Building Regeneration Trees in Distributed Storage Systems with Asymmetric Links

Jun Li, Shuang Yang, Xin Wang
Fudan University, China
Speaker: Dongbo Huang

Oct. 9, 2010

Distributed storage

- **Distributed storage systems**
  - include a large number of storage nodes
  - storage nodes are subject to fail
  - maintain the data integrity by storing redundancy on multiple storage nodes

Distributed storage

- **Distributed storage systems**
  - include a large number of storage nodes
  - storage nodes are subject to fail
  - maintain the data integrity by storing redundancy on multiple storage nodes
- **Maintenance of redundancy**
  - redundancy is subject to loss due to node failures
  - regenerate the lost redundancy in a replacement node, called ***newcomer***

Distributed storage

- **Distributed storage systems**
  - include a large number of storage nodes
  - storage nodes are subject to fail
  - maintain the data integrity by storing redundancy on multiple storage nodes
- **Maintenance of redundancy**
  - redundancy is subject to loss due to node failures
  - regenerate the lost redundancy in a replacement node, called ***newcomer***
- **The efficiency of regeneration depends on how the redundancy is achieved.**

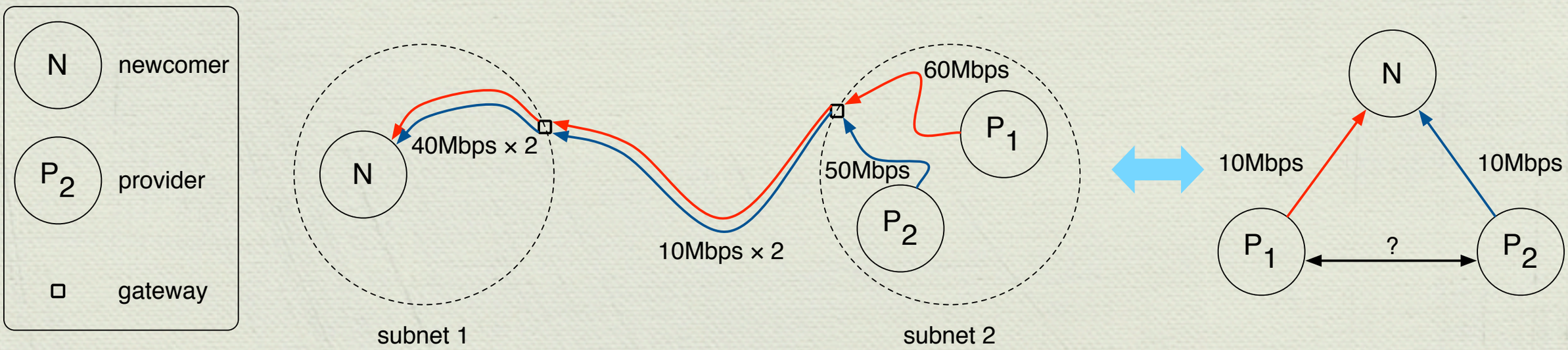
Network coding in distributed storage systems

- **Network coding can improve the data availability**
  - every bit can provide innovative information for data recovery
  - any k blocks among n blocks can recover the original data
    - $c_1B_1 + ... + c_kB_k \rightarrow$ storage node
    - deterministic linear network coding, random linear coding
  - do the job as well as erasure codes (Reed-Solomon, Fountain, etc.)

- **Network coding can improve the data availability**
  - every bit can provide innovative information for data recovery
  - any k blocks among n blocks can recover the original data
    - $c_1B_1 + ... + c_kB_k \rightarrow$ storage node
    - deterministic linear network coding, random linear coding
  - do the job as well as erasure codes (Reed-Solomon, Fountain, etc.)
- **Network coding can improve the efficiency of regeneration**
  - Mapping the newcomer into an intermediate node in a multicast network, network coding resides naturally in the regeneration process [Dimakis *et al.*, INFOCOM 2007]
  - Network coding makes it possible to exploit the network topology during the regeneration [Li *et al.*, INFOCOM 2010]

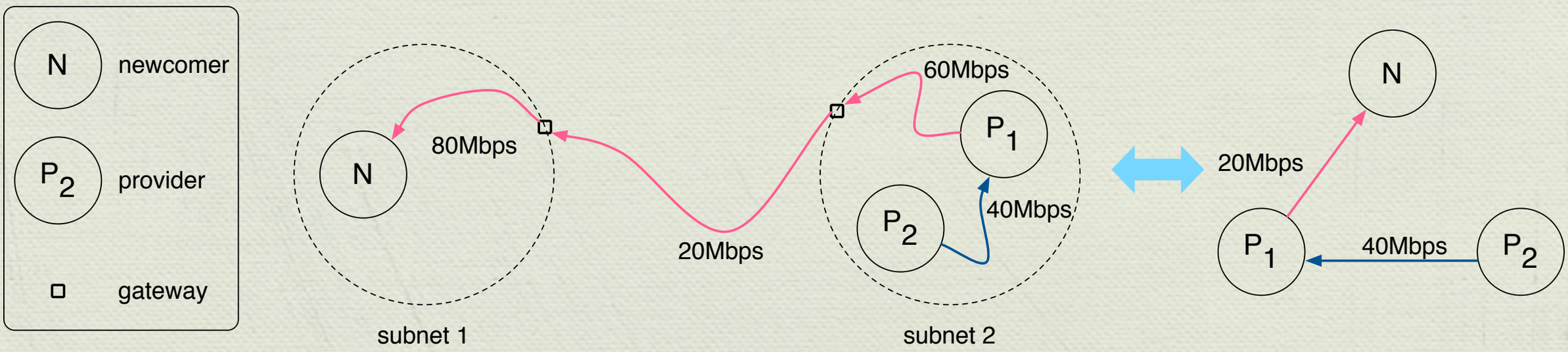
Exploit the network topology

- **Network topology has a significant effect on the available bandwidth between two nodes.**
  - bandwidth of a link inside a subnet is usually more available than that between two subnets
  - the farther away two nodes are, the less available the bandwidth between them are likely to be



