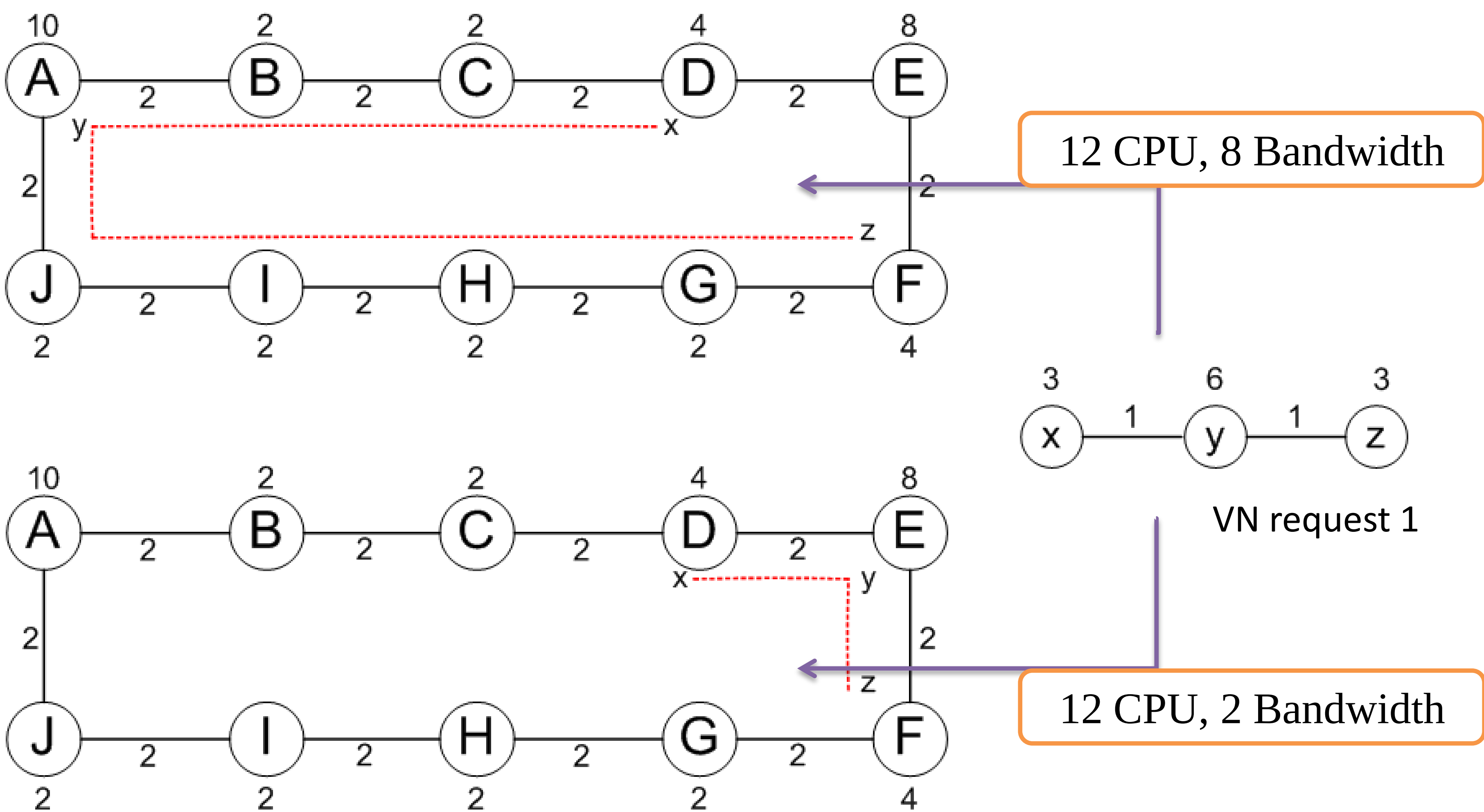
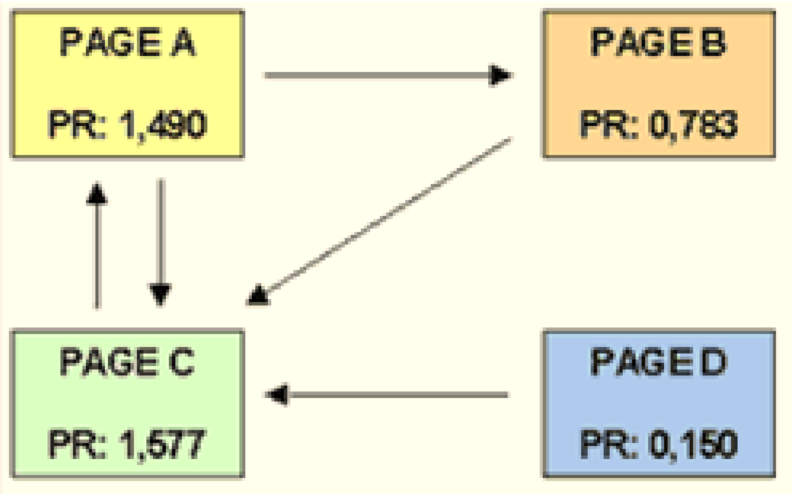
Topology-Aware Node Ranking
-Motivational Example



