Star: A Stable and Robust Membership Protocol

Ajoy Datta, Anne-Marie Kermarrec, Lawrence Larmore, Erwan Le Merrer

STAR: A Stable and Robust Membership Protocol

Ajoy K. Datta[1] Anne-Marie Kermarrec[2] Lawrence L. Larmore[1]
Erwan Le Merrer[2]

[1]School of Computer Science, University of Nevada Las Vegas
[2] INRIA, Rennes Bretagne-Atlantique, France

Abstract

We present the design and analysis of STAR, a fully decentralized self-stabilizing randomized membership protocol building a strongly connected overlay graph with sub-logarithmic diameter and almost homogeneous logarithmic degree. STAR is the first protocol to simultaneously maintain the following properties on the resulting graph G: (i) The graph maintains the *Eulerian* property, namely that the in-degree and out-degree of each node are the same, and that G is strongly connected. (ii) The out-degree of each node automatically converges to an average of $2\ln(n) + O(1)$ without any node knowing the exact size n of the network. (iii) The diameter of the overlay graph will be $\frac{\ln n}{\ln\ln n + \ln 2} + O(1)$ with high probability. (iv) STAR is self-stabilizing. Starting from an arbitrary graph topology, or after disruptive error, STAR will cause the overlay graph to converge to the desired properties.

In the paper, we provide the details of the protocol, analysis of the protocol, and simulation results confirming the analysis and comparing STAR with the closest competitor.

Keywords: Membership protocol, random graph, self-stabilization.

1 Introduction

Context and Objectives. Overlay networks have generated tremendous interest in the research community as well as in the real world, as they are acknowledged to be the core technology needed to build and maintain fully decentralized large scale applications. An overlay network logically connects nodes in a large scale system on top of any physical networking infrastructure. The resulting overlay network can be viewed as a directed graph where there is an edge from a node a to a node b if a is able to communicate with b, typically by knowing its IP address.

Overlay networks are characterized by the property that each node knows a very limited subset of the network. Several overlay network designs exist in the literature, from fully structured designs, such as Pastry [9] or Chord [10], to fully unstructured overlay graphs, the topology of which is closer to those of random graphs [4]. They can be used to search contents, disseminate contents, *etc.* The properties of a given overlay network are of the utmost importance for the applications using it. For example, it is well known [7] that low diameter and nearly homogeneous degree are crucial to ensure fast and well-balanced content dissemination.

Building an overlay network *i.e.,* providing each node with a set of neighbors, is typically achieved through a membership protocol, which should be fully decentralized for scalability reasons. One of the challenges of such fully decentralized membership protocols is to ensure that certain desirable

properties of the overlay graph, such as low diameter or strong connectivity, are maintained as the system evolves over time.

In this paper, we propose the STAR (STAble and Robust) membership protocol, which is fully decentralized. Preliminary calculations, supported by the results of simulations, support our claim that STAR guarantees that, after convergence, each node of the overlay graph has an expected out-degree degree of approximately $2\ln(n)$, where *Min_deg* is a small positive constant fixed in advance, and that the diameter of the overlay graph remains below $\frac{\ln(n)}{\ln(2\ln(n))}+O(1)$ with very high probability. In addition, STAR is robust enough to maintain these properties through repeated node arrivals and departures, as well as isolated node failures. It will also restore the desirable properties from an arbitrary state, such as might result from massive but temporary disruption.

Related Work. There have been a number of membership protocols proposed over the past decade; we give a few details on those most relevant to this paper. Gossip-based membership protocols have been acknowledged as a robust way to built unstructured overlay networks which converge to a topology with characteristics close to those of random graphs. Examples of such protocols are Cyclon, Newcast, Lpbcast or Bhrams. Those protocols are extremely robust in face of failure and cope fairly well with dynamics: failed nodes are gracefully removed from the graph without requiring any specific operation.

A major difference between STAR and some of the other gossip-based protocols is that, in some of those protocols, the out-degree of a node is a parameter of the system, and is not related to the size of the network. Although this might not be an issue in networks where the size does not vary much, it could be problematic if the size varies greatly, causing, for example, network diameter to increase dramatically. Maintaining an out-degree evolving with the size of the system would require continuous monitoring of the system to compute the size of the group (which could become cumbersome) and adjusting the value of the parameter out-degree on all nodes.

SCAMP [6] on the other hand, is a membership protocol which ensures that on average the out-degree nodes is $(c+1)\cdot\ln(n)$, where $c>0$ is a parameter of the system and n the size of the system, without any node being aware of the size of the system. STAR accomplishes the same objective; the expected out-degree of each node is related to the size of the system, without any node knowing that size. Yet there are two major differences between STAR and SCAMP. SCAMP ensures this property only on the average. The motivation was that SCAMP was designed for gossip-based dissemination and theoretical results show that the dissemination succeeds if node degree is related to the size of the system, on the average. This resulted in an initial version of SCAMP where the node degree distributed around the average. Later versions of SCAMP include a rebalancing mechanism to homogenize the degree. STAR maintains nearly homogeneous degree; the degree of each node oscillates dynamically around the average value, with a small variance, Second, SCAMP does not cope with failures and dynamism, while STAR will always to converge to a configuration where the degree and diameter are correct, starting from arbitrary bad configurations, provided the overlay graph remains at least weakly connected.

In [3], a self-stabilizing group membership service protocol was presented. This protocol maintains a spanning tree of the group, and is used as the communication medium to implement services like multicasting. A self-stabilizing and Byzantine tolerant overlay network construction protocol was proposed in [2]; the protocol maintains an average diameter of $\ln(n)$ with a high probability.

Contributions. In this paper, we present the design and analysis of STAR, a fully decentralized self-stabilizing randomized membership protocol building a strongly connected overlay graph with diameter $\frac{\ln n}{\ln \ln n + \ln 2} + O(1)$ with high probability, and almost homogeneous logarithmic degree which averages no more than $2\ln(n) + O(1)$. Finally, starting from an arbitrary topology, or after disruptive error, STAR restores the above properties of the overlay graph of the overlay graph, provided the graph remains at least weakly connected.

Outline of the Paper. In Section 2, we give our model of computation and the problem specification. In Section 3, describe the data structures of STAR, and define what constitutes a legitimate configuration of STAR. In Section 4, we give a formal definition of the non-self-stabilizing version of STAR, together with the results of some simulations. In Section 5, we give protocols which cause STAR to converge toward a "good" topology, *i.e.,*, where the degrees and diameter are as desired. In Section 6, we expand STAR to be self-stabilizing, describing the various protocols used from recovery from an erroneous state.

2 Preliminaries

2.1 Model

We consider a network of any number of nodes. Each node is identified by a unique ID and its IP address. In the remainder of the paper, ID means the pair (ID, IP@). We use an almost synchronous message passing model. A node x can send a message to node y if and only if x has the ID of y in its memory. An enabled node must execute within one time unit by its own clock, and, for some constant Λ, no node's clock run more than Λ times as fast as any other node's clock.

2.2 Problem Specification

We must maintain a directed overlay graph G which links the members of a group, which is a subset of the network of nodes. G must be strongly connected, meaning that for any two nodes x and y, there is a path from x to y. G should have reasonably low and approximately uniform, degree, and low diameter. Nodes must be able to join G (subscribe) and to leave G (unsubscribe). We also want G to be robust, meaning that our algorithm will be able to restore the properties of G from an arbitrary state, provided G is at least weakly connected.

3 STAR: A Stable and Robust Membership Protocol

3.1 Overview of STAR

The heart of STAR is the Balancing protocol, which causes convergence of both diameter and degree by *churning* the edges. Edges are either *active* or *passive*; we can think of passive edges as those which have been marked for deletion, but have not yet been deleted.

Edges are continually removed or marked as *passive* (to be deleted later) and new edges are continually added. The balancing protocol marks an edge (x, y) to be passive only if it determines that there is a *detour* of (x, y), namely a path from x to y which uses only active edges and does

not use the edge (x, y). If it does not find such a detour, it increases the out-degree of x. Protocol 4 also churns edges, since it deletes one active edge and adds two active edges.

The balancing protocol, Protocol 10, is designed to search extensively for a detour of (x, y) if the essential out-degree of x is large, and to severely limit that search if x has small out-degree. Thus, a dynamic equilibrium is achieved, where the essential out-degree of every node tends to oscillate around the average. Simultaneously, the locally balancing protocol, Protocol 9, keeps the essential in-degree to within a constant (not a constant factor) of the essential out-degree.

The balancing protocol is also designed so that, when equilibrium is achieved, the average out-degree of each node will be approximately $2\ln(n)$, and yet there is never a need for any node to compute the value of n.

The balancing protocol achieves the desired diameter merely by the fact that all edges, including "bad" edges, will eventually be removed, while the new edges that are added tend to be randomly placed, and hence "good." The diameter of G^a, the subgraph of G consisting of all nodes and just the active edges, thus converges to approximately the diameter of a random directed graph of the same cardinality and degree, as described in Appendix A.

3.2 The Modules of STAR

STAR consists of a number of modules, each of which we also call a protocol. We list these below, and explain the purpose of each. Protocols 4, 7, 9, 10, 12, and 14 all increase the number of active edges, while Protocols 5 and 10 mark active edges as passive.

