# Always Acyclic Distributed Path Computation

Authors

Saikat Ray, Roch Guérin*, Fellow, IEEE*, Kin-Wah Kwong, and Rute Sofia

Presented By

Abu Hena Al Muktadir

Student ID: 0940007

# Presentation Outline

- Introduction
- Link-state vs. Distance-vector Algorithm
- Transient Loop Problem
- Goal/Objective of this paper
- Previous Studies
- Proposed Algorithm (DIV)
- Simulation Results

# Introduction

- Depending on the mode of information dissemination and subsequent computation using the disseminated information, there are two broad classes of routing algorithms:

1. link-state algorithms (also known as topology broadcast) and
2. distance-vector algorithms

- In both the approach each node selects successor (next-hop) node based on local information.
- The minimum cost path to a destination is formed by concatenating computational results at individual nodes requiring *consistency* across nodes both in computation and in the information.

# Introduction

- Inconsistent information at different nodes can have dire consequences on the possible formation of transient routing loops.
- Routing loops can severely impact network performance, especially in networks with no or limited loop mitigation mechanisms.
- It can cause network wide congestion (with broadcast packet more severe), delaying of control packets terminating the loop.
- Avoiding transient routing loops remains a key requirement for path computation in both existing and emerging network technologies.

# Link-state Algorithm

- Advantages:
- ✓ Exchange state information with reliable flooding.
- ✓ each node independently computes a path to every destination.
- ✓ potential information inconsistency duration across nodes is small.
- Disadvantages:
- ✓ quite high overhead in terms of communication (broadcasting updates),
- ✓ High storage (maintaining a full network map),
- ✓ Large computation (a change anywhere in the network triggers computations at all nodes).
- These are some of the reasons for investigating alternatives as embodied in distance-vector algorithms, which are the focus of this paper.

# Distance-vector Algorithm

- Advantages:

✓ Distance-vector algorithms couple information dissemination and computation.

✓ This can reduce storage requirements (only routing information is stored), communication overhead (no relaying of flooded packets), and computations (a local change needs not propagate beyond the affected neighborhood).

✓ Thus distance-vector algorithms avoid several of the disadvantages of link-state algorithms, which can make them attractive, especially in situations of frequent local topology changes and/or when high control overhead is undesirable.

- Disadvantages:

✓ Dependency on neighboring node's information for path computation extend the period of inconsistent information across nodes.

✓ High network converge time.

✓ Destination unreachable can lead counting-to-infinity problem.

# Objective

- In spite of the benefits of distance-vector based solutions, even in environments where they might be a natural fit, calls for developing approaches to overcome these problems.
- This paper introduces the *distributed path computation with intermediate variables* (DIV) algorithm that enables distributed, light-weight, loop-free path computation.

# How a routing loop can form

For example, in the network given below, **node A** is transmitting data to node **C** via node **B**. If the link between nodes **B** and **C** goes down and **B** has not yet informed node **A** about the breakage, node **A** transmits the data to node **B** assuming that the link **A-B-C** is operational and of lowest cost. Node **B** knows of the broken link and tries to reach node **C** via node **A**, thus sending the original data back to node **A**. Furthermore, node **A** receives the data that it originated back from node **B** and consults its routing table. Node **A**'s routing table will say that it can reach node **C** via node **B** (because it still has not been informed of the break) thus sending its data back to node **B** creating an infinite loop.