Exploit the network topology

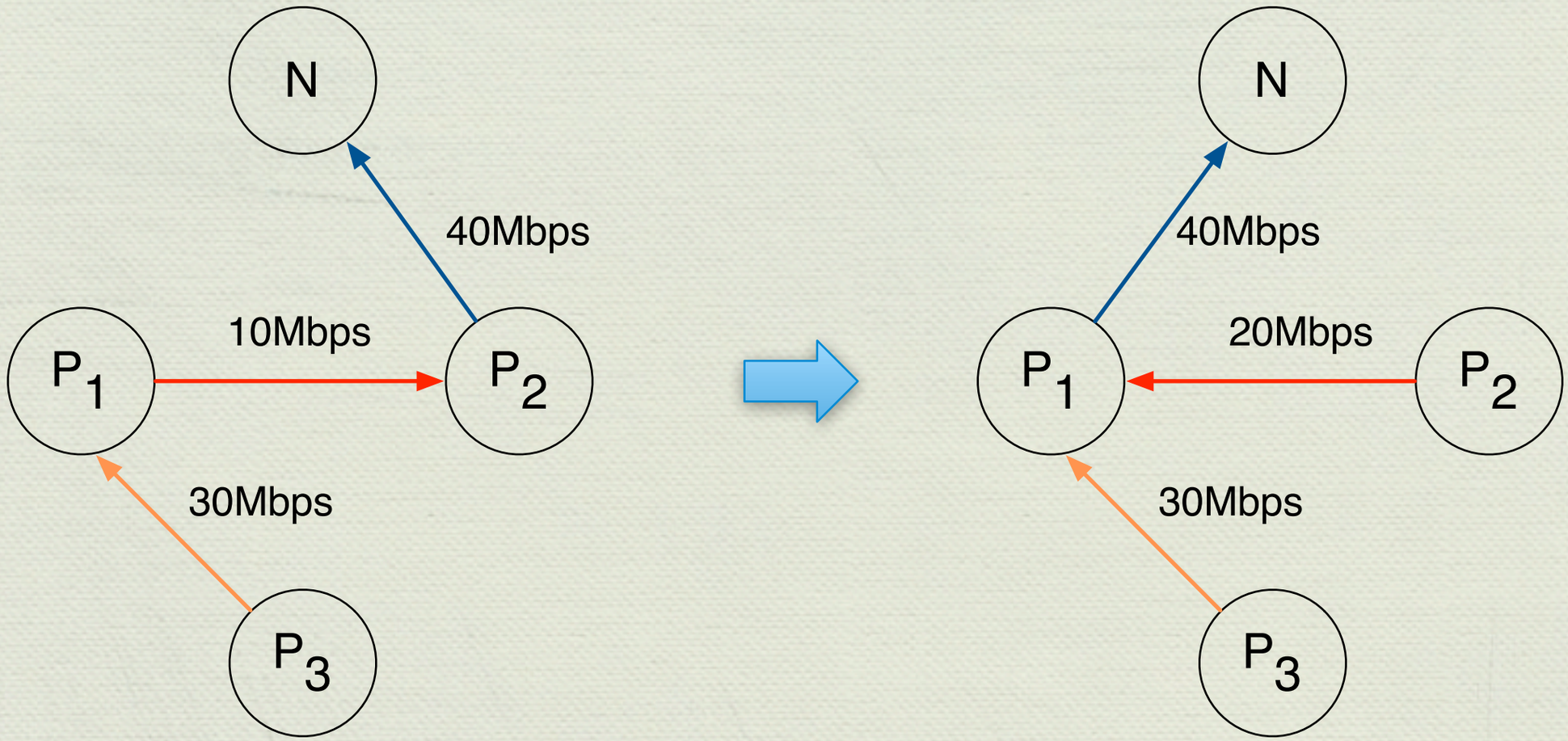
- **Network topology has a significant effect on the available bandwidth between two nodes.**
  - bandwidth of a link inside a subnet is usually more available than that between two subnets
  - the farther away two nodes are, the less available the bandwidth between them are likely to be
- **Bypass the bottleneck links**



However, we have some more observations

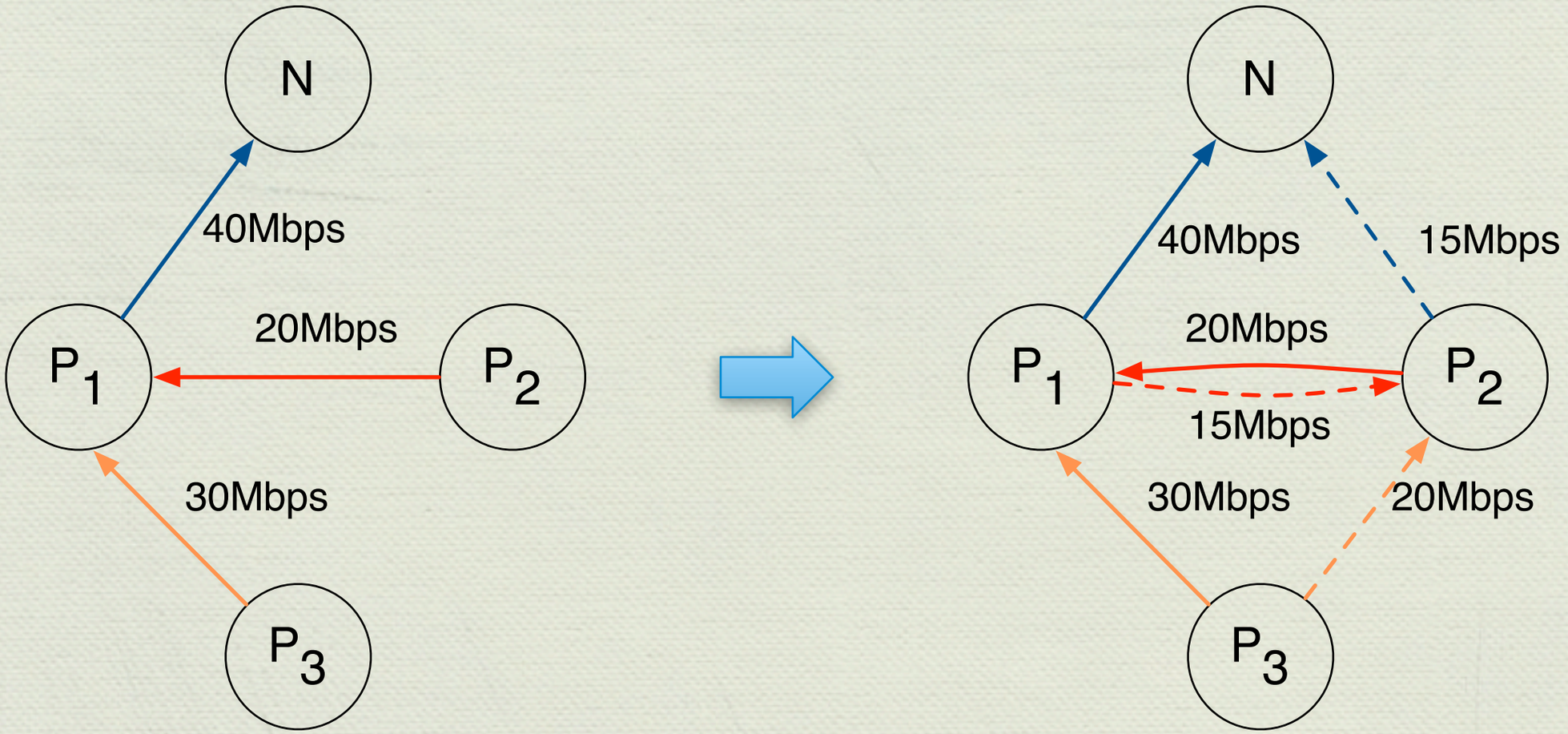
Link with asymmetric bandwidth in two directions

