Compact Routing Schemes for Dynamic Ring Networks

Seminar in Distributed Computing WS 05/06

Jonas Rutishauser

Overview

- **Introduction**
- Overview
- Scheme with Adaption Cost Zero
- Scheme with Linear Adaption Cost
- Scheme With Constant Expected Adaption Cost
- Conclusion

Introduction

- Settings
  - asynchronous dynamically changing ring of processors
  - fault free
- Static techniques
  - significant recomputing on change
- Known dynamic techniques
  - inefficient schemes

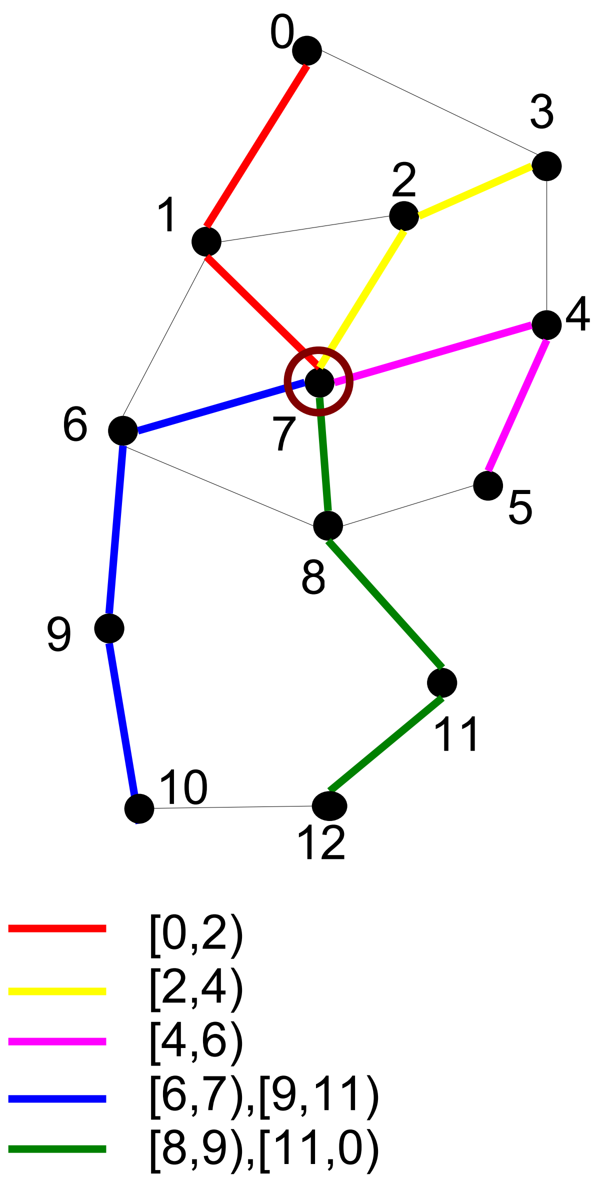
==> **Dynamic Interval Routing**

Overview

- Introduction
- **Overview**
- Scheme with Adaption Cost Zero
- Scheme with Linear Adaption Cost
- Scheme With Constant Expected Adaption Cost
- Conclusion

k-Interval Routing Schemes (k-IRS)

- N-Node Network
  - Nodes labeled from 0 to N-1
  - every arc leaving Node i has k disjoint intervals assigned
  - message from i to j forwarded through arc containing j
  - Space required per Node: O(k*d*log(N))
    d: degree of Node



Dynamic Interval Routing (DIR)

- Nodes labeled from 0 to N-1
- based on the 1-IRS
- not all always on-line
- update procedure on change
  - after going on-line
  - **before going off-line**