Protocols 1 and **2:** Find a Random Edge or Node Starting from *start*. Starting from a given node *start*, which is a parameter of the protocol, using random walk, a random edge or node of G^a is identified. In Protocols 1 and 2, the stationary distributions of the random walks we use are uniform; thus if the walks are sufficiently long, all edges or nodes, respectively, are equally likely to be chosen. We can use shorter walks if they start from a random node, however.

Protocol 3: Satellite of x. Every node x will have a number of *satellites*, that number is a design parameter. Each satellite of x is a mobile agent which is *hosted* by some node y. Protocol 3 controls the movement of a satellite from one host to the next.

Protocol 4: Increment the Degree of y. An active edge (x, z) is chosen elsewhere in G^a, and is deleted, to be replaced by two active edges (x, y) and (y, z). Thus, the in-degree and out-degree of y are both increased by 1. This protocol makes use of Protocol 1 to find the edge (y, z).

Protocol 5: Local Identification of Redundant Edges. This protocol marks any duplicate edges to be passive. Passive edges will eventually be deleted by Protocol 6.

Protocol 6: Delete Passive Edges. This protocol deletes two passive edges, and usually creates one passive edge. In the special case that the two deleted edges are (x, y) and (y, x) for some x and y, no new passive edge is created.

Protocol 7: A Node x Subscribes. A node x which is not currently a member of the group *subscribes*. Its initial action is to send a *subscription message* to some node which is already

a member of the group. In the subscription protocol, x joins G. Protocol 7 makes use of Protocol 4.

Protocol 8: A Node x Unsubscribes. The node x leaves the group. Its former neighbors link together to maintain the strong connectivity of G^a and the parity of G.

Protocol 9: Balance Essential Degrees of a node x. If the number of essential out-edges of any node x differs from the number of essential in-edges of x by more than Max_diff_deg, (which is a positive design parameter), then Protocol 9 makes use of Protocol 2 to give x an additional active out-edge and a passive in-edge or vice-versa, decreasing that difference.

Protocol 10: Balancing Protocol for an Edge (x, y). This protocol is the heart of STAR. Protocol 10 attempts to find a *detour* of an edge (x, y), namely a path of active edges from x to y which does not use the edge (x, y). If it finds a detour, (x, y) is labeled passive, while if it does not, Protocol 4 increments the degree of x. Protocol 10 is biased so as to be more likely to find a detour if x has larger out-degree.

Protocol 11: Balancing Supervisor. This protocol runs continuously, Its purpose is to control the various executions of Protocol 10.

Protocol 12: Restore parity. If x detects that its out-degree is greater than its in-degree, it seeks another node y with the opposite problem, and creates and edge from y to x.

Protocol 13: Node x executes the *will* of a failed neighbor y. If x detects that a neighbor y has failed, and if y has left a valid will, then x executes the instructions in that will.

Protocol 14: If a node detects an error which cannot be corrected by Protocols 12 and 13, it creates an emergency two-way link with every node whose ID it has in its memory.

3.3 Node States

Each node x of G maintains the following data structures and variables.

1. A list of out-neighbors, which we call $x.outview$, one for each out-edge of x. Since G is a multi-graph, a node y can appear in $x.outview$ any number of times. Each out-edge is labeled *active* or *passive*; there can be active and passive edges leading to the same neighbor.

   We will write $actv\_out\_deg(x)$ for the number of active out-edges of x, and $passv\_out\_deg(x)$ for the number of passive out-edges of x. Let $out\_deg(x) = actv\_out\_deg(x) + passv\_out\_deg$.

2. A list of in-neighbors, which we call $x.inview$, one for each in-edge of x. Since G is a multi-graph, a node y can appear in $x.inview$ any number of times. Each in-edge is labeled *active* or *passive*; there can be active and passive edges leading to the same neighbor.

   We will write $actv\_in\_deg(x)$ for the number of active in-edges of x, and $passv\_in\_deg(x)$ for the number of passive in-edges of x. Let $in\_deg(x) = actv\_in\_deg(x) + passv\_in\_deg$.

   We will write $View(x) = x.inview \cup x.outview$, the *view* of x.

3. A list $x.hosts$ of all nodes which are *hosts* of *satellites* of x. A satellite of x is a mobile agent which wanders through the graph, changing hosts frequently. We will explain satellites in detail in Section 4.2. We allow a node to be listed in $x.hosts$ multiple times, indicating that multiple satellites of x are hosted by that node.

The number of satellites of x at any given time is a design parameter.

4. A list $x.guests$ of all nodes whose satellites are hosted by x. A node could be listed multiple times.

5. For each $y \in View(x)$, there is a list of instructions $x.will[y]$. If x ever decides that y has failed, then x executes the instructions in $x.will[y]$.

3.4 Legitimate Configurations of STAR

We say that a configuration of STAR is *legitimate* if the following conditions hold.

1. For any two nodes x and y, $y \in x.outview$ if and only if $x \in y.inview$. Furthermore, the number of active out-edges to y in $x.outview$ is equal to the number of active in-edges from x in $y.inview$; similarly for passive edges.

2. G satisfies the *parity property*, meaning that $out\_deg(x) = in\_deg(x)$ for every node x.

3. The subgraph G^a consisting of all the nodes of the overlay graph G and just the active edges is strongly connected.

   Since this implies that G is also strongly connected, conditions 2 and 3 imply that G is Eulerian.

4. If x and y are nodes, $y \in x.hosts$ if and only if $x \in y.guests$. Furthermore, the multiplicity of y in $x.hosts$ is equal to the multiplicity of x in $y.guests$.

A configuration is still legitimate if the above conditions are temporarily violated because a node has not yet executed a destined action. For example, if $y \in x.inview$ and $x \notin y.outview$, the configuration could still be legitimate if y is destined to receive a message instructing it to add x to its out-view, or if y has received that message and has not yet acted upon it. We will not list all possible situations of this type, but, as we introduce the various modules of STAR, it will become clear what they are.

4 Modules of STAR

We now present the modules (protocols) of STAR. In some cases, they can be in conflict. In Section 6.5, we show how to resolve these conflicts.

4.1 Finding a Random Node or Edge

Protocols 1 and 2 find a random edge and a random node, respectively, of G^a. These protocols are called by other protocols of STAR as needed.

Each of those two protocols has in input parameter, a node *start*, were we begin a random walk in G. In principle, if the random walk is long enough, and we stop after a randomly chosen number of steps, it does not matter which node is chosen as *start*. Because G is Eulerian, the stationary distribution of the random walk uses every edge equally often, and uses every node with a frequency that is proportional to its degree, as we discuss in Appendix B. However, the length of the random walk can be chosen to be shorter if *start* itself is a randomly chosen node.

In Protocol 1, our random walk stops at edges. All edges are equally likely if the length of the walk is long enough. If we stop at a passive edge, we restart the walk, and repeat the process until we stop at an active edge.

Protocol 1 makes use of a parameter L, which is a design parameter; the larger L, the longer closer to random our choice of edge will be, but the longer it will take to execute.

Protocol 1: Find a Random Edge Starting from *start*.

1: $y \leftarrow start$
2: **repeat**
3: $T \leftarrow L \cdot degree(start)$
4: **while** $T > 0$ **do**
5: $z \leftarrow y$
6: Choose $y \in z.outview$ uniformly at random {Use the multiplicities of the edges.}
7: With probability $\frac{1}{2}$: $T \leftarrow T - 1$
8: **end while**
9: **until** (z, y) is active
10: **return** (z, y)

In Protocol 2, our random walk stops at nodes. The probability that we stop at a given node is proportional to the degree of that node, so we bias the search to increase the probability that we stop at a node with low degree. This bias, which precisely cancels the non-uniformity of the stationary distribution of the process, is shown in Line 5 of the code.

Protocol 2 also makes use of a design parameter L.

Protocol 2: Find a Random Node Starting from *start*.

1: $T \leftarrow L$
2: $y \leftarrow start$
3: **while** $T > 0$ **do**
4: Choose $y \in y.outview$ uniformly at random {Use the multiplicities of the edges.}
5: with probability $\frac{1}{2degree(y)}$: $T \leftarrow T - 1$
6: **end while**
7: **return** y

4.2 Satellites

Associated with each node x will be a number of mobile agents, called *satellites* of x. Each satellite of x is *hosted* by some node in the system, and changes host frequently. The number of satellites of each node is a design decision.

Each satellite changes its host frequently, using a random walk, and thus will eventually be distributed around G. When it is necessary for x to find a random node or edge by using Protocol 1 or 2, we will start the walk at a satellite of x, thus helping the choice made by the protocol to be random.

The protocol for one satellite s of x, hosted at a node $host(s)$, is as follows.

Protocol 3: Satellite of x

1: **loop** {forever}
2: x sends the message, "Change host," to s. { The message actually goes to $host(s)$, of course.}
3: Using Protocol 2, where $L = 1$ and $host(s)$ is the start node, a new node y is selected.
4: $host(s)$ sends the message, "You are the new host of s," to y.
5: $host(s)$ sends the message, "The new host of s is y," to x.
6: y sends the message, "I am the new host of s," to x.
7: x updates its host list, $host(s) \leftarrow y$.
8: **end loop**

4.3 Incrementing the Number of Edges

At various points in STAR, we will want to either increase the number of edges in the overlay graph G, by increasing the degree of a specific node, say y. We need to also maintain the parity condition of G, so that we cannot simply insert or delete an edge. When we need to increment the number of edges, we delete one edge and insert two edges. We illustrate these changes in Figure 4.1 below.