- more than 78% links in PlanetLab are asymmetric
- exploit the direction of the link where the bandwidth is more available



Parallel transmission

- most links are not exploited
- the bandwidth on most links are not fully utilized, due to the bottleneck link



Regeneration tree with asymmetric links

Network model

- storage nodes store blocks that are linear combinations of the source blocks, such that any k storage nodes suffice to recover the original file
- N: newcomer (the node which regenerate a lost block)
- $P_1, ..., P_k$: providers (the storage node which provide data to the newcomer)
- bandwidth on each link between two nodes are **asymmetrically** available

Network model

- storage nodes store blocks that are linear combinations of the source blocks, such that any k storage nodes suffice to recover the original file
- N: newcomer (the node which regenerate a lost block)
- $P_1, ..., P_k$: providers (the storage node which provide data to the newcomer)
- bandwidth on each link between two nodes are **asymmetrically** available

Regeneration tree

- N obtains a linear combination of all blocks of all providers
- a provider may encode and relay the data from other providers
- time spent during the regeneration depends on the end-to-end link with the least available bandwidth
- finding the optimal regeneration tree is equivalent to finding maximum bottleneck spanning tree in a directed graph

Finding the optimal regeneration tree

- Gabow and Tarjan's dichotomic search algorithm
- a spanning tree exists if and only if the graph is connected
- search the bottleneck edge dichotomically
- time: $O(k^2 \log k)$

Finding the optimal regeneration tree

- Gabow and Tarjan's dichotomic search algorithm
- a spanning tree exists if and only if the graph is connected
- search the bottleneck edge dichotomically
- time: $O(k^2 \log k)$

Algorithm 1 Splitting algorithm for the problem of the optimal regeneration tree in $G(k)$ [13]. Let $E(k) = \{e_1, e_2, \ldots, e_{k(k+1)}\}$, where $\omega(e_1) \geq \omega(e_2) \geq \ldots \geq \omega(e_{k(k+1)})$.

1: $\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |E(k)|$
2: **while** $\theta_1 \neq \theta_2$ **do**
3: $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
4: $E_1 \leftarrow \{e | \omega(e_m) \leq \omega(e), e \in E(k)\}$
5: **if** every node in $G(V(k), E_1)$ has a directed path towards V_0 **then**
6: $\theta_2 \leftarrow m$
7: **else**
8: $\theta_1 \leftarrow m + 1$
9: **end if**
10: **end while**
11: $\lambda^* \leftarrow \omega(e_{\theta_1})$
12: return a spanning tree in $G(V(k), \{e | \omega(e) \geq \lambda^*\})$

Finding the optimal regeneration tree

- Gabow and Tarjan's dichotomic search algorithm
- a spanning tree exists if and only if the graph is connected
- search the bottleneck edge dichotomically
- time: $O(k^2 \log k)$

Algorithm 1 Splitting algorithm for the problem of the optimal regeneration tree in $G(k)$ [13]. Let $E(k) = \{e_1, e_2, \ldots, e_{k(k+1)}\}$, where $\omega(e_1) \geq \omega(e_2) \geq \ldots \geq \omega(e_{k(k+1)})$.

1: $\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |E(k)|$
2: **while** $\theta_1 \neq \theta_2$ **do**
3: $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
4: $E_1 \leftarrow \{e | \omega(e_m) < \omega(e), e \in E(k)\}$
5: **if** every node in $G(V(k), E_1)$ has a directed path towards V_0 **then**
6: $\theta_2 \leftarrow m$
7: **else**
8: $\theta_1 \leftarrow m + 1$
9: **end if**
10: **end while**
11: $\lambda^* \leftarrow \omega(e_{\theta_1})$
12: return a spanning tree in $G(V(k), \{e | \omega(e) \geq \lambda^*\})$

Finding the optimal regeneration tree

- Gabow and Tarjan's dichotomic search algorithm
- a spanning tree exists if and only if the graph is connected
- search the bottleneck edge dichotomically
- time: $O(k^2 \log k)$

Algorithm 1 Splitting algorithm for the problem of the optimal regeneration tree in $G(k)$ [13]. Let $E(k) = \{e_1, e_2, \ldots, e_{k(k+1)}\}$, where $\omega(e_1) \geq \omega(e_2) \geq \ldots \geq \omega(e_{k(k+1)})$.

1: $\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |E(k)|$
2: **while** $\theta_1 \neq \theta_2$ **do**
3: $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
4: $E_1 \leftarrow \{e | \omega(e_m) < \omega(e), e \in E(k)\}$
5: **if** every node in $G(V(k), E_1)$ has a directed path towards V_0 **then**
6: $\theta_2 \leftarrow m$
7: **else**
8: $\theta_1 \leftarrow m + 1$
9: **end if**
10: **end while**
11: $\lambda^* \leftarrow \omega(e_{\theta_1})$
12: return a spanning tree in $G(V(k), \{e | \omega(e) \geq \lambda^*\})$

Regeneration time

- pipelined transmission: *symbol-by-symbol, other than block-by-block*
- depends on the bottleneck link with the least available bandwidth
- regeneration time = block size / bottleneck bandwidth

Bottleneck link

- the bottleneck link *e* is the one with the least available bandwidth in the tree
- there are totally $(k+1)^2$ links available, and *e* may be the i^{th} maximal link among them with probability p(k+1, i)
- solve p(k+1, i) recursively (refer to our paper for details)

