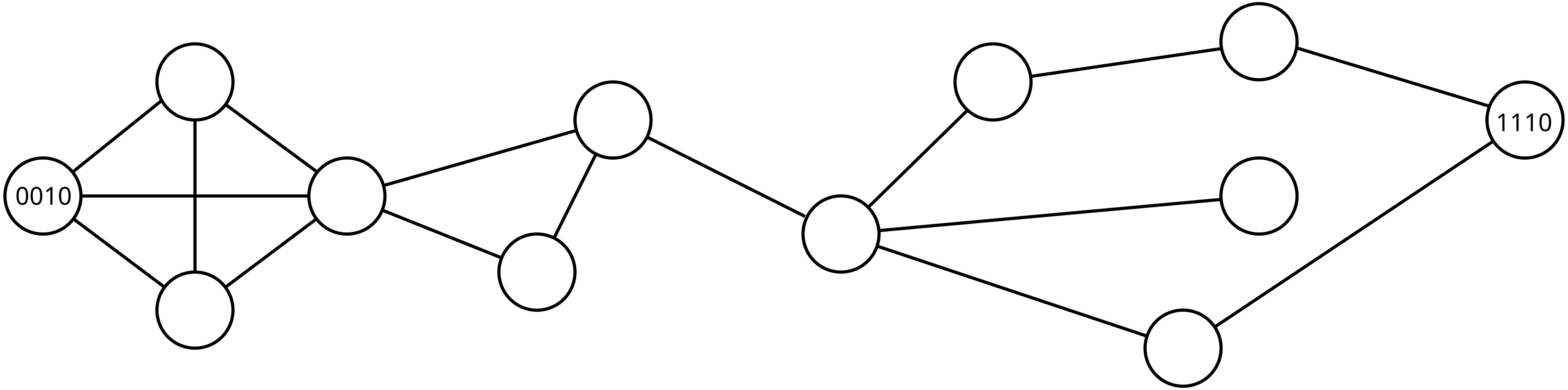
Space-stretch tradeoff in routing revisited

Tolik Zinovyev

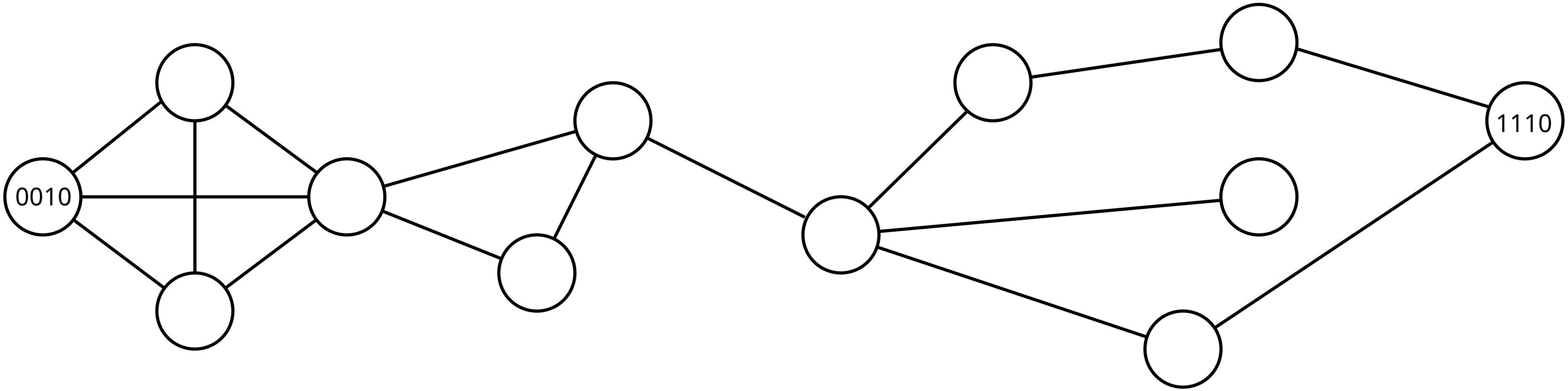
Routing in computer networks, model

- Undirected graph (network)



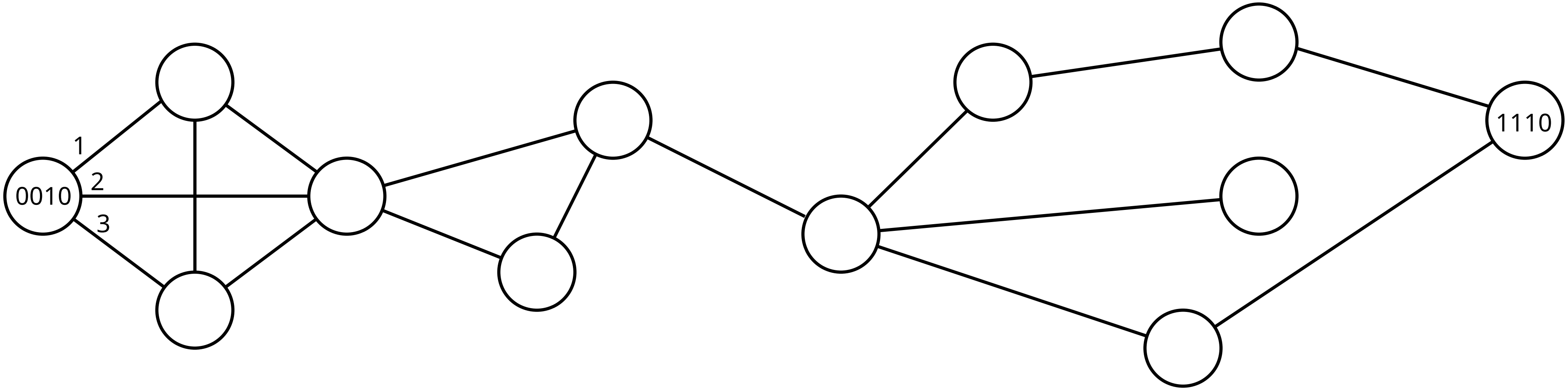
Routing in computer networks, model

- Undirected graph (network)
- Nodes have labels (a binary string)



Routing in computer networks, model

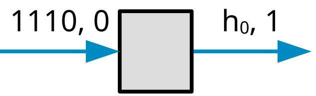
- Undirected graph (network)
- Nodes have labels (a binary string)
- Nodes have ports

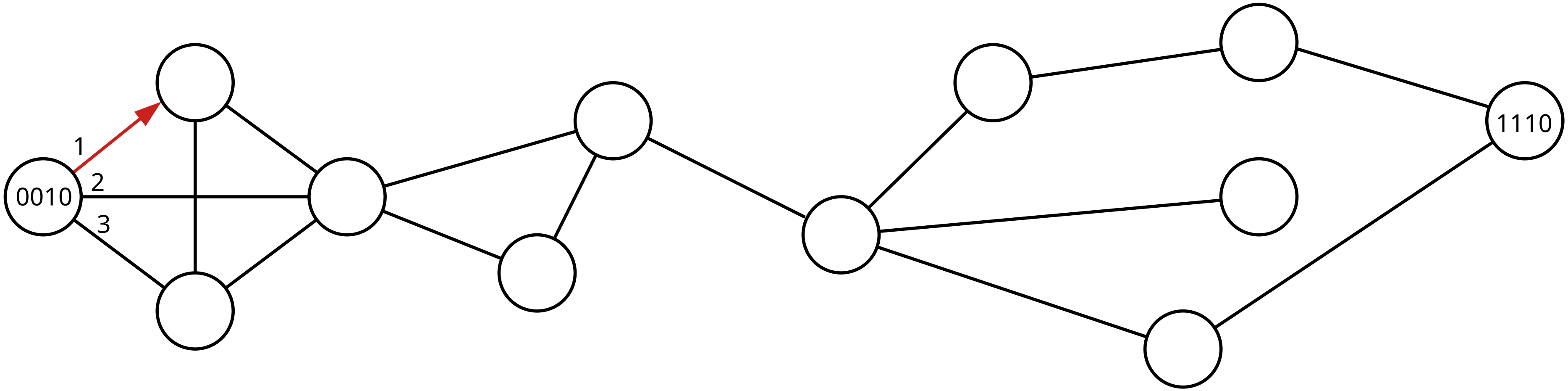


Routing in computer networks, model

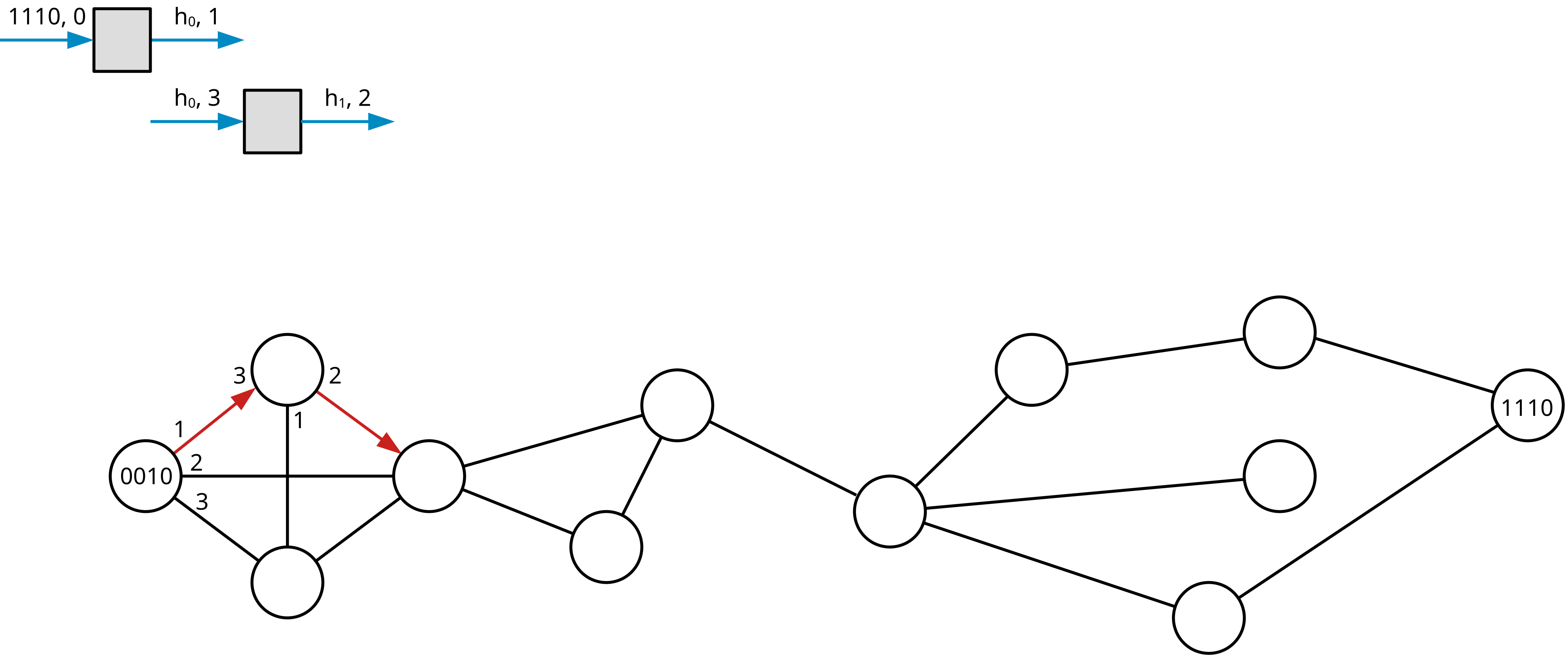
- Undirected graph (network)
- Nodes have labels (a binary string)
- Nodes have ports
- Nodes have routing programs

Routing in computer networks, model

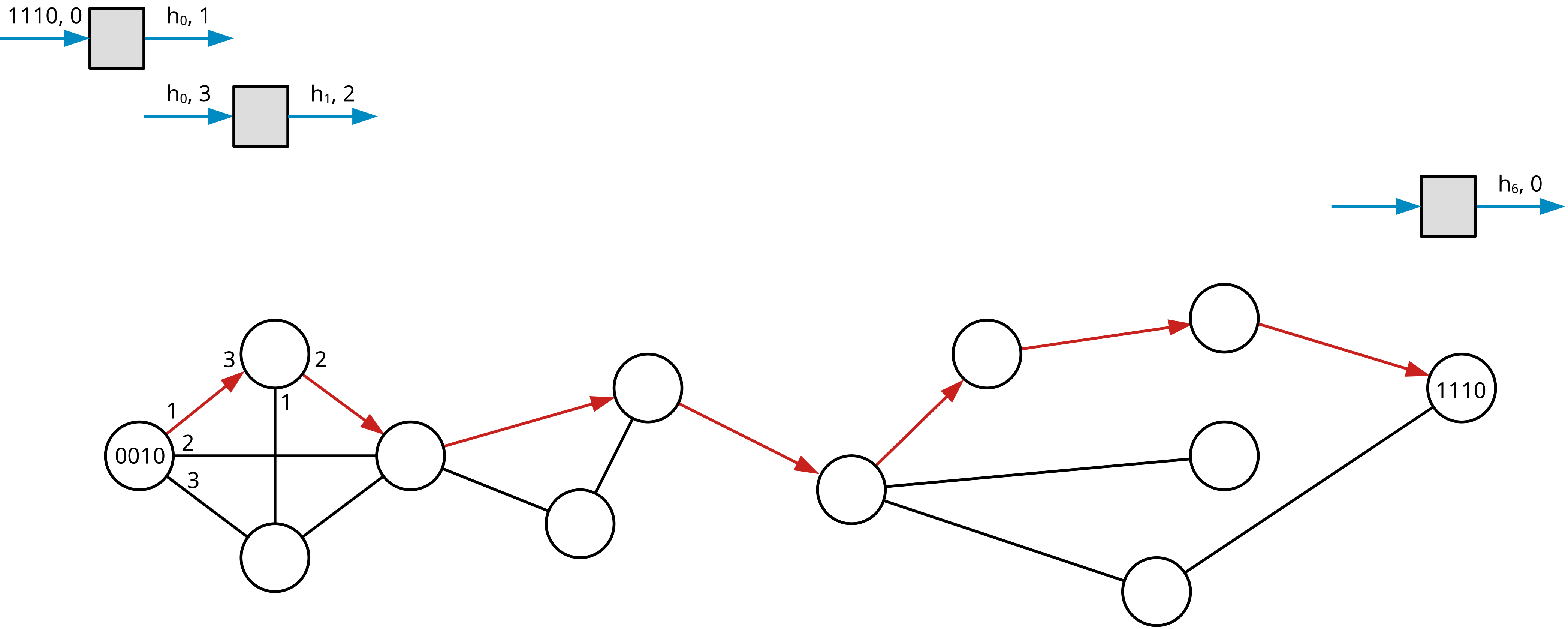




Routing in computer networks, model



Routing in computer networks, model



Routing characteristics

- Space usage (program size)
- Routing stretch
  - Route length / (shortest) distance