Topology-Aware Node Ranking
-Basic Idea



PageRank:
The importance of a web page is determined not only by its own contents but also its neighbors'

Observation:
The importance of a substrate node is determined not only by its own resource but also its neighbors'

Topology-Aware Node Ranking -Details

- A node has a higher rank if it has more highly-ranked neighbors
- Combined Node resources $H(u) = CPU(u) \sum_{l \in L(u)} BW(l),$
- Initial node rank $NR^{(0)}(u) = \frac{H(u)}{\sum_{v \in V} H(v)}$
- Jumping probability $p_{uv}^J = \frac{H(v)}{\sum_{w \in V} H(w)}$
- Forwarding probability $p_{uv}^F = \frac{H(v)}{\sum_{w \in nbr_1(u)} H(w)}$

$$NR^{(t+1)}(v) = \sum_{u \in V} p_{uv}^J \cdot p_u^J \cdot NR^{(t)}(u) + \sum_{u \in nbr_1(v)} p_{uv}^F \cdot p_u^F \cdot NR^{(t)}(u)$$

Topology-Aware Node Ranking -Details

- network of n nodes $V = \{v_1, v_2, \cdots, v_n\}$ $NR_i^{(t)} = NR^{(t)}(v_i)$

$$NR^{(t+1)} = \mathbf{T} \cdot NR^{(t)}$$

- where T is a one step transition matrix of the Markov chain

$$\mathbf{T} = \begin{pmatrix} p_{11}^J & p_{12}^J & \cdots & p_{1n}^J \\ p_{21}^J & p_{22}^J & \cdots & p_{2n}^J \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1}^J & p_{n2}^J & \cdots & p_{nn}^J \end{pmatrix} \cdot \begin{pmatrix} p_1^J & 0 & \cdots & 0 \\ 0 & p_2^J & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n^J \end{pmatrix} + \begin{pmatrix} 0 & p_{12}^F & \cdots & p_{1n}^F \\ p_{21}^F & 0 & \cdots & p_{2n}^F \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1}^F & p_{n2}^F & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} p_1^F & 0 & \cdots & 0 \\ 0 & p_2^F & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n^F \end{pmatrix}$$

Topology-Aware Node Ranking -Details

- network of n nodes $V = \{v_1, v_2, \cdots, v_n\}$ $NR_i^{(t)} = NR^{(t)}(v_i)$

$$NR^{(t+1)} = \mathbf{T} \cdot NR^{(t)}$$

- where T is a one step transition matrix of the Markov chain

Algorithm 1 The NodeRank Computing Method

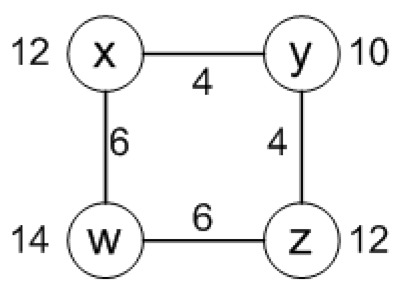
1: Given a positive value ϵ, $i \leftarrow 0$
2: **repeat**
3: $\quad NR^{(i+1)} \leftarrow \mathbf{T} \cdot NR^{(i)}$
4: $\quad \delta \leftarrow \|NR^{(i+1)} - NR^{(i)}\|$
5: $\quad i++$
6: **until** $\delta < \epsilon$

$$\mathbf{T} = \begin{pmatrix} p_{11}^J & p_{12}^J & \cdots & p_{1n}^J \\ p_{21}^J & p_{22}^J & \cdots & p_{2n}^J \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1}^J & p_{n2}^J & \cdots & p_{nn}^J \end{pmatrix} \cdot \begin{pmatrix} p_1^J & 0 & \cdots & 0 \\ 0 & p_2^J & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n^J \end{pmatrix} + \begin{pmatrix} 0 & p_{12}^F & \cdots & p_{1n}^F \\ p_{21}^F & 0 & \cdots & p_{2n}^F \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1}^F & p_{n2}^F & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} p_1^F & 0 & \cdots & 0 \\ 0 & p_2^F & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n^F \end{pmatrix}$$

