Optimizing resource sharing in node-capacitated overlay networks

Jimmy Leblet, Fabio Pianese, Gwendal Simon

Optimizing Resource Sharing in Node-Capacitated Overlay Networks

Jimmy Leblet Fabio Pianese* Gwendal Simon
Computer Science Department
Institut TELECOM; TELECOM Bretagne
France

Abstract

A frequently occurring problem in the field of distributed systems is to determine an allocation of a generic resource so that the demand of every node in the system is fulfilled. Node-capacitated graphs, *i.e.* graphs where the weight of a node corresponds to the amount of resources it can give to the system, provide an appealing model that can be readily applied to real-world applications, such as peer-to-peer and grid-based systems. In this paper, we propose a model for this problem and show that it can be reduced to a problem of maximizing a flow in a bipartite network. We then describe a distributed fault-tolerant algorithm which allows each peer to compute the allocation of its resources using only local knowledge. Finally, we show that computing the maximal flow in a bounded-degree graph is NP-complete, which means that optimizing resource sharing in bounded-degree overlays is still an open problem.

1 Introduction

In this paper, we put forth a generic model for resource allocation in node-capacitated networks. Our model considers an oriented *underlay* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each node x in $\mathcal{V}$ is characterized by an amount of *resources* it owns $r(x)$ and a demand $d(x)$, and where each edge in $\mathcal{E}$ can support the transmission of any quantity of these resources from one node to another. Resources are an abstract representation of some capability, such as data storage, upload bandwidth, power, *etc.* We will assume that resources can be partitioned and that there exists an integer unit of measure (*e.g.* the byte for storage resource). In economics, these resources are referred to as *rival* resources [3]. We aim to build, on top of this underlay, an oriented weighted *overlay* graph $G = (V, E, w)$ where:

1. all nodes of the underlay are comprised in the overlay ($V = \mathcal{V}$)
2. a non-empty subset of the underlay links are used as the edges of the overlay ($E \subseteq \mathcal{E}$), and
3. the weight $w(e)$ associated with an edge $e = (x \rightarrow y) \in E$ corresponds to the amount of resources that are given by node x to node y.

We focus on the hardest case where the global amount of demand is equal to the owned resources: $\sum_{x \in \mathcal{V}} d(x) = \sum_{x \in \mathcal{V}} r(x)$. More precisely, we focus on a *family of weight functions* $\mathcal{W}$ such that

*The Author was visiting researcher at TELECOM Bretagne at the time this work was performed.

(i) the sum of weights of edges from a node x is equal to $r(x)$, that is every node delivers all the resources it owns, formally $\sum_{e=(x\rightarrow\cdot)} w(e) = r(x)$ and (ii) the quantity of resources received by a peer x corresponds to its *demand* $d(x)$, formally $\forall x \in \mathcal{V}, \sum_{e=(\cdot\rightarrow x)} w(e) = d(x)$. Any weight function in $\mathcal{W}$ may be seen as a complete allocation of resources from peers to other peers, so is a basis for an optimal resource sharing into what thus forms an *overlay*. The challenge is to determine a weight function fulfilling every demand in a distributed manner.

1.1 Applicative Goals

A number of problems related to networked systems involve at some level the allocation of a scarce resource. We will now focus our attention on the applicative interest of two particular sub-families of $\mathcal{W}$, which allow a ready application of this model in several well-known contexts.

In the first sub-family of $\mathcal{W}$ that we examine, called *perfectly reciprocal* and noted W_{reci}, the demand of nodes is equal to the resources they own, that is $\forall x \in \mathcal{V}, d(x) = r(x)$. In short, an overlay fulfilling these demands guarantees that a node receives exactly the same amount of resources it gives. A typical application is a form of distributed data backup system where, due to external constraints, the scarce rival resource is the storage space [7]. The goal of each entity x in the system is storing a finite amount of data $d(x)$ over the other entities. As storage space $r(x)$ can only be provided by individual users of the system, the system-wide storage capacity cannot exceed the sum of the capacities $R = \sum_{x\in\mathcal{V}} r(x)$ that the users make available, *i.e.* $\sum_x d(x) \leq \sum_x r(x)$. In this scenario, any weight allocation in W_{reci} provides a fair constraint on the relationship between the resources a node requires and offers. Such allocation, proportional to the individual contribution, is expected to build an appealing enforcement mechanism that could address the problem of free-riders raised in [8]. Such policies can be obviously extended to similar applications and contexts: for instance, in the case of Video-on-Demand (VoD) streaming in cooperative settings [21], it may be required that nodes which provide a larger amount of upload capacity $r(x)$ be served video data with a higher priority. The use of a W_{reci} weight allocation policy on the available upload capacity could produce effects that are similar to those achieved by tit-for-tat incentive policies [5].

The second sub-family of $\mathcal{W}$, called *equally shared* and noted W_{equa}, considers that each node receives exactly the same amount of resources. As W_{equa} is in $\mathcal{W}$, an equally shared weight function guarantees that the amount of resources received by a node is equal to the sum of the resources owned by all nodes R divided by the number of nodes in $\mathcal{G}$. An interesting scenario is the dimensioning and control of power grid infrastructures, especially "microgrids" (MG) [17]. These emergent examples of decentralized power generation and distribution systems can operate both as part of the standard power grid and as isolated systems. Apart from the economic and environmental advantages of local power generation, the decentralized operation of power sources may dampen the effect of large-scale power outages [1]. MGs are a collection of power generator equipment (solar panel, wind turbines, batteries, *etc.*) and loads, interconnected by power lines. Each "node" x of the MG can thus be characterized by a power output $r(x)$ and a power demand $d(x)$. When disconnected from the global power grid, the MG has to satisfy the usual condition $\sum_x d(x) = \sum_x r(x)$. In such a scenario, which we may imagine as an emergency situation, a desirable allocation strategy would try to split the available power equally among all the nodes of the MG: the use of any weight allocation in W_{equa} would enable the controller of each node to route its power output to its grid neighbors according to this egalitarian strategy. This allocation strategy could be also interesting when extended to other applications, such as live video streaming in cooperative peer-to-peer network environments. Basically, a live streaming system strives to allocate the upload capacity $r(x)$

available at each peer so that every peer in the system can receive the video at its original full rate $d(x)$, which is the same for all peers.