Bottleneck link

- the bottleneck link *e* is the one with the least available bandwidth in the tree
- there are totally $(k+1)^2$ links available, and *e* may be the i^{th} maximal link among them with probability p(k+1, i)
- solve p(k+1, i) recursively (refer to our paper for details)
- bandwidth distribution: U [0.3Mbps, 140Mbps] (from PlanetLab)
- expected available bandwidth of the i^{th} maximal link obtained by the theorem in order statistics

Case study: STAR vs. TREE

- STAR: conventional regeneration
- TREE: tree-structured regeneration

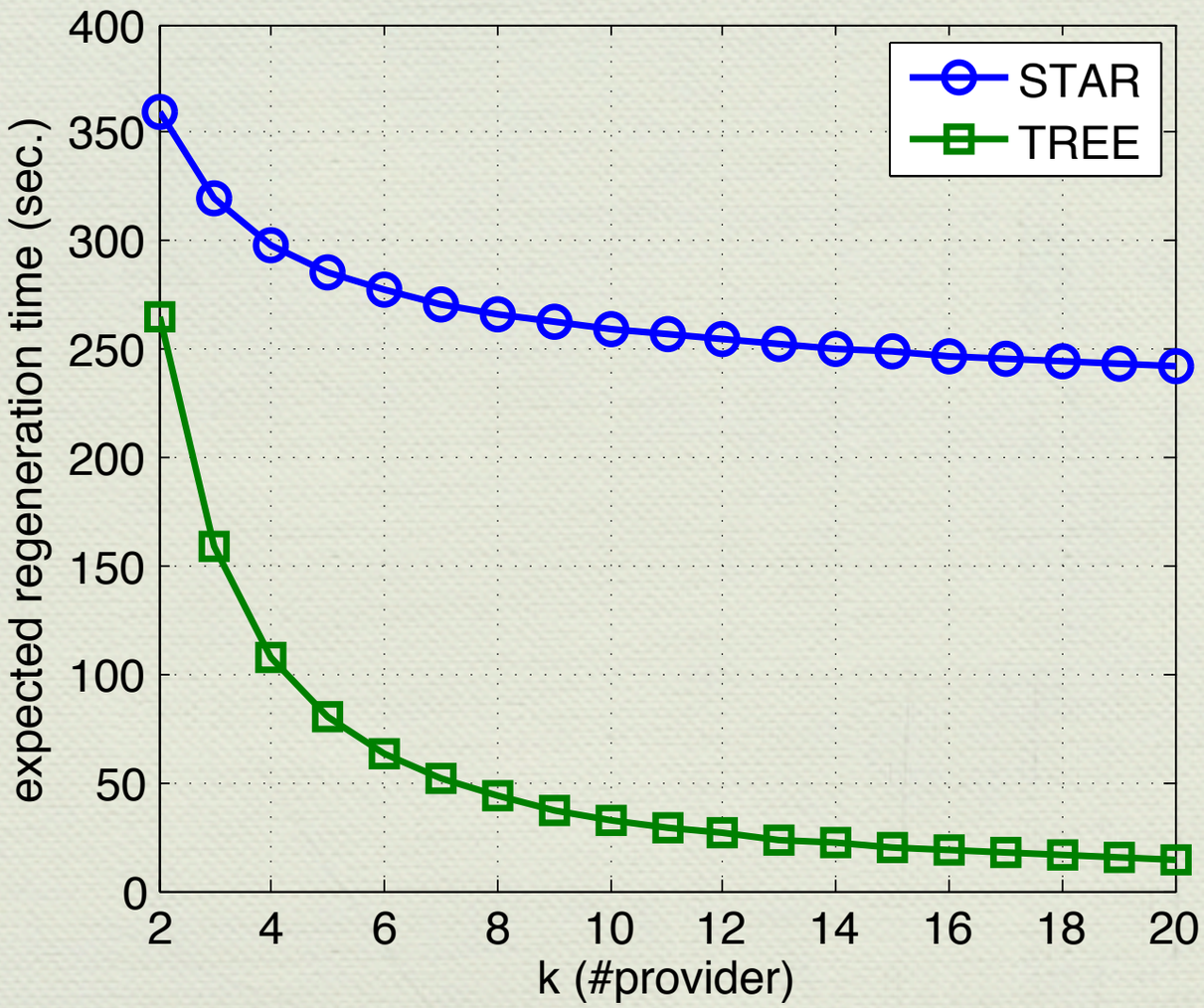
Bottleneck link

- the bottleneck link *e* is the one with the least available bandwidth in the tree
- there are totally $(k+1)^2$ links available, and *e* may be the i^{th} maximal link among them with probability p(k+1, i)
- solve p(k+1, i) recursively (refer to our paper for details)

Case study: STAR vs. TREE

- STAR: conventional regeneration
- TREE: tree-structured regeneration

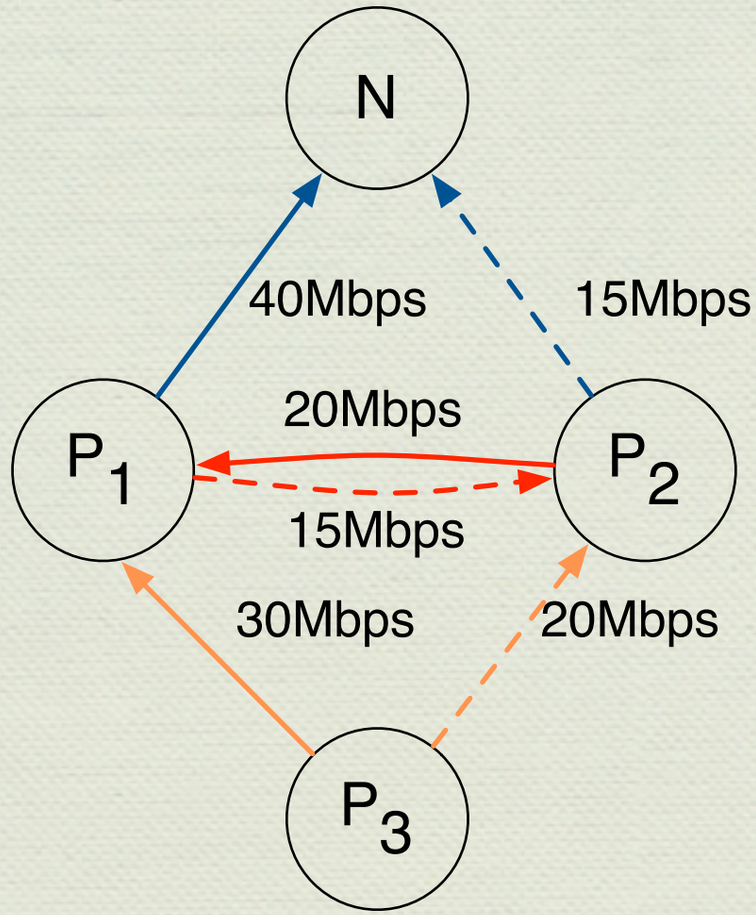
- bandwidth distribution: U [0.3Mbps, 140Mbps] (from PlanetLab)
- expected available bandwidth of the i^{th} maximal link obtained by the theorem in order statistics