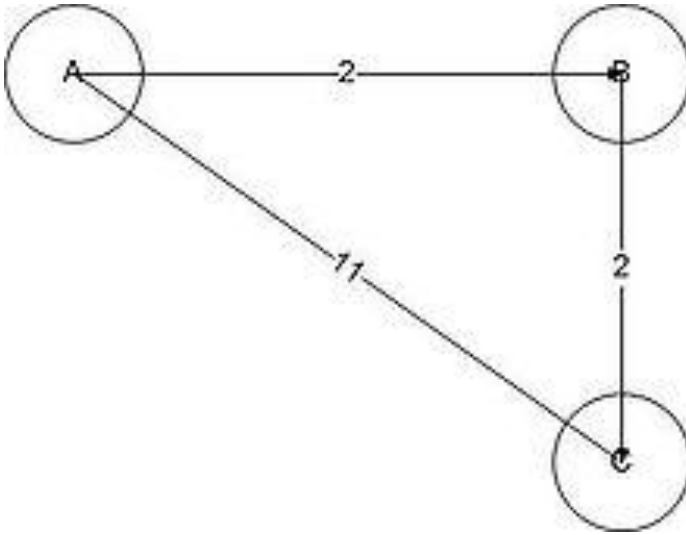
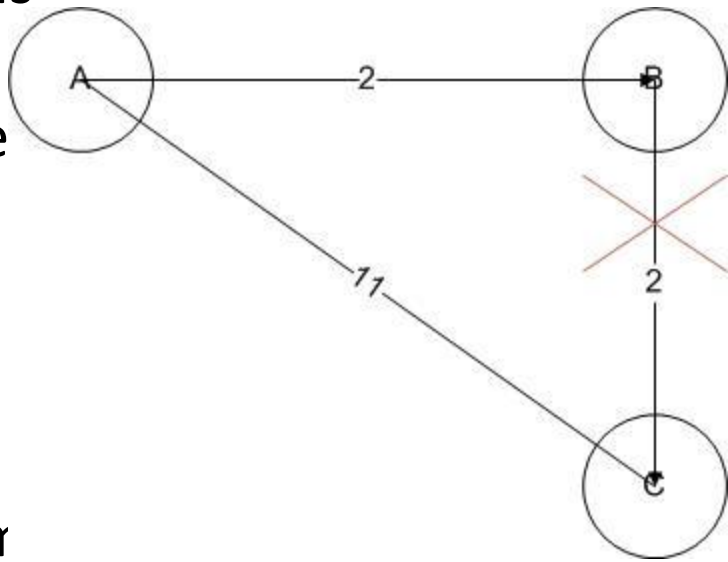



Figure 1

# How a routing loop can persist

- Consider now what happens if both the link from A to C and the link from B to C vanish at the same time (this can happen if node C has crashed). A believes that C is still reachable through B, and B believes that C is reachable through A. In a simple reachability protocol, such as EGP, the routing loop will persist forever.
- In a naïve distance vector protocol, such as RIP, the loop will persist until the metrics for C reach *infinity*.(the maximum no. of routers that a packet can traverse in RIP is 15. 16 is considered infinity and the packet is discarded)



# Previous Works

# *The Common Structure*

- Most previous distance-vector algorithms free from transient loops follow a common structure:
- Nodes exchange update-messages to notify their neighbors of any change in their own cost-to-destination (for any destination).
- If the cost-to-destination decreases at a node, the algorithms allow updating its neighbors in an arbitrary manner; these updates are called *local* (asynchronous) updates.
- However, after an increase in the cost-to-destination of a node, these algorithms require that the node potentially update all its upstream nodes *before* changing its current successor; these are *synchronous* updates.
- Algorithms differ in handling situations where during the propagation of a node's cost-to-destination update to its upstream nodes, its cost-to-destination changes.

# *Diffusing Update Algorithm (DUAL)*

- In DUAL, each node maintains, for each destination, a set of neighbors called the *feasible successor set*.
- An example of the feasible successor set to be the set of all neighbors whose current cost-to-destination is *less* than the minimum cost-to-destination seen so far by the node.
- A node can choose any neighbor in the feasible successor set to be the successor (next-hop) without causing a routing loop.
- If the neighbor through which the cost-to-destination of the node is minimum is in the feasible successor set, then that neighbor is chosen as the successor.
- If the current feasible successor set does not include the best successor, the node initiates a synchronous update procedure, known as a *diffusing computation.*

# *Diffusing Update Algorithm (DUAL)*

- The node sends queries to all its neighbors with its cost-to-destination through the current successor.
- From this point onwards the node does not change its successor until the diffusing computation terminates.
- Each neighbor replies to the query by sending their own cost-to-destination if they themselves have a feasible successor after they update the set following the new information received from the initiator node.
- Otherwise, they themselves send out queries and wait for the replies before replying to the original query.
- Finally if there are multiple overlapping updates—i.e., if a new link-cost change occurs when a node is waiting for replies to a previous query—the node uses a *finite state machine* to process these multiple updates sequentially.