1.2 Related Work

Most problems related with weighted graphs describe the weights on edges as distances (*travelling salesman problem*), capacities (*maximum flow problem*), costs (*minimum spanning tree*), etc. On the other hand, vertex-weighted graphs are quite less common. Here is a set of selected works. The *facility location problem* [6] considers the cost of transportation of commodities (which is represented by weights on edges) and the cost for locating facilities (which are represented by weights on vertices). The goal is to choose the best locations for facilities, *i.e.* the locations minimizing the overall cost. The *maximum balanced connected partition problem* [4] looks for a partition of a vertex weighted graph into two sub-graphs so that both subgraphs are connected and the overall weight is optimally balanced on both subgraphs. As described in [11], the generalization of this partitioning problem, called *node capacitated graph partitioning problem*, can be used to model memory requirement in compiler design application, the number of finite element meshes in finite element computation or the cell size in the design of electronic circuits.

Several recent studies consider node-capacitated graphs, especially in relation to the problem of routing when a capacity constraint must be upheld at the nodes. For example, [13] provides a strongly polynomial-time algorithm for the node capacitated ring routing problem and [16] presents upper bounds for oblivious routing in undirected networks with node capacities. Other studies focus on the problem of broadcasting live streams of data in unstructured network with node capacity constraints. For instance, [20] proposes a fully distributed algorithm for the *node capacitated broadcast problem* and show that this algorithm achieves the optimal rate in some network classes. This work is probably the closest in scope to ours, but our goal differs as we investigate the more generic problem of local resource sharing where each peer can be both a source and a sink.

A number of studies have dealt with overlay building on top of a given underlay in a distributed manner. The predominant approach considers a selfish setting in which every peer determines the best "rewiring" strategy to optimize some local utility function. These efforts, pioneered by [10], define a *network creation game* in which nodes have to establish links to distant nodes at a minimum cost while maximizing a distance-based quality of service metrics (also, the NP-hardness of the problem is thereby proved). Finally, [18] defines the selfish neighbor selection problem and provides some heuristics based on the k-median problem on asymmetric distance when the out-degree of a peer (that is the number of connections it can establish) is bounded.

1.3 Summary

We first show that the problem of building an optimal overlay on top of a given node-capacitated underlay can be reduced to a problem of maximizing a flow into a network. It is well known that this problem can be solved in polynomial time in a centralized manner. However, few distributed algorithms have been proposed to solve this problem. The first contribution of this paper, described in Section 2 is a distributed algorithm especially designed for our problem. This algorithm does not require a knowledge of the number of active participants and is self-stabilizing. Moreover, we show that it will return a valid solution, if a solution exists, for any given demand function.

Then we study bounded outdegree overlays. We argue that an explicit limitation of the number of active connections at every peer is among the most expected features when building an overlay.

We prove in Section 3 that building a bounded-outdegree maximal flow over a given underlay graph is unfortunately NP-complete. This result is the second main contribution of our paper and means that the problem of building an optimal bounded-outdegree overlay on top of a given underlay is still open, as our reduction to a maximal flow problem cannot be applied in this case.

2 Algorithm

2.1 Preambles and Notations

In a maximal flow problem, the goal is to find the maximal value that a flow between a single source and a single sink can achieve in a network where each edge $(x \to y)$ has a maximal nominal capacity $c((x \to y))$. The two most famous algorithms that achieve an optimal solution are the Ford-Fulkerson [12] and the Edmonds-Karp [9] algorithms. These algorithms have a time complexity in $\mathcal{O}(m \cdot f)$ and $\mathcal{O}(n \cdot m^2)$ respectively, where n is the number of vertices of the flow network, m the number of edges and f the value of the maximal flow.

For a network $N = (V, E, c)$, a flow $f : E \to \mathbb{R}^+$ is said to be valid if we have $f(e) \leq c(e)$ for every arc $e \in E$ and $\sum_{e=(\cdot \to x)} f(e) = \sum_{e=(x \to \cdot)} f(e)$ for every node $x \in V$ which is neither the source nor the sink. The *residual graph* of this flow is defined as the weighted directed graph $\mathcal{R}_f = (V, E_f, u)$ of remaining capacities on arcs of N. An arc $e_f = (x \to y)$ belongs to E_f if:

- either e_f belongs to E and $f(e_f) < c(e_f)$ and then capacity $u(e_f)$ is equal to $c(e_f) - f(e_f)$
- or the arc $e_f^{rev} = (y \to x)$ belongs to E and $f(e_f^{rev}) \neq 0$, then capacity $u(e_f)$ is $f(e_f^{rev})$.

If there exists a path from the source to the sink in $\mathcal{R}_f$ then we can add to the flow along this path a positive value of flow equal to the minimum capacity of arcs along this path. The flow is maximal when no such path exists anymore. The Ford-Fulkerson algorithm consists of computing $\mathcal{R}_f$, then seeking a path in the residual graph, finally increasing the flow along this path and do it again until there is no path from the source to the sink in $\mathcal{R}_f$. The algorithm begins with the null flow which is obviously valid.

2.2 Reduction to Maximal Flow Problem

We now give some definitions that allow to transform the problem of the overlay creation into a max-flow problem. With an oriented overlay $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a resource distribution function r and a demand function d, we associate a *network* $N(\mathcal{G}, r, d) = (V', E', c)$. The set V' contains a sink q, a source s and, for every peer $x \in \mathcal{V}$, two vertices x^+ and x^-. Let V^+ be the set $\{x^+ : x \in \mathcal{V}\}$ and $V^- = \{x^+ : x \in \mathcal{V}\}$. Formally, we have $V' = V^+ \cup V^- \cup \{s, q\}$.

The set of oriented edges E' linking the vertices of the network can be seen as three distinct subsets. The first one contains n edges from the source to each vertex x^+. The capacity of an edge $(s \to x^+)$ is the amount of resources $r(x)$ the peer x can supply. The second one includes n edges from each vertex x^- to the sink. The capacity does here depend on the demand $d(x)$. For example, a weight function w in W_{reci} aims to fulfill $(x^- \to q)$ equal to $r(x)$ for all x, although it is $\frac{R}{n}$ for a weight function in W_{equa}. Finally, in the third subset of edges, we assign one edge from x^+ to y^- if there is an edge $(x \to y)$ in the overlay graph. The capacity of this edge is infinite[1]. Thus we can

[1]This transformation can cover a peer-to-peer system where some links are constrained by setting the capacity to the limitation of this link instead of ∞.