Parallel regeneration trees

Edge-disjoint parallel regeneration trees

- **Build *t* edge-disjoint regeneration trees**
  - edges of one tree are disjoint with those of other trees
  - each tree works in parallel to regenerate 1/t block
  - available bandwidth $\omega_1, \omega_2, ..., \omega_t$
  - $\min\{\omega_1, \omega_2, ..., \omega_t\}$ is maximized



Edge-disjoint parallel regeneration trees

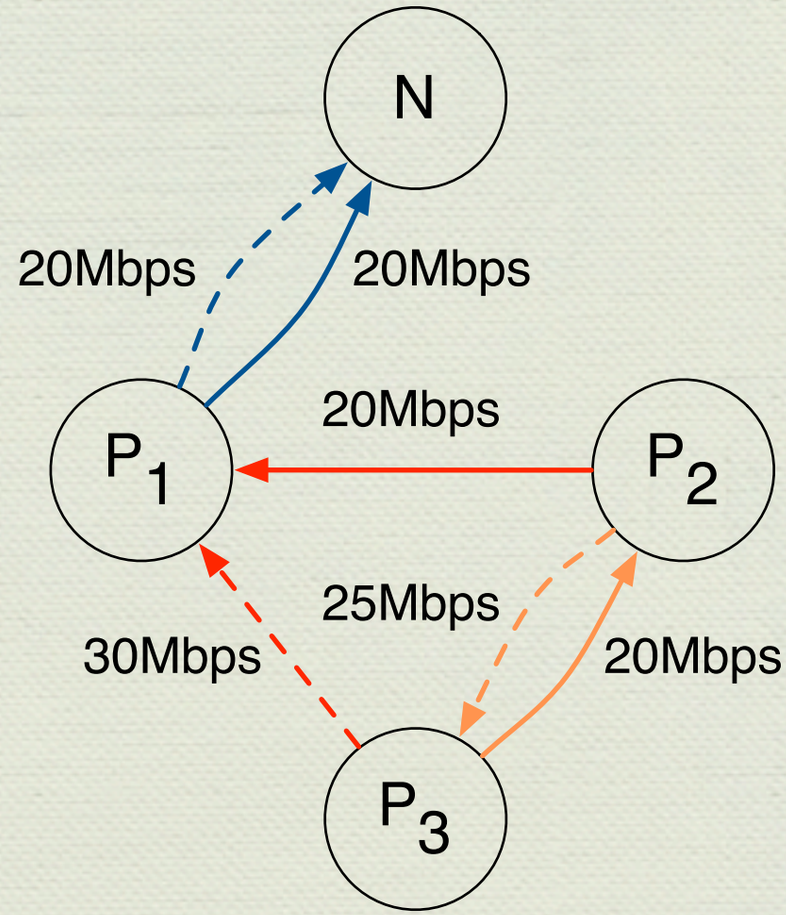
- **Splitting algorithm for finding *t* edge-disjoint regeneration trees with optimal bottleneck bandwidth**
  - search the bottleneck edge dichotomically
  - use Tarjan's algorithm to check whether there exist t edge-disjoint trees
  - use Gabow and Manu's algorithm to obtain the t edge-disjoint trees with the obtained bottleneck edge
  - in polynomial time

Algorithm 2 Splitting algorithm for the problem of finding t edge-disjoint parallel regeneration trees in $G(k)$.

1: $\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |E(k)|$
2: **while** $\theta_1 \neq \theta_2$ **do**
3: $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
4: $E_1 \leftarrow \{e | \omega(e_m) \leq \omega(e), e \in E(k)\}$
5: **if** t edge-disjoint parallel regeneration trees exist in $G(V(k), E_1)$ **then**
6: $\theta_2 \leftarrow m$
7: **else**
8: $\theta_1 \leftarrow m + 1$
9: **end if**
10: **end while**
11: $\lambda^* \leftarrow \omega(e_{\theta_1})$
12: return t regeneration trees in $G(V(k), \{e | \omega(e) \geq \lambda^*\})$

Edge-sharing parallel regeneration trees

- **Build *t* edge-sharing regeneration trees**
- multiple edges from different trees may share the same link to improve the link utilization
- each tree works in parallel to regenerate 1/t block
- available bandwidth $\omega_1, \omega_2, ..., \omega_t$
- $\min\{\omega_1, \omega_2, ..., \omega_t\}$ is maximized
- the sum of bottleneck bandwidths of the regeneration trees sharing one link can not exceed the available bandwidth of that link



Edge-sharing parallel regeneration trees

- **Splitting algorithm for finding *t* edge-sharing regeneration trees with optimal bottleneck bandwidth**
  - map the problem to that of finding t edge-disjoint regeneration trees
    - given a bottleneck bandwidth λ, we split the link with available bandwidth ω with ⌊ω/λ⌋ edges
  - search the bottleneck edge dichotomically as finding t edge-disjoint trees
  - in polynomial time

Algorithm 3 Splitting algorithm for the problem of finding t edge-sharing parallel regeneration trees in $G(k)$.