# *Loop Free Invariance (LFI) Algorithms*

- A pair of invariances, based on the cost-to-destination of a node and its neighbors, called *Loop Free Invariances* (LFI).
- If nodes maintain these invariances, then no transient loops can form.
- Update mechanisms are required to maintain the LFI conditions.
- *Multiple-path Partial-topology Dissemination Algorithm* (MPDA) uses a link-state type approach with LFI and *Multipath Distance Vector Algorithm* (MDVA) that uses a distance vector type approach.

# Overview of DIV

- DIV combines advantages of both DUAL and LFI.
- DIV lays down a set of rules on existing routing algorithms to ensure their loop-free operation at each instant.
- This rule-set is not predicated on shortest path computation, so DIV can be used with other path computation algorithms as well.

# Node Value Assignment

- For each destination, DIV assigns a *value* to each node in the network.
- Some typical value assignments are as follows: (i) cost-to-destination, (ii) the minimum cost-to-destination seen by the node from timet=0, or (iii) the number of next-hop neighbors for the destination.
- One restriction is imposed on the value assignment: a node that does not have a path to a destination must assign a value of "infinity" (the maximum possible value) to itself.
- Intuitively, this restriction prevents other nodes from using it as a successor.
- This restriction turns out to be crucial for avoiding counting-to-infinity problems in shortest path environments.

# Selecting successor from neighbor

- DIV allows a node to choose one of its neighbors as a successor only if the value of that neighbor is less than its own value.
- this is called the *decreasing value property* of DIV. This ensures that no routing loop can ever form.
- The hard part of DIV is enforcing the decreasing value property.

# Update Messages

- Each node must update its neighbors about its own current value by means of update messages.
- In DUAL these update messages are asynchronous, information at various nodes may be inconsistent and may lead to the formation of loops.
- DIV lays down specific update rules that guarantee loop freedom.
- DIV accomplishes this task by maintaining several intermediate variables that hold a replica of the value of a node at its neighbors and vice versa, and exchanging messages between neighboring nodes.
- the update mechanism sends update messages and for some of them, requires an acknowledgment from the neighbor.

# ACK Messages

- Depending on the rules for sending acknowledgments, DIV can be operated in one of the following two modes: (i) the *normal mode*, and (ii) the *alternate mode*.
- In the normal mode, a neighbor can hold on to sending an acknowledgment until its own value is adjusted appropriately.
- In the alternate mode, on the other hand, the neighbor immediately sends the acknowledgment, but could temporarily lose all paths (to that particular destination).

# Description of DIV

- There are four aspects to DIV:

(i) the variables stored at the nodes;

(ii) two ordering invariances that each node maintains;

(iii) the rules for updating the variables; and (iv) two semantics for handling nonideal message deliveries (such as packet loss or reordering).

# *The Intermediate Variables*

- Suppose that a node x is a neighbor of node y.
- There are three aspects to each of these variables: whose value is this? who believes in that value? and where is it stored?
- Accordingly, we define V(x;y|x) to be the value of node x as known (believed) by node y stored in node x; similarly V(y;x|x) denotes value of node y as known by node x stored in node x.

# *The Intermediate Variables*

Thus, assuming node $x$ has $n$ neighbors, $\{y_1, y_2, \ldots, y_n\}$, it stores, for each destination:

1) its own value, $V(x; x|x)$;
2) the values of its neighbors as known to itself, $V(y_i; x|x)$ $[y_i \in \{y_1, y_2, \ldots, y_n\}]$;
3) and the value of itself as known to its neighbors $V(x; y_i|x)$ $[y_i \in \{y_1, y_2, \ldots, y_n\}]$.