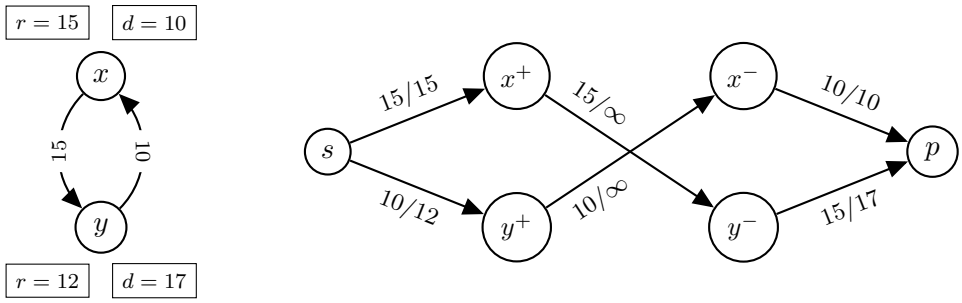

Figure 1: Transformation of an overlay containing two peers x and y.

define E' as $E' = \{(s \to x^+), (x^- \to q) : x \in \mathcal{V}\} \cup \{(x^+ \to y^-) : (x \to y) \in \mathcal{E}\}$. A transformation of a basic overlay containing only two peers to its associated network is represented in Fig. 1.

weight function $w \in \mathcal{W}$ to apply to the overlay $\mathcal{G}$. Indeed, let f be a flow achieving the maximum value. The function w can be defined as, for every arc $(x \to y) \in \mathcal{E}$, let $w(x \to y)$ be equal to $f(x^+ \to y^-)$. The amount of exchanged resources by w is then exactly the value of f.

For any demand distribution d such that $\sum_{x \in \mathcal{V}} d(x) \leq \sum_{x \in \mathcal{V}} r(x)$, the maximal value of a maximal flow is equal to the sum of demands because the capacity of the links from the nodes in V^+ to the nodes in V^- is infinite. Therefore, by the definition of the flow conservation, if the value of f is equal to the sum of demands, we obtain that w reaches the maximal demands. More generally, any maximal flow f of $N(\mathcal{G}, r, d)$ allows to determine an associated weight function for $\mathcal{G}$ such that the demand $d(x)$ for every x in $\mathcal{V}$ is fulfilled if and only if the value of f is the sum of demands. In other words, an optimal resource allocation can be immediately deduced from a computation of the maximum flow, which is known to be done in polynomial time.

In a maximal flow problem, the goal is to find the maximal value that a flow between a single source and a single sink can achieve in a network where each edge $(x \to y)$ has a maximal nominal capacity $c(x \to y)$. The two most famous algorithms that achieve an optimal solution are the Ford-Fulkerson [12] and the Edmonds-Karp [9] algorithms. These algorithms have a time complexity in $\mathcal{O}(m \cdot f)$ and $\mathcal{O}(n \cdot m^2)$ respectively, where n is the number of vertices of the flow network, m the number of edges and f the value of the maximal flow.

Such a transformation from a problem related with node-weighted graphs to a maximal flow on edge-weighted graphs is not unusual. Typically, several recent studies have used the same method for the problem of routing [13, 16] or broadcasting [20] when a capacity constraint must be upheld at the nodes. We emphasize however that the problem we tackle here has never been formulated with such a graph-based model where each vertex is associated with two weights. On the contrary, works dealing with close problems have used powerful but costly techniques, such as integer programming, to provide approximate algorithms [19]. In comparison, this elegant though simple model gives exact solutions in polynomial time.

2.3 Algorithm

Exploiting the full capacity of a given peer-to-peer overlay would be of real interest if it could be achieved without global view by peers themselves. In other words, we are looking for a distributed algorithm able to cope with transient peers and to rely only on local information from immediate neighborhood. Basically, known distributed algorithms for the max-flow problem are based on

either the Ford-Fulkerson method, but only for synchronous systems to our knowledge [2, 14], or the preflow-push method [15]. In their seminal article, Goldberg and Tarjan emphasize that their algorithm admits efficient distributed and parallel implementations. However, it is neither tolerant to faults nor scalable. In particular, two criteria are required to end the preflow-push algorithm, but both require to know the number of active nodes. On the other hand, the Ford-Fulkerson algorithm does not require such global knowledge, hence it appears as a more appealing basis, though it has to be substantially adapted to fit with our model.

We describe now a distributed and self-stabilizing algorithm that allow peers to compute a global optimal resource allocation. This algorithm requires the only knowledge of the immediate neighborhood, and is able to eventually reach a global optimal state from any configuration. The originality of our algorithm is that, according to Ford-Fulkerson principle, not only each peer tries to find a way to supply its resources but also it pushes back its received resources in a kind of residual overlay graph.

2.3.1 Algorithm Description

We assume each node can communicate directly with other nodes via bidirectional links. Each node runs the same local algorithm and may start executing the algorithm either at any arbitrary time or upon receiving a message triggering algorithm execution.

When a node x joins the system, it only knows its demand $d(x)$, its resources $r(x)$ and its neighborhood $\mathcal{N}(x)$. Besides, x also knows its *remaining offer* $\rho(x)$ and its *remaining demand* $\delta(x)$. That is, these two variables give the value of, respectively, what amount of resources is still available at x, and what amount of resources x is still demanding. The algorithm is based on *activations*. A node x is activated once either it receives a message from one of its neighbors, or the failure of a neighbor is detected. The protocol consists of only three messages: `require`, `supply` and `back`.

Each node manages a buffer to store the requests of other nodes. When a node has to select one of its requesters, it can use any policy to choose it in its buffer, typically first-in-first-out. When a peer x joins the system, it sends a `require` message with its demand to all its neighbors, so that it will be marked as a requesting peer, and it will hopefully be eventually served. Once it is fulfilled, it should alert its neighbors that they have to discard the entry on their buffer. This is done by a `require` message with a null value. In the following, we do not give details on buffer management or variables update in order to let the reading as easy as possible.