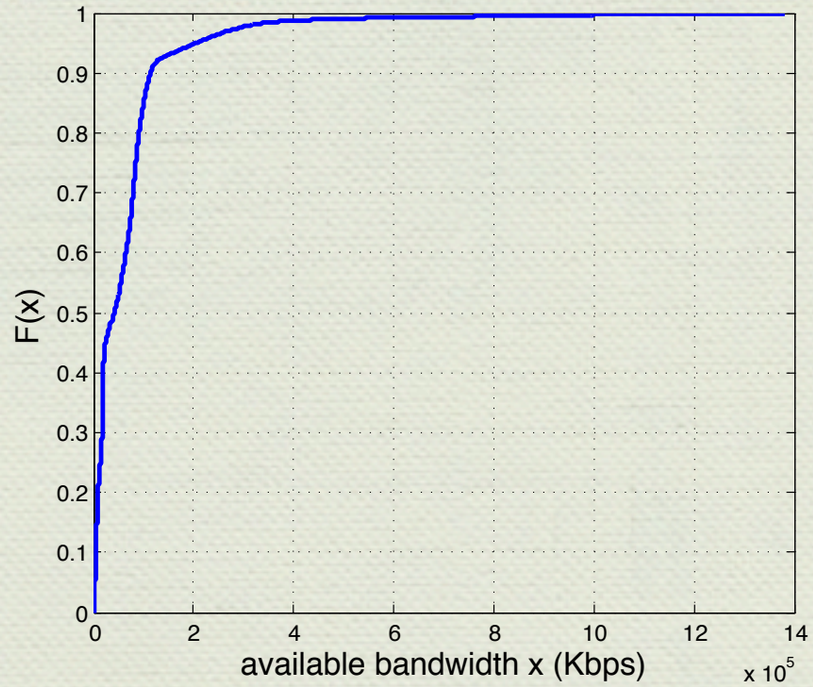
1: $\theta_1 \leftarrow 1$, $\theta_2 \leftarrow |W|$
2: **while** $\theta_1 \neq \theta_2$ **do**
3: $m \leftarrow \lfloor \frac{1}{2}(\theta_1 + \theta_2) \rfloor$
4: Let E_1 be a multiset of $S(w_m) = \{e \times n_{w_m}(e) | e \in E(k)\}$
5: **if** t edge-disjoint parallel regeneration trees exist in $G(V(k), E_1)$ **then**
6: $\theta_2 \leftarrow m$
7: **else**
8: $\theta_1 \leftarrow m + 1$
9: **end if**
10: **end while**
11: $\lambda^* \leftarrow w_{\theta_1}$
12: return t regeneration trees in $G(V(k), S(\lambda^*))$

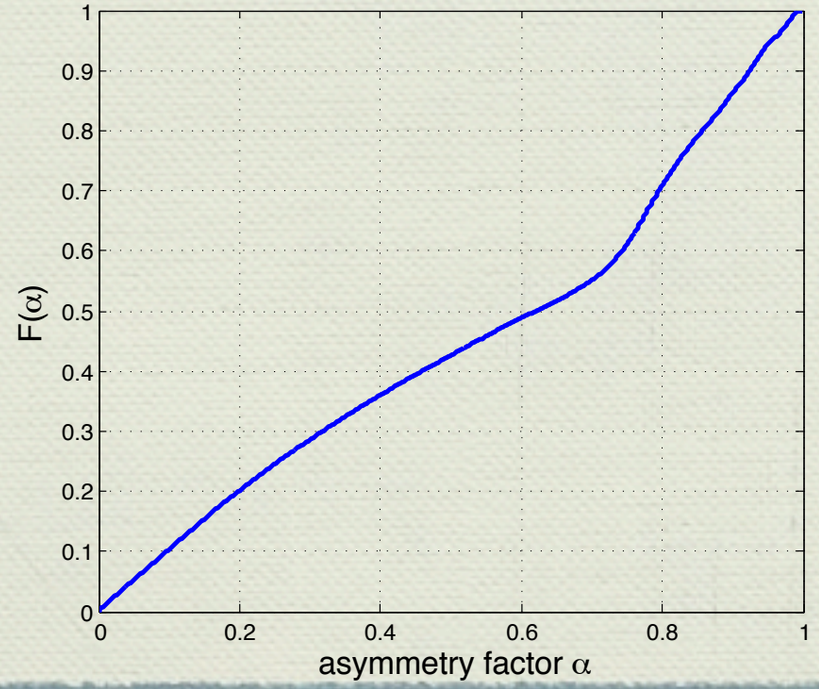
Simulation

Static network environment

- **Bandwidth measured in PlanetLab**
  - asymmetry factor:
    - on a link with available bandwidth BW_1 and BW_2 on two directions, respectively

$$\alpha = \frac{|\mathrm{BW}_1 - \mathrm{BW}_2|}{\max\{\mathrm{BW}_1, \mathrm{BW}_2\}}.$$



