Engineering a new loop-free shortest paths routing algorithm

Gianlorenzo D'Angelo **Mattia D'Emidio** Daniele Frigioni
Vinicio Maurizio

Department of Electrical and Information Engineering
University of L'Aquila, Italy
–
WEB: informatica.ing.univaq.it/demidio
MAIL: mattia.demidio@univaq.it

SEA 2012

Outline

1. Motivation
2. The new algorithm
3. Experimental evaluation
4. Conclusion and future research

Outline

1. Motivation
2. The new algorithm
3. Experimental evaluation
4. Conclusion and future research

The problem

Distributed minimum cost paths computation

- Crucial problem in today's practical applications
- Most important approach to solve the problem: **distributed all-pairs shortest paths problem**

Dynamic version

- Highly relevant in practice
- Widely studied in the literature, many solutions have been developed over the years

The problem

Distributed minimum cost paths computation

- Crucial problem in today's practical applications
- Most important approach to solve the problem: **distributed all-pairs shortest paths problem**

Dynamic version

- Highly relevant in practice
- Widely studied in the literature, many solutions have been developed over the years

Known solutions

Link-State

- Each node stores the whole graph topology and computes locally the SPs
- Each topology change needs to be notified to all the nodes
- High space/computational complexity
- OSPF protocol used in the Internet

Distance-Vector

- Each node computes the SPs by interacting with neighbors
- Broadcast a topology change only to affected nodes
- High message complexity but store very little information
- Classical Bellman-Ford method used in RIP protocol

Known solutions

Link-State

- Each node stores the whole graph topology and computes locally the SPs
- Each topology change needs to be notified to all the nodes
- High space/computational complexity
- OSPF protocol used in the Internet

Distance-Vector

- Each node computes the SPs by interacting with neighbors
- Broadcast a topology change only to affected nodes
- High message complexity but store very little information
- Classical Bellman-Ford method used in RIP protocol

Recent developments

Recently

- Renewed interest in devising new **efficient light-weight** distributed SP solutions for **Large Scale Ethernet Networks**
- Focus on Distance-vector algorithms as an attractive alternative to link-state solutions when:
  - **scalability** is mandatory
  - routing devices have **limited** storage/computation capabilities

Recent developments

Recently

- Renewed interest in devising new **efficient light-weight** distributed SP solutions for **Large Scale Ethernet Networks**
- Focus on Distance-vector algorithms as an attractive alternative to link-state solutions when:
  - **scalability** is mandatory
  - routing devices have **limited** storage/computation capabilities

State-of-the-art Distance-Vector algorithm

DUAL

- Part of CISCOs widely used EIGRP protocol
- Free of looping phenomena + fully dynamic
- Uses **SNC** (Source Node Condition) and Diffuse-Computation
- Good message complexity but high space complexity
- Requires $O(\Phi \cdot \Delta)$ messages and $\Theta(n \cdot \Delta)$ space where:
  - Φ: total number of nodes *affected* by a set of updates
  - Δ: maximum node degree

Our motivation

Develop a loop-free algorithm with a smaller space complexity than **DUAL**, able to send less messages at least in practically interesting scenarios

State-of-the-art Distance-Vector algorithm

DUAL

- Part of CISCOs widely used EIGRP protocol
- Free of looping phenomena + fully dynamic
- Uses **SNC** (Source Node Condition) and Diffuse-Computation
- Good message complexity but high space complexity
- Requires $O(\Phi \cdot \Delta)$ messages and $\Theta(n \cdot \Delta)$ space where:
  - Φ: total number of nodes *affected* by a set of updates
  - Δ: maximum node degree

Our motivation

Develop a loop-free algorithm with a smaller space complexity than **DUAL**, able to send less messages at least in practically interesting scenarios

Outline

1 Motivation

2 The new algorithm

3 Experimental evaluation

4 Conclusion and future research

Our contribution

Loop Free Routing

New distributed shortest paths routing algorithm which:

- is loop-free and fully dynamic
- combines **SNC** with a two phases distributed computation
- has the same message complexity of **DUAL**
- requires less memory: $O(n + \phi \cdot \Delta)$, where ϕ is the maximum number of destinations for which a node is *affected*

Basic Idea

Nodes:

- do not permanently store neighborhood data
- interact with neighbors when such information are needed

Our contribution

Loop Free Routing

New distributed shortest paths routing algorithm which:

- is loop-free and fully dynamic
- combines **SNC** with a two phases distributed computation
- has the same message complexity of **DUAL**
- requires less memory: $O(n + \phi \cdot \Delta)$, where ϕ is the maximum number of destinations for which a node is *affected*

Basic Idea

Nodes:

- do not permanently store neighborhood data
- interact with neighbors when such information are needed

What **SNC** (**S**ource **N**ode **C**ondition) is?

Sufficient condition for loop-freedom

- When a node has to choose a new via, if **SNC** is used, the graph induced by the vias is free of loops
- For a generic node v, a neighbor $u \in N(v)$ satisfies **SNC**, at a certain time t, if and only if
  - $u = \arg\min_{k \in N(v)} \{\mathtt{D}_k[s](t) + w^t(v,k)\}$
  - $\mathtt{D}_u[s](t) < \mathtt{D}_v[s](t)$

If no neighbors that satisfy **SNC** exist

$\Rightarrow$ Other methods to compute loop-free paths are required

- **DUAL** – Diffuse-Computation + Finite State Machine
- **LFR** – Local-Computation + Global-Computation + Queue

What **SNC** (**S**ource **N**ode **C**ondition) is?