The `require` message contains a path ψ and a value Δ. The pseudo-code of the treatment at reception is detailed in Figure 2. This message is the result of a chain of messages initiated by a peer x_k whose demand is not fulfilled at the time it sends the first message. The first message of the chain is `require` with $\psi = [x_k]$ and $\Delta = \delta(x_k)$. A negative value Δ means "*please send me some resources*". This message is sent to all neighbors of x_k. Consider now the peer x_{k-1} receiving this message (line 1-10). If its remaining offer is not null, it can obviously supply x_k (lines 2-5). If it has already allocate its whole offer or if the remaining offer was not sufficient to serve x_k, it can consider re-allocating its offer in a different way, for example, supplying x_k instead of supplying another neighbor. In order to investigate such an opportunity (lines 6-10), the peer x_{k-1} then forwards the message `require` with $\psi = [x_{k-1}, x_k]$ and a new computed Δ whose value is now positive. A positive value means "*give me back my resources, I want to supply* x_k". The message is sent to all peers that are already supplied by x_{k-1}. The peer x_{k-2} receiving this message (lines 11-19) can forward the `require` message again with $\psi = [x_{k-2}, x_{k-1}, x_k]$ and a new computed Δ

```
1  if Δ negative then
2      if ρ > 0 then
3          send "supply, min(ρ, Δ)" to x_1
4          Δ ⟵ Δ − min(ρ, Delta)
5      end if
6      if Δ ≠ 0 then
7          S = {y ∈ N(x) : w(x → y) > 0}
8          send "require, [x, x_1, ··· , x_k], −Δ" to S
9      end if
10 end if
11 if Δ positive then
12     if δ < 0 and w(x_1 → x) > 0 then
13         send "back,min(Δ, w(x_1 → x))" to x_1
14         Δ ⟵ Δ − min(Δ, w(x_1 → x))
15     end if
16     if Δ ≠ 0 then
17         send "require, [x, x_1, ··· , x_k], −Δ" to N(x)
18     end if
19 end if
```

Figure 2: Reception "`require`, $[x_1, \cdots x_k]$, Δ" (node x)

whose value is now negative. The receiver should interpret this message as "*send me some resources so that I can free the resources x_{k-1} is supplying to me and let x_{k-1} supply x_k*". And so on.

The `supply` message contains an amount of resources. It is sent by a peer that (i) has been activated by the reception of a message, (ii) has stored a message `require` with a negative value $-\Delta$, and (iii) has some resources to offer. This message is actually a reservation of the resources from the sender of this message to the destination, which is the requiring neighbor. The amount of resources is free, but it can obviously not be greater than the remaining offer, nor than Δ. At reception, the treatment is detailed in Figure 3.

The `back` message does also contain an amount of resources. It is sent by a peer that (i) has been activated by the reception of a message `supply` with an amount of resources ω, and (ii) is fulfilled. This situation may occur in two cases. First, when two messages `supply` are concurrently arriving. In this case, the peer sends immediately these resources back to the sender (lines 8-9 in Figure 3). In the other case, this peer has forwarded a message `require` with a positive value because a re-allocation is required (lines 6-7 in Figure 3). The amount of resources is easy to compute, it is the excess of resources. Note that a peer does never receive more resources than its demand: as soon as it receives a `supply` message while it is already fulfilled, it forwards the resources in excess.

2.3.2 Algorithm Interpretation

We give another interpretation of this algorithm, based on the aforementioned transformation. This interpretation is the core of the proofs that this algorithm terminates and the final allocation is optimal.

Each peer cares about its associated nodes x^+ and x^-. The goal is to detect some existing

```
1  δ_new ⟵ δ − Ω
2  if (δ > 0 and δ_new = 0) then
3       send "require,∅, 0" to N(x)
4  end if
5  if δ_new < 0 then
6       if (∃ a requiring z ∈ N(x) with positive Δ) then
7            send "back, −δ_new" to z
8       else
9            send "back, −δ_new" to y
10      end if
11 end if
```

Figure 3: Reception "`supply`, Ω" from y (node x)

paths from both nodes to the sink with a non-empty capacity. The remaining offer and demand can be here transformed into one unique variable called $Excess$ for, respectively, the node x^+ and x^-. That is, a positive $Excess(x^+)$ means that the node x does not share all its available resources. The value $Excess(x^+)$ is initially set to $r(x)$. The evolution of $Excess(x^-)$ is more complex. Its value is initially set to $-d(x)$, so it is negative. While $Excess(x^-)$ is negative, the demand $d(x)$ is not fulfilled. A positive value of $Excess(x^-)$ can occur and means that x receives more resources that it demands. The discussion related with Figure 3 describes these cases. The goal is to reach a state where both $Excess(x^+)$ and $Excess(x^-)$ are null, which is equivalent to both remaining offer and demand null.

The protocol can easily be explained in this model. When a peer x sends a `require` message with a negative value Δ, it means that this message has been sent by the node x^- although a positive value Δ means a message from x^+. The `supply` message can only be sent by x^+, and `back` message by x^-. The use of these latter message is based on Ford-Fulkerson principle: the flow is also circulating on the residual graph created by the allocations.

We denote by $\psi(x^+)$ (respectively $\psi(x^-)$) the current path from the node x^+ (respectively x^-) to the sink. A path to the sink is obviously equivalent to a path to a node y^- which is requiring resources. We denote by $\omega(\psi(x^+))$ (respectively $\omega(\psi(x^-))$) the amount of resources that can be allocated to the path $p(x^+)$ (respectively $p(x^-)$).

The computation of paths, which is done at every activation, is described in Alg. 4. For the node x^+ (lines 2-8), the challenge is to find a node y^- either whose demand is not fulfilled, or which knows a way to reach a not fulfilled node z^-. For x^-, a negative $Excess(x^-)$ leads trivially to $psi(x^-) = [x^-]$ (line 12), otherwise the path $p(x^-)$ depends on an existing node y^+ such that both $psi(y^+)$ is not empty and y sends resources to x (line 14-17). If no neighbor of x exhibits any path to the sink, then x has no path anymore (lines 7 and 19).

The second step is to send flow along paths. A node x° with $\circ \in \{+, -\}$ can send flow if it has both a positive excess $Excess(x^\circ)$ and a valid path $p(x^\circ) \neq \emptyset$. It then sends the maximal amount of resources it can, *i.e.* the smallest value between $\delta(x^\circ)$ and $Excess(x^\circ)$, to the next node in $p(x^\circ)$. If it still has some available resources, it then computes a new path $p(x^\circ)$ and sends, if possible, flow along this new path. The node sends flow along new and not yet used paths until it has no more available resources or no more known paths to fill. It can then update its variables and become inactive.

```
1  − · · · − x+ − · · · −
2  pick u ∈ N(x) such that ψ(u−) ≠ ∅ and x+ ∉ ψ(u−)
3  if u exists then
4       ψ(x+) ← ψ(u−) and add x+ to ψ(x+)
5       ω(ψ(x+)) ← ω(ψ(u−))
6  else
7       ψ(x+) ← ∅; ω(ψ(x+)) ← 0
8  end if
9
10 − · · · − x− − · · · −
11 if Excess(x−) < 0 then
12      ψ(x−) ← [x] ; ω(ψ(x−)) ← −Excess(x−)
13 else
14      pick u ∈ N(x) such that ψ(u+) ≠ ∅ and x− ∉ ψ(u+) and w((u → x)) ≠ 0
15      if u exists then
16           ψ(x−) ← ψ(u+) and add x− to ψ(u+)
17           ω(ψ(x−)) ← min (ω(ψ(u+)), w((u → x)))
18      else
19           ψ(x−) ← ∅ ; ω(ψ(x−)) ← 0
20      end if
21 end if
```