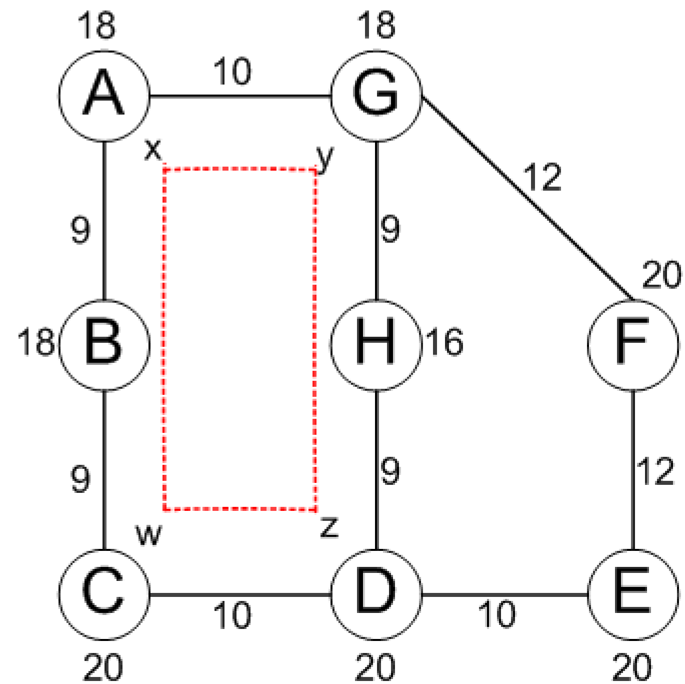
Mapping

- Phase 1: node-to-node
  - Sort VN nodes according to their CPU requirements
  - Sort SN nodes according to their *MCRank*
  - Maximum first matching
- Phase 2: link-to-link
  - shortest path
    - y-z: G-H-D ?

      G-F-E-D ?
  - k-shortest path
    - multiple edges



VN request 1



substrate network

Mapping
RW-MaxMatch

Algorithm 2 RW-MaxMatch Node Mapping Stage

Compute the NodeRank values of all nodes in both G_s and G_v using Algorithm 1 with a given value of ϵ.
Sort the nodes in G_s according to their NodeRank values in non-increasing order.
Sort the nodes in G_v according to their NodeRank values in non-increasing order.
Map virtual nodes to substrate nodes using the L2S2 mapping procedure.
if node resource constraints satisfied **then**
    return NODE_MAPPING_SUCCESS
else
    return NODE_MAPPING_FAILED
end if

Algorithm 3 RW-MaxMatch Link Mapping Stage

if path unsplittable **then**
    Map virtual links using the k-shortest path algorithm.
else
    Map virtual links using the multi-commodity flow algorithm.
end if
if link resource constraints satisfied **then**
    return LINK_MAPPING_SUCCESS
else
    return LINK_MAPPING_FAILED
end if

Mapping
RW-BFS

Algorithm 4 RW-BFS

Compute the NodeRank values of all nodes in both G_s and G_v using Algorithm 1 with a given value of ϵ.
Construct the breadth-first searching tree of G_v.
Sort all the nodes in G_v in non-increasing order in each level of the breadth-first tree according to their NodeRank values.
backtrack_count = 0
Select a positive integer Δ.
for each node N_v^i in G_v **do**
    Build the candidate substrate node list fo
    if Match(N_v^i)==1 **then**
        Match(N_v^{i+1})
    else
        if backtrack_count <= Δ **then**
            Match(N_v^{i-1})
            backtrack_count++
        else
            return BFS_FAILED
        end if
    end if
end for
return BFS_SUCCESS

Algorithm 5 The Details of Match Function

if N_v^i is the root **then**
    map N_v^i onto the substrate node with the largest NodeRank
    return MATCH_SUCCESS
end if
k = 1
while $k < Max\_Hop$ **do**
    for each N_s^j which satisfies the k-hop constraint from its mapped parent node in the candidate substrate node list of N_v^i **do**
        if pair(N_v^i, N_s^j) meets all the capacity constraints **then**
            return MATCH_SUCCESS
        end if
    end for
    k++
end while
return MATCH_FAILED