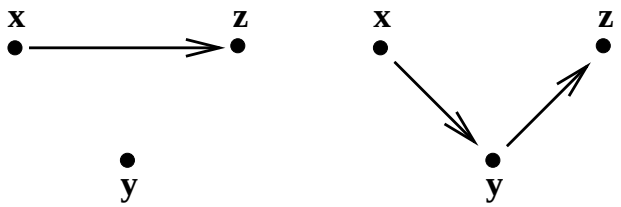

Figure 4.1: Incrementing the Degree of Node y.
Edge (x, z) Replaced by Edges (x, y) and (y, z).

Protocol 4: Increment the Degree of y, starting at *start*

1: **repeat**
2: Choose a random edge (x, z), using Protocol 1, starting from *start*
{We need to start the random search from a specific node, *start*}
3: **until** $y \notin \{x, z\}$
4: Insert active edge (x, y) into G
5: Insert active edge (y, z) into G
6: Delete the edge (x, z) from G.

Note that the edge (x, z) must be disjoint from y. Thus, Protocol 4 cannot be used if $n < 3$. Note that the Protocol preserve parity.

4.4 Active and Passive Edges

Every edge is either labeled *active* or *passive*. We cannot arbitrarily delete edges from G without violating the parity condition. Instead, STAR marks edges to be passive. Passive nodes will eventually be deleted when that can be done without violating parity.

An active edge (x, y) is defined to be *redundant* if its removal does not cause G^a to lose its strong connectivity. If we identify a redundant edge, we can label it passive, since if G^a was strongly connected, it will remain strongly connected.

We have two protocols which identify redundant edges and mark them passive: Protocol 5 which marks all duplicate edges as passive, and Protocol 10, given in Section 5, which marks an edge (x, y) passive if it finds a detour, *i.e.,* another path from x to y.

Although an individual passive edge cannot be deleted, Protocol 6 deletes a pair of passive edges, such as (x, y) and (y, z) and inserts a new passive edge (x, z). If parity holds before an execution of Protocol 6, it will still hold after that execution.

Protocol 5: Local Identification of Redundant Edges

1: **loop** {forever}
2: **for** each active edge (x, y) **do**
3: **if** there exists more than one active edge (x, y) **then**
4: mark all but one copy of (x, y) passive
5: **end if**
6: **end for**
7: **end loop**

We can insist that Protocol 5 act instantly whenever a duplicate active edge is inserted by Protocol

4, 7, 8, 9, 12, or 14.

Protocol 6: Delete Passive Edges

```
1: loop {forever}
2:   for each node y do
3:     if there are passive edges (x, y) and (y, x) for some node x then
4:       Delete edges (x, y) and (y, x)
5:     else if there exist passive edges (x, y) and (y, z) for some nodes y and z then
6:       Delete edges (x, y) and (y, z)
7:       Insert passive edge (x, z)
8:     end if
9:   end for
10: end loop
```

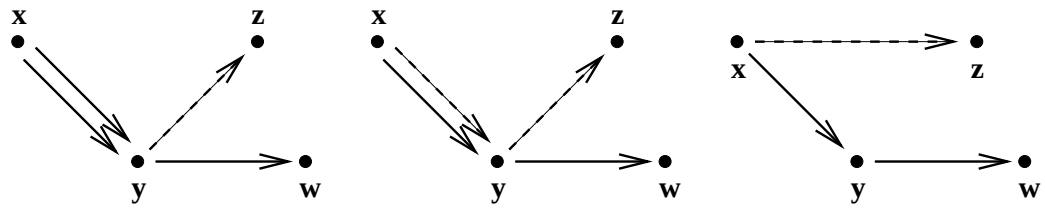

Figure 4.2: Protocol 5 followed by Protocol 6

4.5 Node Subscription

A new node x *subscribes* to the group by sending a *subscription message* to an arbitrary member of the group, the *contact node*. We make no assumptions about how x selects the contact node. Perhaps every new subscriber selects the same contact node, or at the other extreme, perhaps a new subscriber picks a contact node at random. Our protocol is designed to work properly regardless of how the contacts are selected.

The contact node forwards the subscribe message to a randomly chosen node, w, which we call the *indirect contact node*, and which will be in charge of attaching x to the group. We use Protocol 2 to find w, starting from the contact node. Let d be the degree of the indirect contact node.

The purpose of choosing an indirect contact node instead of using the initial contact node is to avoid bias in the choice of d; we want the expected value of d to be the average degree of a node in G. Since it is possible for every new subscriber to select the same contact node, it would introduce bias if the degree of this contact node were larger or smaller than average.

In order to meet the conditions given in the hypothesis of Theorem 4.1, we need to increase the number of edges of G by $d + 2$ when we add the node x. However, since we want x to have the same degree as every other node, we should only add d out-edges to x. These two goals are achieved by executing the following protocol.

Protocol 7: A Node x Subscribes: w is the indirect contact node

1: **if** w is the only node in G **then**
2: Create *Min_deg* edges from x to w.
3: Create *Min_deg* edges from w to x.
4: **else**
5: $d \leftarrow \max\{degree(w), Min\_deg\}$ {Guard against creating nodes of small degree}
6: **loop** {d times}
7: Use Protocol 4 to increment the active degree of x
8: **end loop**
9: **for** i from 1 to 2 **do**
10: Use Protocol 2 to pick a random node s_i
11: Use Protocol 4 to increment the active degree of s_i
12: **end for**
13: **end if**
14: Initialize satellites of x, hosted by w

Figure 4.3 illustrates Protocol 7 in the case that $d = 3$.

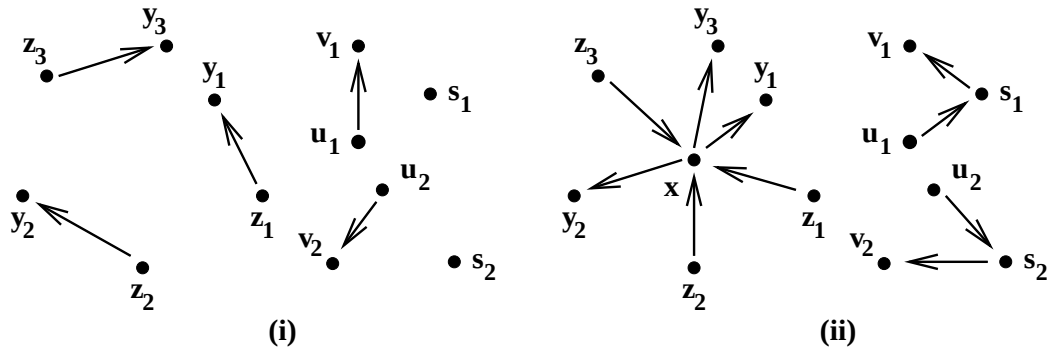

Figure 4.3: The node x subscribes, in a case where $d = 3$. The three edges chosen in Line 7 are (z_1, y_1), (z_2, y_2), and (z_3, y_3). The two edges chosen in Lines 10 and 11 are (u_1, v_1) and (u_2, v_2).

4.6 Evolution of the Overlay Graph Starting from a Single Node

In this section, we do not introduce another protocol. Instead, we describe the evolution of the overlay graph G under the following simplifying assumptions.

1. Initially, G contains exactly one node.
2. When a new node subscribes, the indirect contact node is randomly chosen among nodes that are already in G, and Protocol 4 is used to attach the new node to G.

3. No node fails or unsubscribes.

4. We do not mark any edge passive.

Let x_1 be the original node of G, and let Let x_i be the i^{th} node of G in order of joining G.

Recall that $H_m = \sum_{i=1}^{m} \frac{1}{i}$ is the m^{th} *harmonic number.*

Theorem 4.1 *If G has cardinality $n \geq 2$, then, for all $1 \leq i \leq n$:*
(a)$degree(x_i) \geq Min\_deg$.
(b)$Exp(degree(x_i)) = Min\_deg - 3 + 2H_n$.

Note that the statement of Theorem 4.1 is stronger than the statement that the expected average degree of G is $Min\_deg - 3 + 2H_n$, because Theorem 4.1 implies that there is no bias between the nodes that joined G earlier and the nodes that joined G later.

Proof: By induction on n. For $n = 2$, we are given, by Lines 2 and 3 of Protocol 7, that $degree(x_i) = Min\_deg = Min\_deg - 3 + 2H_2$, for $i = 1, 2$, and we are done.

Suppose $n > 2$. Let x_i, for $1 \leq i < n$, be the indirect contact node when x_n joins G.

We now consider the steps of Protocol 7. By the inductive hypothesis, $degree(w) \geq Min\_deg$, and thus $d = degree(w)$. After Line 5, the expected value of d will be the average degree of a node of G, which is $Min\_deg - 3 + 2H_{n-1}$ by the inductive hypothesis.

After execution of the loop given in Lines 6–8, x_n is attached to G, and $degree(x_n) = d$. The expected average value of the degree of a node in G is now

$$\frac{n-1}{n}(Mindeg - 3 + 2H_{n-1}) + \frac{1}{n}Exp(d) = Min\_deg - 3 + 2H_{n-1}$$

.