Figure 4: Computing Paths

As our target environment are asynchronous distributed systems where processes can crash at any time, a path $p(x^\circ)$ may have disappeared in the current residual graph. So this node will send resources to a node $y^\bullet$ which does not necessarily belong to a path to the sink. But, if a path from x° exists to the sink, then the flow sent to $y^\bullet$ will eventually go back to x° and go along the correct path. So if a basic failure detector can eventually notify the crash of a neighbor, a node retrieving the resources it gave to this faulty neighbor can update its neighborhood and re-become active. The system will thus re-converge to a new maximal solution.

2.4 Algorithm Proof

We could use in the following proofs a fully realistic model including various communication time and various computing time, but, mostly for clarity, we assume in the sequel that messages have no travelling time, that algorithm running time is null and that messages are immediately treated. Therefore, for a time t and a time $t + \epsilon$ with $\epsilon > 0$, there is a unique finite sequence $(x_1, \cdots, x_l)$ of activated nodes. At a given time t, a working configuration C_t is defined by, for each $x \in V$, the value of $\delta_t(x^+)$, $\delta_t(x^-)$, $p_t(x^+)$, $p_t(x^-)$, $Excess_t(x^+)$, $Excess_t(x^-)$ and for each $y \in \mathcal{N}(x)$ the value of $w_t((x \to y))$. Each system configuration is associated with a unique valid flow f_t. If no node x^- exhibits a positive $Excess_t(x^-)$, then the valid flow f_t is directly deduced from w_t and we denote by $\mathcal{R}_t$ the corresponding residual graph.

We now explain, for the purpose of proofs, how to obtain f_t if there is at least one node x^- with a positive $Excess_t(x^-)$. We first assume a total order on nodes, for example $V = \{x_1, \cdots, x_n\}$. A node x^- having a positive excess should discard this excess. Let $j \leq n$ be the smallest index

1 $-\cdots-$ reduce $Excess(x^+)$ $-\cdots-$
2 $\mathcal{N}_{sent} = \emptyset$
3 $-\cdots-$ Set of nodes to whom x^+ has already sent resources $-\cdots-$
4 **while** $(Excess(x^+) > 0)$ and$(p(x^+) \neq \emptyset)$ **do**
5 let y^- be the second element of $p(x^+)$
6 $w((x \rightarrow y)) += \min(Excess(x^+), \delta(x^+))$
7 $Excess(x^+) -= \min(Excess(x^+), \delta(x^+))$
8 add y to the set $\mathcal{N}_{sent}$
9 `msg` to y: $w((x \rightarrow y)) = \min(Excess(x^+), \delta(x^+))$
10 run Alg. 4 lines 1-8
11 **end while**
12 $-\cdots-$ reduce $Excess(x^-)$ $-\cdots-$
13 $\mathcal{N}_{sent} = \emptyset$
14 **while** $(Excess(x^-) > 0)$ and $(p(x^-) \neq \emptyset)$ **do**
15 let y^+ be the second element of $p(x^-)$
16 $w((y \rightarrow x)) -= \min(Excess(x^-), \delta(x^-))$
17 $Excess(x^-) -= \min(Excess(x^-), \delta(x^-))$
18 `msg` to y: $w((y \rightarrow x)) = Excess(x^-)$
19 add y to the set $\mathcal{N}_{sent}$
20 run Alg. 4 lines 9-20
21 **end while**

Figure 5: Sending Flow Along Paths

such that $\sum_{i=1}^{j} w_t((x_i \to x)) \geq Excess(x^-)$. The excess of x^- can be discard easily by somehow "refusing" the resources given by all nodes in $\{x_1^+, \cdots, x_j^+\}$. More precisely, we set the flow $f((x_i^+ \to x^-))$ to 0 for all $1 \leq i < j$ and the flow $f((x_j^+ \to x^-))$ is set with the rest of the excess, formally $\sum_{i=1}^{j} w_t((x_i \to x)) - Excess(x^-)$. Then, the flow is not conserved at all nodes impacted by this operation, *i.e.* all nodes in $\{x_1^+, \cdots, x_j^+\}$. Therefore, we reset the flow between the source and these nodes so that the flow is conserved.

For the clarity of following proofs, let $ExcessNodes_t$ be the set of nodes having positive excess and $GlobalExcess_t$ the sum of all positive excesses $GlobalExcess_t = \sum_{x^\circ \in ExcessNodes_t} Excess(x^\circ)$.

Lemma 1. *For any time t and for any $\epsilon > 0$, we have $GlobalExcess_t \geq GlobalExcess_{t+\epsilon}$.*

The proof of this lemma is detailed in Appendix.

Lemma 2. *At a given time t, if there exists a path $p = [s, x_1^+, x_2^-, \cdots, x_k^-, q]$ in $\mathcal{R}_t$ from a node x_1^+ to a node x_k^-, then either we have $Excess_t(x_1^+) > 0$ or there exists $y^- \neq x_2^-$ such that we have both $w_t((x_1 \to y)) > 0$ and $Excess_t(y^-) \geq w_t((x_1 \to y))$.*

Proof. The proof directly follows from the definition of $\mathcal{R}_t$. □

By considering inactive nodes and shortest paths in $\mathcal{R}_t$ and by Lemma 2, we obtain the following theorem, and thus the following corollary.

Theorem 1. *At a given time t, if all nodes are inactive then there is no path from s to q in $\mathcal{R}_t$.*

Corollary 1. *At a given time t, if all nodes are inactive then the flow induced by w_t is maximal.*

Theorem 2. *At a given time t, let $p = [s, x_1^+, \cdots, x_k^-, q]$ a shortest path in $\mathcal{R}_t$ from s to q, then there exists $\epsilon > 0$ such that there is no path from s to q in $\mathcal{R}_{t+\epsilon}$ beginning by x_1^+ and ending by x_k^-. Moreover, we have $GlobalExcess_t > GlobalExcess_{t+\epsilon}$.*

Outline of the Proof. Let $p = [s, x_1^+, \cdots, x_k^-, q]$ a shortest path in $\mathcal{R}_t$ from s to q. Assume, by contradiction, that for every $\epsilon > 0$ there exists a path from s to q in $\mathcal{R}_{t+\epsilon}$ beginning by x_1^+ and ending by x_k^-. First of all, as p exists, we have by Theorem 1 that there is at least one node that will become active. Moreover, as there always exists a path from s to q in $\mathcal{R}_{t+\epsilon}$ containing both x_1^+ and x_k^-, we obtain an infinite sequence S of activated nodes after the time t.

