Weekly Report #7**
**From November 10, 2014 to November 15, 2014**

---

Working time of Julian TANKE: 20 h 00 m
Working time of : 0 h 00 m
**Work done.**
— Paper: Lightweight Probabilistic Broadcast
— Start writing report: "A Scalable Network Topology for Massive Collaborative Editing"
**Work not yet done.**
**Communication with the client.**
**Next week planning.**
— Start with a draft of the report
— Researches to be conducted

---

**Weekly Report #8**
**From November 17, 2014 to November 22, 2014**

---

Working time of Julian TANKE: 30 h 30 m
Working time of : 0 h 00 m
**Work done.**
— Write Introduction of Paper: "Topology Management for Massive Collaborative Editing"
— Change the reasearch topic: We figured that some fundamental issues MUST be addressed before the old topic (Topology Management). New Topic: Handshaking gossiping, as this is essential for the implementation.
**Work not yet done.**
**Communication with the client.**

**Next week planning.**
— Write the report

---

**Weekly Report #9**
**From November 24, 2014 to November 28, 2014**

---

Working time of Julian TANKE: 30 h 30 m
Working time of : 0 h 00 m
**Work done.**
— Write report: "Massive Broadcasting with WebRTC"
**Work not yet done.**
**Communication with the client.**
**Next week planning.**
— Evaluating the different algorithms under WebRTC

---

**Weekly Report #10**
**From December 1, 2014 to December 5, 2014**

---

Working time of Julian TANKE: 30 h 30 m
Working time of : 0 h 0 m
**Work done.**
— Start implementing Handshaking-Prototype in WebRTC
— Start implementing CYCLON-Prototype in WebRTC (based on the Handshaking-Prototype)
— A SCAMP-Prototype for WebRTC is developed in parallel by Brice Nedelec
**Work not yet done.**
— Implementation Newscast
**Communication with the client.**
— We switched the PeerSim implementation with the actual WebRTC implementation to better fit into the current workflow with the collaborative editor.
**Next week planning.**

---

**Weekly Report #11**
**From December 8, 2014 to December 12, 2014**

---

Working time of Julian TANKE: 10 h 0 m
Working time of : 0 h 0 m
**Work done.**
— Bugfixing Handshaking-Prototype in WebRTC
— Bugfixing in CYCLON WebRTC - massive problems with concurrency as the shuffling function must be split in two parts (sending and receiving).
**Work not yet done.**

— Implementation Newscast

**Communication with the client.**

**Next week planning.**

---

**Weekly Report #12**
**From December 15, 2014 to December 19, 2014**

---

Working time of Julian TANKE: 4 h 30 m

Working time of : 0 h 0 m

**Work done.**

— Bugfixing Handshaking-Prototype in WebRTC
— Bugfixing in CYCLON WebRTC - massive problems with concurrency as the shuffling function must be split in two parts (sending and receiving).

**Work not yet done.**

— Implementation Newscast, We decided to skip this algorithm as it was only used to show a "poorly" performing algorithm

**Communication with the client.**

— We dropped Newscast from our research

**Next week planning.**

---

**Weekly Report #13**
**From December 22, 2014 to December 26, 2014**

---

Working time of Julian TANKE: 0 h 00 m
Working time of : 0 h 00 m
**Work done.**
**Work not yet done.**

— Bugfixing in CYCLON WebRTC - massive problems with concurrency as the shuffling function must be split in two parts (sending and receiving).

**Communication with the client.**
**Next week planning.**

---

**Weekly Report #14**
**From December 29, 2014 to January 2, 2015**

---

Working time of Julian TANKE: 0 h 0 m
Working time of : 0 h 0 m
**Work done.**
**Work not yet done.**

— Bugfixing in CYCLON WebRTC - massive problems with concurrency as the shuffling function must be split in two parts (sending and receiving).

**Communication with the client.**
**Next week planning.**

**Weekly Report #15**
**From January 5, 2015 to January 9, 2015**

Working time of Julian TANKE: 3 h 00 m

Working time of : 0 h 0 m

**Work done.**

— Bugfixing in CYCLON WebRTC - massive problems with concurrency as the shuffling function must be split in two parts (sending and receiving). Current solution: Locking.