The loop in Lines 9–12 increase the degree of two nodes of G, which are chosen at random. For each i, the expected increase of $degree(x_i)$ is $\frac{2}{n}$, and the value of $degree(x_i)$ is thus

$$Min\_deg - 3 + 2H_{n-1} + \frac{2}{n} = Min\_deg - 3 + 2H_n$$

Thus (b) holds. Since the degree of no node decreases, (a) also still holds. □

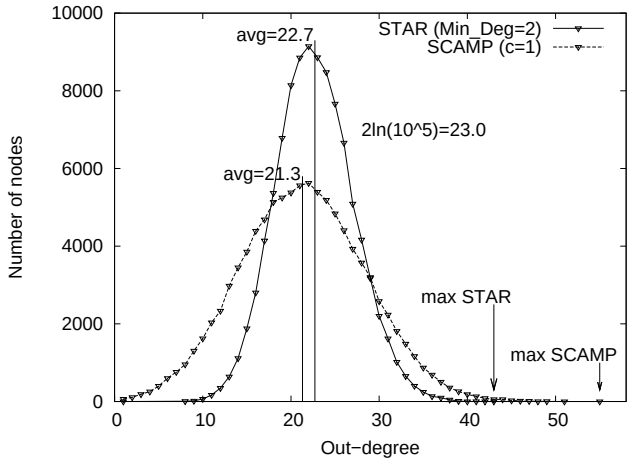

Figure 4.4: Distribution of out-degrees for STAR and SCAMP, networks resulting from $n = 10^5$ subscriptions

We now highlight the characteristics of STAR, bootstrapping from 1 to 10^5 nodes. Simulation using PeerSim produces Figure 4.4, which plots the degree distribution of nodes for both STAR and SCAMP, for a typical run. Those two protocols have been parametrized so their expected out-degree is the same: $2\ln(n)$ for STAR ($Min\_deg = 2$) and $(c+1)\ln(n)$ with $c = 1$ for SCAMP. As expected, the out-degrees of STAR are more sharply distributed around the expected value. Table 1 indicates that the diameter of G grows more slowly under STAR than under SCAMP.

Size	10^2	10^3	10^4	10^5
SCAMP	5	7	8	8
STAR	4	4	5	6

Table 1: Diameters of generated graphs on typical runs

4.7 Unsubscribing

Unsubscribe. When a node x unsubscribes, strong connectivity of G^a are maintained.

Protocol 8: A Node x Unsubscribes

1: Let $v_1, v_2, \ldots v_d$ be the in-neighbors of x. {active or passive}
2: Let $y_1, y_2, \ldots y_d$ be the out-neighbors of x. {active or passive}
3: **if** (x, y_i) is active for all i **then**
4: **for** i from 1 to $d-1$ **do**
5: Insert active edge (y_i, y_{i+1})
6: **end for**
7: Insert active edge (y_d, y_1)
8: **else** {By the input condition, (v_i, x) is active for all i}
9: **for** i from 1 to $d-1$ **do**
10: Insert active edge (v_i, v_{i+1})
11: **end for**
12: Insert active edge (v_d, v_1)
13: **end if**
14: **for** i from 1 to d **do**
15: **if** (v_i, x) and (x, y_i) are both active **then**
16: Insert active edge (v_i, y_i)
17: **else**
18: Insert passive edge (v_i, y_i)
19: **end if**
20: Delete (v_i, x)
21: Delete (x, y_i)
22: **end for**

The edges $\{(y_i, y_{i+1})\}$ and (y_d, y_1), or $\{(v_i, v_{i+1})\}$ and (v_d, v_1), form a cycle, which we call the *unsubscription cycle*. Figure 4.5 shows this action in a case where $d = 3$ and all (x, y_i) are active, and Figure 4.6 shows the other case. In both figures, the dashed arrows represent edges which are marked passive.

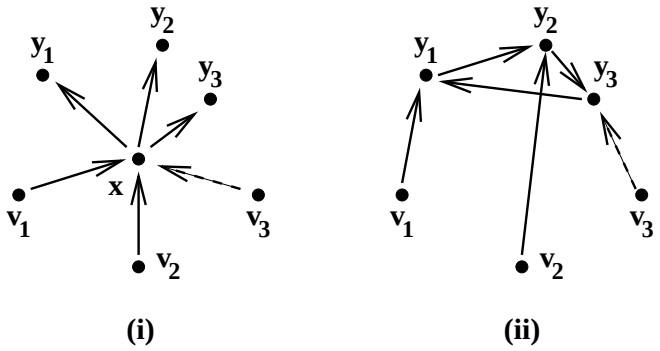

Figure 4.5: Unsubscribing x, if x has no passive out-edges.
Passive edges are shown as dashed arrows.

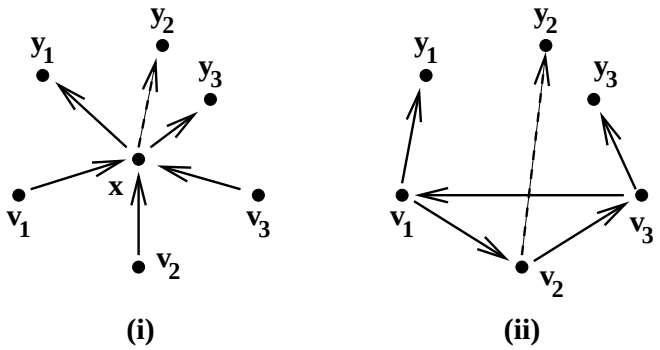

Figure 4.6: Unsubscribing x, if x has no passive in-edges.
Passive edges are shown as dashed arrows.

Lemma 4.2 *If G satisfies the parity condition, then G satisfies the parity condition after one execution of Protocol 8.*

Proof: Each node in the unsubscription cycle increases its in-degree and out-degree by one each. Each other node retains the same in-degree and out-degree. □

Recall that G^a is the subgraph of G consisting of all nodes and only active edges.
Lemma 4.3 *If G^a is strongly connected, then G^a remains strongly connected after one execution of Protocol 8.*

Proof: Suppose $a \neq x$ and $b \neq x$ are nodes. We need to show there is a path from a to b in G^a after the protocol executes. By hypothesis, there is a simple path σ in G^a from a to b before the protocol executes. If σ does not contain x, then we are done.

Otherwise, σ contains two consecutive edges (v_i, x) and (x, y_j) for some i, j. If $j = i$, then simply replace those two edges by (v_i, y_i) in σ. Otherwise, replace the two edges by the edge (v_i, y_i),

followed by the path from y_i to y_j in the unsubscription cycle, or by the path from v_i to v_j in the unsubscription cycle, followed by the edge (v_j, y_j). □

5 The Balancing Mechanism of STAR.

5.1 The Issues

In this section, we consider the problem of maintaining the topology of G. The topology of G could be badly initialized, or it could change to an undesirable state by repeated executions of subscription and unsubscription, or by node failures. We will give a *balancing mechanism*, consisting of Protocols 9, 10, 11, which will restore the topology from any legitimate state.

If there are two or more copies of the same active edge, Protocol 5 is enabled to execute, marking all but one of those copies passive. We can assume that it executes instantly whenever a duplicate active edge is inserted. If $passv\_in\_deg(x) > 0$ and $passv\_out\_deg(x) > 0$, Protocol 6 is enabled to execute, decreasing each by one, We can assume that it executes as soon as possible whenever that condition holds.

Good Topology. Intuitively, the topology of G^a is *good* if, in addition being strongly connected, G^a is approximately a random directed graph. This implies that the following three (loosely defined) criteria are satisfied.

- The average value of $actv\_out\_deg(x)$ (which is necessarily the same as the average value of $actv\_in\_deg(x)$) is approximately equal to the desired value, which we have chosen to be $2\ln(n)$.
- The variances of $actv\_out\_deg(x)$ and $actv\_in\_deg(x)$ are not too great.
- The diameter of G^a is small.

The diameter of G^a that we strive to achieve will be approximately the expected diameter of a random directed graph of average degree $2\ln(n)$, which is $\frac{\ln n}{\ln\ln n + \ln 2} + O(1)$.

It is important to realize that STAR maintains the above three criteria with high probability, but that it maintains strong connectivity of G^a with certainty, provided there are no node failures or other errors. The parity property of G is also maintained with certainty, again barring errors.

5.2 The Local Balance Protocol

Protocol 9, ensures that the active out-degree and active in-degree of each node remain approximately the same. If the number of active out-edges of a node x exceeds the number of active in-edges by more than *Max_diff_deg*, a positive number that is a design parameter, then x chooses a random node y, and if y has more active in-edges than active out-edges, a new active edge (x, y) and a new passive edge (y, x) are inserted; this has the effect of improving the balance of active edges for both x and y. A similar action occurs if the number of active in-edges of x exceeds the number of active out-edges by more than *Max_diff_deg*.

Protocol 9: Balance Essential Degrees of x

```
1: loop {forever}
2:   if actv_in_deg(x) > actv_out_deg(x) + Max_diff_deg then
3:                     {Max_diff_deg ≥ 1 is a design parameter}
4:     Choose a random node y, using Protocol 2.
5:     if actv_in_deg(y) < actv_out_deg(y) then
6:       Insert active edge (x, y) into G.
7:       Insert passive edge (y, x) into G.
8:     end if
9:   else if actv_out_deg(x) > actv_in_deg(x) + Max_diff_deg then
10:    Choose a random node y, using Protocol.
11:    if actv_in_deg(y) > actv_out_deg(y) then
12:      Insert passive edge (x, y) into G.
13:      Insert active edge (y, x) into G.
14:    end if
15:  end if
16: end loop
```

The combined effect of Protocols 6 and 9 will be to keep the passive degrees of each node low.