Sufficient condition for loop-freedom

- When a node has to choose a new via, if **SNC** is used, the graph induced by the vias is free of loops
- For a generic node v, a neighbor $u \in N(v)$ satisfies **SNC**, at a certain time t, if and only if
  - $u = \arg\min_{k \in N(v)} \{\mathtt{D}_k[s](t) + w^t(v,k)\}$
  - $\mathtt{D}_u[s](t) < \mathtt{D}_v[s](t)$

If no neighbors that satisfy **SNC** exist

$\Rightarrow$ Other methods to compute loop-free paths are required

- **DUAL** – Diffuse-Computation + Finite State Machine
- **LFR** – Local-Computation + Global-Computation + Queue

LFR Data structures

Each node v stores, for each destination $s \in V$:

- $\mathtt{D}_v[s]$ and $\mathtt{VIA}_v[s]$: estimated distance and via
- $\mathtt{STATE}_v[s]$: the state of node v wrt s
  - $\mathtt{STATE}_v[s] = \mathit{true} \Rightarrow v$ is processing a weight increase operation wrt s
- $\mathtt{UD}_v[s]$: the upper bound distance to s, that is the distance to s through $\mathtt{VIA}_v[s]$
  - $\mathtt{UD}_v[s]$ is always greater or equal than $\mathtt{D}_v[s]$; if v is passive, they coincide

Plus, a temporary data structure, to implement **SNC**

- $\mathtt{tempD}_v$: allocated for a certain s only when v is active wrt s, deallocated when v turns passive wrt s
  - entry $\mathtt{tempD}_v[u][s]$ contains $\mathtt{UD}_u[s]$, for each $u \in N(v)$

LFR Data structures

Each node v stores, for each destination $s \in V$:

- $\mathtt{D}_v[s]$ and $\mathtt{VIA}_v[s]$: estimated distance and via
- $\mathtt{STATE}_v[s]$: the state of node v wrt s
  - $\mathtt{STATE}_v[s] = \mathit{true} \Rightarrow v$ is processing a weight increase operation wrt s
- $\mathtt{UD}_v[s]$: the upper bound distance to s, that is the distance to s through $\mathtt{VIA}_v[s]$
  - $\mathtt{UD}_v[s]$ is always greater or equal than $\mathtt{D}_v[s]$; if v is passive, they coincide

Plus, a temporary data structure, to implement **SNC**

- $\mathtt{tempD}_v$: allocated for a certain s only when v is active wrt s, deallocated when v turns passive wrt s
  - entry $\mathtt{tempD}_v[u][s]$ contains $\mathtt{UD}_u[s]$, for each $u \in N(v)$

Messages

Three kinds of messages

- *update*: notifies that the distance to a node is changed
- *get.dist*: asks for the current `UD[]`
- *get.feasible.dist*: asks for a loop-free distance if no neighbors satisfy **SNC** (induces the target node to activate, if needed)

Behaviour

We analyze the algorithm in the two possible cases:

- the weight of an edge $\{x_i, y_i\}$ increases
- the weight of an edge $\{x_i, y_i\}$ decreases

Messages

Three kinds of messages

- *update*: notifies that the distance to a node is changed
- *get.dist*: asks for the current `UD[]`
- *get.feasible.dist*: asks for a loop-free distance if no neighbors satisfy **SNC** (induces the target node to activate, if needed)

Behaviour

We analyze the algorithm in the two possible cases:

- the weight of an edge $\{x_i, y_i\}$ increases
- the weight of an edge $\{x_i, y_i\}$ decreases

If the weight of an edge $\{x_i, y_i\}$ changes

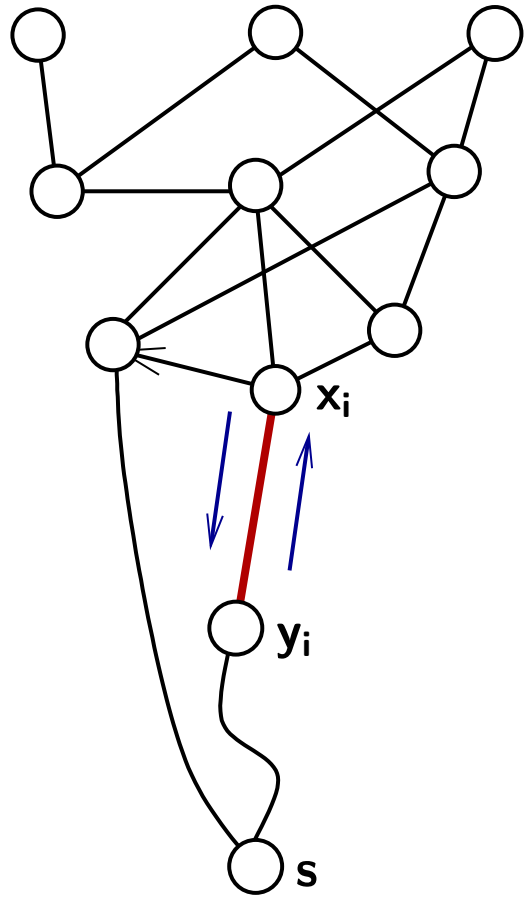
The algorithm starts

1: Node x_i sends, for each $s \in V$, a message $update(x_i, \mathtt{UD}_{x_i}[s])$ to y_i

2: Node y_i sends, for each $s \in V$, a message $update(y_i, \mathtt{UD}_{y_i}[s])$ to x_i

If the nodes are passive

- $\mathtt{UD}_{x_i}[s] \equiv \mathtt{D}_{x_i}[s]$
- $\mathtt{UD}_{y_i}[s] \equiv \mathtt{D}_{y_i}[s]$



If a node v receives an $update(u, \mathtt{UD}_u[s])$ from $u \in N(v)$

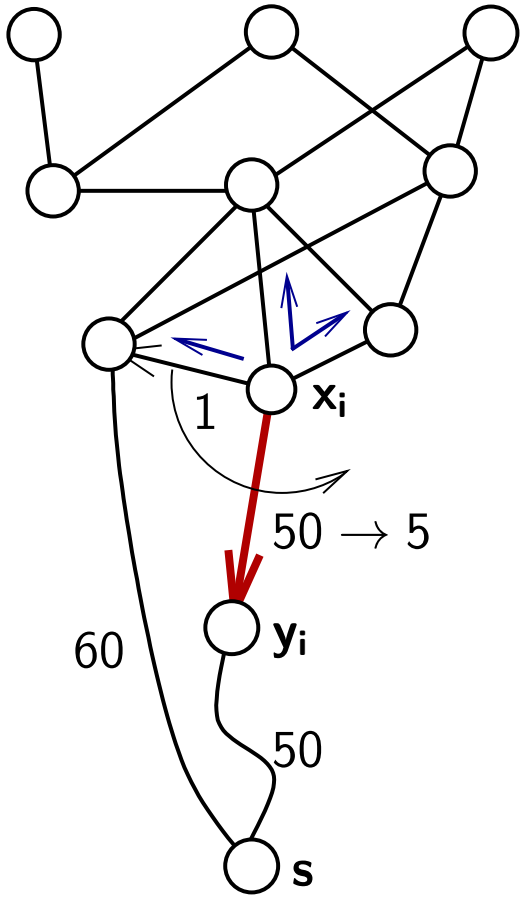
1: If $\mathtt{D}_v[s] \equiv \mathtt{UD}_u[s] + w(v,u)$: discard the message