Model, continued

- Adversarial / <u>non-adversarial</u> labels (bit strings)
  - Adversarial: labels given, more practical
  - Non-adversarial: designer labels, often used in adversarial labels routing schemes

Model, continued

- Adversarial / <u>non-adversarial</u> labels (bit strings)
  - Adversarial: labels given, more practical
  - Non-adversarial: designer labels, often used in adversarial labels routing schemes
- Adversarial / non-adversarial ports (1..deg)
  - Adversarial: ports given
  - Non-adversarial: designer ports

Known results + contribution

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	$< 5/3$	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Known results + contribution

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	< 5/3	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	< 2k+1	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	< 2k+1	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Known results + contribution

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	$< 5/3$	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are $[n]$
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	$< 2k+1$	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	$< 2k+1$	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	$< 2k+1$	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

Adversarial ports and labels

Some upper bounds

Work	Stretch	Local memory (bits)
Thorup and Zwick (2001)	4k - 5	$\tilde{O}(n^{1/k})$ on every node
Chechik (2013)	ck, c<4	$\tilde{O}(n^{1/k} \log D)$ on every node

Non-adversarial labels, adversarial ports

Work	Stretch	Local memory (bits)
Abraham et al. (2008)	3	$\tilde{O}(\sqrt{n})$ on every node
Abraham et al. (2006)	O(k)	$\tilde{O}(n^{1/k})$ on every node

Adversarial labels, adversarial ports

Lower bounds with non-adversarial ports

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	$< 5/3$	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	$< 2k+1$	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	$< 2k+1$	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	$< 2k+1$	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

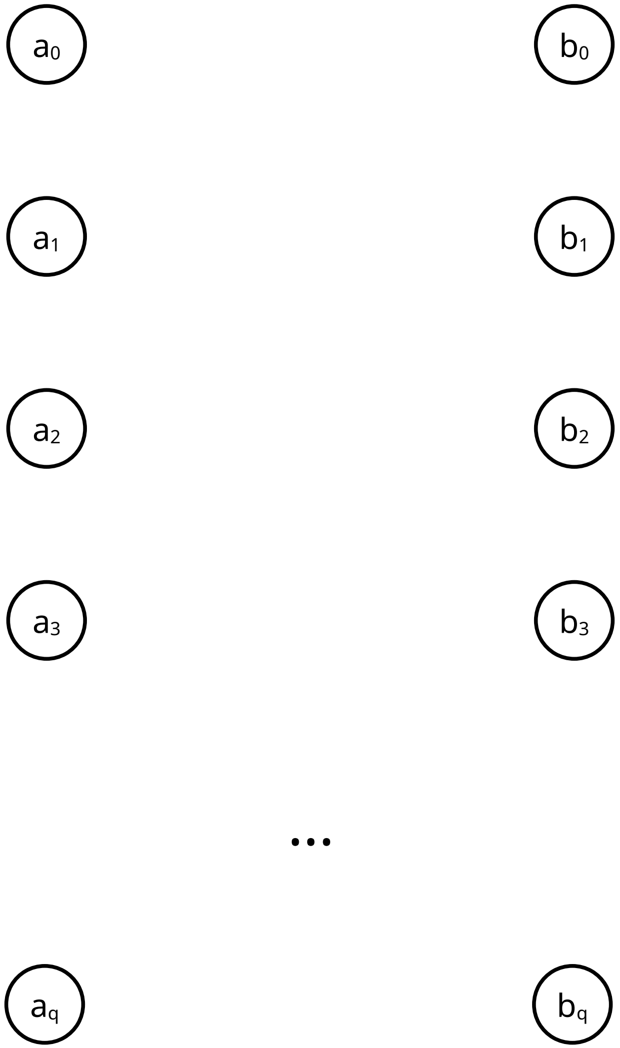
Adversarial ports and labels

Thm: there is a graph with n vertices for which all stretch < 3 routing schemes have a routing program of size $\Omega(n)$ bits

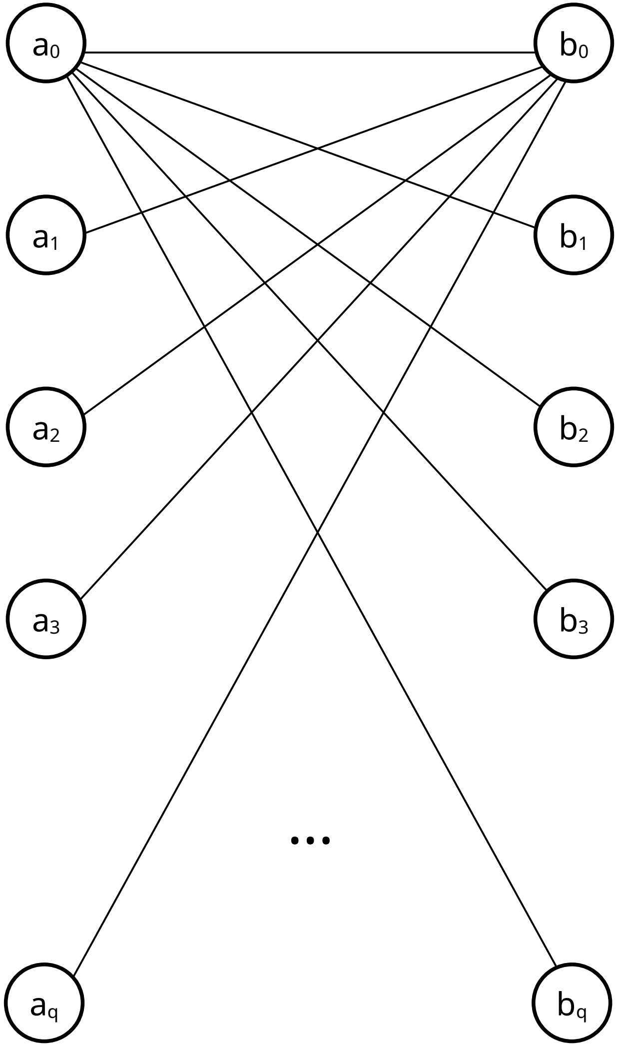
Lower bounds with non-adversarial ports

- Idea: consider a family of graphs and show that $2^{\Omega(n^2)}$ routing schemes are needed to satisfy all graphs
- Then $\Omega(n^2)$ bits are necessary to describe some routing scheme
- Then at least one routing program must be $\Omega(n)$ bits

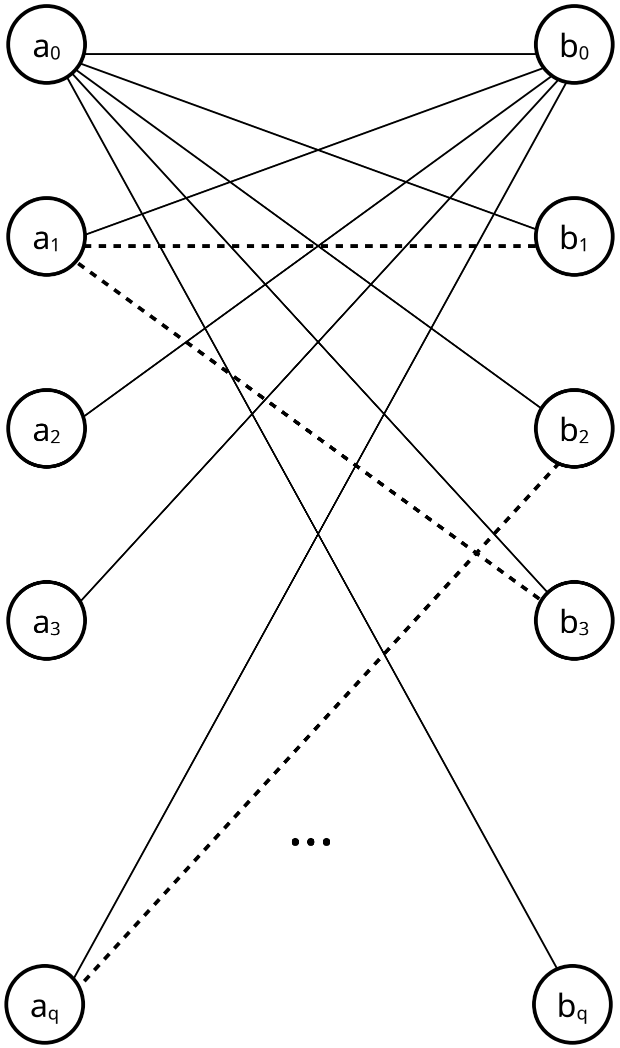
- Let $n = 2q + 2$, and consider a family of bipartite graphs with parts A and B



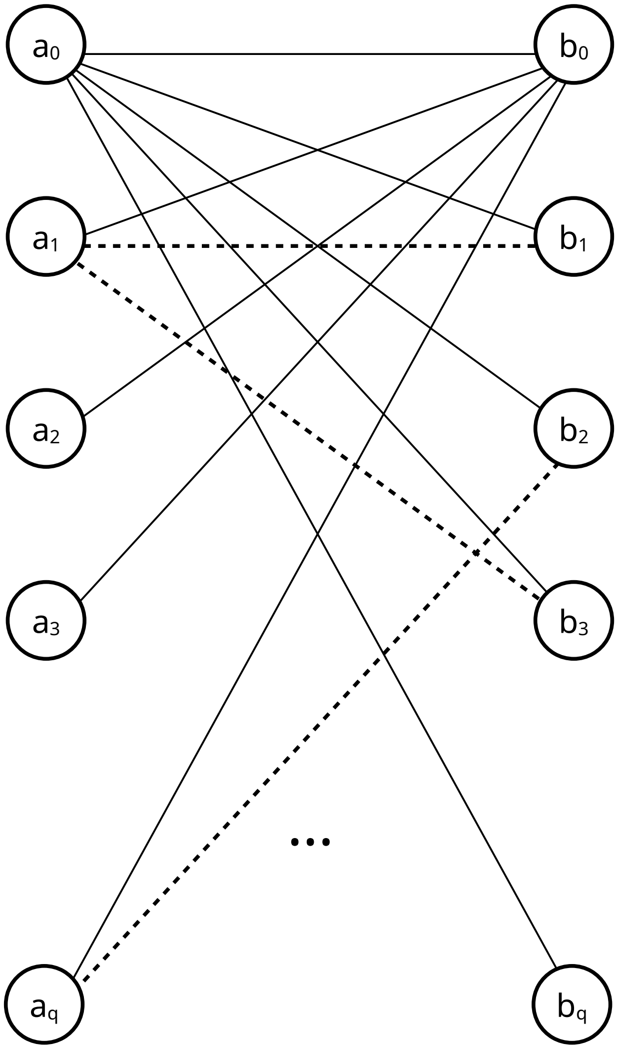
- Let $n = 2q + 2$, and consider a family of bipartite graphs with parts A and B
- a_0 is connected to all b_i, b_0 is connected to all a_i



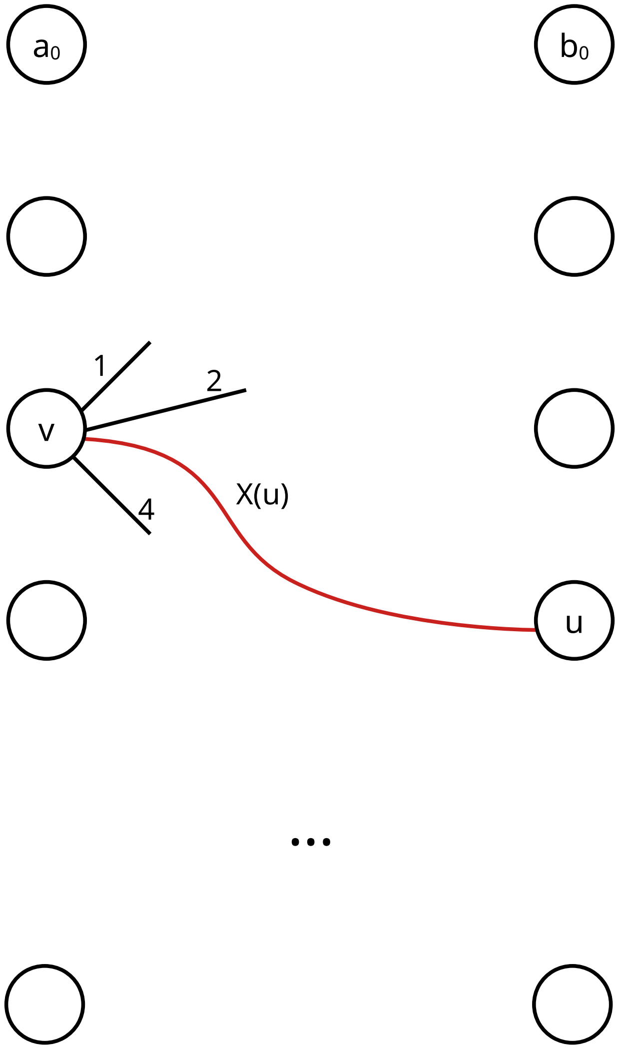
- Let $n = 2q + 2$, and consider a family of bipartite graphs with parts A and B
- a_0 is connected to all b_i, b_0 is connected to all a_i
- All other nodes are connected arbitrarily (2^{q^2} graphs)



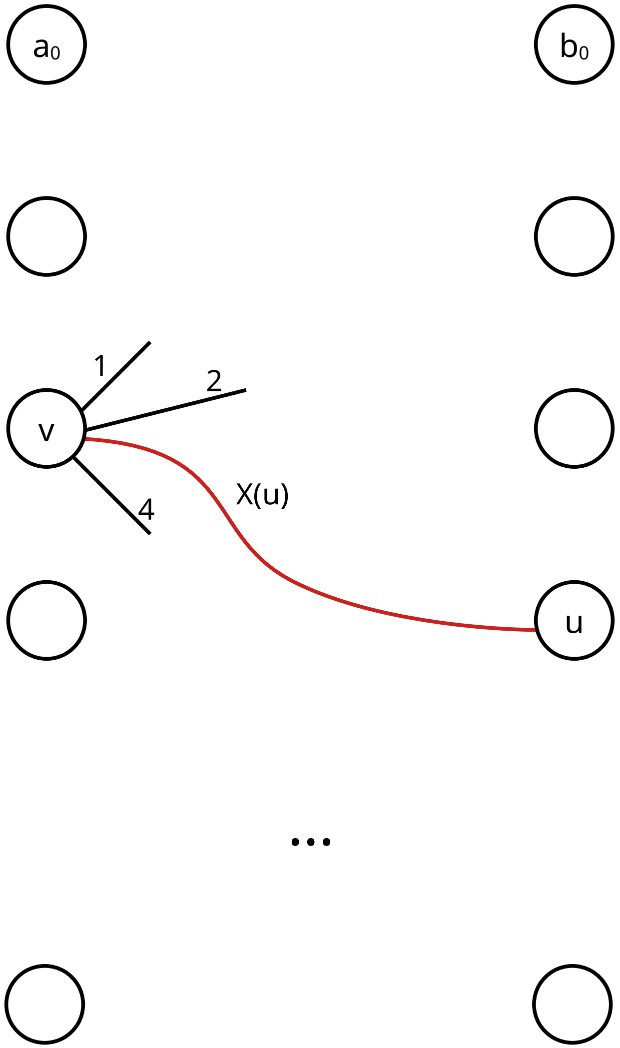
- Let $n = 2q + 2$, and consider a family of bipartite graphs with parts A and B
- a_0 is connected to all b_i, b_0 is connected to all a_i
- All other nodes are connected arbitrarily (2^{q^2} graphs)
- Routing to a neighbor must take the shortest path!