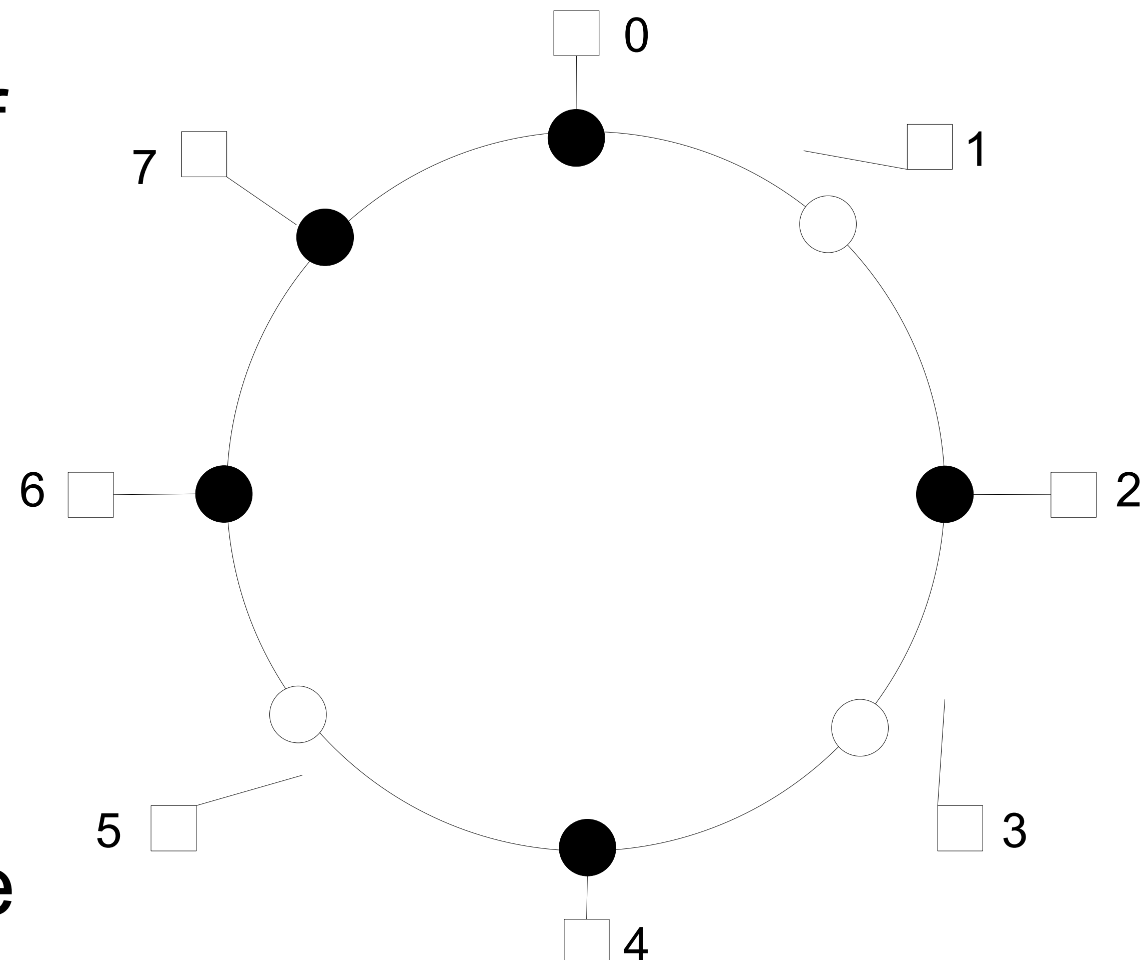
Definitions

- *pending*: processor coming on-line or going off-line but not completed update procedure
- *non-/active*: completed update procedure
- *quiescence*: all processors are either active or non-active

- Correct:
  - Message travels only a bounded number of steps
  - receiver receives the message if he was active during the entire lifetime of the message

System

- bidirectional ring of N processors
- FIFO-Queues
- global Orientation
- N Switches
- 0 always active
- n: number of active and pending processors
- closed switches have cost 1 others 0