2: If $\mathtt{D}_v[s] > \mathtt{UD}_u[s] + w(v,u)$
- 2.1: set $\mathtt{D}_v[s] = \mathtt{UD}_u[s] + w(v,u)$, $\mathtt{VIA}_v[s] = u$
- 2.2: forward the change by sending $update(v, \mathtt{UD}_v[s])$ to all the neighbors

If a node v receives an $update(u, \mathtt{UD}_u[s])$ from $u \in N(v)$

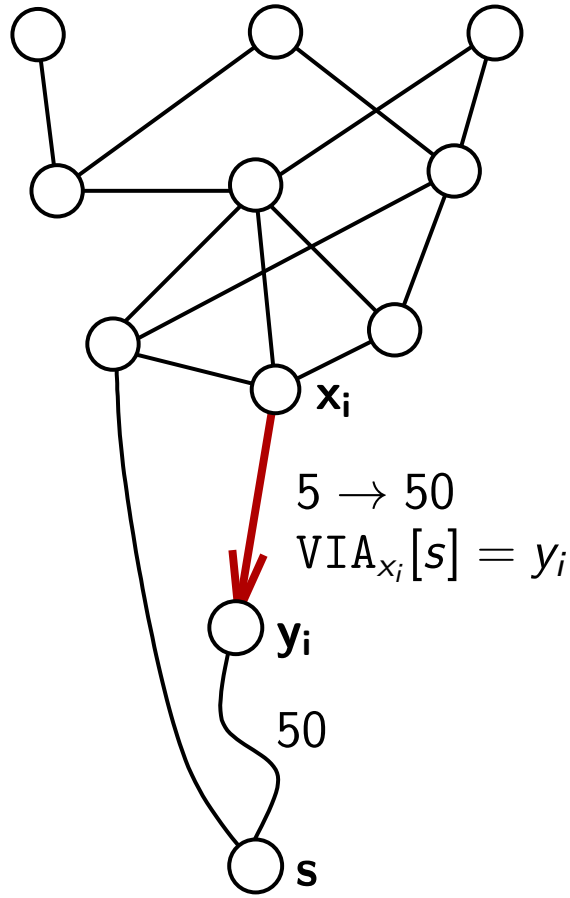
1: If $\mathtt{D}_v[s] \equiv \mathtt{UD}_u[s] + w(v,u)$: discard the message

2: If $\mathtt{D}_v[s] > \mathtt{UD}_u[s] + w(v,u)$
- 2.1: set $\mathtt{D}_v[s] = \mathtt{UD}_u[s] + w(v,u)$, $\mathtt{VIA}_v[s] = u$
- 2.2: forward the change by sending $update(v, \mathtt{UD}_v[s])$ to all the neighbors



If a node v receives an $update(u, \mathtt{UD}_u[s])$ from $u \in N(v)$

3: If $\mathtt{D}_v[s] < \mathtt{UD}_u[s] + w(v,u)$ and $\mathtt{VIA}_v[s] = u$
  3.1: Perform Local-Computation
  3.2: If Local-Computation fails, perform Global-Computation



If a generic node v performs Local-Computation wrt s

1: Node v turns active and sends *get.dist* to each $u \in N(v) : u \neq \mathtt{VIA}_v[s]$

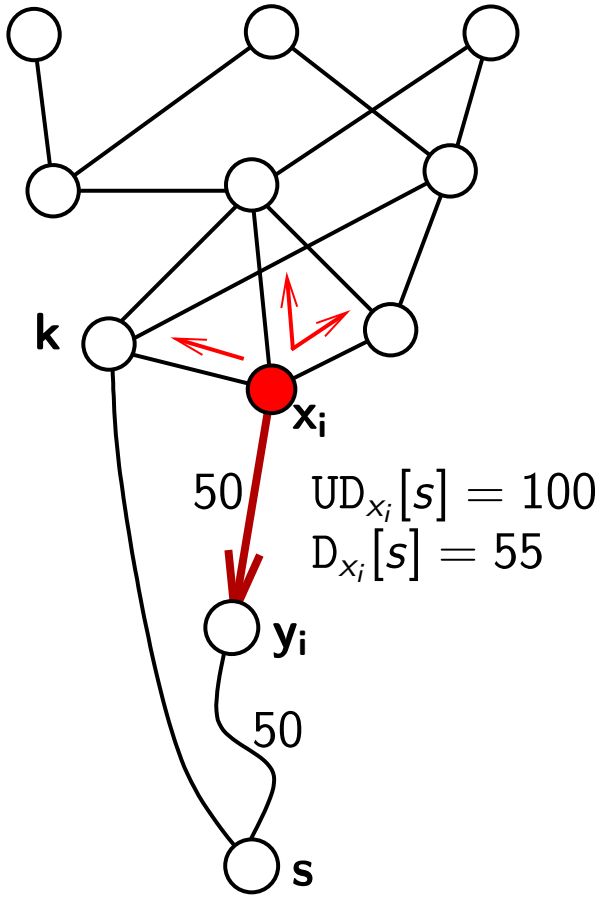
2: Each neighbor immediately replies with $\mathtt{UD}_u[s]$

3: If a neighbor k that satisfies **SNC** exists, node v:

3.1: updates its data structure

3.2: turns passive and forwards the change

4: Otherwise, v performs Global-Computation



If a generic node v performs Local-Computation wrt s

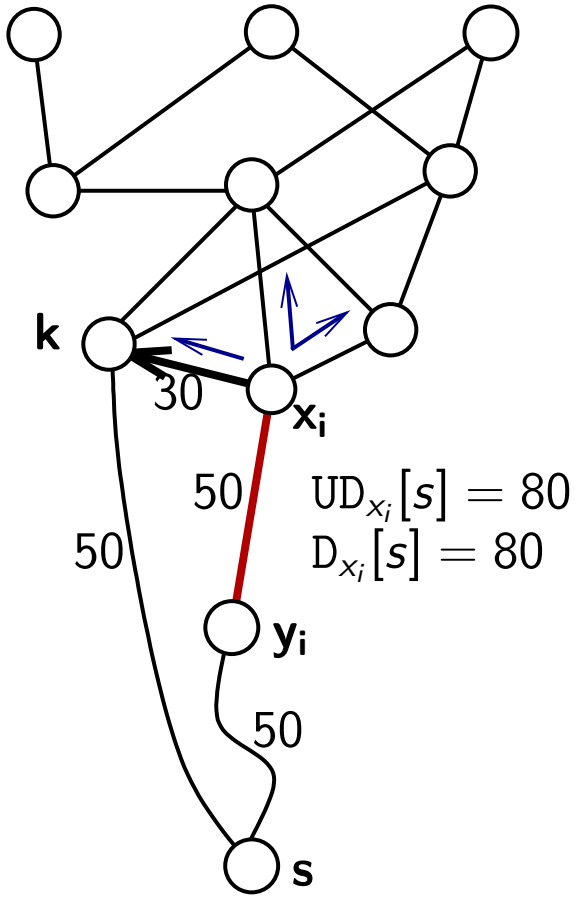
1: Node v turns active and sends *get.dist* to each $u \in N(v) : u \neq \mathtt{VIA}_v[s]$