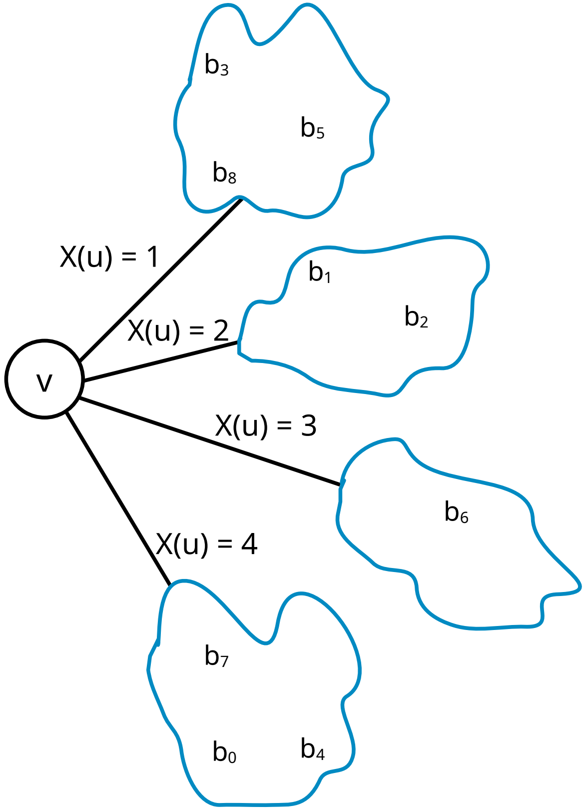
- Let's show that a single routing scheme cannot support many graphs
- Fix a routing scheme and node labels, and count supported graphs
- Take any $v \in A \setminus \{a_0\}$, define $X : B \to \mathbb{N} \cup \{0\}$



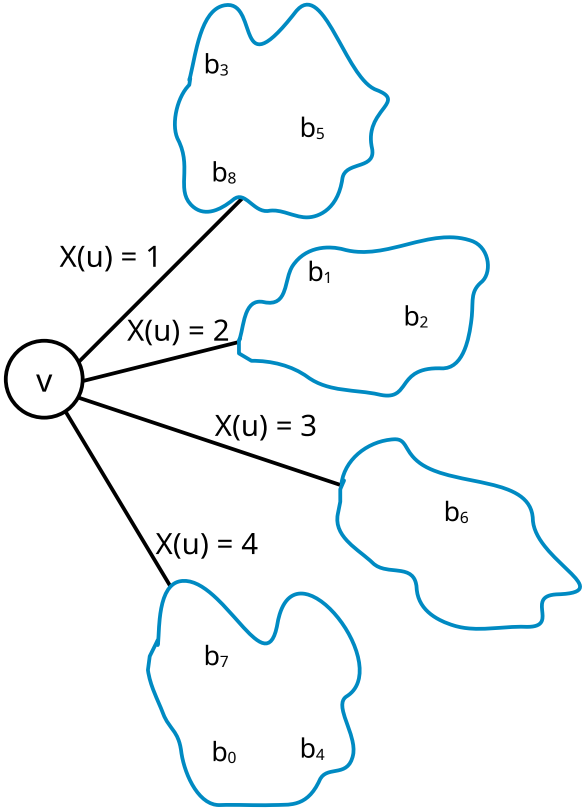
- Let's show that a single routing scheme cannot support many graphs
- Fix a routing scheme and node labels, and count supported graphs
- Take any $v \in A \setminus \{a_0\}$, define $X : B \to \mathbb{N} \cup \{0\}$
- Let $M = \max\{X(u) : u \in B\}$
- $\deg(v) \geq M$; otherwise, routing through undefined port
- $\deg(v) \leq M$; otherwise, some port is unused



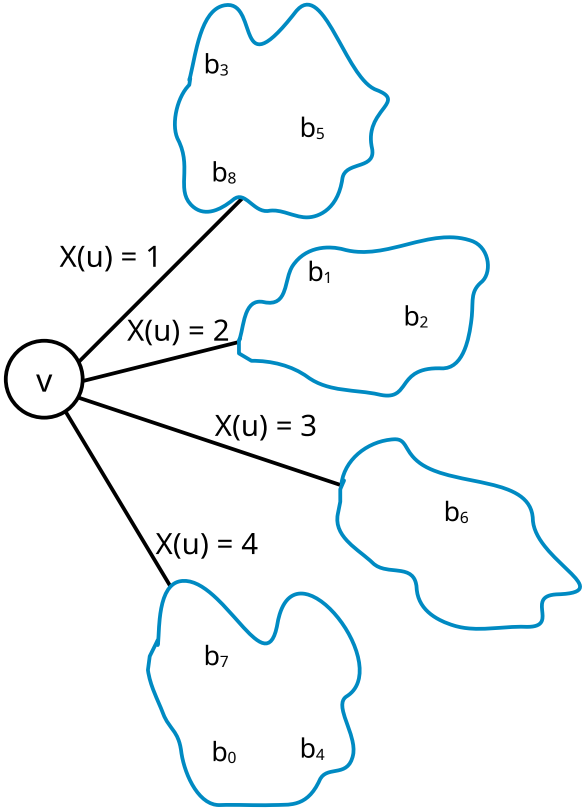
- Let’s show that a single routing scheme cannot support many graphs
- Fix a routing scheme and node labels, and count supported graphs
- Take any $v \in A \setminus \{a_0\}$, define $X : B \to \mathbb{N} \cup \{0\}$
- Let $M = \max\{X(u) : u \in B\}$
- $\deg(v) \geq M$; otherwise, routing through undefined port
- $\deg(v) \leq M$; otherwise, some port is unused
- Function $X(u)$ partitions B based on outgoing port number
- Neighbor behind port i must be in part i



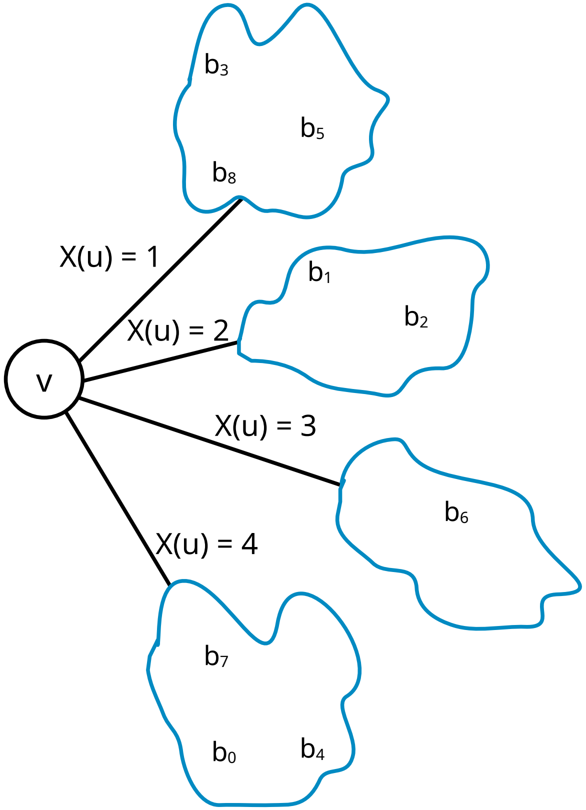
- Let s_i be the size of part i: $s_i = |\{u \in B : X(u) = i\}|$
- s_i possible neighbors behind port i
- Neighbors of v can take at most Πs_i possible values



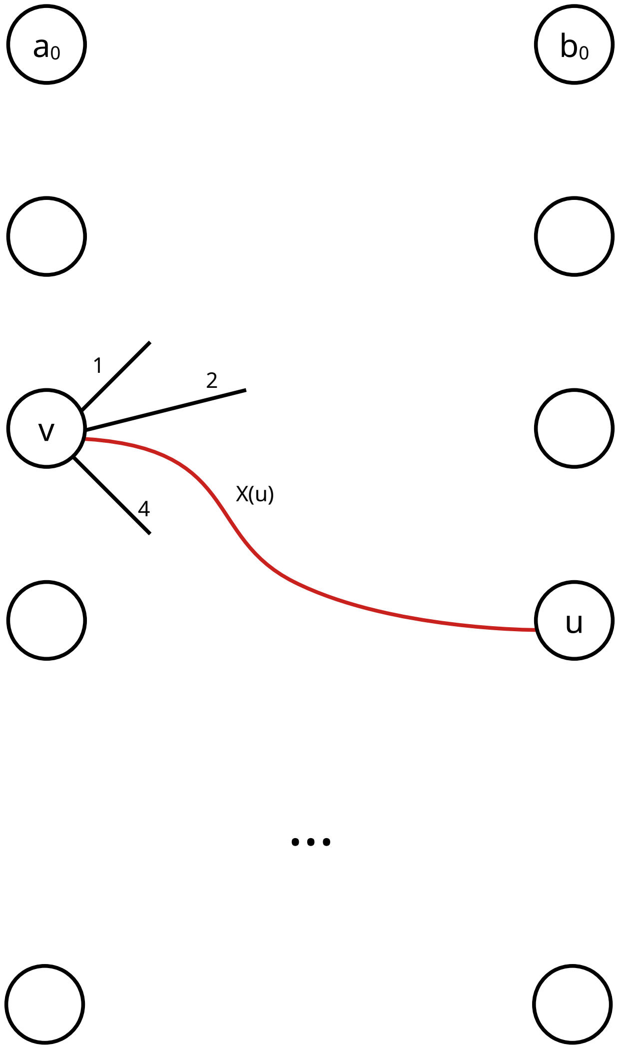
- Let s_i be the size of part i: $s_i = |\{u \in B : X(u) = i\}|$
- s_i possible neighbors behind port i
- Neighbors of v can take at most Πs_i possible values
- Also know that $\Sigma s_i = |B| = q+1$
- Geometic mean ≤ arithmetic mean: $(\Pi s_i)^{1/M} \leq (\Sigma s_i) / M$
- So, $\Pi s_i \leq ((q+1) / M)^M$



- Let s_i be the size of part i: $s_i = |\{u \in B : X(u) = i\}|$
- s_i possible neighbors behind port i
- Neighbors of v can take at most Πs_i possible values
- Also know that $\Sigma s_i = |B| = q+1$
- Geometic mean ≤ arithmetic mean: $(\Pi s_i)^{1/M} \leq (\Sigma s_i) / M$
- So, $\Pi s_i \leq ((q+1) / M)^M$
- Differentiating... Maximized when $M = (q+1) / e$
- $\Pi s_i \leq e^{(q+1)/e}$



- Let s_i be the size of part i: $s_i = |\{u \in B : X(u) = i\}|$
- s_i possible neighbors behind port i
- Neighbors of v can take at most Πs_i possible values
- Also know that $\Sigma s_i = |B| = q+1$
- Geometic mean ≤ arithmetic mean: $(\Pi s_i)^{1/M} \leq (\Sigma s_i) / M$
- So, $\Pi s_i \leq ((q+1) / M)$^M
- Differentiating… Maximized when $M = (q+1) / e$
- $\Pi s_i \leq e^{(q+1)/e}$
- Any routing scheme with fixed labels satisfies at most $e^{(q+1)/e \cdot q}$ graphs in the family



- Routing schemes necessary:
  (# graphs) / (# graphs per routing scheme) / (# node labels)
- Logarithm is in $\Omega(n^2)$

Lower bounds with non-adversarial ports

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	$< 5/3$	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	$< 2k+1$	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	$< 2k+1$	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	$< 2k+1$	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

Adversarial ports and labels

- $\Omega(n)$ bits of storage on arbitrary constant fraction of nodes
- Same neighbor configurations argument, but more careful counting

Lower bounds with adversarial ports

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	$< 5/3$	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	$< 2k+1$	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	$< 2k+1$	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	$< 2k+1$	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

Adversarial ports and labels

- In 2001 Thorup and Zwick claim $\Omega(n^{1/k})$ bits of storage for stretch $<2k+1$
- Proof idea: reduction from a lower bound for approximate distance oracles
- Distance oracles report approximate distances in a graph

Girth conjecture

- The lower bound for distance oracles relies on existence of dense graphs with large girth

Girth conjecture

- The lower bound for distance oracles relies on existence of dense graphs with large girth

def: graph G has girth s iff all cycles in G have length $\geq$ s

Girth conjecture

- The lower bound for distance oracles relies on existence of dense graphs with large girth
- Girth conjecture by Paul Erdős: there exists a graph with
  - n nodes
  - $\Omega(n^{1+1/k})$ edges
  - girth 2k+2
- Proven for k=1,2,3,5; weaker results for other k

def: graph G has girth s iff all cycles in G have length $\geq$ s

Lower bound for distance oracle

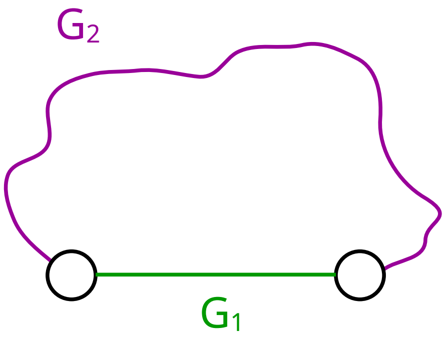
Prove: distance oracles with stretch < 2k+1 require m bits of storage

- Take a graph with n nodes, m edges and girth 2k+2
- All subsets of edges form a family of 2^m graphs