First, we claim that there is at least one node from the set $\{x_1, \cdots, x_k\}$ that is activated after the time t, that is more formally that there exists $x_i \in V$ such that we have both $x_i^\circ \in p$, with $\circ \in \{+, -\}$, and $x_i \in S$. Indeed, assume by contradiction that there is no nodes from $\{x_1, \cdots, x_k\}$ that belongs to S. Then, as p is a shortest path in $\mathcal{R}_t$ from s to q ending by x_k^-, we have that $Excess_{t+\epsilon}(x_k^-) < 0$, and thus $p_{t+\epsilon}(x_k^-) = [x_k^-]$. Now, as x_{k-1}^+ is never activated after the time t and as p is a shortest path in $\mathcal{R}_t$, we have that $p_t(x_{k-1}^+) = p_{t+\epsilon}(x_{k-1}^+) = [x_{k-1}^+, x_k^-]$. By induction, we obtain that $p_t(x_1^+) = p_{t+\epsilon}(x_1^+) = [x_1^+, \cdots, x_{k-1}^+, x_k^-]$ and $p_t(x_2^-) = p_{t+\epsilon}(x_2^-) = [x_2^-, \cdots, x_{k-1}^+, x_k^-]$. By Lemma 2, we have either that $Excess_t(x_1^+) > 0$ or there exists $y^- \neq x_2^-$ such that we have both $w_t((x_1 \to y)) > 0$ and $Excess_t(y^-) \geq w_t((x_1 \to y))$. Thus we obtain that, we have $p_t(y^-) \neq \emptyset$ and $p_t(y^-) = p_{t+\epsilon}(x_y^-)$. Now as either x_1^+ or y^- have positive excess at time t which contradicts that $p_t(x_1^+) \neq \emptyset$ and $p_t(x_2^-) \neq \emptyset$, because after a node x° has been activated we can not have both $Excess(x^\circ) > 0$ and $p(x^\circ) \neq \emptyset$. So either x_1 or y belongs to S. Notice that in the second case, as y is activated and as y^- has positive excess, the fact that $p(y^-)$ is none void implies that $p(y^-)$ will change, and thus y will send a message to x_1, which will become active.

Second, we claim that the node x_1^+ or a node y^- such that $w((x_1 \to y)) > 0$ will have a sending activation after time t, where a sending activation of the node x° is an activation of the node x with a decreasing $Excess(x^\circ)$. This claim directly follows from the proof of the previous claim.

Third, we claim that the node x° that is activated ($x^\circ = x_1^+$ or $x^\circ = y^-$) has a finite number of sending activation. We show that, for a node y_1^- and the node x°, the number of sending activation of x° at a time t such that $p_t(x^\circ) = [x^\circ, y_l, \cdots, y_1^-]$ is finite. This facts follows from the facts that the number of possible paths between x° and y_1^- is finite, that paths contains no cycles and that a sent resource from x° can come back to x° a finite number of times.

From this last claim, we obtain that there exists a time t' such that there is no nodes having sending activation after time t' and thus, by claim 3, that there is no path from s to q in $\mathcal{R}_{t'}$, which is a contradiction. Moreover, as there is a path between s and q in $\mathcal{R}_t$ and as there is no path from s to q in $\mathcal{R}_{t'}$ and as there exists sending activation, we have that there exists y^- such that we have both $Excess_t(y^-) < 0$ and $Excess_{t'}(y^-) < Excess_t(y^-)$, and thus we obtain $GlobalExcess_t > GlobalExcess_{t'}$. □

Proposition 1. *If at a given time, there is no path from s to q in $\mathcal{R}_t$ then there exists $\epsilon > 0$ such that at time $t + \epsilon$ all nodes are inactive.*

From previous theorems and this last proposition, we obtain the following theorem.

Theorem 3. *The algorithm terminates and it will converge to the maximal flow.*

Proof. As the value of *GlobalExcess* decreases with time and as it is a positive integer, from Theorem 2, there exists a time t such that there is no path from s to q in $\mathcal{R}_t$. By Proposition 1, we obtain that the algorithm terminates and by Corollary 1, that the obtained flow is maximal. □

3 Bounded Outdegree Overlays

In this Section we assume a peer x_i to have not more than $\Delta(x_i)$ connections, that is the overlay graph has a bounded out-degree: $|\mathcal{N}^+(x_i)| \leq \Delta(x_i)$. This assumption is not only helpful to reduce the search space for the design of an optimal overlay graph but stems from practical considerations. Indeed, in most real systems the number of connections a peer can establish concurrently is limited, both for scalability and performance reasons.

As for the unbounded case, we can reduce the problem of the overlay construction to a max-flow problem: the constraint on the number of connections a peer can establish becomes a constraint on the out-degree of the network flow. Hence, the problem is to determine the maximum flow of a *edge-capacitated* directed graph under the constraint that for each vertex $x_i \in V$, the out-degree induced by the flow is less or equal to $\Delta(x_i)$. Formally, we express our problem as follows:

Bounded-Outdegree Maximal Flow

INSTANCE : Directed graph $G = (V, A)$, specified vertices s and q, capacity $c(a) \in \mathbb{N}^*$ for $a \in A$, bound on out-degree $\Delta(v) \in \mathbb{N}^*$ for $v \in V$ and positive integer K.

QUESTION : Is there a flow function $f : A \longrightarrow \mathbb{N}$ such that :

(*i*) $f(a) \leq c(a)$ for all $a \in A$,

(*ii*) for each $v \in V \setminus \{s, q\}$, $\sum_{(u,v)\in A} f((u, v)) = \sum_{(v,u)\in A} f((v, u))$, *i.e.* flow is *conserved* at v,

(*iii*) for each $v \in V$, $|\{u \in V : f((v,u)) \neq 0\}| \leq \Delta(v)$, *i.e.* the outdegree of v in the flow is at most $\Delta(v)$,

(*iv*) $\sum_{(u,q)\in A} f((u,q)) - \sum_{(q,u)\in A} f((q,u)) \geq K$, *i.e.* the net flow in q is at least K.