2: Each neighbor immediately replies with $\mathtt{UD}_u[s]$

3: If a neighbor k that satisfies **SNC** exists, node v:

- 3.1: updates its data structure
- 3.2: turns passive and forwards the change

4: Otherwise, v performs Global-Computation



If a generic node v performs Local-Computation wrt s

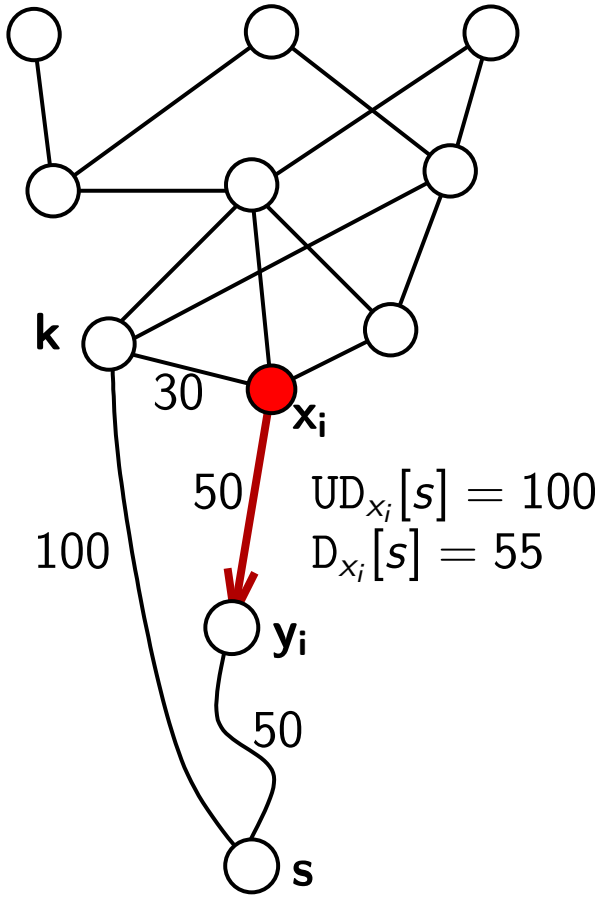
1: Node v turns active and sends *get.dist* to each $u \in N(v) : u \neq \mathtt{VIA}_v[s]$

2: Each neighbor immediately replies with $\mathtt{UD}_u[s]$

3: If a neighbor k that satisfies **SNC** exists, node v:

- 3.1: updates its data structure
- 3.2: turns passive and forwards the change

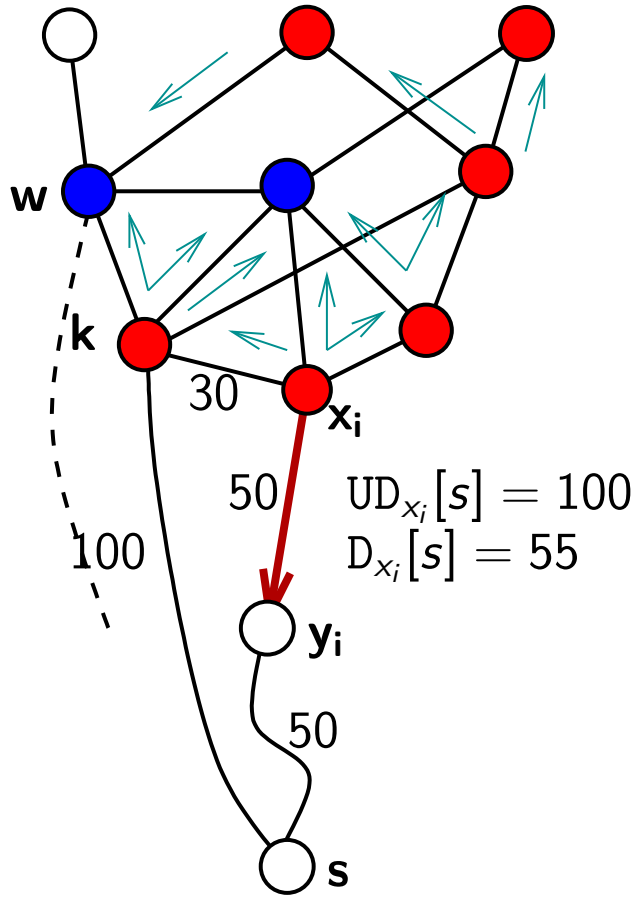
4: Otherwise, v performs Global-Computation



If a generic node v performs Global-Computation wrt s

Global-Computation

- Nodes of network communicate by *get.feasible.dist* messages in order to compute loop-free paths
- Always terminates
- When a node receives the replies to its *get.feasible.dist* messages it:
  - updates its data structures
  - turns passive
  - replies to possible pending *get.feasible.dist* and forwards the change



Handling special cases

- If multiple weight changes occur, each node running **LFR** uses a FIFO queue in order to guarantee:
  - mutual exclusion
  - properly ordered processing of the messages
- While a node is active wrt s, incoming *update* messages are enqueued
- When a node turns passive wrt s, enqueued *update* messages are dequeued in FIFO order

Outline

1 Motivation

2 The new algorithm

3 Experimental evaluation

4 Conclusion and future research

Experimental evaluation

Simulation environment

- Extensive experimental evaluation within OmNet++, a well-known distributed systems simulation enviroment
- We measured the total number of message sent and the space requirements

Implemented algorithms

- We implemented, in C++, **LFR** and **DUAL**

Experimental evaluation

Simulation environment

- Extensive experimental evaluation within OmNet++, a well-known distributed systems simulation enviroment
- We measured the total number of message sent and the space requirements