Lemma 5.1 *If the parity property holds, Protocols 6 and 9 will run until, for each node x, at least one of $passv\_out\_deg(x)$ and $passv\_in\_deg(x)$ is zero and the other is at most Max_diff_deg.*

Proof: Protocol 9 will run until the difference between $actv\_out\_deg(x)$ and $actv\_in\_deg(x)$ is at most Max_diff_deg for every node x. By the parity property, $passv\_in\_deg(x) - passv\_out\_deg(x) = actv\_out\_deg(x) - actv\_in\_deg(x)$. Thus the difference between $passv\_out\_deg(x)$ and $passv\_in\_deg(x)$ is at most Max_diff_deg for ever node x. Protocol 6 will run until at least one of those two degrees is zero, which then implies that the other is at most Max_diff_deg. □

5.3 The Balancing Protocol

The heart of STAR is the *balancing protocol*, Protocol 10. The input to one execution of Protocol 10 is an active edge (x, y). An edge (x, y) initiates an instance of Protocol 10 if instructed to do so by the *balance supervisor*, Protocol 11, which runs continuously.

If that execution of Protocol 10 runs to completion (it may *abort* instead) one of the following will happen.

1. Either x or y will execute Protocol 4, increasing the number of edges of G^a by one.
2. The edge (x, y) will be marked *passive*, decreasing the number of edges of G^a by one.

The effect of many executions of the balancing protocol is to create a dynamic equilibrium, in which the expected degree of a node converges to approximately $2n \ln(n)$. In addition, the

topology of G^a will converge to a topology which is reasonably close to that of a randomized directed graph.

5.3.1 Finding a Detour

The "main" branch of Protocol 10 is the branch consisting of Lines 8 through 27 of the code. The goal of that branch is to find a *detour* for (x, y), namely a path from x to y in G^a which does not use the edge (x, y) itself. If such a detour is found, then (x, y) is marked passive. Figure 5.1 shows a simple example, where there is a detour of (x, y) of length 2.

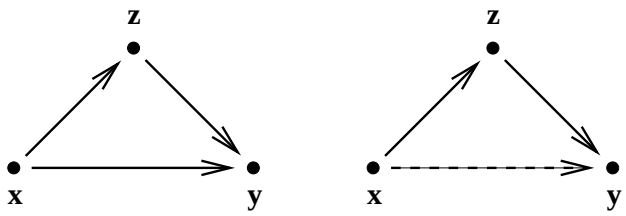

Figure 5.1: Mark (x, y) as passive.

We want G^a to maintain some redundancy; thus we do not want to seek detours too diligently. In fact, the protocol only looks for a detour whose length is not too great; if it fails to find a sufficiently short detour, it gives up the search, and executes Protocol 4.

5.3.2 Blossoms

Let $d = actv\_out\_deg(x)$, and let $\widehat{d} = actv\_in\_deg(y)$. The first step in the main branch of the balancing protocol is to compute a non-negative integer r and a real number $0 \leq \lambda < 1$, using the numbers d and $\hat{d}$ as inputs.

We give the computation of r, and λ in Section 5.4, below, as well as an explanation of those computations. Some values of r, and λ are given in Table 1. For now, we simply assume that r and λ are given.

Given an edge (x, y), define $G^a_{\neg(x,y)}$ to be the directed graph consisting of all nodes of G, together will all active edges of G other than (x, y). Thus, $G^a_{\neg(x,y)}$ is obtained from G^a by deletion of just one edge.

Protocol 10 builds a *forward blossom*, $Forw\_bloss(x, y)$, based at x, which contains some nodes which can be reached from x by paths in $G^a_{\neg(x,y)}$, and also a *backward blossom*, $Back\_bloss(x, y)$, based at y, which contains some nodes which can reach y by paths in $G^a_{\neg(x,y)}$. The radii of both blossoms is approximately r. More specifically stated, $Forw\_bloss(x, y)$ contains all nodes which can be reached from x by paths of length r, a portion (depending on λ) of the nodes that can be reached by paths of length $r + 1$, and no other nodes. A similar rule holds for $Back\_bloss(x, y)$. A detour is then found if these two blossoms have a node in common. We now give the detailed construction of the blossoms.

- For any path σ, define $Start(\sigma)$ and $End(\sigma)$ to be the starting and ending nodes of σ. If $\mathcal{P}$ is a set of paths, let $Starts(\mathcal{P})$ and $Ends(\mathcal{P})$ be the set of starting and ending nodes of members of $\mathcal{P}$, respectively.

- For any two nodes u, v, let $dist^*(u, v)$ be the shortest length of any path in $G^a_{\neg(x,y)}$ from u to v. That is, the shortest length of any path of active edges, not including the edge (x, y).

- For any integer $i \geq 0$ and node u, let $A(u, i) = \{v : dist^*(u, v) = i\}$, and let $B(u, i) = \{v : dist^*(v, u) = i\}$.

- For any integer $i \geq 0$ and node u, let Let $\mathcal{A}(u, i)$ be the set of all edges which start at nodes in $A(u, i)$, and let $\mathcal{B}(u, i)$ be the set of all edges which end at nodes in $B(u, i)$.

- For any $i \geq 0$ and node u, and any probability p, let $\mathcal{A}(u, i, p)$ be a subset of $\mathcal{A}(u, i)$, where each member is randomly and independently chosen to be a member with probability p, and let $\mathcal{B}(u, i, p)$ be a subset of $\mathcal{B}(u, i)$, where each member is randomly and independently chosen to be a member with probability p.

- For any $i \geq 0$ and node u, and any probability p, let

$$\begin{aligned} A(u, i, p) &= Ends(\mathcal{A}(u, i, p)) - \bigcup_{j=0}^{i} A(u, j) \\ B(u, i, p) &= Starts(\mathcal{B}(u, i, p)) - \bigcup_{j=0}^{i} B(u, j) \end{aligned}$$

  Note that we could also write:

$$\begin{aligned} A(u, i, p) &= Ends(\mathcal{A}(u, i, p)) \cap A(u, i + 1) \\ B(u, i, p) &= Starts(\mathcal{B}(u, i, p)) \cap B(u, i + 1) \end{aligned}$$

- Finally, let

$$\begin{aligned} Forw\_bloss(x, y) &= \bigcup_{i=0}^{r} A(x, i) \cup A(x, r, \lambda) \\ Back\_bloss(x, y) &= \bigcup_{i=0}^{r} B(y, i) \cup B(y, r, \lambda) \end{aligned}$$

  We call the sets $A(x, i)$, for $i = 0, 1, \ldots r$, the *layers* of the blossom $Forw\_bloss(x, y)$, and we call $A(x, r, \lambda)$ the *fringe* of the blossom. We define the layers and fringe of $Back\_bloss(x, y)$ similarly. Note that $dist^*(x, u) \leq r \Rightarrow u \in Forw\_bloss(x, y) \Rightarrow dist^*(x, u) \leq r + 1$, and $dist^*(u, y) \leq r \Rightarrow u \in Back\_bloss(x, y) \Rightarrow dist^*(u, y) \leq r + 1$.

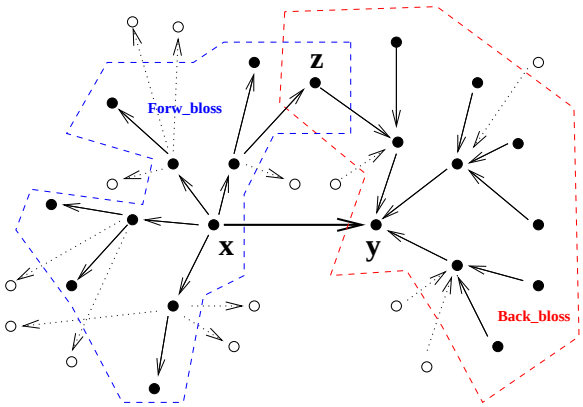

Figure 5.2: Forward and Backward Blossoms. In this figure, $r = 1$. The solid circles represent members of the blossoms. The blossoms contain all nodes of $A(x, 1)$ and $B(y, 1)$, but only some of the nodes of $A(x, 2)$ and $B(y, 2)$. The open circles represent nodes which are in $A(x, 2)$ or $B(y, 2)$, but are not in the blossoms.

The two blossoms intersect, creating a detour of (x, y). The figure is not accurate, since $d = 5$ is inconsistent with $r = 1$. An accurate figure would either be too crowded or fail to show all structure.

Protocol 10: Balancing Protocol for an Edge (x, y).