We first recall the definition of the NP-complete Minimum Cover problem, then we prove that our problem is also NP-complete, even if the directed graph induced by $V \setminus \{s,q\}$ is bipartite.

Minimum cover

INSTANCE : Collection C of subsets of a finite set S, positive integer $L \leq |C|$.

QUESTION : Does C contain a cover for S of size L or less, *i.e.* a subset $C' \subseteq C$ with $|C'| \leq L$ such that every element of S belongs to at least one member of C' ?

Theorem 4. *Bounded-Outdegree Maximal Flow is NP-complete.*

Proof. Given an instance of Bounded-Outdegree Maximal Flow and a flow f, verifying the conditions (i) to (iv) is polynomial in the size of the problem: hence, this problem belongs to NP.

We now transform Minimum Cover to Bounded-Outdegree Maximal Flow. Let $C = \{C_1, \cdots, C_m\}$ a collection of subsets of a finite set $S = \{x_1, \cdots, x_n\}$ and let L a positive integer such that $L \leq |C|$. We define $G = (V, A)$ with $V = \{s, q\} \cup S \cup C$ and $A = \{(s, C_i) : i \in \{1, \cdots, m\}\} \cup \{(x_i, q) : i \in \{1, \cdots, n\}\} \cup \{(C_i, x_j) : i \in \{1, \cdots, m\}, j \in \{1, \cdots, n\} \text{ and } x_j \in C_i\}$. Let $c((s, C_i)) = |C_i|$ and $c(a) = 1$ otherwise and let $\Delta(s) = L$ and $\Delta(v) = |V|$ otherwise and let $K = |S|$ It is clear that the instance of Bounded-outdegree Maximal flow can be constructed in polynomial time. We claim that C contains a cover for S of size L or less if and only if there exists a flow f which respect the conditions (i) to (iv).

Clearly, if we have a minimum cover C' of C of size lower or equal to L, then we can define a mapping ϕ from S to C' such that $x_i \in \phi(x_i)$. Then we can define the flow f as $f((s, C_i)) = 0$ if $C_i \notin C'$ and $f((s, C_i)) = |\phi^{-1}(C_i)|$ otherwise, $f((C_i, x_j)) = 1$ if $C_i = \phi(x_j)$ and 0 otherwise, and $f((x_j, q)) = 1$. One can check that f verifies conditions (i) to (iv) of the definition of Bounded-Outdegree Maximal Flow.

Now, let f a flow that fulfills conditions (i) to (iv) of the definition of Bounded-Outdegree Maximal Flow. One can check that $C' = \{C_i : f((s, C_i)) \neq 0\}$ is a minimum cover of S of size lower or equal to L. Indeed, as $K = |S|$, the definition of the G implies that for every $x_j \in S$, there exists $C_i \in C'$ such that $x_j \in C_i$ (we must have that $f((C_i, x_j)) = 1$). Now as $\Delta(s) \leq L$, we have that $|C'| \leq L$. □

4 Conclusion

Node-capacitated graphs have recently become a hot topic as it appears as an appealing model for peer-to-peer systems where the core network is transparent and resource constraints are on the peers. This paper provides a formal graph-based model for the problem of local resource sharing in node-capacitated underlays and exhibits two families of resource allocation revealing a substantial practical interest. We show that building an overlay such that the demand in resources of every peer is fulfilled can be reduced to computing the maximal flow in a bipartite network. Then we design a distributed fault-tolerant algorithm which is provably correct, but we also show that this algorithm can not be directly used to build bounded-outdegree overlays.

Note however that despite Bounded-outdegree Maximal flow is NP-complete, it does not imply that the problem of building an optimal overlay on top of a given node-capacitated underlay with

constraints on the outdegree of this overlay is NP-complete. It only shows that the technique used here for unbounded outdegree overlays can not be directly applied to bounded outdegree overlays. The problem of determining a bounded outdegree overlay is consequently still open.

References

[1] G. Andersson, P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, J. Sanchez-Gasca, R. Schulz, A. Stankovic, C. Taylor, and V. Vittal. Causes of the 2003 major grid blackouts in north america and europe, and recommended means to improve system dynamic performance. *IEEE Transactions on Power Systems*, 20(4):1922–1928, Nov. 2005.

[2] V. C. Barbosa. *An introduction to distributed algorithms*. MIT Press, 1996.

[3] Y. Benkler. *The Wealth of Networks*. Yale University Press, 2006.

[4] J. Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Inf. Process. Lett.*, 60(5):225–230, 1996.

[5] B. Cohen. Incentives build robustness in bittorrent. In *Proc. of P2Pecon*, 2003.

[6] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey. The uncapacitated facility location problem. In *Discrete location theory*, Wiley-Intersci. Ser. Discrete Math. Optim., pages 119–171. Wiley, New York, 1990.

[7] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *OSDI*, 2002.

[8] L. P. Cox and B. D. Noble. Samsara: honor among thieves in peer-to-peer storage. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP'03)*, pages 120–132, 2003.

[9] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 19(2):248–264, 1972.

[10] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a Network Creation Game. In *Proc. of ACM PODC*, 2003.

[11] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Math. Program.*, 81(2):229–256, 1998.

[12] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[13] A. Frank, Z. Király, and B. Kotnyek. An algorithm for node-capacitated ring routing. *Oper. Res. Lett.*, 35(3):385–391, 2007.

[14] S. Ghosh, A. Gupta, and S. V. Pemmaraju. A self-stabilizing algorithm for the maximum flow problem. *Distrib. Comput.*, 10(4):167–180, 1997.

[15] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. Assoc. Comput. Mach.*, 35(4):921–940, 1988.

[16] M. T. Hajiaghayi, R. D. Kleinberg, H. Räcke, and T. Leighton. Oblivious routing on node-capacitated and directed graphs. *ACM Trans. Algorithms*, 3(4):51, 2007.

[17] N. Hatziargyriou and G. Strbac. Microgrids - a possible future energy configuration. IEA Workshop "Distributed Generation: Key issues, Challenges, Roles", 2004.

[18] N. Laoutaris, G. Smaragdakis, A. Bestavros, and J. W. Byers. Implications of Selfish Neighbor Selection in Overlay Networks. In *Proc. of IEEE INFOCOM*, 2007.

[19] S. Liu, S. Sengupta, M. Chen, J. Li, M. Chiang, and P. A. Chou. Streaming capacity in peer-to-peer networks with topology constraints. Technical report, Princeton University, Sep. 2008.