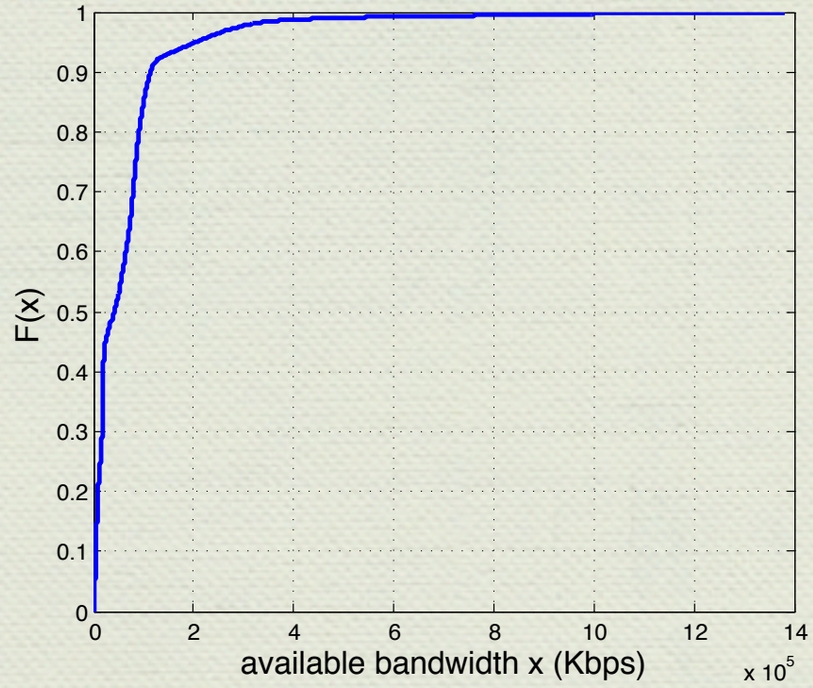


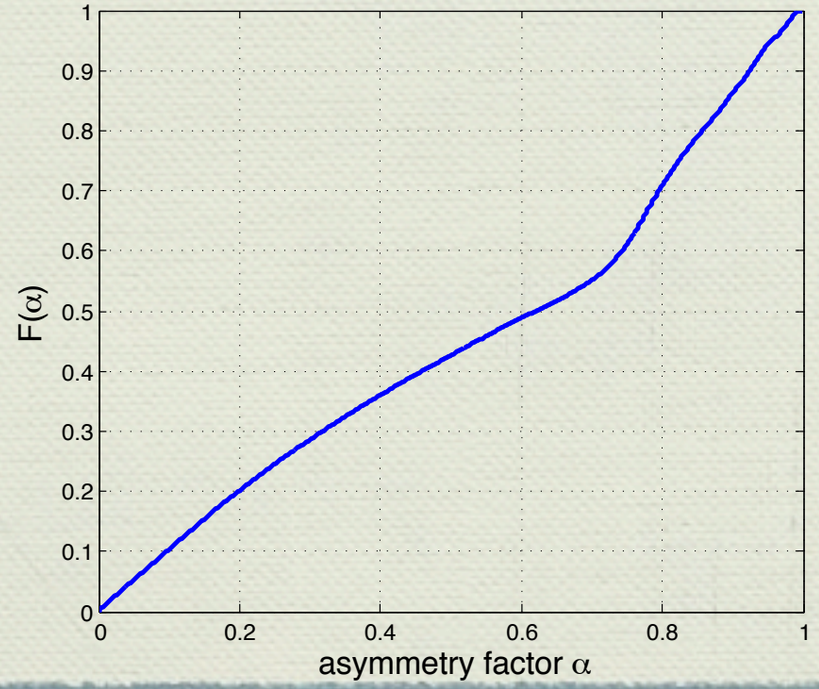
Static network environment

- **Bandwidth measured in PlanetLab**
  - asymmetry factor:
    - on a link with available bandwidth BW_1 and BW_2 on two directions, respectively

$$\alpha = \frac{|\mathrm{BW}_1 - \mathrm{BW}_2|}{\max\{\mathrm{BW}_1, \mathrm{BW}_2\}}.$$

  - assign the available bandwidth of each direction between any pair of two nodes during the regeneration



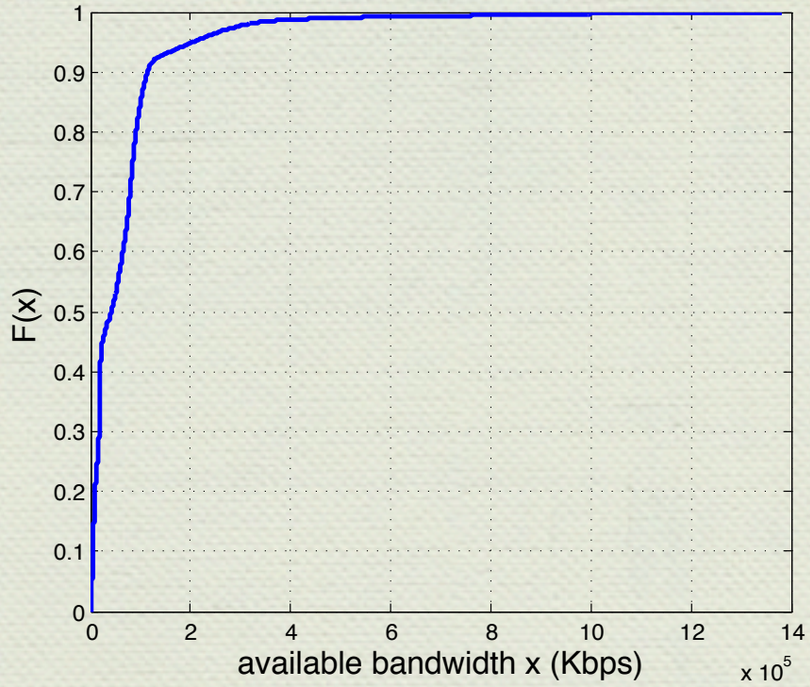


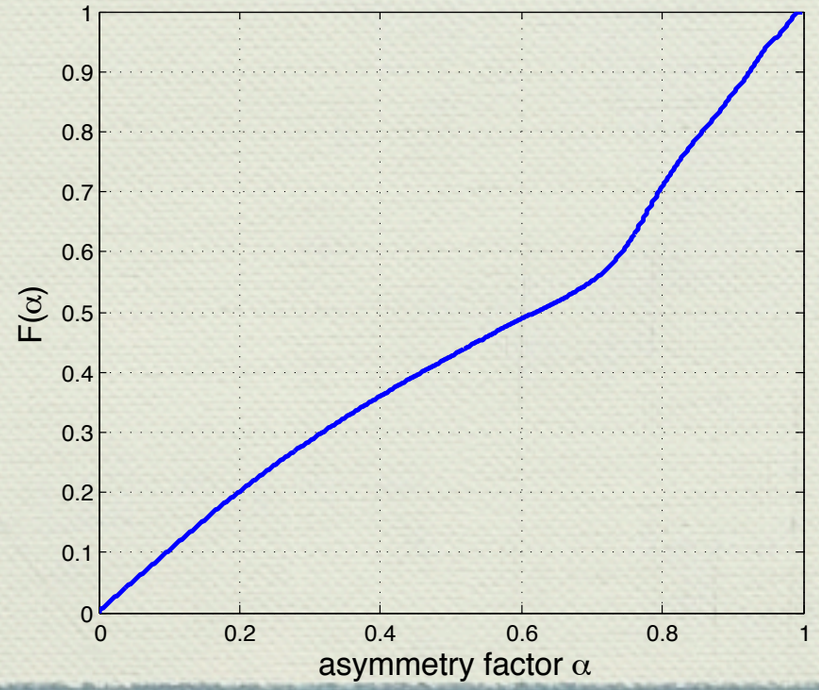
Static network environment

- **Bandwidth measured in PlanetLab**
  - asymmetry factor:
    - on a link with available bandwidth BW_1 and BW_2 on two directions, respectively