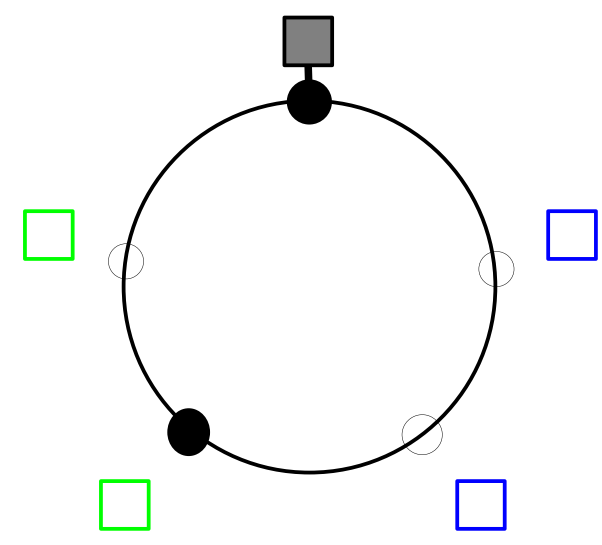


Overview

- Introduction
- Overview
- **Scheme with Adaption Cost Zero**
- Scheme with Linear Adaption Cost
- Scheme With Constant Expected Adaption Cost
- Conclusion

Scheme with Adaption Cost Zero

- divide ring in two halves
- no message when going on-/off-line
- stretch factor: min{ n-1 ,⌊N/2⌋}
- Intervals:
  - $l_i = [i+1 \, mod \, N, i+N/2 \, mod \, N]$
  - $r_i = [i+1+N/2 \, mod \, N, i-1 \, mod \, N]$
- Message: M=(D,r,s,x)
  - D: information
  - r: receiver



s: source

x: times passed 0

Scheme with Adaption Cost Zero

- Properties:
  - space at most O(logN) per Node
    - N, label, two intervals of O(logN) bits
  - adaption cost zero
    - trivial
  - stretch factor at most min{ n-1 ,$\lfloor N/2 \rfloor$}
    - travels always in the same direction
    - at most N/2 active processors
    - can be at most n-1

Overview

- Introduction
- Overview
- Scheme with Adaption Cost Zero
- **Scheme with Linear Adaption Cost**
- Scheme With Constant Expected Adaption Cost
- Conclusion

Scheme with Linear Adaption Cost

- dynamically update intervals
- interval delimited by active opposite processor
- update procedure when going on-/off-line
  - 3 phases
  - phase 1 for sequentializing
  - phase 2/3 for updating values of all processors
- store
  - left label, left opposite, label, opposite, old opposite, even, right label, right opposite

Sequentializing

- messages from higher label can pass
- messages from higher phase can pass
- buffering other messages

- only one processor can pass to phase 2 at a time

Update Procedure

- Send phase 1 message to left
  - collect left and right neighbors values
  - do sequentializing
- getting message back => „won“
  - calculate own values
- start phase 2 and then phase 3
  - propagate the new values the other processors
  - one phase for each direction

Proof (1)

- Lemma
  - at most one processor enters phase 2 at a time
- Proof by contradiction
  - Assume x!=y and both enters phase 2
  - Assume x<y
  - x passed before y got up
  - phase 1 message of x before message of y
  - x gets into phase 2
  - phase 1 message of y can't pass x

Proof (2)

- Lemma
  - all pending processors enter phase 2

- Proof
  - Blocked by other in phase 2/3
    - will continue after other finishes
  - Blocked by higher labeled processor
    - highest will enter phase 2
      - number of higher labeled processors decreases
  - => number of pending processors decreases

Scheme with Linear Adaption Cost

- Properties:
  - stretch factor: 1
    - use opposite for intervals
  - space: O(log*N*) bits per Node
    - constant number of values of O(logN) bits
  - adaption cost per pending processor:
    O(n) messages of O(log*N*) bits
    - 3*n messages (n messages for each phase)
    - constant number of values in messages of O(logN) bits

Overview

- Introduction
- Overview
- Scheme with Adaption Cost Zero
- Scheme with Linear Adaption Cost
- **Scheme With Constant Expected Adaption Cost**
- Conclusion

Scheme with Constant Expected Adaption Cost

- Randomized DIR
- expected stretch factor: 1+1/k,   k≥3
- intervals delimited using estimation of opposite
- update opposite with probability such that:
  - expected adaptation cost: O(k)
  - expected stretch factor: <1+1/k
- update procedure when going on-/off-line

Properties

- Store per processor:
  - own label
  - opposite value

- update uses 3 phases
  - phase 1 to get number of on-line processors
  - phase 2 to get (equally spaced) subset of labels
  - phase 3 to let every processor update their values