1: $d = actv\_out\_deg(x)$
2: $\widehat{d} = actv\_in\_deg(y)$
3: **if** $d \leq Min\_deg$ or $2d \leq \widehat{d}$ **then**
4: Use Protocol 4 to increment the degree of x
5: **else if** $\widehat{d} \leq Min\_deg$ or $2\widehat{d} \leq d$ **then**
6: Use Protocol 4 to increment the degree of y
7: **else** {We now know that $\widehat{d} - 1 \leq 2(d-1)$ and $d - 1 \leq 2(\widehat{d} - 1)$}
8: Compute r and λ
9: $Forw\_bloss \leftarrow \{x\}$
10: $Back\_bloss \leftarrow \{y\}$
11: **for** i from 1 to r **do**
12: **if** $Forw\_bloss \cap Back\_bloss = \emptyset$ **then** {If a detour has already been found, we do not need to compute the remaining layers}
13: Compute $A(x, i)$ and $B(y, i)$
14: $Forwbloss \leftarrow Forw\_bloss \cup A(x, i)$
15: $Backbloss \leftarrow Back\_bloss \cup B(y, i)$
16: **end if**
17: **end for**
18: **if** $Forw\_bloss \cap Back\_bloss = \emptyset$ **then** {If a detour has already been found, we do not need to compute the fringes}
19: Compute $A(x, r, \lambda)$ and $B(y, r, \lambda)$
20: $Forw\_bloss \leftarrow Forw\_bloss \cup A(x, r, \lambda)$
21: $Back\_bloss \leftarrow Back\_bloss \cup B(y, r, \lambda)$
22: **end if**
23: **if** $Forw\_bloss \cap Back\_bloss = \emptyset$ **then**
24: Use Protocol 4 to increment the degree of x
25: **else**
26: Mark (x, y) passive
27: **end if**
28: **end if**

The Balancing Supervisor. Protocol 10 is repeatedly called by Protocol 11, which runs continuously. The probability that a node will begin to search for a passive path is inversely proportional to one plus its passive in-degree. This bias is enforced by using *tickets*; a ticket is a real number in the range $[0, 1)$, and the smaller the ticket the more likely the node is to finish its execution of the protocol. The purpose of this bias is to favor generating passive out-edges at nodes where there are passive in-edges which need to be canceled.

Purpose of Abort. The purpose of abort is to make sure that every node gets a chance to complete an execution of Protocol 10 at least some of the time. Our concern is that, every time x tries to execute the protocol, some of the nodes it needs are busy executing other instances of the protocol. Eventually, x will have the lowest ticket during for an interval that is long enough for it to complete an execution of Protocol 10.

Another purpose of abort is shown in Lines 18–24 of Protocol 11. If (u, v) and (x, y) are both running Protocol 10, and each of those two executions uses the other edge in one of its blossoms, it is possible for each edge to have a detour that uses the other edge. Marking both edges as passive could then cause G^a to lose its strong connectivity. To avoid this possibility, we abort one of the executions.

Protocol 11: Balancing Supervisor

```
 1: for all nodes x of G do
 2:   loop
 3:     Wait one time unit, according to x's clock.
 4:     Pick a random real number rand uniformly in the interval (0, 1].
 5:     biased_rand ← rand / (1+passv_in_deg(x))
 6:     if x is currently executing Protocol 10 and biased_rand < ticket(x) then
 7:       x aborts the current version of Protocol 10.
 8:     end if
 9:     if x is not currently executing Protocol 10 then
10:       ticket(x) ← biased_rand
11:       Choose a random active out-edge (x, y) of x
12:       x begins execution of a new instance of Protocol 10 for (x, y)
13:     end if
14:     if some node is involved in the execution of too many instances of Protocol 10
                                  and ticket(x) is the largest ticket of all those instances then
15:       {"Too many" means that the memory of that node is overloaded.}
16:       x aborts its current execution of Protocol 10.
17:     end if
18:     if an edge (u, v) is used for one of the blossoms
                                  and Protocol 10 is running for edge (u, v) then
19:       if ticket(u) < ticket(x) then
20:         Instruct x to abort its execution of Protocol 10
21:       else
22:         Instruct u to abort its execution of Protocol 10
23:       end if
24:     end if
25:   end loop
26: end for
```

In Line 14, "too many" means more than that node can handle, given its space limitations.

5.4 Computing r and λ.

In computing r and λ, the protocol makes the assumptions[1] that $actv\_out\_deg(z) = d$ for every node z, that $d = 2\ln(n)$, and that G^a is a randomly chosen directed graph with those properties. Write $\bar{n} = \ln(d/2)$, the assumed value of n.

For any $i \geq 0$, let $X(x,i) = \bigcup_{j=0}^{i} A(x,i)$, and let $Y(y,i) = \bigcup_{j=0}^{i} B(x,i)$. Thus, there is a detour of (x,y) of length at most $2i$ if and only if $X(x,i) \cap Y(y,i) \neq \emptyset$.

By definition, we have:

Remark 5.2 *If $i + j = k + \ell$, then $A(x,i) \cap B(y,j) = \emptyset \iff A(x,k) \cap B(y,\ell)$*

Corollary 5.3

$$Forw\_bloss(x,y) \cap Back\_bloss(x,y) = \emptyset \iff$$

$$\left(\bigcup_{i=0}^{r} A(x,i) \cap B(y,i)\right) \cup \left(\bigcup_{i=1}^{r} A(x,i) \cap B(y,i-1)\right) \cup$$

$$\Big(A(x,r,\lambda) \cap B(y,r)\Big) \cup \Big(A(x,r) \cap B(y,r,\lambda)\Big) \cup \Big(A(x,r,\lambda) \cap B(y,r,\lambda)\Big) = \emptyset$$

We now define four sequences of real numbers, $\{a_i\}$, $\{x_i\}$, $\{b_i\}$, and $\{y_i\}$ as follows.

1. $a_0 = x_0 = b_0 = y_0 = 1$
2. For $i = 1$, we define
$$a_1 = d - 1$$
$$x_1 = d$$
$$b_1 = \widehat{d} - 1$$
$$y_1 = \widehat{d}$$
3. For any $i \geq 2$:
$$a_i = \left(e^{d/2} - x_{i-1}\right)\left(1 - e^{-a_{i-1}d/\bar{n}}\right)$$
$$x_i = x_{i-1} + a_i$$
$$b_i = \left(e^{d/2} - y_{i-1}\right)\left(1 - e^{-b_{i-1}d/\bar{n}}\right)$$
$$y_i = y_{i-1} + b_i$$
4. Finally, let
$$\alpha = \left(e^{d/2} - x_r\right)\left(1 - e^{-a_r\lambda d/\bar{n}}\right)$$
$$\beta = \left(e^{d/2} - y_r\right)\left(1 - e^{-b_r\lambda d/\bar{n}}\right)$$
$$\gamma = \left(e^{d/2} - y_r\right)\left(1 - e^{-b_r\lambda(1-\lambda)d/\bar{n}}\right)$$

Lemma 5.4

[1] These assumptions could be wildly incorrect; they simply form the basis for the computation of r and λ.

(a) *The expected cardinality of* $A(x,0,\lambda) \cap B(y,0,\lambda)$ *is* $\frac{(d-1)(\widehat{d}-1)\lambda^2}{\bar{n}-1}$.

(b) *The expected cardinality of* $A(x,1) \cap B(y,1)$ *is* $\frac{(d-1)(\widehat{d}-1)}{\bar{n}-1}$.

(c) *For any* $2 \le i \le r$ *the expected cardinality of* $A(x,i) \cap B(y,i)$ *is approximately* $\frac{a_i b_i}{\bar{n}}$.

(d) *For any* $2 \le i \le r$ *the expected cardinality of* $A(x,i) \cap B(y,i-1)$ *is approximately* $\frac{a_i b_{i-1}}{\bar{n}}$.

(e) *The expected cardinality of* $A(x,r,\lambda) \cap B(y,r)$

(f) *The expected cardinality of* $\Big(A(x,r,\lambda) \cap B(y,r)\Big) \cup \Big(A(x,r) \cap B(y,r,\lambda)\Big)$ *is approximately* $\frac{\alpha b_r + a_r \beta}{\bar{n}}$

(g) *The expected cardinality of* $A(x,r,\lambda) \cap B(y,r,\lambda)$ *is approximately* $\frac{\alpha\gamma}{\bar{n}}$

Choice of r and λ. For any integer $r \ge 0$ and real λ,

$$F(r,\lambda) = \begin{cases} \frac{(d-1)(\widehat{d}-1)\lambda^2}{\bar{n}-1} & \text{if } r = 0 \\ \frac{(d-1)(\widehat{d}-1)}{\bar{n}-1} + \sum_{i=2}^{r} \left(\frac{a_i b_i + a_i b_{i-1}}{\bar{n}}\right) + \frac{\alpha b_r + a_r \beta + \alpha\gamma}{\bar{n}} & \text{if } r \ge 1 \end{cases}$$

Choose an integer $r \ge 0$ and a real number λ, where $0 \le \lambda < 1$, such that

$$F(r,\lambda) \quad = \quad \ln 2 \tag{1}$$

Lemma 5.5 *For any* $d, \widehat{d} \ge 2$*, there is a unique choice of* ϱ *and* λ *which satisfies Equation* (1).

Proof: For any real number $x > 0$, define $f(x) = F(\lfloor x \rfloor, x - \lfloor x \rfloor)$. The lemma follows immediately from the following three claims:

Claim I: $\lim_{x \to 0} f(x) = -\infty$

Claim II: $\lim_{x \to \infty} f(x) = \infty$

Claim III: f is continuous and strictly monotone increasing.

Claims I and II are trivial.

Claim III follows from two statements:

1. For any $r \ge 0$, $F(r,1) = F(r+1,0)$.
2. For fixed $r \ge 0$, $F(r,\lambda)$ is a continuous and strictly monotone increasing function of λ in the range $\lambda \ge 0$.

The first statement can be verified algebraically by substitution into the formula, and the second statement follows from elementary calculus. □

Lemma 5.6 *If* G^a *is a random directed graph, if* d *is approximately equal to* $2\ln(n)$*, which is approximately the average degree of* G^a*, then the probability that Forw_bloss* $\cap$ *Back_bloss* $= \emptyset$ *is approximately* $\frac{1}{2}$.