Lower bound for distance oracle

Prove: distance oracles with stretch < 2k+1 require m bits of storage

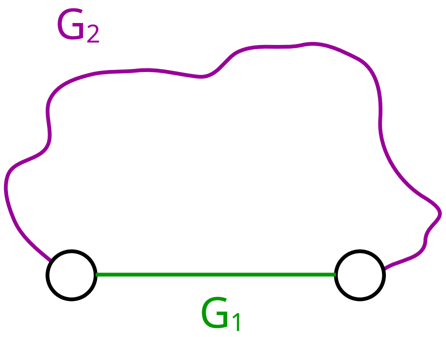
- Take a graph with n nodes, m edges and girth 2k+2
- All subsets of edges form a family of 2^m graphs
- Two distinct graphs require distinct distance oracles



Lower bound for distance oracle

Prove: distance oracles with stretch < 2k+1 require m bits of storage

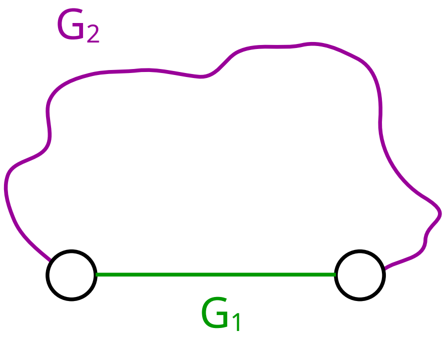
- Take a graph with n nodes, m edges and girth 2k+2
- All subsets of edges form a family of 2^m graphs
- Two distinct graphs require distinct distance oracles
- Need 2^m distinct distance oracles ⇒ m bits for some



Lower bound for distance oracle

Prove: distance oracles with stretch < 2k+1 require m bits of storage

- Take a graph with n nodes, m edges and girth 2k+2
- All subsets of edges form a family of 2^m graphs
- Two distinct graphs require distinct distance oracles
- Need 2^m distinct distance oracles ⇒ m bits for some
- By girth conjecture take $m = \Omega(n^{1+1/k})$



Reduction from distance oracles

- Suppose there is a routing scheme with low space complexity and low stretch
- Construct a distance oracle that simulates routing

Reduction from distance oracles

- Suppose there is a routing scheme with low space complexity and low stretch
- Construct a distance oracle that simulates routing
- Does not work in the standard model (ports in 1..deg(v))

Lower bounds with adversarial ports

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	< 5/3	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	< 2k+1	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	< 2k+1	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

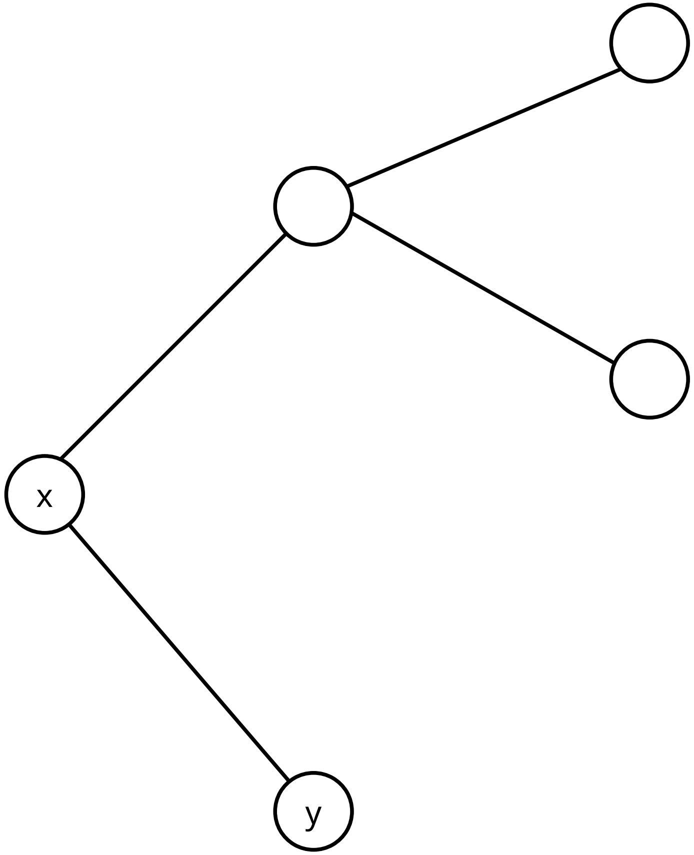
Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	< 2k+1	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

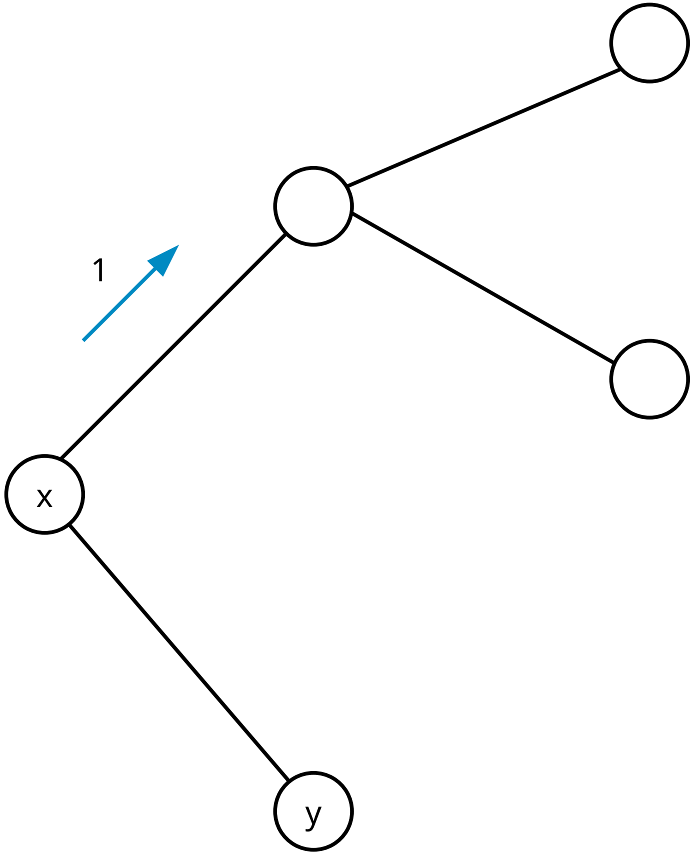
Adversarial ports and labels

- Thm: if graph with n vertices, m edges, girth 2k+2 exists, then routing with stretch < 2k+1 requires $\Omega(m/n \log(m/n))$ bits at some node
- Cor: routing with stretch < 2k+1 requires $\Omega(n^{1/k} \log n)$ bits at some node

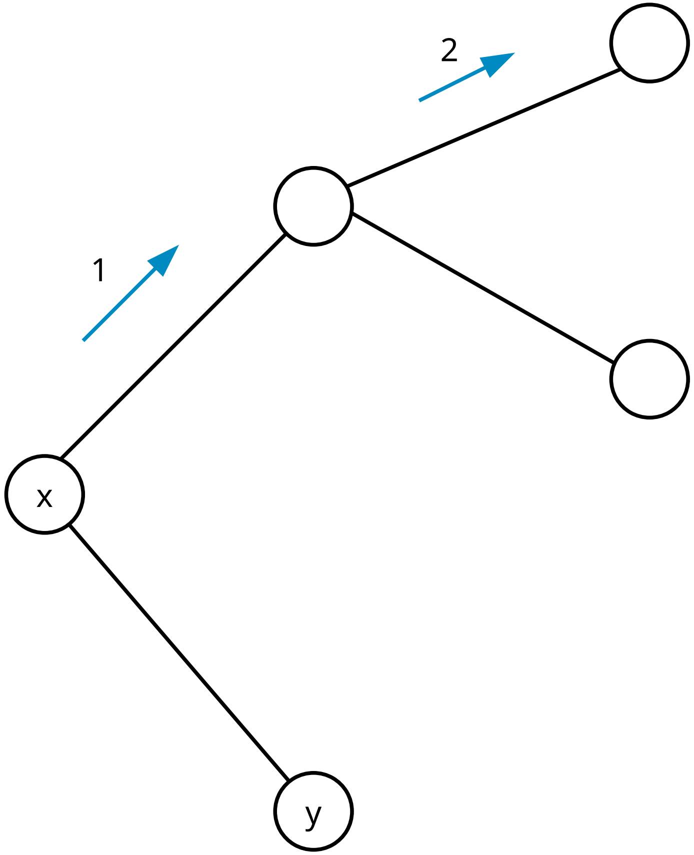
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



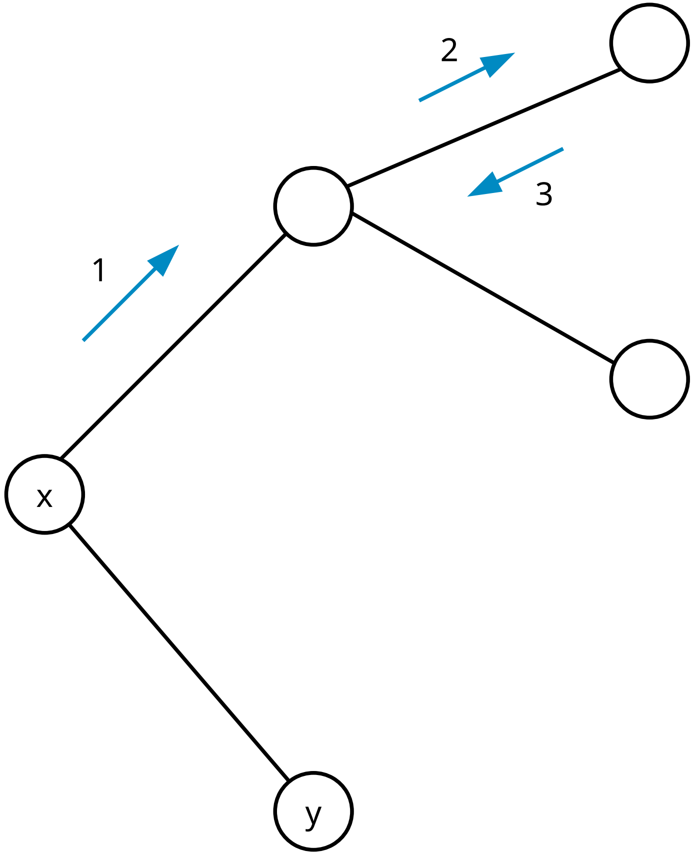
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



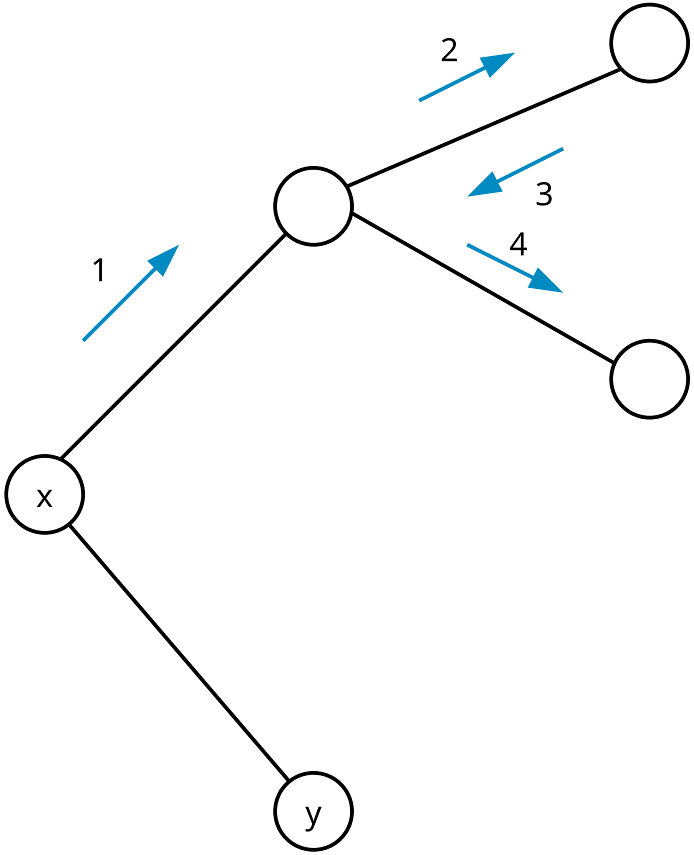
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



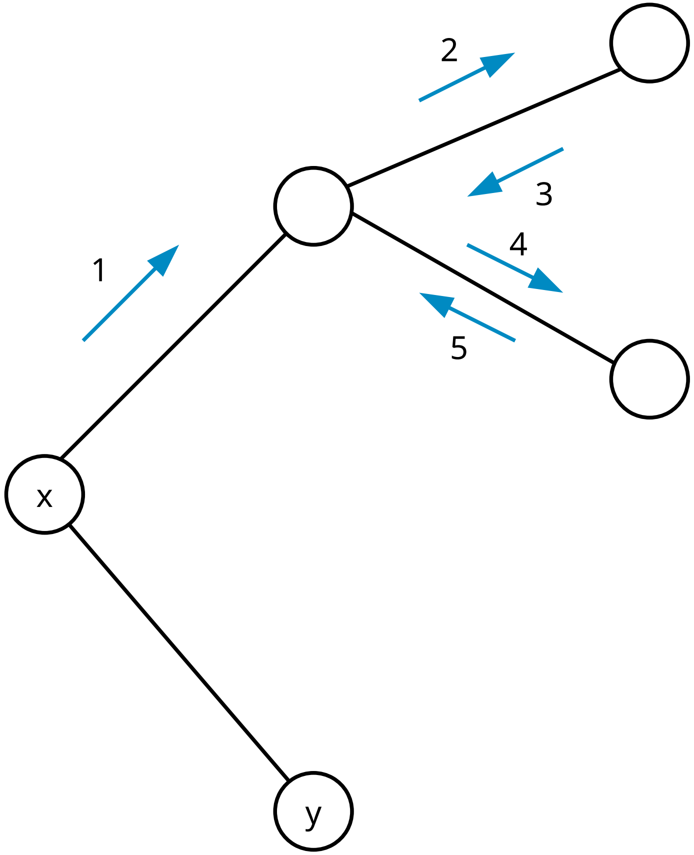
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