That is, $2n + 1$ values for each destination. This is $O(N \cdot d)$ storage complexity in a network with $N$ destination nodes and average node degree $d$. The variables $V(y_i; x|x)$ and $V(x; y_i|x)$ are called intermediate variables since they endeavor to reflect the values $V(y_i; y_i|y_i)$ and $V(x; x|x)$, respectively. In steady state, DIV ensures that $V(x; x|x) = V(x; y_i|x) = V(x; y_i|y_i)$.

# *The Invariances*

- DIV requires each node to maintain at all times the following two invariances based on its set of *locally stored variables*.

*Invariance 1: The value of a node is not allowed to be more than the value the node thinks is known to its neighbors. That is*

$$V(x; x|x) \leq V(x; y_i|x) \quad \text{for each neighbor } y_i. \qquad (1)$$

*Invariance 2: A node can choose one of its neighbors as a successor only if the value of is less than the value of as known by node ; i.e., if node is the successor of node , then*

$$V(x; x|x) > V(y; x|x). \qquad (2)$$

Thus, due to Invariance 2, a node $x$ can choose a successor only from its *feasible successor set* $\{y_i|V(x; x|x) > V(y_i; x|x)\}$. The two invariances reduce to the LFI conditions if the value of a node is chosen to be its current cost-to-destination.

# Message Update Rules

- There are two operations that a node needs to perform in response to network changes: (i) decreasing its value and (ii) increasing its value.
- DIV uses two corresponding update messages, Update::Dec and Update::Inc, and acknowledgment (ACK) messages in response to Update::Inc (no ACKs are needed for Update::Dec).

# *Decreasing Value*

*Decreasing Value:* Decreasing value is the simpler operation among the two. The following rules are used to decrease the value of a node $x$ to a new value $V_0$:

- Node $x$ first simultaneously decreases the variables $V(x; x|x)$ and the values $V(x; y_i|x)\ \forall i = 1, 2, \ldots, n$, to $V_0$;
- Node $x$ then sends an Update::Dec message to all its neighbors that contains the new value $V_0$.
- Each neighbor $y_i$ of $x$ that receives an Update::Dec message containing $V_0$ as the new value updates $V(x; y_i|y_i)$ to $V_0$.

# Increasing Value

*Increasing Value:* Increasing value is potentially a more complex operation, however, conceptually it is simply an inverse operation: in the decrease operation a node first decreases its value and then notifies its neighbors; in the increase operation, a node first notifies its neighbors (and waits for their acknowledgments) and then increases its value. In particular, a node $x$ uses the following rules to increase its value to $V_1$:

- Node $x$ first sends an Update::Inc message to all its neighbors.
- Each neighbor $y_i$ of $x$ that receives an Update::Inc message sends an acknowledgment message (ACK) when it is able to do so according to the rules explained in details below (Section III-B5C). When $y_i$ is ready to send the ACK, it first modifies $V(x; y_i|y_i)$, changes successor if necessary (since the feasible successor set may change), and then sends the ACK to $x$; the ACK contains the sequence number of the corresponding Update::Inc message and the new value of $V(x; y_i|y_i)$. Note that in this case it is essential that node $y_i$ changes successor, if necessary, *before* sending the ACK.
- When node $x$ receives an ACK from its neighbor $y_i$, it modifies $V(x; y_i|x)$ to $V_1$. At any time, node $x$ can choose any value $V(x; x|x) \leq V(x; y_i|x), \forall i = 1, 2, \ldots, n$.

# Sending ACKs

*Rules for Sending Acknowledgment: The Two Modes:* We now describe how a node decides whether it can send an ACK in response to an Update::Inc message. There are two possibilities: each possibility leads to a distinct behavior of the algorithm, which we refer to as modes.

