THE OHIO STATE UNIVERSITY

An Eventually Perfect Failure Detector on ADD Channels Using Clustering

Laine Rumreich and Dr. Paul Sivilotti

Overview

- Development of a cluster-based algorithm to improve complexity for an eventually perfect failure detector
  - Structured as a series of superpositioned layers
  - Simulation shows performance based on various topologies

Results

1. Development of a cluster-based algorithm
2. Complexity reduced from O(*En*log*n*) to O(*En*)
3. Simulation reveals improvement for some network topologies

Background - ◊P

- Eventually must provide *only* correct information about crashed processes
  - Strong completeness: Eventually, every crashed process is suspected (by every process)
  - Eventual strong accuracy: No correct process is suspected after some finite prefix
- Oracle is both powerful and implementable

Previous Work

- Average Delayed/Dropped (ADD) channel

An ADD channel from nodes p to q, given unknown constants K and D, satisfies the following two properties:

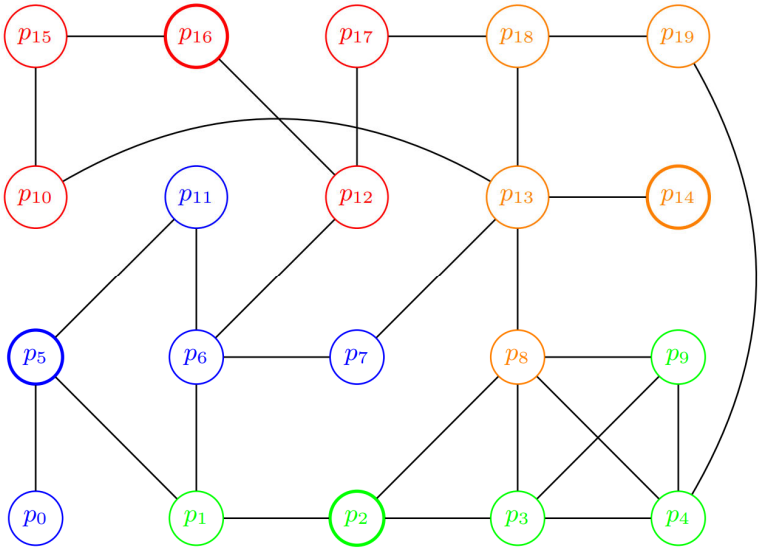
1. *The channel does not create or duplicate messages*
2. *For every K consecutive messages sent by p to q, at least one is delivered to q within D time*

Previous Work

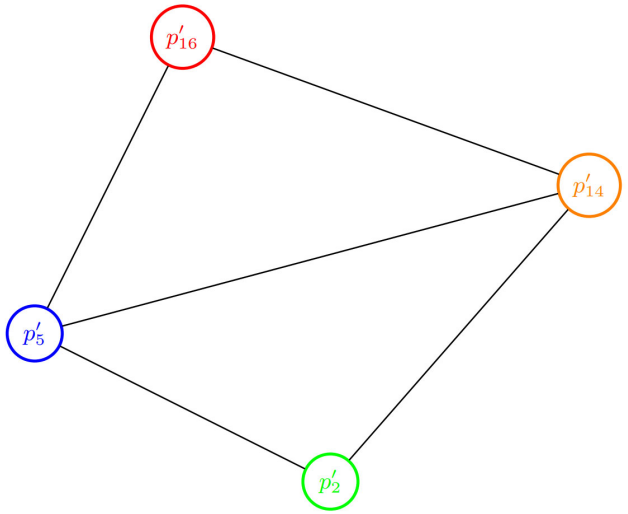
- ◊P on ADD channels for completely connected graphs [1]
- Arbitrarily connected network [2]
- Performance improvement [3] using
  - Heartbeat-based approach
  - Time-to-live values for messages

[1] Sastry, S., Pike, S.: Eventually perfect failure detectors using add channels.
[2] Kumar, S., Welch, J.: Implementing ◊P with bounded messages on a network of add channels.
[3] Vargas, K., Rajsbaum, S., Raynal, M.: An eventually perfect failure detector for networks of arbitrary topology connected with add channels using time-to-live values.