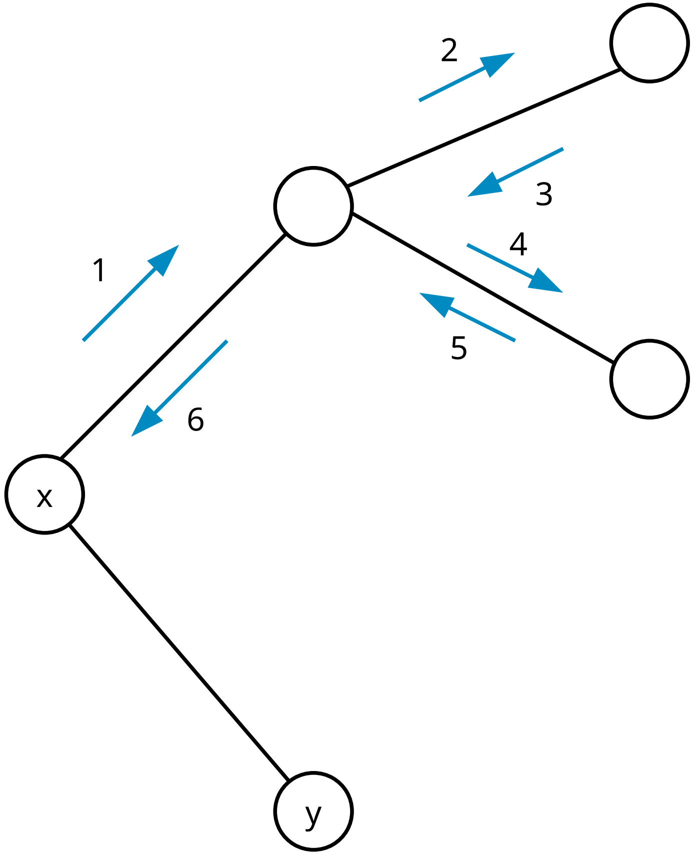
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



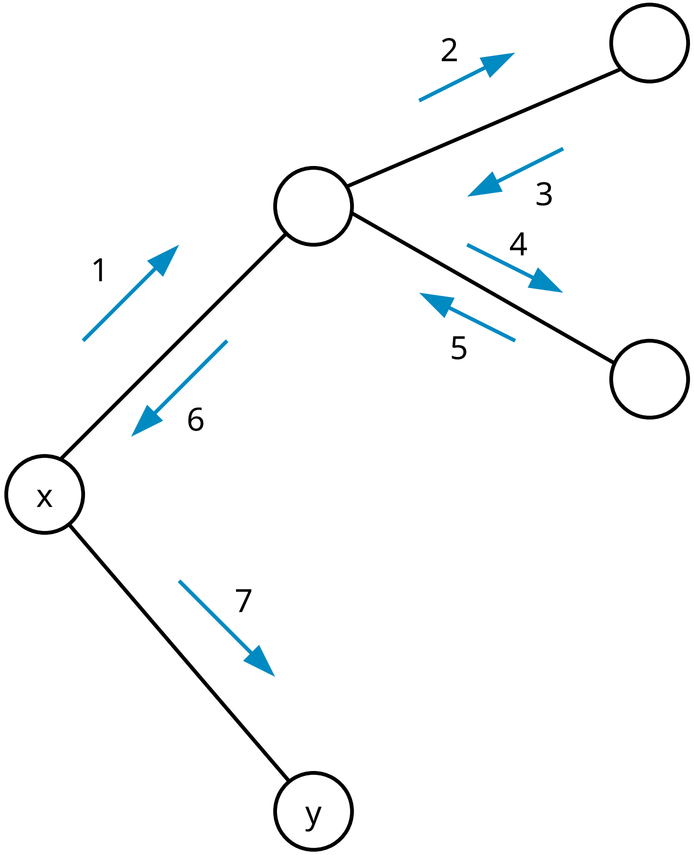
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



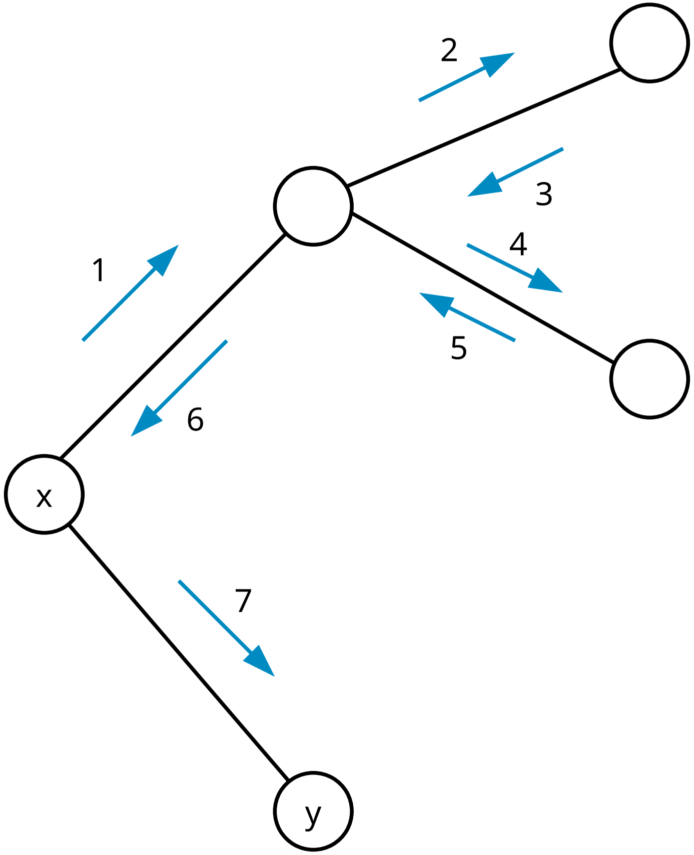
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



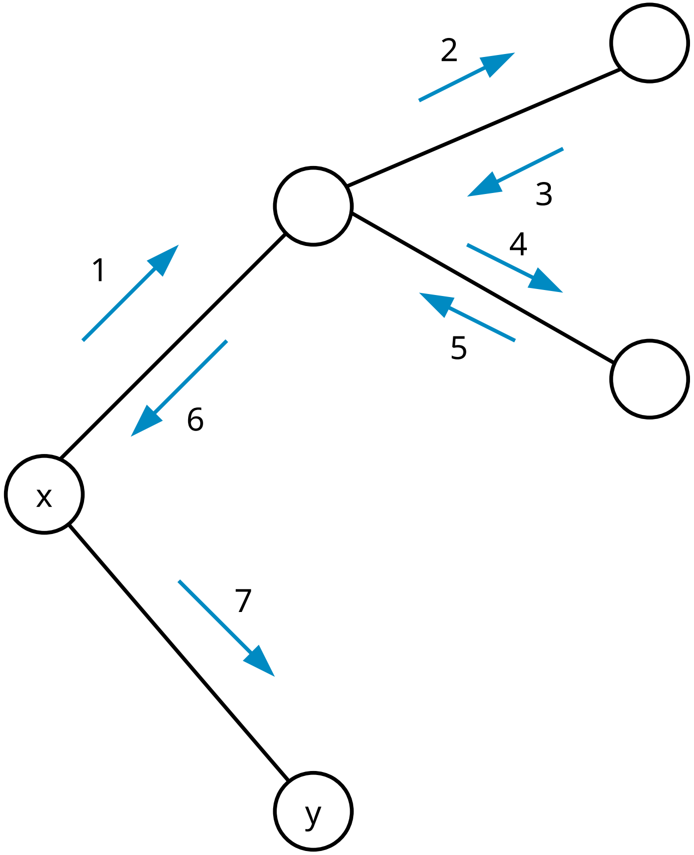
Routing to a neighbor with stretch <2k+1 in a graph with girth 2k+2:



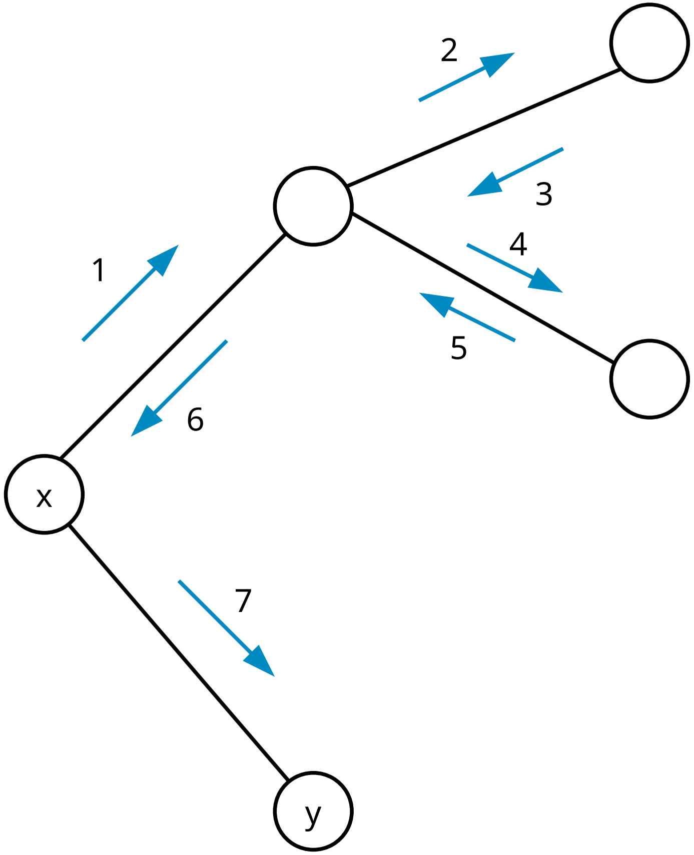
- Choose a graph with n nodes, m edges and girth 2k+2
- Consider all possible port assignments



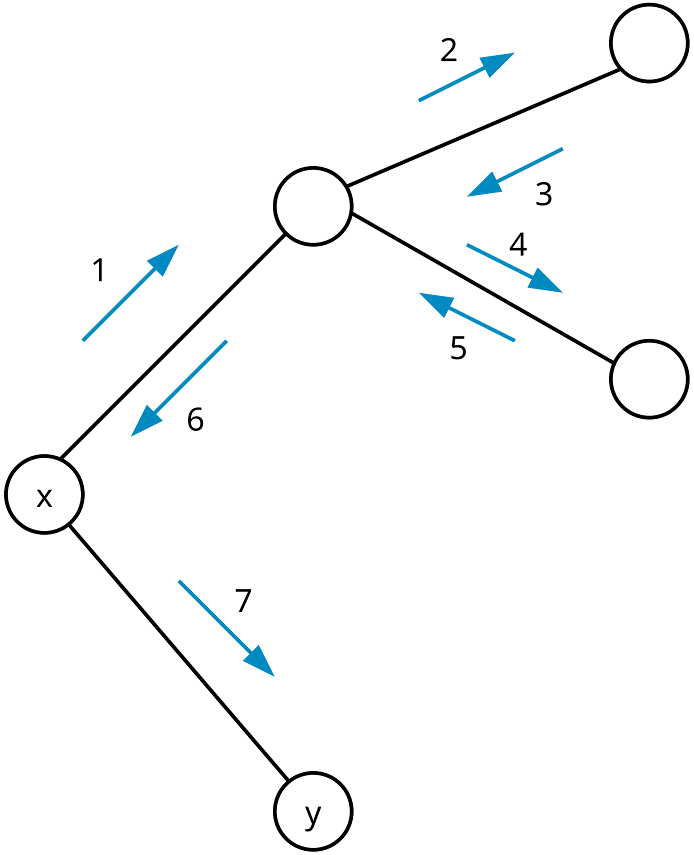
- Choose a graph with n nodes, m edges and girth 2k+2
- Consider all possible port assignments
- Let m_i be the degree of vertex i
- There are $\Pi m_i!$ different port assignments
- $\Pi m_i! = 2^{\Omega(m \log(m/n))}$



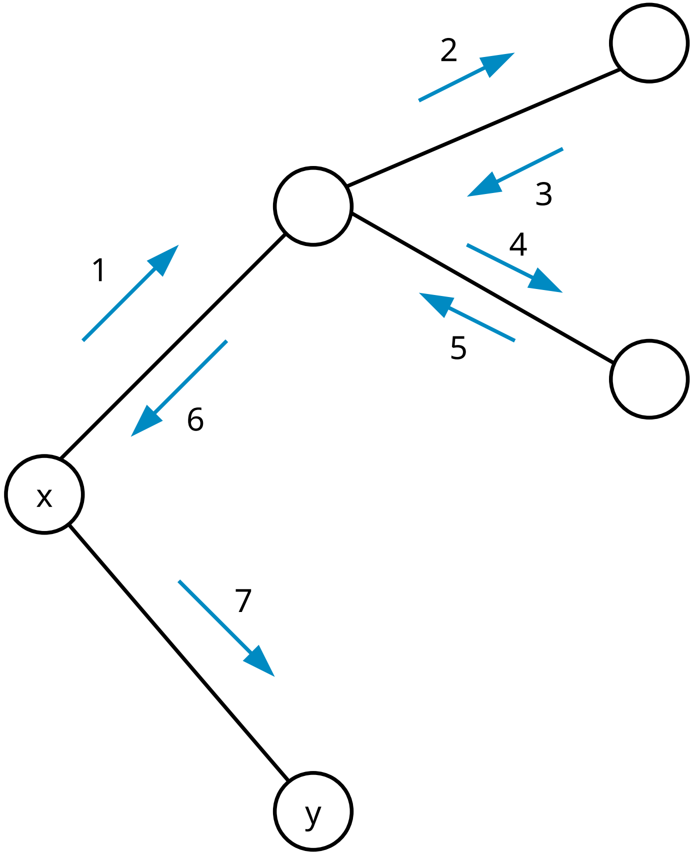
- Choose a graph with n nodes, m edges and girth 2k+2
- Consider all possible port assignments
- Let m_i be the degree of vertex i
- There are $\Pi m_i!$ different port assignments
- $\Pi m_i! = 2^{\Omega(m \log(m/n))}$
- Want to prove that a single routing scheme cannot support many port assignments



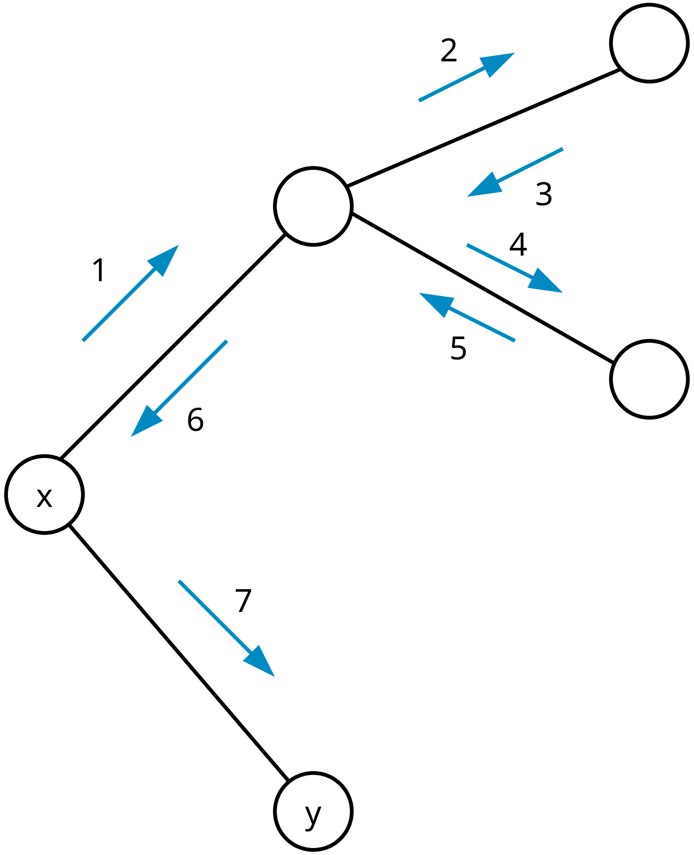
- Construct an extractor program that given a routing scheme, node labels and an advice string returns the port assignment
- If the advice string is short, intuitively the routing scheme must contain much information