Implemented algorithms

- We implemented, in C++, **LFR** and **DUAL**

Input data

Graphs

- Real-world graphs (G_{IP}), provided by the Cooperative Association for Internet Data Analysis (CAIDA)
- Artificial random graphs (G_{ER}), generated by the Erdòs-Renyi algorithm

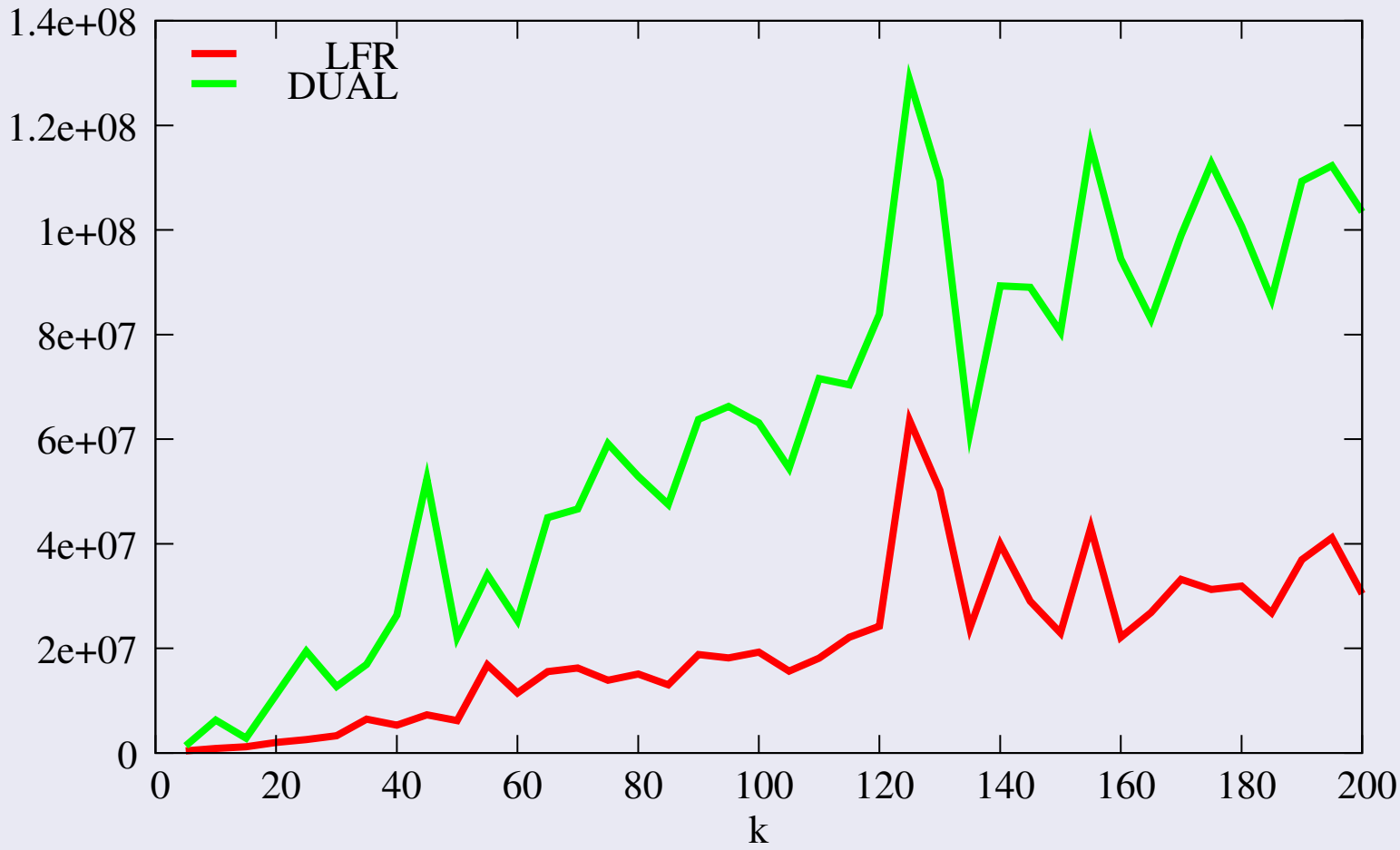
Executed tests

Executed tests

- We ran simulations in different dynamic **realistic** settings, where the weight of an edge represents the delay needed to a packet to traverse it
- Due to **DUAL**'s high memory requirements we used CAIDA and Erdòs-Renyi instances with maximum number of nodes equal to 8000 and 2000, respectively
- We show the results for:
  - a CAIDA graph with $n = 8000$, $m = 11141$ $(m/n \approx 1.4)$ and number of modifications $k \in \{5, 10, ..., 200\}$
  - artificial graphs with number of nodes $n = 2000$, *density* ranging from 0.01 to 0.61 and fixed number of modifications $k = 200$
- The other instances give similar results