Proof: (Sketch) We can show that the cardinality of the set given in Corollary 5.3 has close to a Poisson distribution, with mean approximately $\ln 2$. We then apply Theorem C.1. □

$d = \widehat{d}$	r	λ	$\bar{n}$
3	0	0.3702	4.48
4	0	0.2873	7.39
5	0	0.2820	12.18
6	0	0.3138	20.09
7	0	0.3822	33.12
8	0	0.5083	54.60
9	0	0.8105	90.02
10	1	0.0134	148.41
11	1	0.0291	244.69
12	1	0.0456	403.43
13	1	0.0635	665.14
14	1	0.0833	1096.63
15	1	0.1056	1808.04
16	1	0.1311	2980.96

$d = \widehat{d}$	r	λ	$\bar{n}$
17	1	0.1605	4914.77
18	1	0.1947	8103.08
19	1	0.2349	13359.73
20	1	0.2821	22026.47
21	1	0.3381	36315.50
22	1	0.4046	59874.14
23	1	0.4839	98715.77
24	1	0.5788	162754.79
25	1	0.6926	268337.29
26	1	0.8294	442413.39
27	1	0.9943	729416.37
28	2	0.0069	1202604.28
29	2	0.0150	1982759.26
30	2	0.0242	3269017.37

Table 1: Values of r and λ for a few choices of d, where we also assume that $\widehat{d} = d$. The last column shows the value of $\bar{n}$ for which $d = 2\ln(\bar{n})$.

6 Self-Stabilization of STAR

The network could be initialized in an illegitimate configuration, or an illegitimate configuration could result from an *error*, meaning corruption of the memory of one or more nodes, or complete node failures. From any configuration, as long as G is at least weakly connected, STAR will converge to a legitimate state in finite time, given that no additional errors occur during that time.

6.1 Detecting Failure

Each node x sends a *heartbeat* message to each of its neighbors during every unit time interval, according to its own clock. We assume that there is some global constant Λ such that no node's clock runs faster than Λ times any other node's clock. Thus, if x does not receive a heartbeat message from y within any interval of length Λ, x concludes that y has failed.

Similarly, x can detect the failure of any member of $x.hosts$ or $x.guests$, since Protocol 3 requires continual back-and-forth messages between a node and its guest or host.

Linking to a Failed Node. At times, a protocol will execute the instruction to create a link between a node x and some other node y when that node has already failed. In that case, x treats y as a failed neighbor.

Failure of a Satellite Link. The failure of a guest or host is the simplest type of error to recover from. If $y \in x.guests$ fails, x does nothing. If $y \in x.hosts$ fails, then the satellite that y hosted is also dead, and thus x must create a replacement satellite.

6.2 Restoration of Parity

Suppose that parity does not hold. Then there is some node x for which $in\_deg(x) < out\_deg(x)$, and there must be at least one node y for which $out\_deg(y) < in\_deg(y)$. As a satellite of x continually host, using a random walk, eventually it must be hosted by y, at which point parity will be partially restored, through Protocol 12.

Protocol 12: Restore Parity to x

```
1: loop {forever}
2:    for all y ∈ x.hosts do
3:       if in_deg(x) < out_deg(x) and out_deg(y) < in_deg(y) then
4:          Insert passive edge (y, x) into G.
5:       end if
6:    end for
7: end loop
```

6.3 Recovery from Single Node Failure

Our protocol can handle node failure, as long as no two adjacent nodes fail simultaneously. Actually, if a node x fails and if y is a neighbor of x, STAR only works if, after a failure of x, y does not fail for an interval of time long enough for the overlay graph to recover from the failure of x.

Writing a Will. As each node x adds any other node y to its view, it must send a will to y, and it may also need to send an updated copy of its will to some of its other neighbors. The will that x sends to a given neighbor consists simply of the instructions that that neighbor would execute if x decided to unsubscribe, as given in Protocol 8.

Let us consider the case illustrated in Figure 4.5(i). The node x will send the message to each v_i, "In case I fail, link to y_i," and will send the message to each y_i, "In case I fail, add v_i and y_{i-1} to your in-view, and link to y_{i+1}," (where $i-1$ and $i+1$ are taken modulo $degree(x)$, of course). Thus, if x fails, the neighbors of x will join together in the manner shown in Figure 4.5(ii), exactly as they would have if x unsubscribed.

This simple recovery method, given here as Protocol 13, will restore the overlay graph to a legitimate state as long as failures of nodes are total, *i.e.,* a failing node shuts down completely, never sending another message, and if failures are isolated in time or space, meaning that no two adjacent nodes fail simultaneously, and if one fails, the recovery protocol completes itself before any neighbor node fails.

In a realistic implementation and deployment of our protocol, we expect the recovery phase for a failure to be on the order of a very few seconds, as simple relinking between direct neighbors is implied. We then argue that additional failure in the same small neighborhood, of size approximately $2\ln(n)$, during this short time interval is rare.

Protocol 13: A Node x Executes the Will of a Failed Neighbor y

1: **if** x has a valid will for y **then**
2: x executes all instructions in that will.
3: **end if**

6.4 Recovery from Complex Failure

We now consider what the self-stabilizing protocol will do if a node discovers that the network is in an illegitimate state, and the situation is not simply a parity error, or the failure of a node that left a valid will. If a node x detects that it is in such a state, it will try to guarantee connection with all nodes whose addresses are known to it, by executing Protocol 14. These include all nodes in $x.inview$, $x.outview$, $x.hosts$, and $x.guests$, and all nodes whose IDs are included in the wills of any neighbors of x.

Protocol 14: Emergency Linking of x

1: **for** every node y known to x **do**
2: Insert active edge (x, y) into G.
3: Insert active edge (y, x) into G.
4: **end for**

We remark that execution of Protocol 14, after a massive failure of the network, could introduce far more edges than necessary; Protocol 9 will introduce even more edges in an attempt to balance the active in and out-degrees of each node. But after legitimacy is restored, Protocol 11, running continuously, will "clear out" the extra edges by marking them passive, and then most of them will be deleted by Protocol 6. In fact, after Protocols 6 and 9 have done their work, the total number of passive edges in the entire overlay graph cannot exceed $\frac{n\ Max\_diff\_deg}{2}$.

We summarize the recovery steps which take place when a massive error occurs, provided no other errors occur during that recovery. For simplicity, we assume that no nodes subscribe or unsubscribe during these steps.

1. All nodes that do not detect the error continue their normal behavior.
2. Every node that detects the error executes Protocol 14. (If G is not connected at this point, recovery is not possible.)
3. Protocol 12 runs until parity is restored. Protocols 6, 9, and 10 could be running concurrently. At this point, the overlay graph is in a legitimate state.
4. Protocol 9 runs until the difference between the active in and out-degrees of all nodes are at most Max_diff_deg. Protocols 6 and 10 could be running concurrently.

5. Protocol 10 runs until the average degree of a node is approximately $2\ln(n)$, and the diameter is $\frac{\ln n}{\ln\ln n+\ln 2} + O(1)$. Protocol 6 could be running concurrently.

6. Protocol 6 runs until there are at most $\frac{nMax\_diff\_deg}{2}$ passive edges in the overlay graph.

6.5 Priorities

The priority problem occurs when a node is involved in the execution of more than one protocol (or more than one instance of the same protocol) simultaneously, and those instructions conflict.

We first consider only the case that no node failure or other error occurs, and that each node has sufficient memory to execute all actions requested by the messages it receives. We will deal with the more complex situation of possible error, or limited node capacity in 6.5.2.

6.5.1 Resolving Error-Free Conflict

Random Walk Conflict. If a node is participating in a random walk protocol, namely Protocols 1 or 2, at the same time that its out-view changes, by insertion or deletion of an edge, then it simply executes the commands in arbitrary order. Which it does first will effect the random walk, but since the outcome is random anyway, that is not a problem.

Active Edge Conflict. The most complex kinds of conflict are those than involve an active edge, say (x, y). Protocol 1 cannot return the edge (x, y) if that edge is currently involved in Protocol 4, Protocol 8, or Protocol 10. In particular, these means that two concurrent instances of Protocol 1 may not return the same edge.

If Protocol 1 selects an edge (x, y) and (x, y) is currently in use as one of the edges of the BFS spanning tree of one of the blossoms of some instance of Protocol 10, then 1 simply waits until Protocol 10 releases (x, y), and then returns it.

If Protocol 1 selects an edge (x, y) and (x, y) is currently in use by Protocol 4 or 8, if (x, y) is the input parameter of an instance of Protocol 10, or if another instance of 1 has already selected that edge and is waiting, then the protocol abandons (x, y) and continues its random walk.

If an instance of Protocol 10 needs to use an edge to build one of its BFS spanning trees, and that edge is currently in use by another instance of Protocol 10 in one of its spanning trees, there is no conflict. If that edge is currently in use by Protocol 4 or 8, or if there is a current instance of Protocol 1 waiting to return that edge, then Protocol 10 simply "by-passes" it, treating it as if it were not there while building the blossom.

A more severe conflict occurs if an edge is the input parameter of one instance of Protocol 10, and another instance of the same protocol wants to use the same edge for one of its blossoms. In this case, the instance of Protocol 10 with the larger ticket aborts, releasing all its edges.

Passive Edge Conflict We can assume that Protocol 5 executes instantly whenever possible, which means, at the same instant that a duplicate active edge is inserted. Thus, that protocol cannot be in conflict with any other.