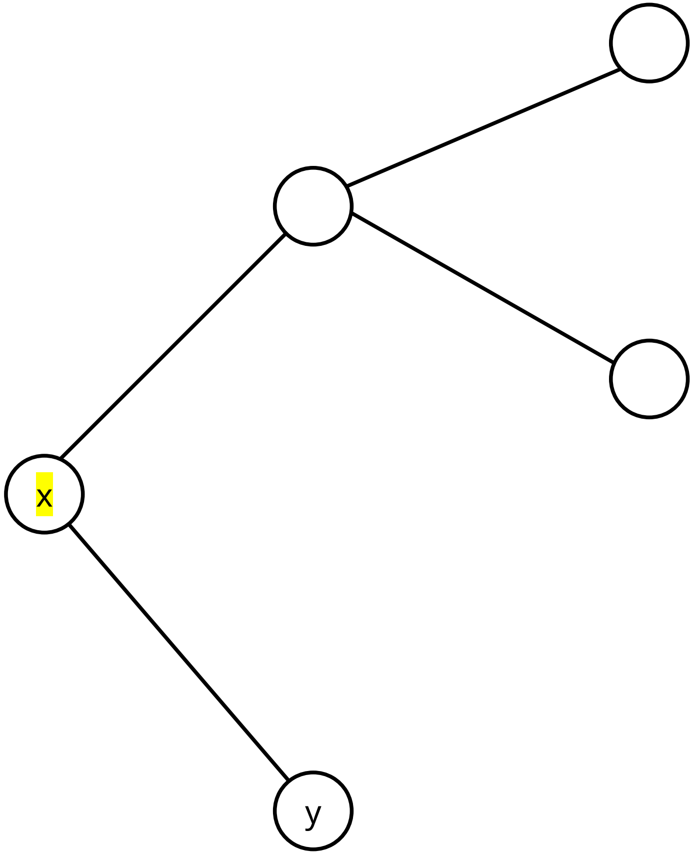
- Construct an extractor program that given a routing scheme, node labels and an advice string returns the port assignment
- If the advice string is short, intuitively the routing scheme must contain much information
- More formally:
  - fix routing scheme and node labels
  - then extractor maps advice string to port assignment
  - ∀ supported port assignments ∃ advice string
  - all advice strings are short ⇒ not many supported port assignments



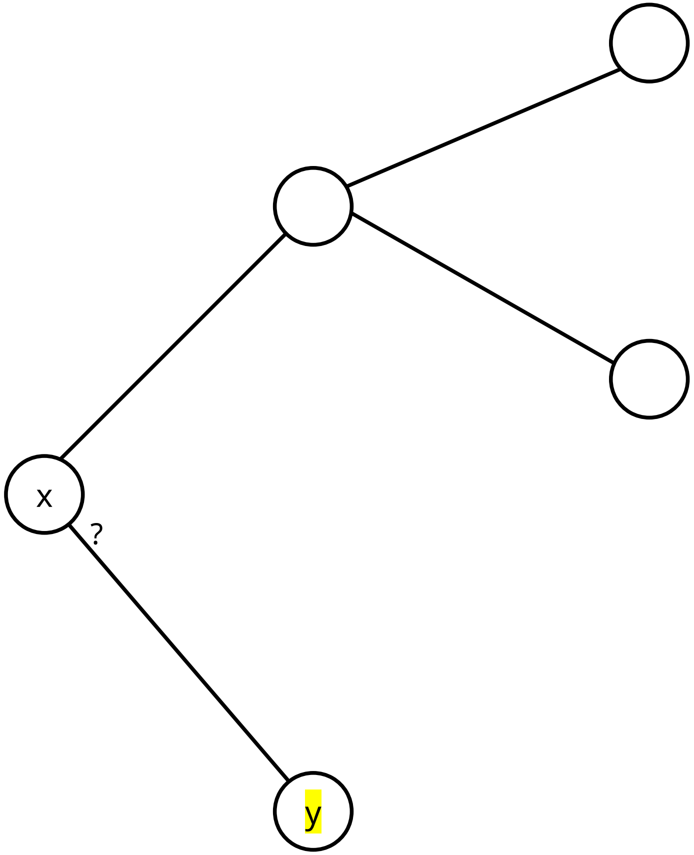
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← <u>read</u> the number of hops
  - for r times:
    - invoke routing program at x
    - <u>read</u> the next hop if unknown
    - let z be the next hop
    - <u>read</u> the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



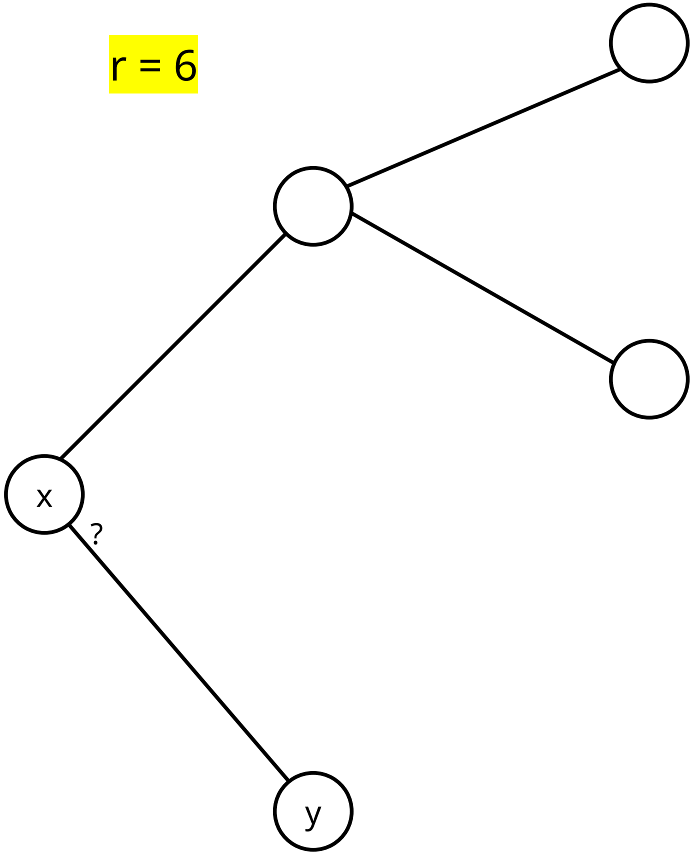
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source's port towards destination „for free“



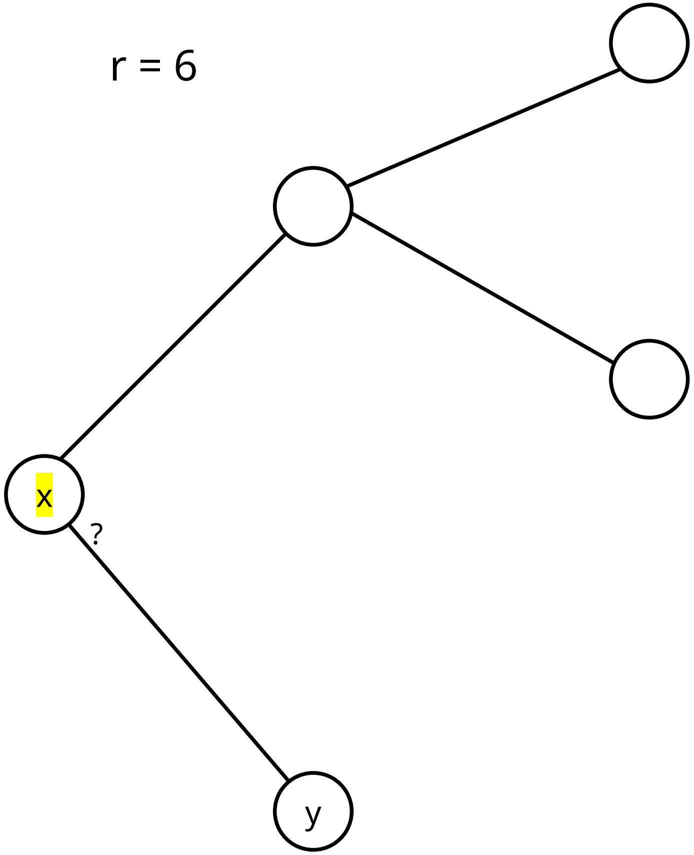
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source's port towards destination „for free“



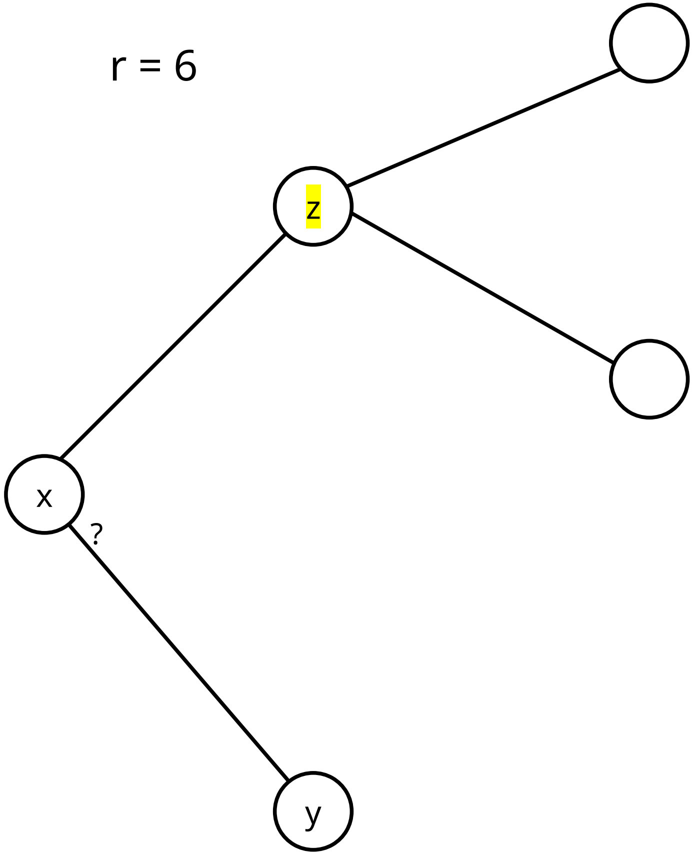
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



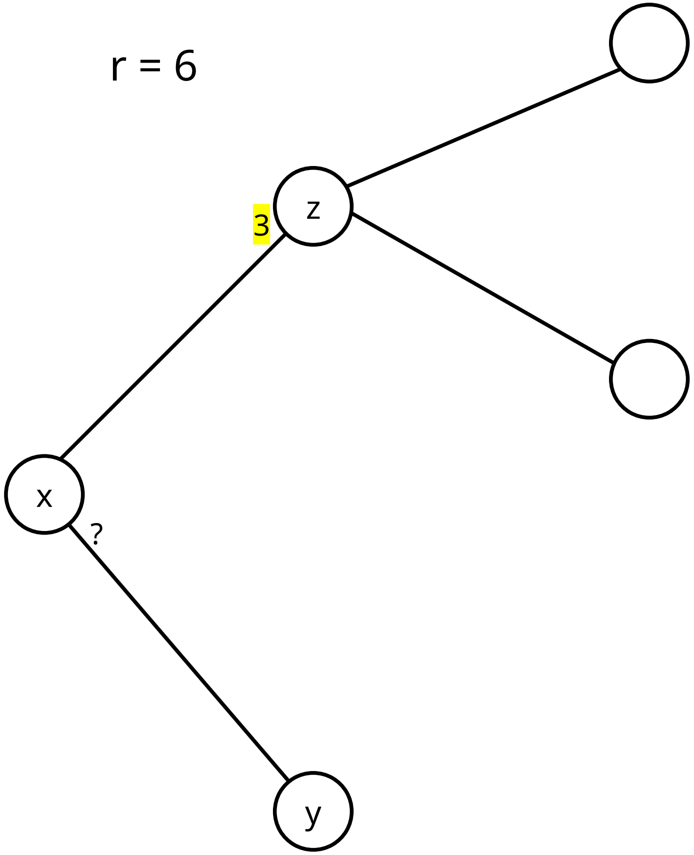
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



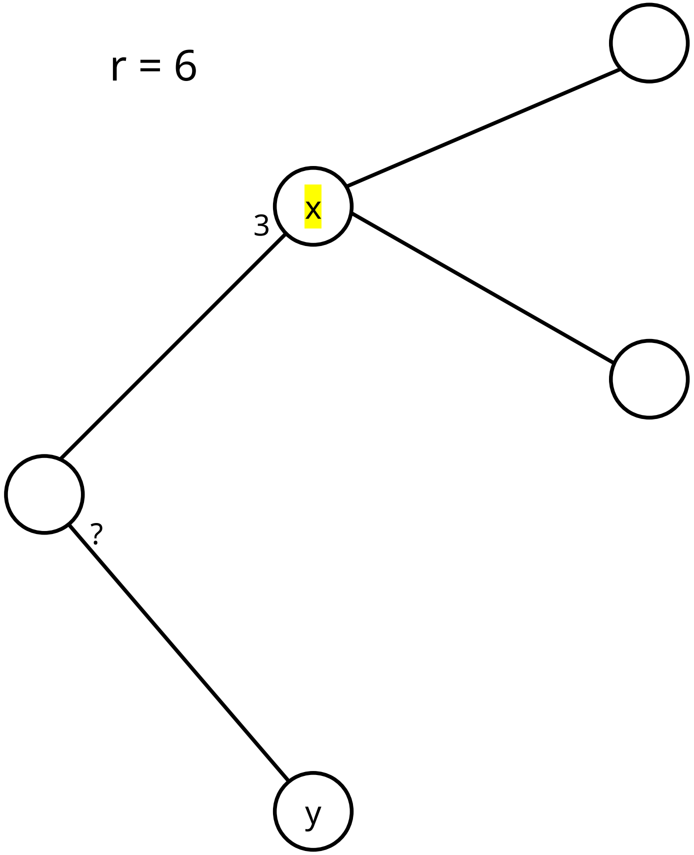
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