[20] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *IEEE INFOCOM 2007*, pages 1073–1081, 2007.

[21] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello. Push-to-peer video-on-demand system: Design and evaluation. *IEEE Journal on Selected Areas in Communications*, 25(9), 2007.

Appendices

A Proofs

Lemma. *For any time t and for any $\epsilon > 0$, we have $GlobalExcess_t \leq GlobalExcess_{t+\epsilon}$.*

Outline of the Proof. In order to prove this lemma, we have only to consider nodes before and after that they become active. Let x be an inactive node and let t the time before than x becomes active. We denote by $C_{t'}$ the configuration of the system after that the node x has running the algorithm and that the messages sent by x have been treated. So, the only nodes such that the *Excess* values could have been changed are in $\mathcal{N}(x) \cup \{x\}$. We then obtain that we have both $\sum_{y\in\mathcal{N}(x)} Excess_t(y^+) + Excess_t(x^+) = \sum_{y\in\mathcal{N}(x)} Excess_{t'}(y^+) + Excess_{t'}(x^+)$ and $\sum_{y\in\mathcal{N}(x)} Excess_t(y^-) + Excess_t(x^-) = \sum_{y\in\mathcal{N}(x)} Excess_{t'}(y^-) + Excess_{t'}(x^-)$. So the difference $GlobalExcess_{t'} - GlobalExcess_t$ is the sum of excess of nodes having negative excess at time t and positive excess at time t'. As the only nodes that have negative excess at time t and positive excess at time t' are nodes belonging to the set $\{y^- : y \in \mathcal{N}(x)\}$, we obtain that :

$$GlobalExcess_{t'} - GlobalExcess_t = \sum_{\substack{y\in\mathcal{N}(x) \\ Excess_t(y^-)<0 \\ Excess_{t'}(y^-)\geq 0}} Excess_t(y^-)$$

Thus, we obtain that $GlobalExcess_{t'} \leq GlobalExcess_t$ □

Theorem. *At a given time t, if all nodes are inactive then there is no path from s to q in $\mathcal{R}_t$.*

Proof. Assume by contradiction that at time t all nodes are inactive and that there exist paths from s to q in $\mathcal{R}_t$. Let $p = [s, x_1^+, \cdots, x_j^-, q]$ a shortest path from s to q in $\mathcal{R}_t$. Notice that by definition of N, we have $j \geq 2$. First of all, as p exists we have that $Excess_t(x_j^-) < 0$ and thus that $p_t(x_j^-) = [x_j^-]$. As x_j is inactive and as p is a shortest path in $\mathcal{R}_t$, we have that $p_t(x_{j-1}^+) = [x_{j-1}^+, x_j^-]$. Now, as x_{j-1} is inactive and as p is a shortest path in $\mathcal{R}_t$, we have that $p_t(x_{j-2}^-) = [x_{j-2}^-, x_{j-1}^+, x_j^-]$. By induction, one can prove that this remains true for every $i \in \{1, \cdots, j\}$. So we obtain that $p_t(x_1^+) = [x_1^+, \cdots, x_j^-]$ and $p_t(x_2^-) = [x_2^-, \cdots, x_j^-]$. By Lemma 2, we have either that $Excess_t(x_1^+) > 0$ or there exists $y^- \neq x_2^-$ such that we have both $w_t((x_1 \to y)) > 0$ and $Excess_t(y^-) \geq w_t((x_1 \to y))$. Thus as we have both $p(x_1^+)$ and $p(y^-)$ are none void and by Lemma 2 one of them have positive excess, this contradicts the fact that both nodes x_1 and y are inactive. □

Corollary. *At a given time t, if all nodes are inactive then the flow induced by w_t is maximal.*

Proof. This corollary directly follows from the proof of the Ford-Fulkerson's algorithm, from the fact that the flow induced by w_t is a valid flow in the network N and from Theorem 1. □

Proposition. *If at a given time, there is no path from s to q in $\mathcal{R}_t$ then there exists $\epsilon > 0$ such that at time $t + \epsilon$ all nodes are inactive.*

Proof. We denote by F the set of nodes having negative excess, that is $F = \{y^- : Excess(y^-) < 0\}$. By definition of F, we have that for any $\epsilon > 0$ and for any $y^- \in F$, $p_{t+\epsilon} = [y^-]$ and $\delta_{t+\epsilon}(y^-) =$

$-Excess_{t+\epsilon}(y^-) = -Excess_t(y^-)$. This means that nodes belonging to F send no more message to its neighbors. We denote by F_1 the set of nodes that are adjacent in $\mathcal{R}_t$ to at least one element of F, so there exists $\epsilon_1 > 0$ such that after the time $t + \epsilon_1$ the values of $p(x^+)$ and $\delta(x^+)$ does not change anymore for $x^+ \in F_1$. This means that nodes belonging to F_1 send no more message to its neighbors after the time $t + \epsilon_1$. In the same way, we can define F_i as the set of nodes such that the minimal distance in $\mathcal{R}_t$ between a node from F and this node is i. By induction, one can show that there exists $\epsilon_i > 0$ with $\epsilon_i \geq \epsilon_{i-1}$ such that after the time $t + \epsilon_i$ the values of $p(x)$ and $\delta(x)$ does not change anymore for $x \in F_i$. As the number of nodes in the system is finite, there exists an integer j such that we have $F_j = F_{j+1}$. So, at time $t + \epsilon_j$, we have that all nodes from F_j does not send message anymore. Clearly, we obtain that F_j is the set of nodes x such that there exists a path in $\mathcal{R}_t$ from x to q.

Now, let G the set of nodes x such that $p_{t+\epsilon_j} = \emptyset$. As there is no path from s to q in $\mathcal{R}_t$ and thus in $\mathcal{R}_{t+\epsilon_j}$, we obtain that G is not empty. Moreover, nodes from G send no more message to its neighbors after the time $t + \epsilon_j$. Now, again, if we define G_i as the set of nodes such that the minimal distance in $\mathcal{R}_t$ between a node from G and this node is i, we obtain that there exists $\epsilon'_i \geq \epsilon'_{i-1}$ (with $\epsilon'_0 = \epsilon_j$) such that all nodes from G_i does not send message anymore. If we denote by k the integer such that we have $G_k = G_{k+1}$, we obtain that $F_j \cup G_k =$ is the set of nodes of the system. and thus, that at time $t + \epsilon'_j$ all nodes are inactive. □