**Work not yet done.**

— Applying testing framework for WebRTC to measure the protocols. We decided to freeze this part and remove it from the paper as the WebRTC implementations (SCAMP and CYCLON) where more complex than expected

**Communication with the client.**

— Start implementing SCAMP (no handshakes) in PeerSim
— Start implementing SCAMP (handshakes) in PeerSim
— Start implementing CYCLON (no handshakes) in PeerSim
— Start implementing CYCLON (handshakes) in PeerSim

**Next week planning.**

**Weekly Report #16**
**From January 12, 2015 to January 16, 2015**

Working time of Julian TANKE: 14 h 30 m
Working time of : 0 h 0 m
**Work done.**

— Start implementing CYCLON (handshakes) in PeerSim
— implementing CYCLON (no handshakes) in PeerSim

**Work not yet done.**
**Communication with the client.**

— Start implementing SCAMP (no handshakes) in PeerSim
— Start implementing SCAMP (handshakes) in PeerSim

**Next week planning.**

— Start measuring the PeerSim implementations

**Weekly Report #17**
**From January 19, 2015 to January 24, 2015**

Working time of Julian TANKE: 25 h 30 m
Working time of : 0 h 0 m
**Work done.**

— Start implementing CYCLON (handshakes) in PeerSim
— implementing CYCLON (no handshakes) in PeerSim

**Work not yet done.**

**Communication with the client.**

**Next week planning.**

— Start implementing SCAMP (no handshakes) in PeerSim
— Start implementing SCAMP (handshakes) in PeerSim
— Start implementing SCAMPLON in PeerSim
— Measuring the PeerSim implementations
— Write report

**Work not yet done.**

**Communication with the client.**

**Next week planning.**

— Bufixes SCAMP (no handshakes) in PeerSim
— Bufixes SCAMP (handshakes) in PeerSim
— Bufixes CYCLON (no handshakes) in PeerSim
— Bufixes CYCLON (handshakes) in PeerSim
— Bugfixes implementing SCAMPLON in PeerSim
— Measuring the PeerSim implementations
— Write report

---

**Weekly Report #18**
**From January 26, 2015 to January 30, 2015**

---

Working time of Julian TANKE: 15 h 30 m
Working time of : 0 h 0 m

**Work done.**

— Start implementing SCAMP (no handshakes) in PeerSim
— Start implementing SCAMP (handshakes) in PeerSim
— Bufixes CYCLON (no handshakes) in PeerSim
— Bufixes CYCLON (handshakes) in PeerSim
— Start implementing SCAMPLON in PeerSim

---

**Weekly Report #19**
**From February 2, 2015 to February 6, 2015**

---

Working time of Julian TANKE: 35 h 30 m
Working time of : 0 h 0 m

**Work done.**

— Bufixes SCAMP (no handshakes) in PeerSim
— Bufixes SCAMP (handshakes) in PeerSim
— Bufixes CYCLON (no handshakes) in PeerSim
— Bufixes CYCLON (handshakes) in PeerSim
— Measuring the PeerSim implementations
— Write report

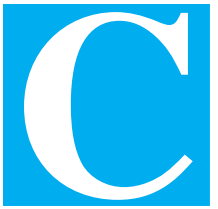
**Work not yet done.**

**Communication with the client.**
**Next week planning.**

Table B.1 summarises the rate at which the projects advances. Let us recall that the "planned minimal theoretical time" equals the time indicated in the programme plus 20% of additional time corresponding to personal work, i.e., outside the timetable. The high limit corresponds to 50% of aditional personal work.