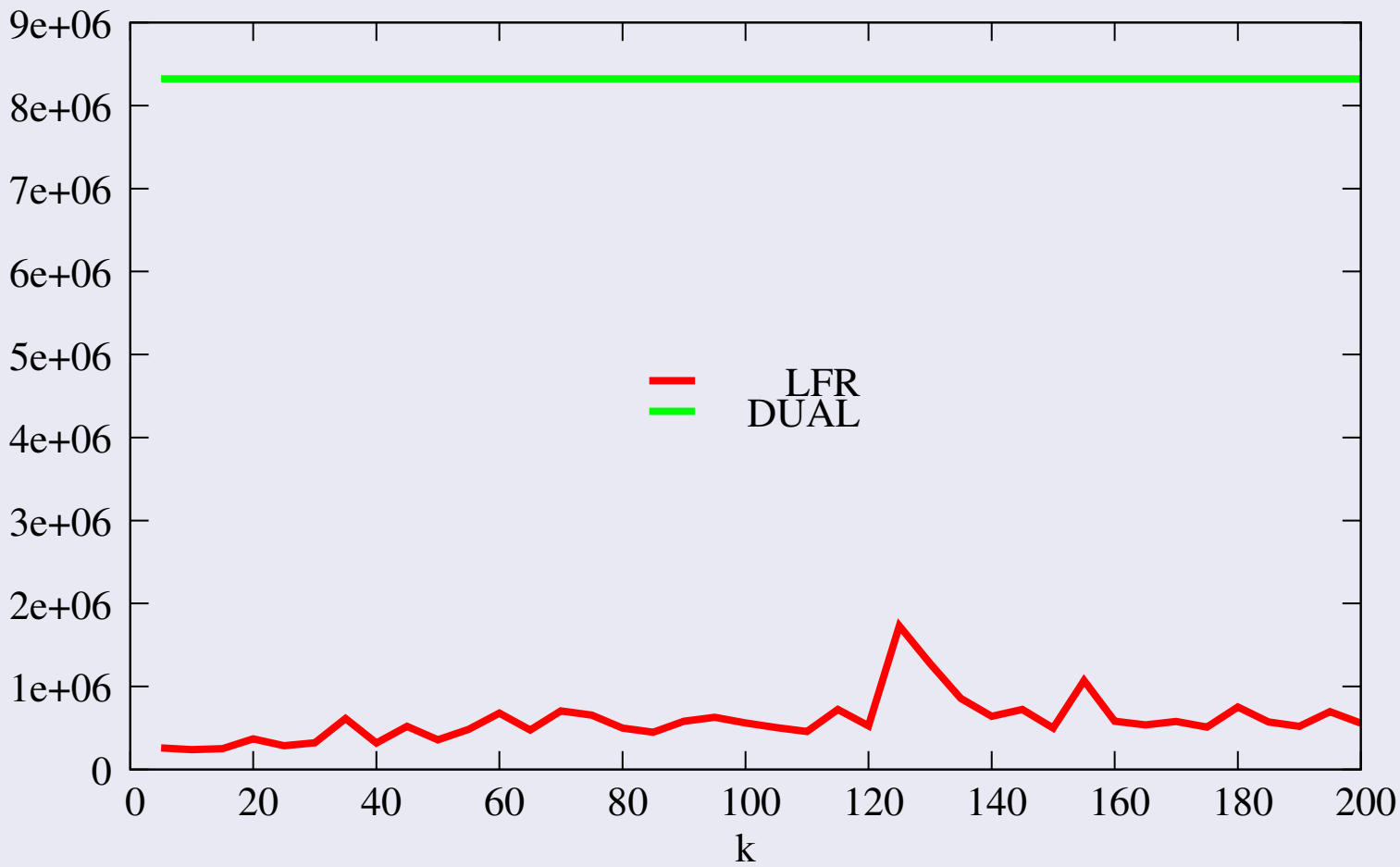
Results – Message complexity in real-world instances

Number of messages sent - G_{IP_8000}



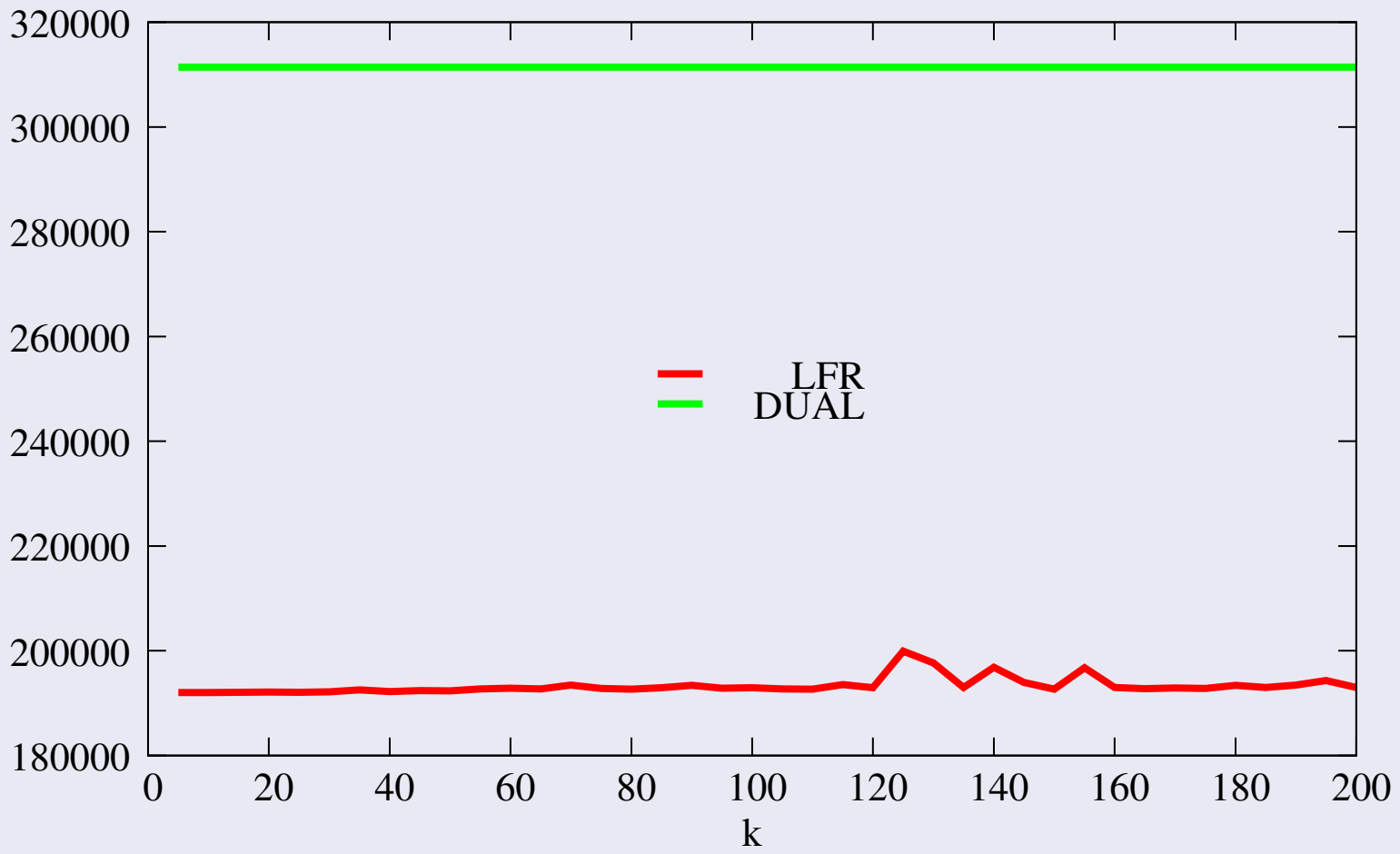
Results – Space complexity in real-world instances