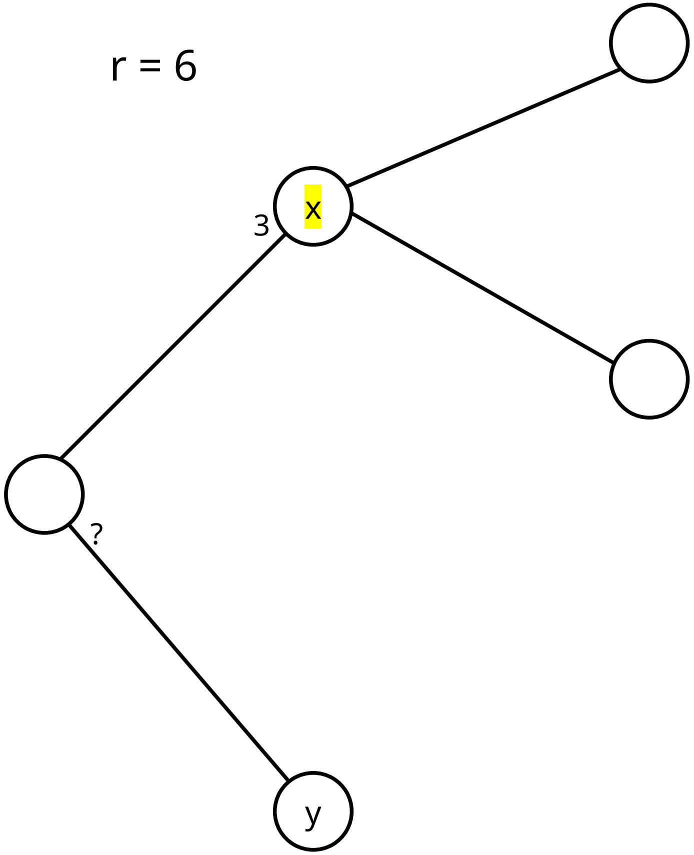
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source's port towards destination „for free“



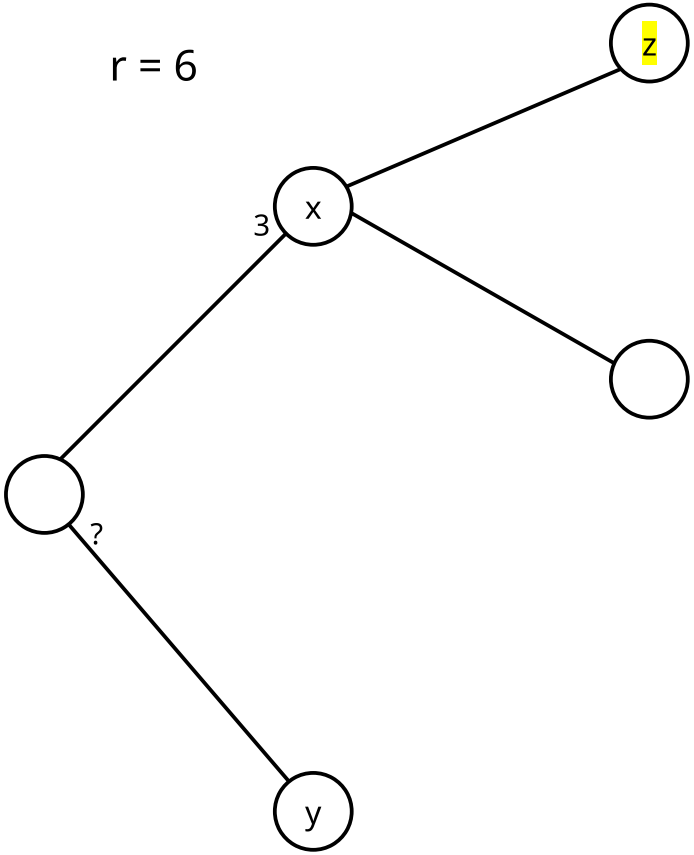
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source's port towards destination „for free"



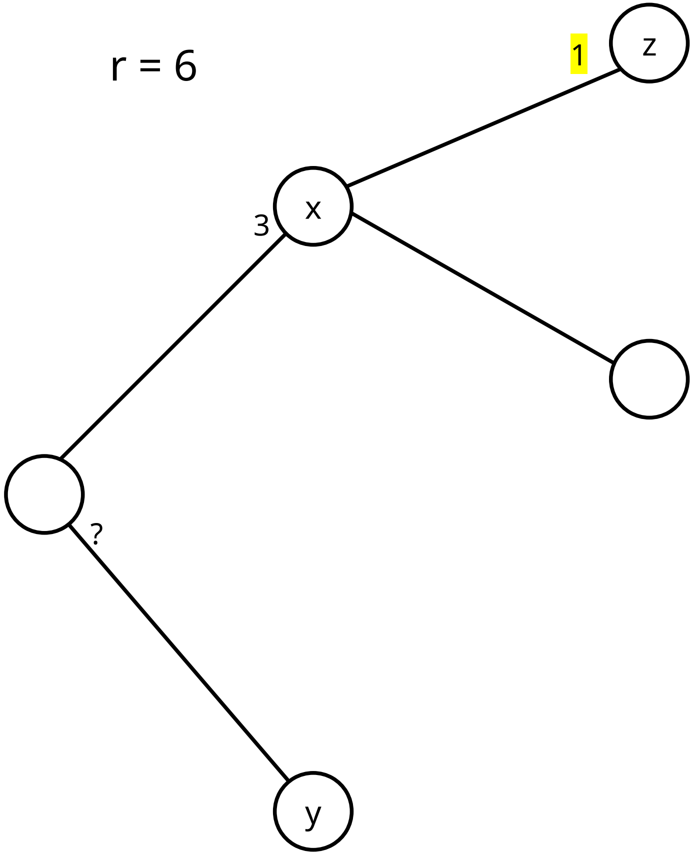
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← <u>read</u> the number of hops
  - for r times:
    - invoke routing program at x
    - <u>read</u> the next hop if unknown
    - let z be the next hop
    - <u>read</u> the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



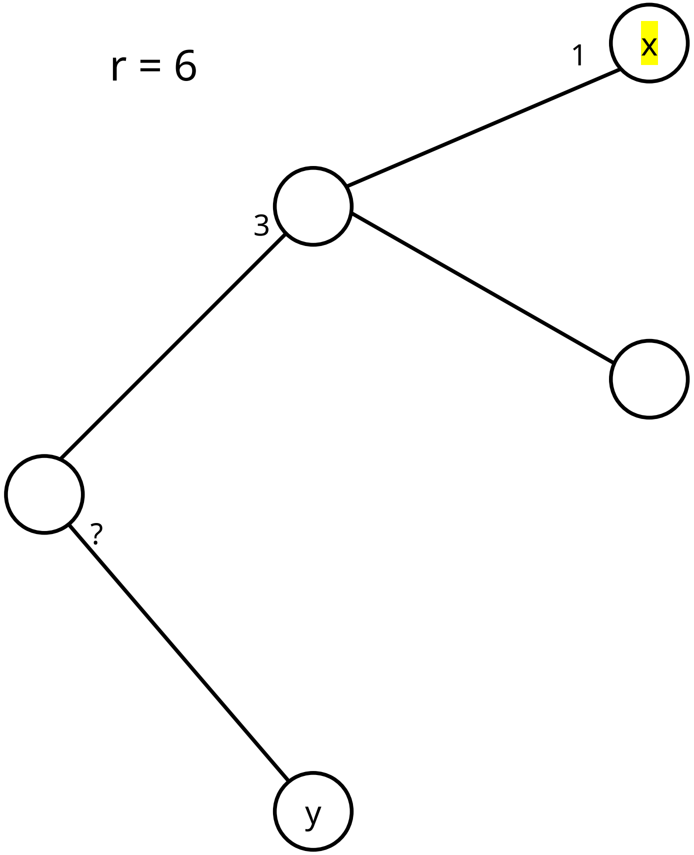
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← <u>read</u> the number of hops
  - for r times:
    - invoke routing program at x
    - <u>read</u> the next hop if unknown
    - let z be the next hop
    - <u>read</u> the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



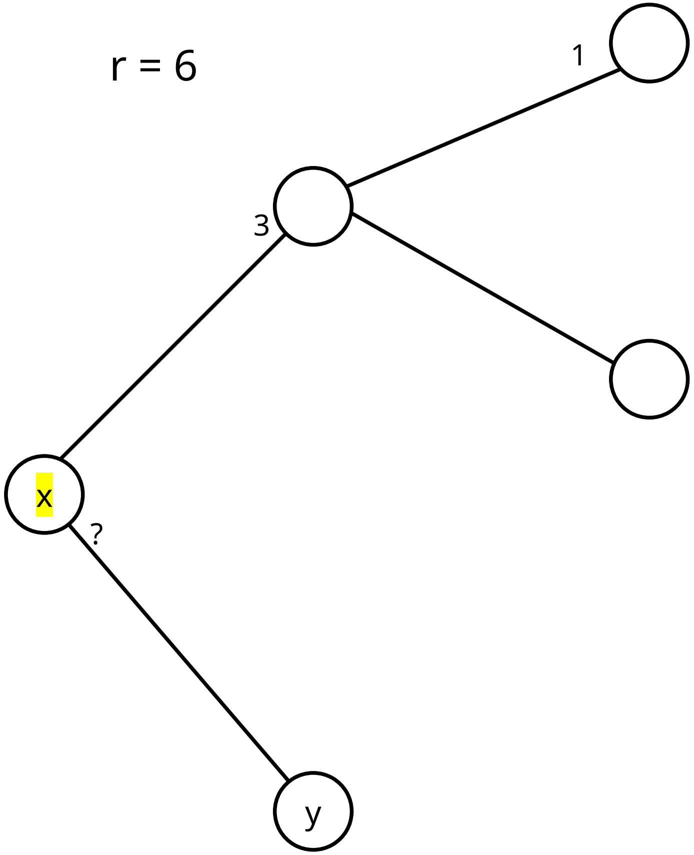
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← read the number of hops
  - for r times:
    - invoke routing program at x
    - read the next hop if unknown
    - let z be the next hop
    - read the incoming port at the next hop if unknown
    - x ← z
  - learn source's port towards destination „for free“



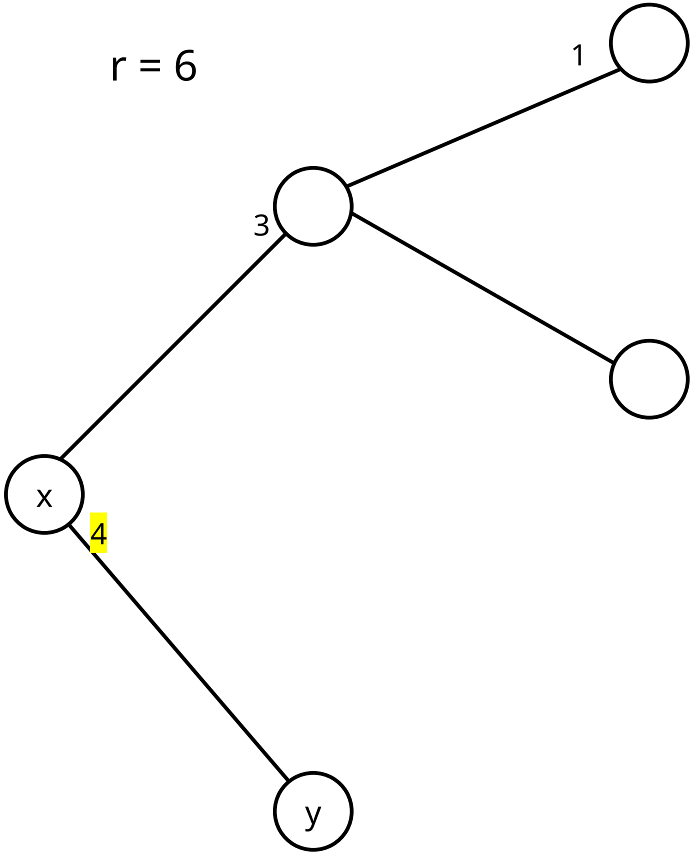
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← <u>read</u> the number of hops
  - for r times:
    - invoke routing program at x
    - <u>read</u> the next hop if unknown
    - let z be the next hop
    - <u>read</u> the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



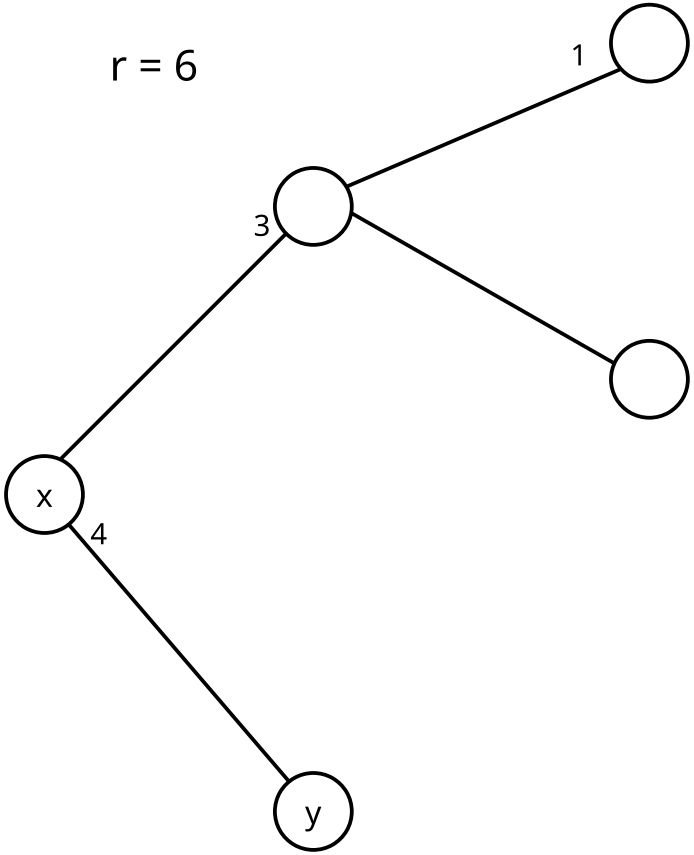
- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← <u>read</u> the number of hops
  - for r times:
    - invoke routing program at x
    - <u>read</u> the next hop if unknown
    - let z be the next hop
    - <u>read</u> the incoming port at the next hop if unknown
    - x ← z
  - learn source’s port towards destination „for free“