Suppose that node $y_i$ received an Update::Inc message from node $x$. Recall that node $y_i$ must increase $V(x; y_i|y_i)$ before sending an ACK. However, increasing $V(x; y_i|y_i)$ may remove node $x$ from the feasible successor set at node $y_i$. If node $x$ is the only preferred node in the feasible successor set of node $y_i$, then node $y_i$ may lose its path if $V(x; y_i\ y_i)$ is increased without first increasing $V(y_i; y_i|y_i)$. In such a case node $y_i$ has two options: 1) first increase $V(y_i; y_i|y_i)$ and then increase $V(x; y_i|y_i)$ and send the ACK to node $x$; or 2) increase $V(x; y_i|y_i)$, send ACK to node $x$ and then increase $V(y_i; y_i|y_i)$. If a node uses option 1), we say that DIV is operating in its *normal mode*; if a node uses option 2), we say that DIV is operating in *alternate mode*.

# *Semantics for Handling Message Reordering*

*Semantic 1: A node ignores an update message that comes out-of-order ( i.e., after a message that was sent later).*

*Semantic 2: A node ignores outstanding ACKs after issuing an Update::Dec message.*

- These semantics are enforced using the embedded sequence numbers in the update messages.

# PERFORMANCE EVALUATION

In this section, we consider three shortest paths algorithms, DBF, DUAL (using SNC as its feasibility condition), and DIV (using DBF to compute value updates), and compare their performance in terms of loop avoidance[9] and convergence time. The simulations are performed on random graphs with fixed average degree of 5, but in order to generate a reasonable range of configurations, a number of other parameters are varied. Networks with sizes ranging from 10 to 90 nodes are explored in increments of 10 nodes. For each network-size, 100 random graphs are generated. Link costs are drawn from a bimodal distribution: with probability 0.5 a link cost is uniformly distributed in [0,1]; and with probability 0.5 it is uniformly distributed in [0,100]. For each graph, 100 random link-cost changes are introduced, again drawn from the same bimodal distribution. All three algorithms are run on the same graphs and sequences of changes. Processing time of each message is random: it is 2 s with probability 0.0001, 200 ms with probability 0.05, and 10 ms otherwise.