Protocol 6 can be in conflict with another instance of Protocol 6. This can occur when edges (x, y), (y, z) and (z, w) are all passive, for nodes x, y, z, w. In this case, both y and z will execute

the protocol, and the order is irrelevant, since the outcome of those two executions will be to replace those three passive edges by the one passive edge (x, w).

If (x, y), (y, z) and (y, w) are all passive, for nodes x, y, z, w, then y could execute Protocol 6 in two different ways. This conflict is resolved by randomly picking one of them.

Protocols 8 and 6 could also be in conflict. In this case, Protocol 6 will always have priority.

Node Conflict If Protocol 8 has begun, and is waiting to delete a node x, then Protocol 2 is not allowed to return that node x. In this situation, the random walk simply continues.

6.5.2 Resolving Conflict in Case of Error

The general rule is that fixing errors takes precedence over other actions. although we try to avoid creating new errors by aborting a protocol when its data structure is in an intermediate state. We now consider the problems that result when a node that is currently involved in some action fails.

Suppose that, in the process of executing some protocol, a node x has instructed another node y to do something, and y then fails.

- If y is the current node in a random walk, then x simply starts a new random walk.
- If y is the node supposed to be returned by an instance of Protocol 2, or is one end of an edge to be returned by an instance of Protocol 1, then x simply restarts the search protocol.
- If a node involved in the insertion of an edge fails, then the node at the other end of that edge will detect failure, and will delete that edge from its view. This will create a parity error, which will later be resolved by Protocol 12.
- If a node involved in an instance of Protocol 10 fails, that execution of the protocol is aborted.
- Protocol 14 has priority over everything else. If a node x decides to execute that protocol, it aborts all other executions it is involved in. Protocol 14 cannot be aborted.

6.5.3 Memory Overload

Suppose each node x has limited memory. If more requests come to x than it can handle, it either stores the additional requests or sends a "busy" message back to the node which initiated that request. If the request involved an execution of Protocol 10, that execution is aborted, but in other cases, the request is simply resent.

7 Conclusion

The protocol STAR given in this paper constructs an overlay graph with the asymptotically minimal degree needed for strong connectivity of a random directed graph, as well as the asymptotically minimal diameter that can be achieved for such a graph. The overlay graph constructed by STAR is robust enough to recover from substantial corruption, including node failure.

References

[1] Fan Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

[2] S. Dolev, E. N. Hoch, and R. van Renesse. Self-stabilizing and byzantine-tolerant overlay network. In *OPODIS 2007*, pages 343–357, 2007.

[3] Shlomi Dolev and Elad Schiller. Comminication adaptive self-stabilizing group membership service. *Transactions on Parallel and Distributed Systems*, 14(7):709–720, 2003.

[4] P. Erdös and A. Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[5] T. I. Fenner and A. M. Frieze. On the connectivity of random m-orientable graphs and digraphs. *Combinatorica*, 2:347–359, 1982.

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

[7] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, 2003.

[8] C. J. H. McDiarmid. General percolation and random graphs. *Adv. Apl. Prob.*, 13:40–60, 1982.

[9] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[10] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

Appendix

A The Uniform Random Graph $G_{n,p}$

For any integer $n \geq 0$ and any $0 \leq p \leq 1$, Let $G_{n,p}$ be the random directed graph (V, E) where

1. $|V| = n$
2. If $x, y \in V$, then the probability that $(x, y) \in E$ is p.
3. (Independence) The events $(x, y) \in E$ are independent.

Let $P_{n,p}$ be the probability that $G_{n,p}$ is strongly connected. For fixed n, the value of $P_{n,p}$ rises from 0 to 1 as p increases, but it increases very rapidly near the value of $\frac{\ln(n)}{n}$. For values of p only slightly smaller than $\frac{\ln(n)}{n}$, $P_{n,p}$ is very close to 0, while for values of p only slightly larger than $\frac{\ln(n)}{n}$, $P_{n,p}$ is very close to 1.

From [5] and [8], we have:

Lemma A.1 *If $k > 1$ is a constant, then*

$$\lim_{n\to\infty} P_{n,(k\ln(n)/n)} = 1$$

Lemma A.2 *If $k > 1$ is a constant then, for sufficiently large n, the diameter of $G_{n,(k\ln(n)/n)}$ is $\frac{\ln n}{\ln\ln n + \ln 2} + O(1)$ with high probability.*

B Uniform Stationary Distributions

Definitions. Suppose that $G = (V, E)$ is a directed multi-graph. Let $V = \{v_1, v_2, \ldots v_n\}$.

1. G satisfies the *parity condition* if $in\_deg(v_i) = out\_deg(v_i)$ for every v_i.

Theorem B.1 *A directed multi-graph which satisfies the parity condition is strongly connected if and only if it is weakly connected.*

2. A directed graph is *Eulerian* if it is connected and satisfies the parity condition.
3. A *distribution* on G is a vector of probabilities $P = (p_1, p_2, \ldots p_n)$ such that $\sum p_i = 1$. The distribution P is *uniform* if $p_i = \frac{1}{n}$ for all i.
4. A *Markov process* on G is a matrix of probabilities $Q = \{q_{i,j}\}_{1\leq i,j\leq n}$ such that $\sum_{j=1}^{n} q_{i,j} = 1$ for all i, and such that $q_{i,j} = 0$ if $(v_i, v_j) \notin E$.
5. The *principle eigenvalue* of any Markov process is 1.
6. A *stationary distribution* of a Markov process Q is any distribution W which is also a principle eigenvector of the process, namely $QW = W$.

7. If G has no node of out-degree zero, then the *fair Markov process* on G is the process $Q = \left\{ \frac{m_{i,j}}{out\_deg(x_i)} \right\}$ where $m_{i,j}$ is the number of edges from v_i to v_j

8. A *fair distribution* of G is a distribution $W = (w_1, \ldots w_n)$ which is a principle eigenvector of the fair Markov process on G.

Lemma B.2 *A strongly connected directed multi-graph has a unique fair distribution.*

Lemma B.3 *If G is Eulerian, and if $W = (w_1, \ldots, w_n)$ is the fair distribution on G, then*

$$w_i = \frac{out\_deg(v_i)}{\sum_{j=1}^{n} out\_deg(v_j)}$$

for all i.

B.1 Random Walks in Eulerian Graphs

We define a *fair random walk* in a directed graph to be a random walk $v^0, v^1, \ldots$ in G such that v^{k+1} is a uniformly randomly chosen out-neighbor of v^k, for all $k \geq 0$.

From Chung [1], we have the following two lemmas.

Lemma B.4 *If G is Eulerian, each edge occurs approximately equally often in a sufficiently long fair random walk in G.*

Lemma B.5 *If G is Eulerian, the frequency of each node is approximately proportional to the degree of that node in a sufficiently long fair random walk in G.*

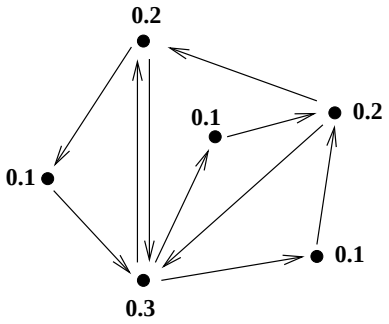

Figure B.1: Stationary Distributions on Nodes in an Eulerian Directed Graph

Figure B.1 shows a Markov process, indicated by a directed graph G. Since G is Eulerian, the stationary distribution, indicated by numbers at each node, is proportional to the degree of the node. Each edge has a flow, which is the probability that the state uses that edge during a step. Since G is Eulerian, each edge has the same flow, which is $\frac{1}{10}$ since there are ten edges.

B.2 A Non-Eulerian Example

The stationary distribution for a non-Eulerian graph can be very far from that of an Eulerian graph, even if the difference is a single edge. Consider, for example, the graph shown in Figure

B.2. It is "almost" Eulerian, meaning that the addition of just one edge, (x, y) makes it Eulerian, and yet the stationary distribution decreases exponentially as we move to the right. When we insert just the one edge (x, y), we obtain the Eulerian distribution.

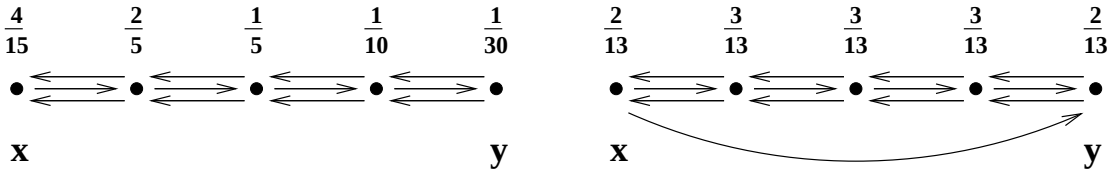

Figure B.2: Non-Eulerian and Eulerian graphs. The stationary distribution of each is shown as the numbers above the nodes.

C Poisson Distributions

A random variable X is said to have a *Poisson* distribution with mean M if X is always a non-negative integer, and if, for any non-negative integer i, the probability that $X = i$ is $\frac{\mu^i}{i!}e^{-\mu}$.

Theorem C.1 *If $X_1, \ldots X_n$ are independent and random variables, where each X_i has the value 1 with probability ε_i and the value 0 with probability $1 - \varepsilon_i$, then the distribution of $X = \sum_{i=1}^{n} X_i$ is close to a Poisson distribution with mean $\mu = \sum_{i=1}^{n} \varepsilon_i$ if $\frac{\varepsilon_i}{\mu}$ is small for each i.*