Clustering Algorithm



Network Topology with Overlay Network



Network Topology with Abstraction of Overlay Nodes

A Superpositioning Approach

Layer Name	Variables Defined	Variables Accessed
1. Heartbeat - Leader		cluster, cluster_suspect, leaders, leader_clusters
2. Forwarding Overlay		cluster, leader_clusters
3. Leader Notification	leaders, leader_clusters	cluster, leader
4. Leader Election	leader	cluster, cluster_suspect
5. Heartbeat - cluster	cluster, cluster_suspect	

A Superpositioning Approach

Layer Name	Variables Defined	Variables Accessed
1. Heartbeat - Leader		cluster, cluster_suspect, leaders, leader_clusters
2. Forwarding Overlay		cluster, leader_clusters
3. Leader Notification	leaders, leader_clusters	cluster, leader
4. Leader Election	leader	cluster, cluster_suspect
5. Heartbeat - cluster	cluster, cluster_suspect	

Algorithm 4 Leader Election Node p

Constants:

1: $T, n, neighbors$

Variables:

2: $clock() \leftarrow 1$

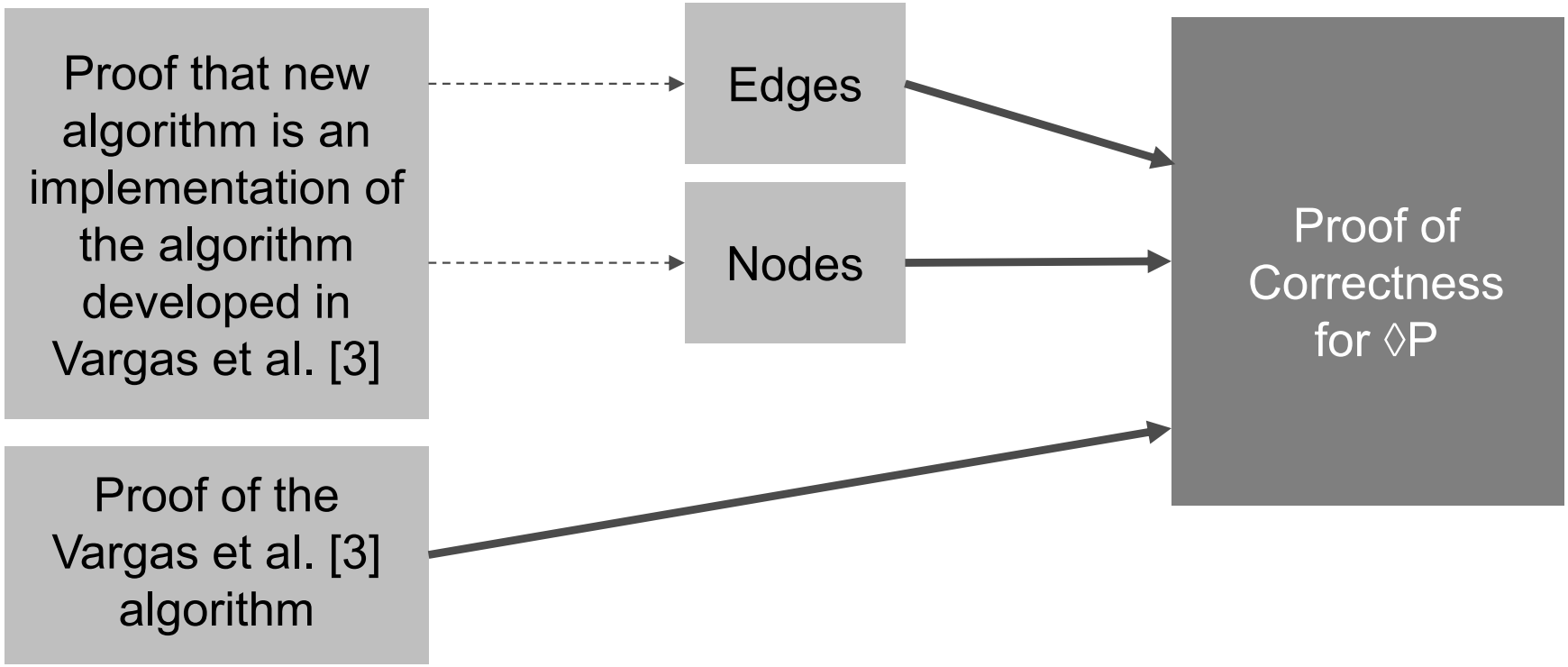
3: $leader, cluster, cluster\_suspect$

4: **if** $cluster\_suspect[leader]$ **then**

5: $\quad leader \leftarrow \Omega(cluster)$

6: **end if**