Maximum space occupancy - $G_{IP-8000}$



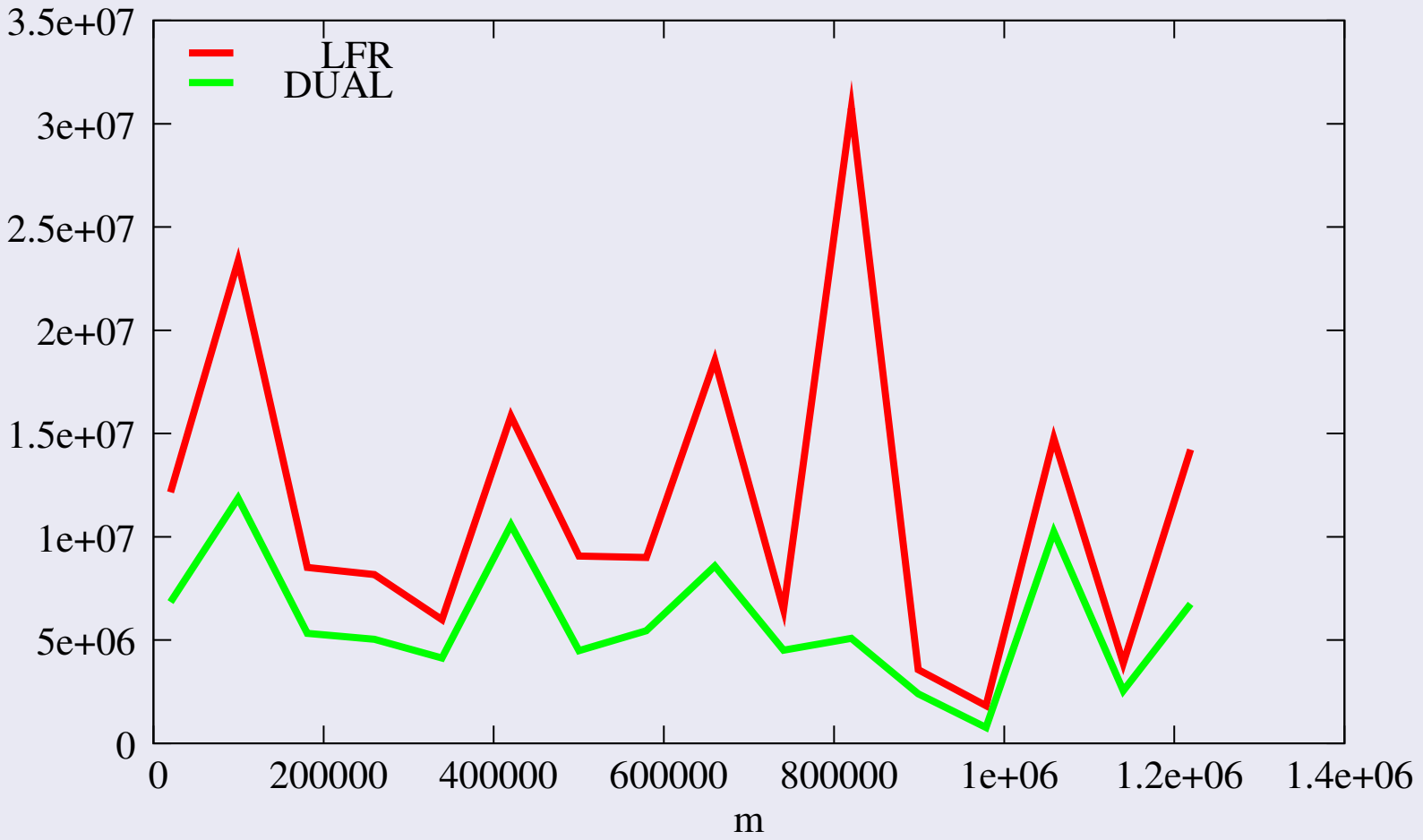
Results – Space complexity in real-world instances

Average space occupancy - $G_{IP-8000}$



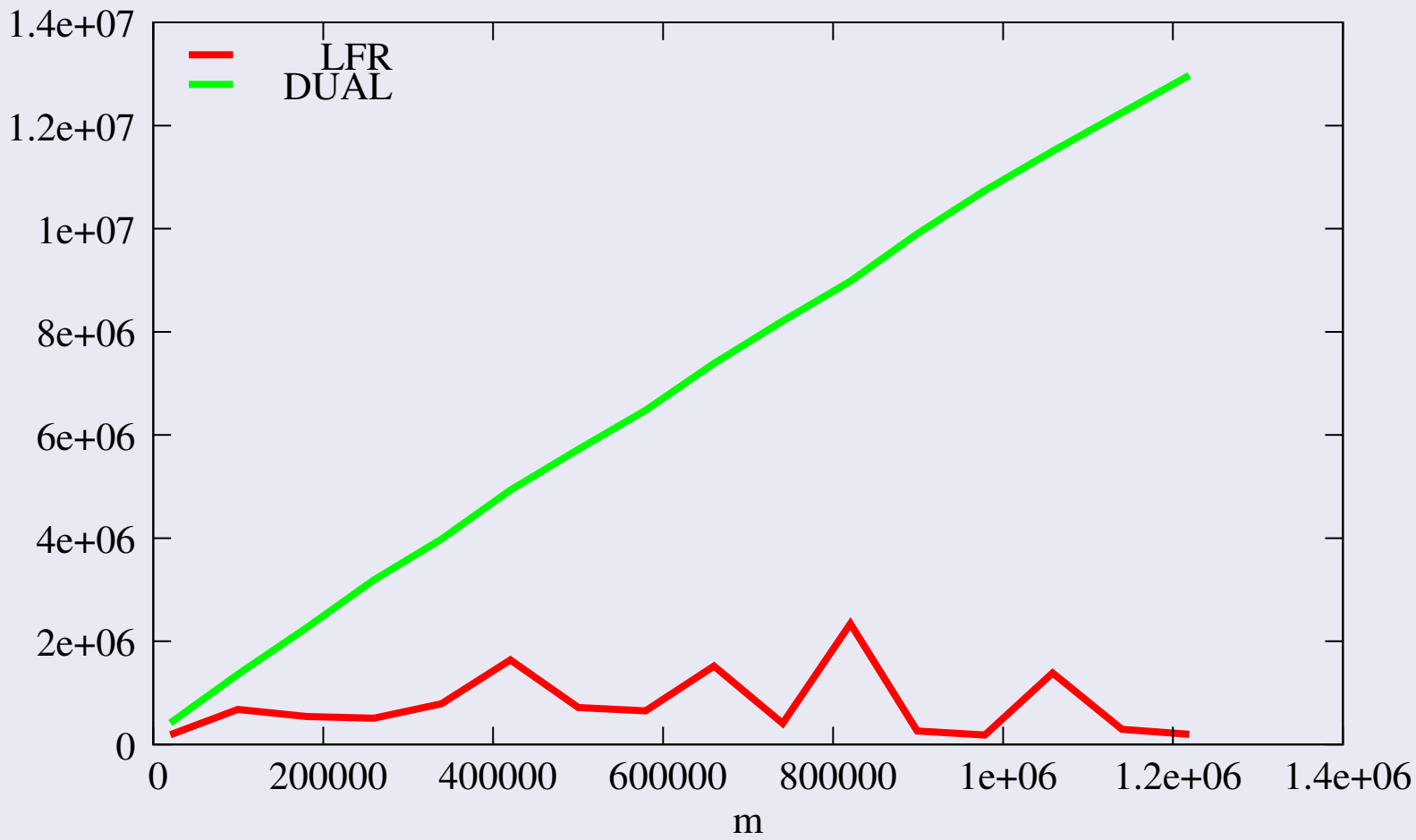
Results – Message complexity in artificial instances