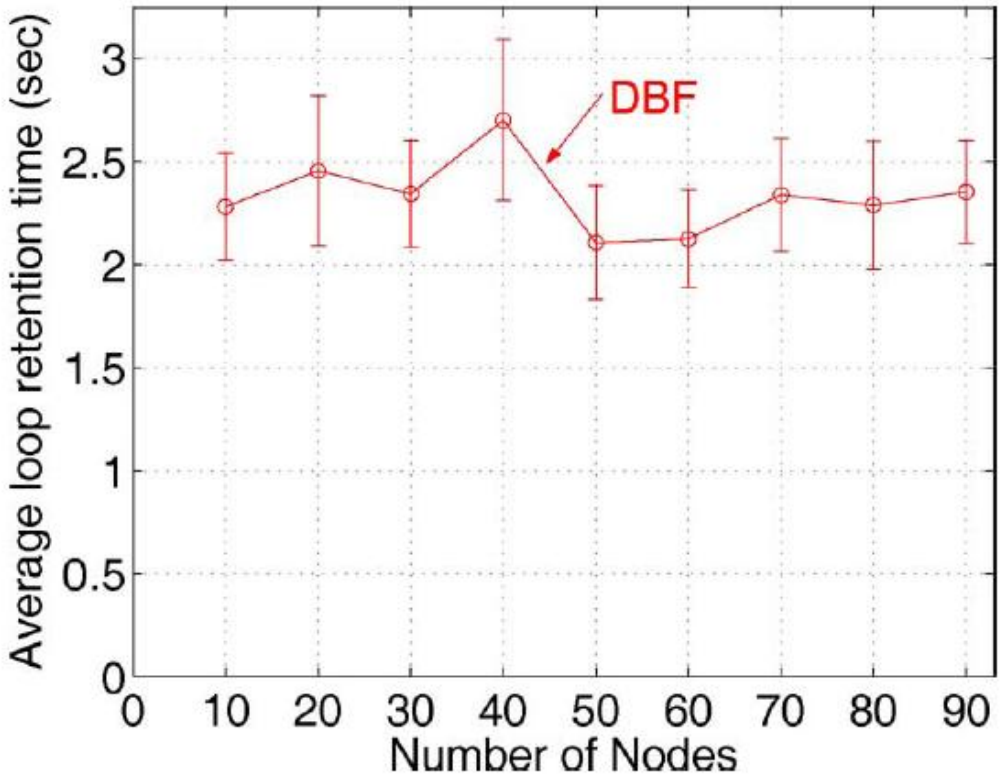



Fig. 6. Mean loop-retention time. No loops are found with DUAL or DIV.

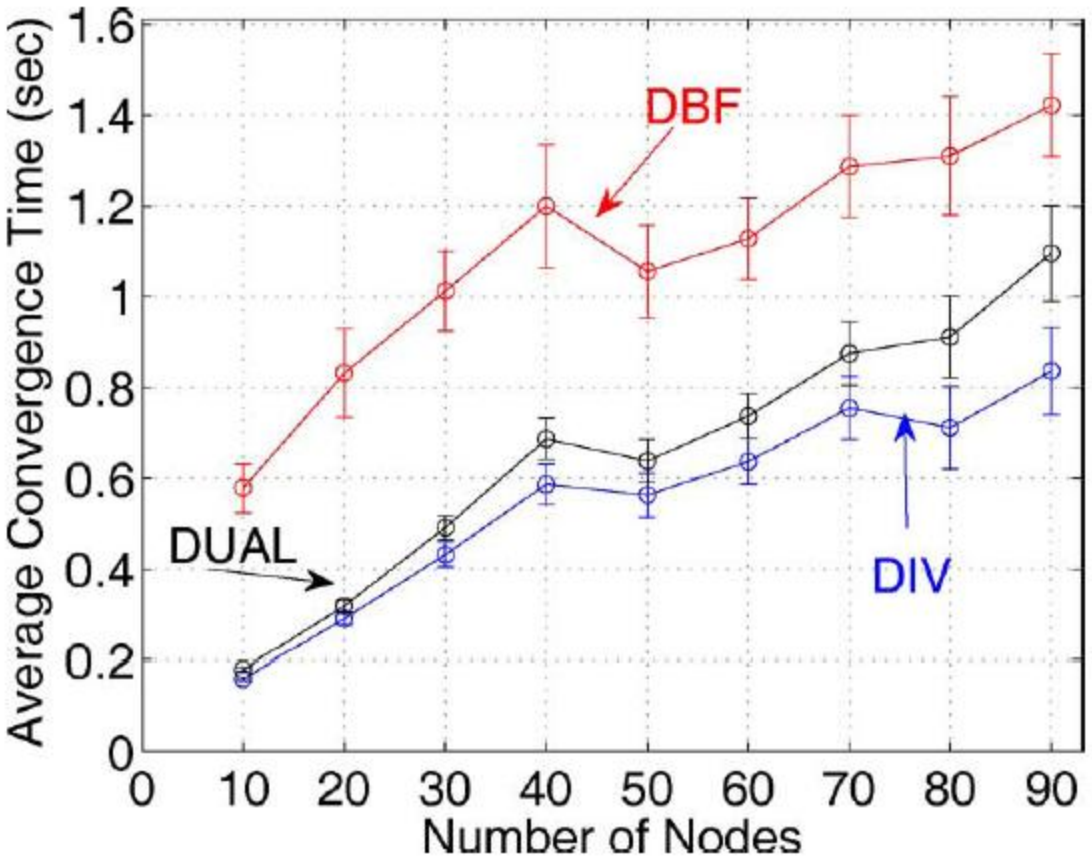



Fig. 7. Mean convergence time.