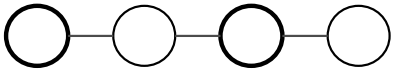
Correctness Proof



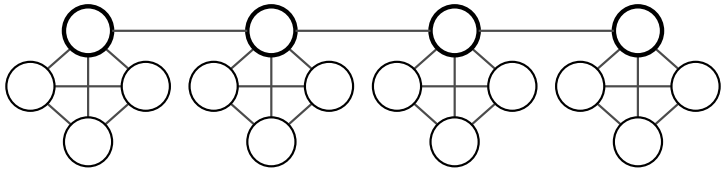
Simulation

- Python script to simulate the cluster-based algorithm and the best previous implementation
- Tests of varying topologies and group sizes on 100 nodes

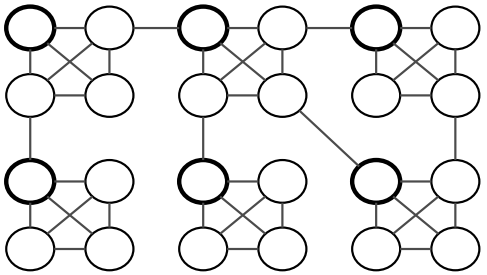
Simulation - Topologies



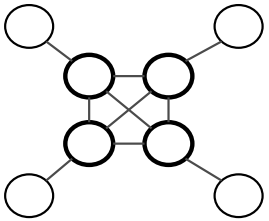
Chain



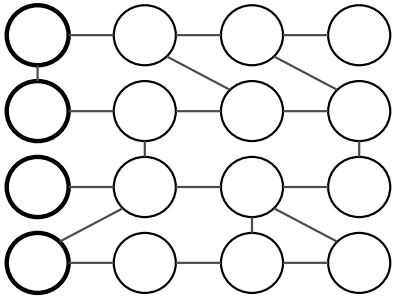
Chain with Connected Clusters



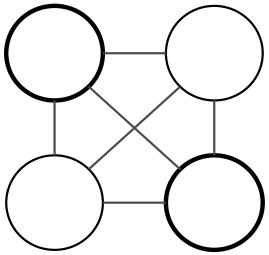
Connected Clusters



Connected Leaders



Average Connectedness



Fully Connected

Simulation – Convergence Results

Topology	Original Algorithm 1-1/M-1/M-M	Cluster Algorithm 1-1/M-1/M-M
Chain	223/223/295	38/38/44
Chain with Connected Clusters	1/4/4	3/8/8
Fully Connected Clusters	4/4/4	6/6/6
Fully Connected Leaders	1/4/4	3/5/44
Average Connectedness	1/4/4	3/5/44
Fully Connected	1/1/3	3/5/5

Conclusions

- Reduction in the message size complexity of the best previous implementation of ◊P
  - O(*En*log*n*) down to O(*En*)
- Simulation demonstrates a similar time to convergence for the cluster-based algorithm compared to the previous implementation