Update Procedure (1)

- send request (A message) to left processor
  - label, number of on-line processors and opposite
  - if receiving a phase 1 message count as active
    - values will be set later in phase 2/3 and get active
- get values from left processor (R message) or phase 3 message
- flip coin if should start an update
  - probability of update: min{1,10k/ñ}

    (ñ number of active processors got previously)
  - shouldn't update => active

Update procedure (2)

- send phase 1 message
  - content
    - counter n_0 of active processors
    - counter n_1 of pending processors
    - own label
  - use sequentializing from previous algorithm
    - not winning processors will update with the phases 2/3 of the winning processor and get active afterwards
- get phase 1 message back
  - $n = n_0+n_1$

Update procedure (3)

- send phase 2 message
  - collect label of every n/10k -th processor
    => stores $n*(10k/n) \leq 20k$ labels
- get phase 2 message back
  - calculate opposite using labels of phase 2 message

Update procedure (4)

- send phase 3 message
  - content
    - labels of phase 2 message
    - n
  - update active and pending processors opposite and n value
- get phase 3 message back
  - become active

Proof

- Lemma
  - every pending processor will go on-/off-line after some time
- Proof
  - 4 cases after sending first message to left

    a) receive message back from left, flip coin and get tail
      - become active

    b) receive message back, flip coin and get head
      - enter phase 1 and rest similar to previous algorithm

    c) receives phase 1 message
      - will participate update and get active afterwards

    d) receives phase 2/3 message
      - wait until end of update and than flip coin => a) or b)

Properties (1)

- expected amortized number of messages: O(k) of O(k*log*N*) bits
  - message size
    - max O(k) values of size O(logN)
  - number of messages:
    - A pending processor is responsible for at most
      - one A message and 1 R message
    - A processor sends per update at most
      - two phase 1 message
        - its own (got R) or from other (got phase 1)
        - winners message
      - one phase 2/3 message

Properties (2)

- update phases have probability min{1,10k/n}
- let n' = changes since last update
- update is responsible for 3(n+n') messages
  - => cost ≤ $6+min\{1,\frac{10k}{n}\}\cdot 3\frac{(n+n')}{1+n'}=O(k)$

Properties (3)

- space: at most O(log*N*) bits per node
  - constant number of values of size O(logN)
- expected stretch factor: 1+1/k
  - consider at quiescent state
  - last update done by processor i
  - n_0 = active processors counted by i
  - n_1 = pending processors counted by i
  - n_2 = change of size in the ring since last update

Proof (1)

- each processor in n_2 flips coin with head probability of $\min\{1, 10k/(n_0+n_1)\}$
- expected value of $n_2 \le (n_0+n_1)/10k$
- let $v = (n_0+n_1)$, $D=v/(10k)$
- at most $\lambda=v/D$ labels are collected in phase 2
- collected labels are: $V=\{v_0,v_1,...,v_{\lambda-1}\}$
- let v_j in V be first processor after x or x self
- $op(x) = v_{(j+\lambda/2)\mathrm{mod}\lambda}$

Proof (2)

- minimum distance between x and op(x):
  - 1+(λ/2-1)D≥1+(λ/2-3/2)D≥1+(v/(2D)-3/2)D
- in worst case the distance decreases by n_2
- => stretch factor bounded to:

$$\frac{v-(1+(v/(2D)-3/2)D)}{1+(v/(2D)-3/2)D-v/(10k)} \leqslant \frac{v/2+3D/2}{v/2-3D/2-v/(10k)} \leqslant \frac{10k+3}{10k-5}$$

- => stretch factor bounded by 1+1/k for k≥3

Overview

- Introduction
- Overview
- Scheme with Adaption Cost Zero
- Scheme with Linear Adaption Cost
- Scheme With Constant Expected Adaption Cost
- **Conclusion**

Conclusion

- works also with rings of ring networks
- randomized algorithm isn't tested in practice
- must know N before
- every Node has his fixed place in the ring

- interval-routing seems only be useful for
  - special topologies like rings and trees
  - or if space is expensive
- the intervals are calculated using a tree in other topologies

Questions?