- Initially port assignment is completely unknown
- while some node has incomplete port assignment
  - let x be a node with incomplete port assignment with the largest degree
  - let y be a neighbor with unknown outgoing port
  - r ← <u>read</u> the number of hops
  - for r times:
    - invoke routing program at x
    - <u>read</u> the next hop if unknown
    - let z be the next hop
    - <u>read</u> the incoming port at the next hop if unknown
    - x ← z
  - learn source's port towards destination „for free"



- Bit string has length B
- $B \lesssim (1-1/(4k)) \Sigma m_i \log m_i$
- Compare to the „bit-size" of a port assignment:
  $\approx \Sigma m_i \log m_i$
- The difference is $\Omega(\Sigma m_i \log m_i) = \Omega(m \log(m/n))$

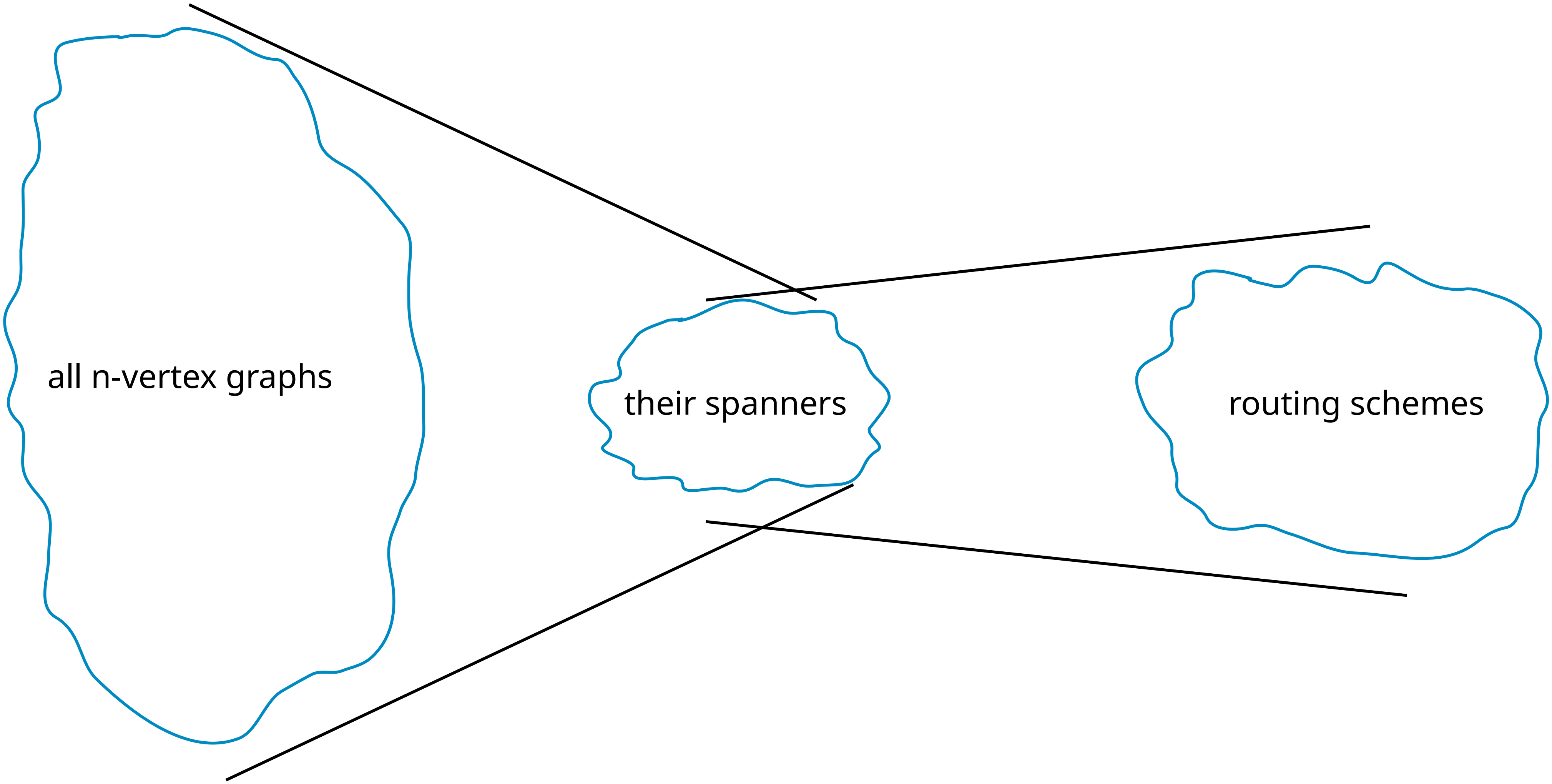


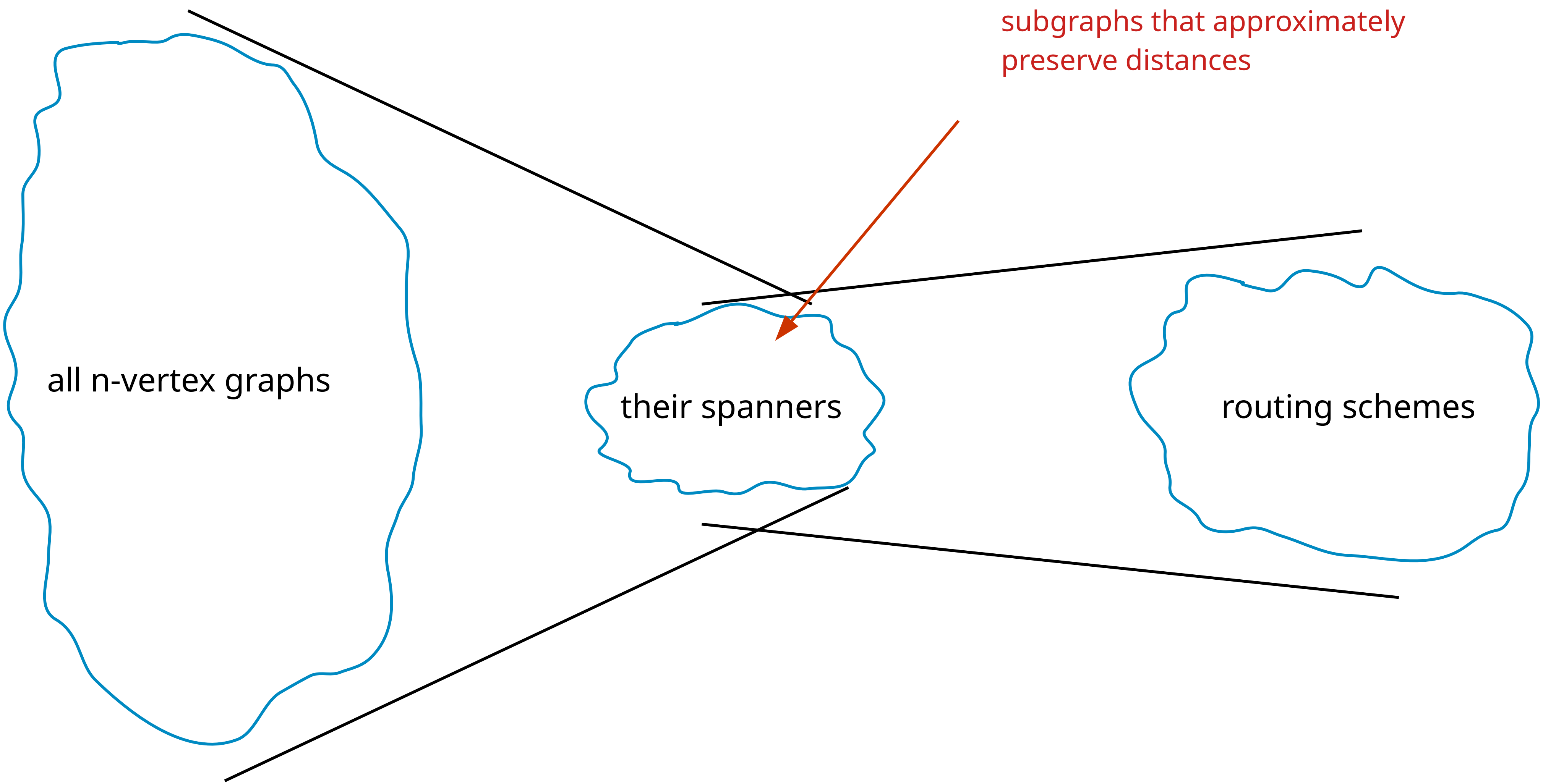
Girth conjecture requirement

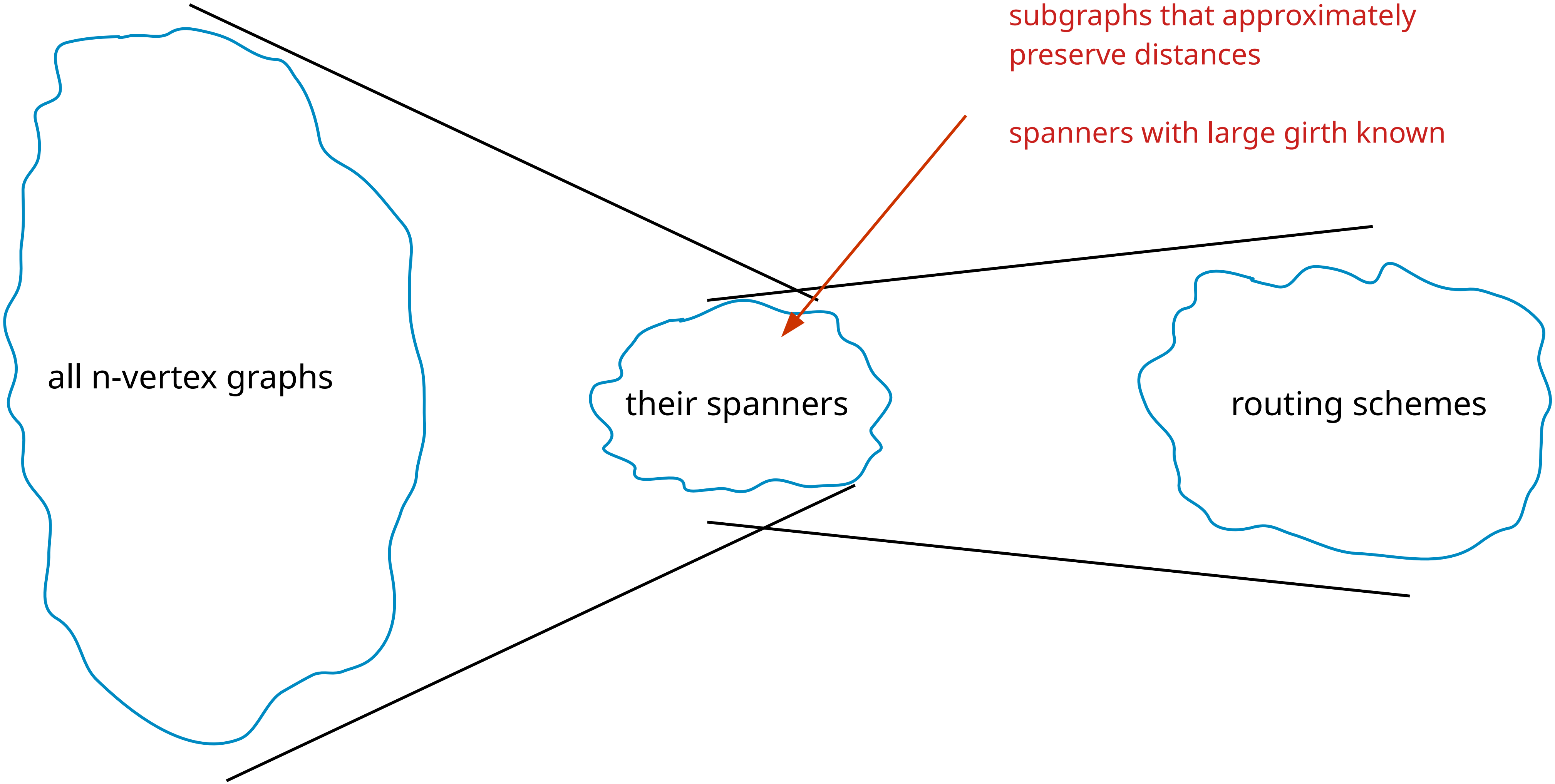
- All known approaches: prove that many routing schemes are necessary

Girth conjecture requirement

- All known approaches: prove that many routing schemes are necessary
- If $2^{\Omega(n^{(1+1/k)} \log n)}$ routing schemes are needed to satisfy all n-vertex graphs with stretch $< 2k+1$
  then there exists a graph with $\Omega(n^{1+1/k})$ edges and girth $2k+2$


all n-vertex graphs
their spanners
routing schemes


subgraphs that approximately preserve distances
all n-vertex graphs
their spanners
routing schemes


subgraphs that approximately
preserve distances
spanners with large girth known
all n-vertex graphs
their spanners
routing schemes

Conclusion

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	< 5/3	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	< 2k+1	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	< 2k+1	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	< 2k+1	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

Adversarial ports and labels

Conclusion

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	< 5/3	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	< 2k+1	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	< 2k+1	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	< 2k+1	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

Adversarial ports and labels

Conclusion

Work	Stretch	Local memory (bits)	Notes
Gavoille and Perennes (1996)	$< 5/3$	$\Omega(n \log n)$ on $\Omega(n)$ nodes	Node labels are [n]
Buhrman et al. (1996)	1	$\Omega(n)$ on $\Omega(n)$ nodes	
Gavoille and Gengler (2001)	< 3	$\Omega(n)$ on some node	complex proof
This paper	< 3	$\Omega(n)$ on cn nodes, $\forall 0<c<1$	

Non-adversarial (+ adversarial) ports

Work	Stretch	Local memory (bits)	Notes
Peleg and Upfal (1989)	$s \geq 1$	$\Omega(n^{1/(s+2)})$ on some node	
Thorup and Zwick (2001)	$< 2k+1$	$\Omega(n^{1/k})$ on some node	Works in a different model; relies on conjecture
This paper	$< 2k+1$	$\Omega(n^{1/k} \log n)$ on some node	relies on conjecture

Adversarial ports

Work	Stretch	Local memory (bits)	Notes
Abraham et al. (2006)	$< 2k+1$	$\Omega((n \log n)^{1/k})$ on some node	requires weighted graphs

Adversarial ports and labels

- Many routing schemes needed ⇒ dense graphs with large girth
- Can we overcome girth conjecture with a different approach?