| | Planned Time | | Julian TANKE | | | | | |
|---|---|---|---|---|---|---|---|---|
| | low | high | weekly | Σ | % | weekly | Σ | % |
| Week | h : m | h : m | h : m | h : m | | h : m | h : m | |
| 1 | 10 : 00 | 12 : 30 | 6 : 00 | 6 : 00 | 60 (48) | 0 : 00 | 0 : 00 | 0 (0) |
| 2 | 20 : 00 | 25 : 00 | 8 : 30 | 14 : 30 | 72 (58) | 0 : 00 | 0 : 00 | 0 (0) |
| 3 | 30 : 00 | 37 : 30 | 17 : 00 | 31 : 30 | 105 (84) | 0 : 0 | 0 : 00 | 0 (0) |
| 4 | 40 : 00 | 50 : 00 | 22 : 30 | 54 : 00 | 135 (108) | 0 : 00 | 0 : 00 | 0 (0) |
| 5 | 50 : 00 | 62 : 30 | 6 : 00 | 60 : 00 | 120 (96) | 0 : 00 | 0 : 00 | 0 (0) |
| 6 | 60 : 00 | 75 : 00 | 14 : 00 | 74 : 00 | 123 (98) | 00 : 00 | 0 : 00 | 0 (0) |
| 7 | 70 : 00 | 87 : 30 | 20 : 00 | 94 : 00 | 134 (107) | 0 : 00 | 0 : 00 | 0 (0) |
| 8 | 80 : 00 | 100 : 00 | 30 : 30 | 124 : 30 | 155 (124) | 0 : 00 | 0 : 00 | 0 (0) |
| 9 | 90 : 00 | 112 : 30 | 30 : 30 | 155 : 00 | 172 (137) | 0 : 00 | 0 : 00 | 0 (0) |
| 10 | 100 : 00 | 125 : 00 | 30 : 30 | 185 : 30 | 185 (148) | 0 : 0 | 0 : 00 | 0 (0) |
| 11 | 110 : 00 | 137 : 30 | 10 : 0 | 195 : 30 | 177 (142) | 0 : 0 | 0 : 00 | 0 (0) |
| 12 | 120 : 00 | 150 : 00 | 4 : 30 | 200 : 00 | 166 (133) | 0 : 0 | 0 : 00 | 0 (0) |
| 13 | 130 : 00 | 162 : 30 | 0 : 00 | 200 : 00 | 153 (123) | 0 : 00 | 0 : 00 | 0 (0) |
| 14 | 140 : 00 | 175 : 00 | 0 : 0 | 200 : 00 | 142 (114) | 0 : 0 | 0 : 00 | 0 (0) |
| 15 | 150 : 00 | 187 : 30 | 3 : 00 | 203 : 00 | 135 (108) | 0 : 0 | 0 : 00 | 0 (0) |
| 16 | 160 : 00 | 200 : 00 | 14 : 30 | 217 : 30 | 135 (108) | 0 : 0 | 0 : 00 | 0 (0) |
| 17 | 170 : 00 | 212 : 30 | 25 : 30 | 243 : 00 | 142 (114) | 0 : 0 | 0 : 00 | 0 (0) |
| 18 | 180 : 00 | 225 : 00 | 15 : 30 | 258 : 30 | 143 (114) | 0 : 0 | 0 : 00 | 0 (0) |
| 19 | 190 : 00 | 237 : 30 | 35 : 30 | 294 : 00 | 154 (123) | 0 : 0 | 0 : 00 | 0 (0) |

Table B.1: Advance of the project with respect to the planned minimal theoretical time (respectively, a high involvement)



# Self-assessment

## C.1 First Half

1. report;
2. oral presentation;
3. results.

This allows to evaluation your own level of satisfcation at the end of the first part of the project, consisting of:

1. preliminary study;
2. bibliographic study;
3. general design of a solution.

You can discuss in some details these various points.

**General remark**

While trying to implement certain Algorithms with the chosen technology we noticed a signifcant problem in the initial project boundaries. The initial research topic was tackling topology management in a peer-to-peer network. The goal was to create a single network for managing multiple documents for a massive collaborative editor - topolgy management and social network algorithms would make sure that the number of unnecassary messages is low and dissemination is fast.

However, this algorithms are build up on basic gossip protocols (like peer sampling service) and while implementing prototypes we realized that this basic algorithms will behave different in the infrastructure we want run and that they change the complexity of a connection. This lead us to abandon the initial research topic and instead focus on this new, elementary problem.

**preliminary study**

...

**bibliographic study**

As said in *General remark*, we changed the research topic mid-project. Luckily most of the papers where still relevant, as the major topic is still gossiping.

**general design of solution**

There is no solution so far, rather, we have to evaluate which of the gossiping algorithms hold up best when applied with handshake.

## C.2 Second Half

I want to remark that I totally underestimated the complexity of implementing the gossiping algorithms in real world applications. While in the papers the implementation seems to be easy in reality many more factors must be considered, *especially* when it comes to parallelism. This hit me twice: first in the WebRTC implementation (which is much more complex than the PeerSim-Java implementation) but also in the PeerSim implementation -> This is way it took me way longer than I planned initially.

Unfortunatly, I could not implement and measure the behavior of the algorithms on WebRTC yet. However, the outcome of the new Algorithm SCAMPLON was a positive surprise as such a good result was net expected - as it allows double/triple/... edges towards the same peer we were expecting it to degrade and and become quite clustered with high clustering coefficient: but the opposite was the case! The next step must be to transform the Algorithm into a handshaking version and validate that its properties stay true. Than it can be transformed into WebRTC and further tests can be applied.

Last but not least: I was fighting with the latex and I could not fix the following issues: I could not remove the second "Writer" and, in Abstract, the Keywords are heavily misaligned.