Number of messages sent - $G_{ER-2000}$



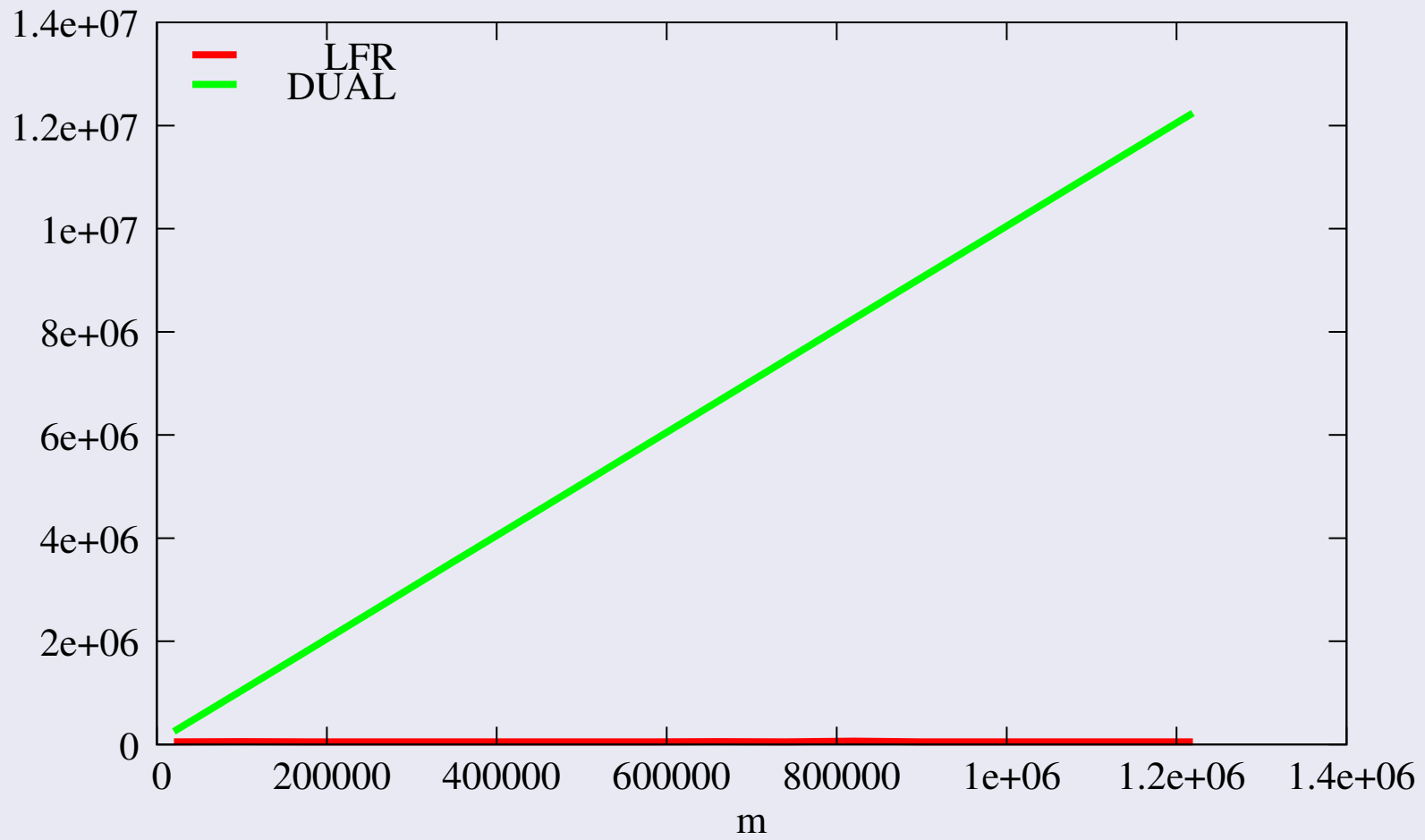
Results – Space complexity in artificial instances

Maximum space occupancy - $G_{ER-2000}$



Results – Space complexity in artificial instances

Average space occupancy - $G_{ER-2000}$



Outline

1 Motivation

2 The new algorithm

3 Experimental evaluation

4 Conclusion and future research

Conclusion

The new algorithm

- We proposed **LFR**, a new fully dynamic loop-free shortest paths routing algorithm which:
  - has the same message complexity of **DUAL**
  - has a better space complexity

Experimental contribution

- We implemented and tested **LFR** and **DUAL**
- We showed that:
  - **LFR** is always the best choice in terms of space requirements
  - **LFR** is the best choice in terms of messages sent in the CAIDA real-world networks

Conclusion

The new algorithm

- We proposed **LFR**, a new fully dynamic loop-free shortest paths routing algorithm which:
  - has the same message complexity of **DUAL**
  - has a better space complexity

Experimental contribution

- We implemented and tested **LFR** and **DUAL**
- We showed that:
  - **LFR** is always the best choice in terms of space requirements
  - **LFR** is the best choice in terms of messages sent in the CAIDA real-world networks

Future research

- Improve **LFR**, like e.g. by:
  - reducing the message/space complexity while preserving loop-freedom
- Design new algorithms tailored to be effective in other real world practically interesting scenarios
- Extend the experimental evaluation to other real-world/artificial instances

Thank you for your attention - Q&A