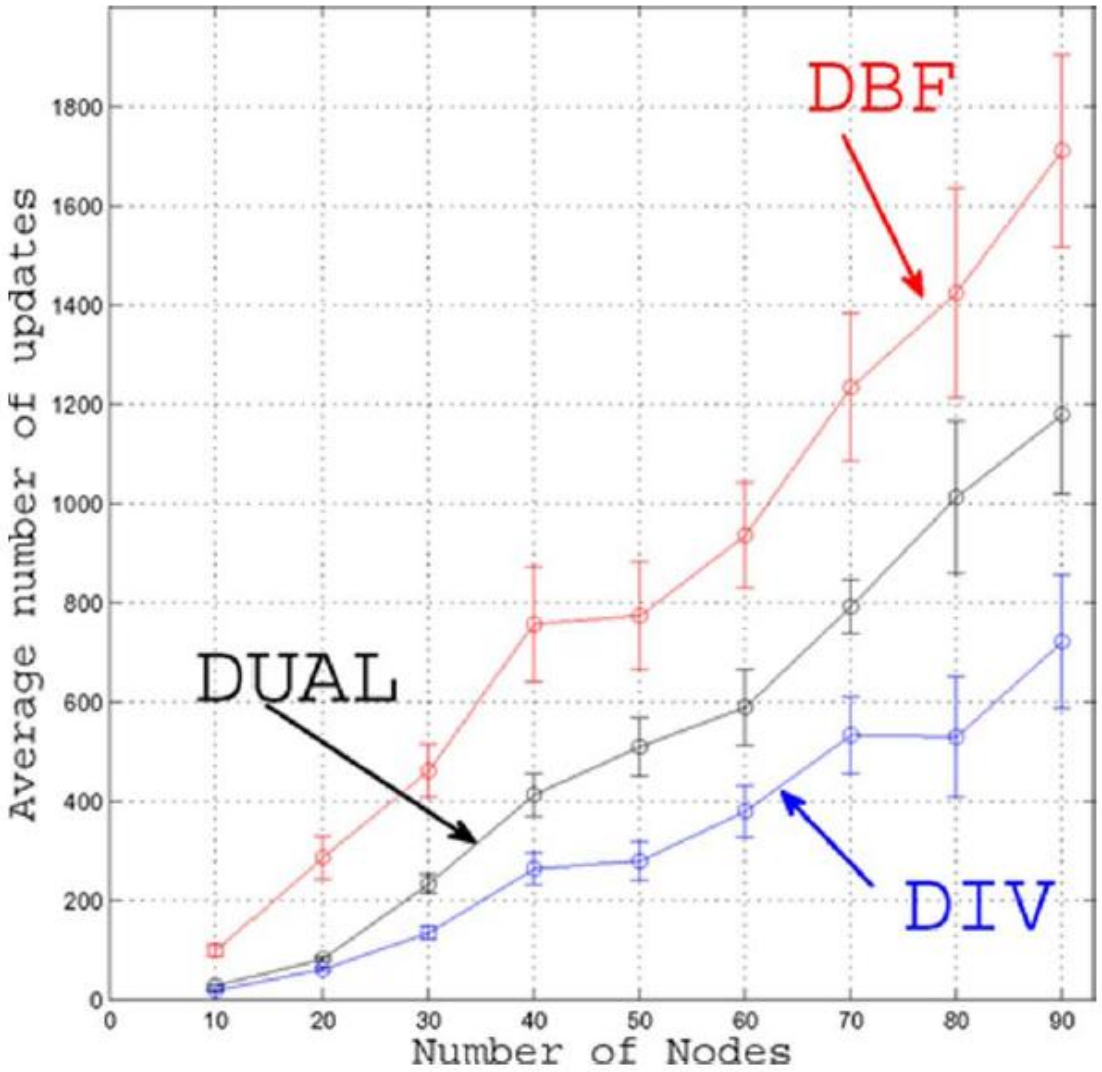



Fig. 8. Number of update messages required for convergence.

# Questions?

# THANK YOU VERY MUCH