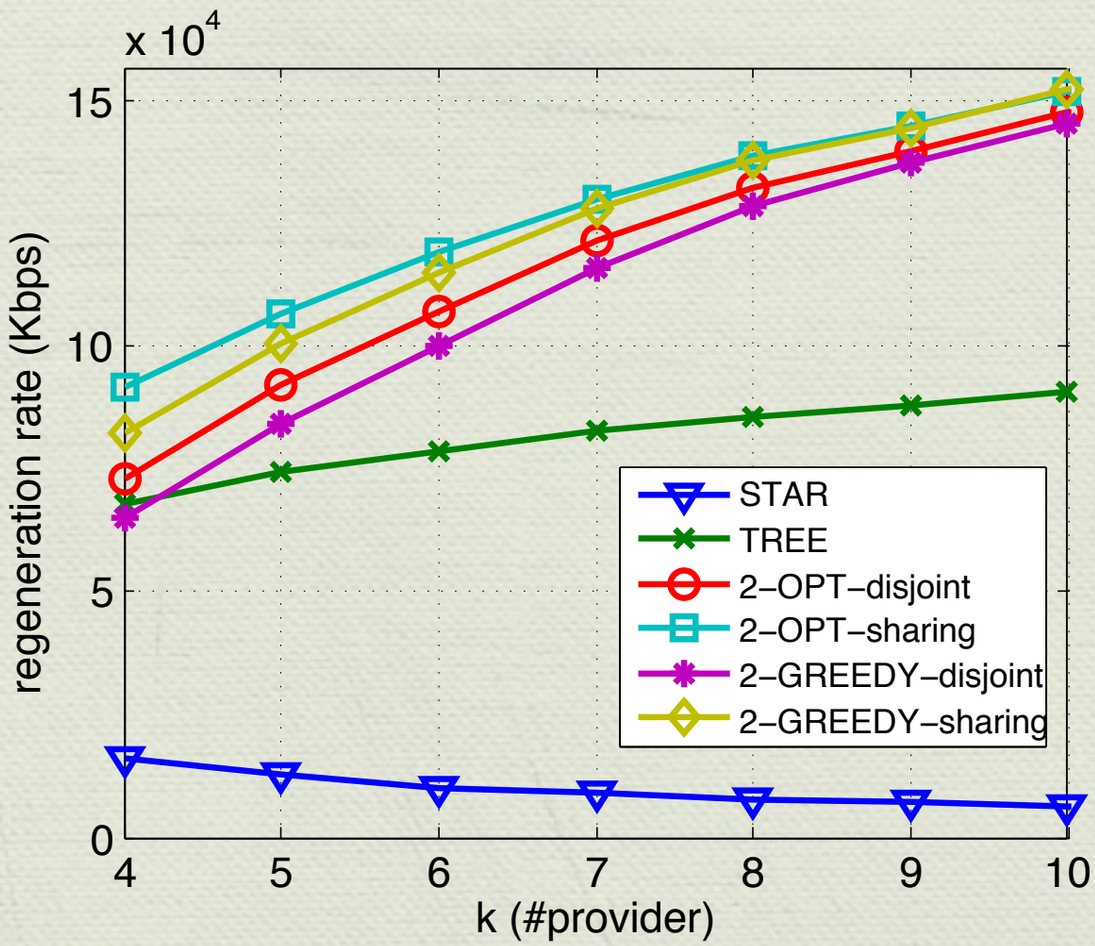
$$\alpha = \frac{|\mathrm{BW}_1 - \mathrm{BW}_2|}{\max\{\mathrm{BW}_1, \mathrm{BW}_2\}}.$$

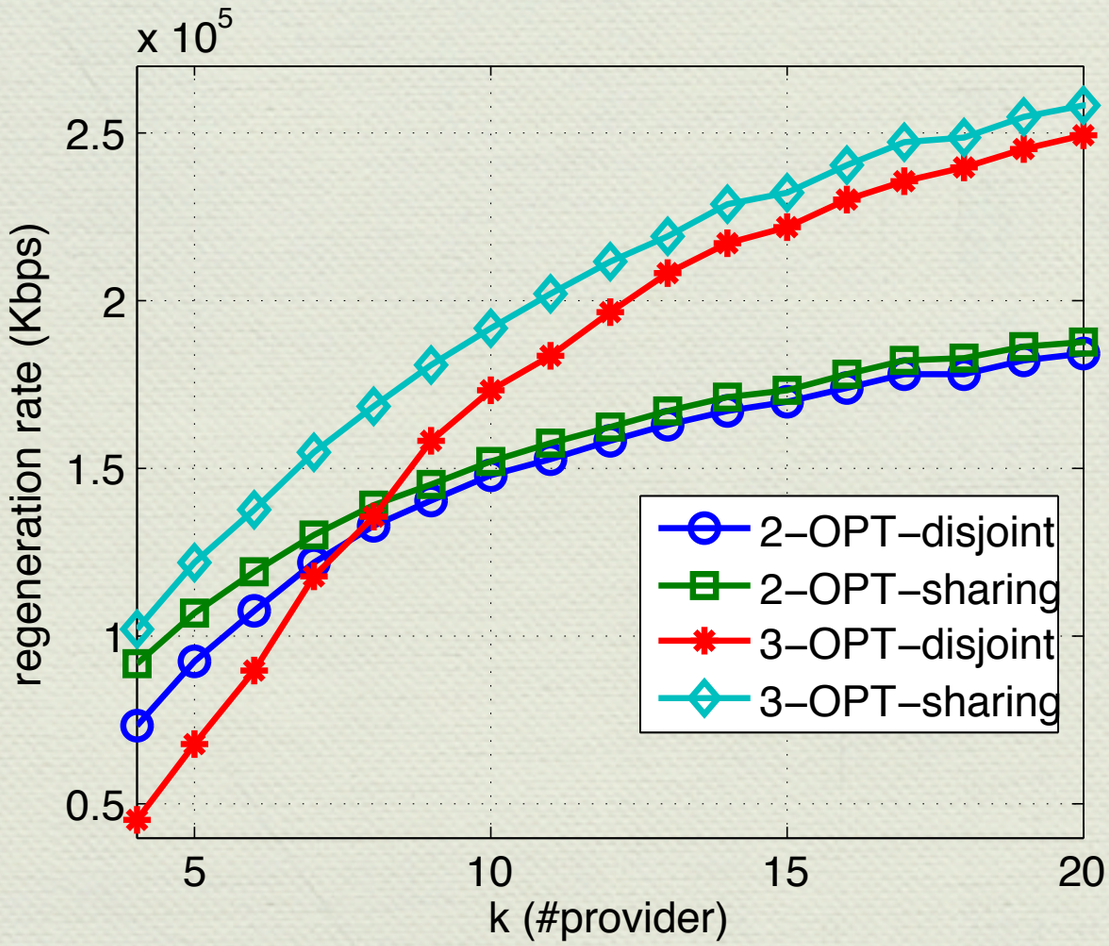
  - assign the available bandwidth of each direction between any pair of two nodes during the regeneration
  - compare the regeneration rate of
    - STAR, TREE, t-OPT-disjoint, t-OPT-sharing
    - t-GREEDY-disjoint, t-GREEDY-sharing





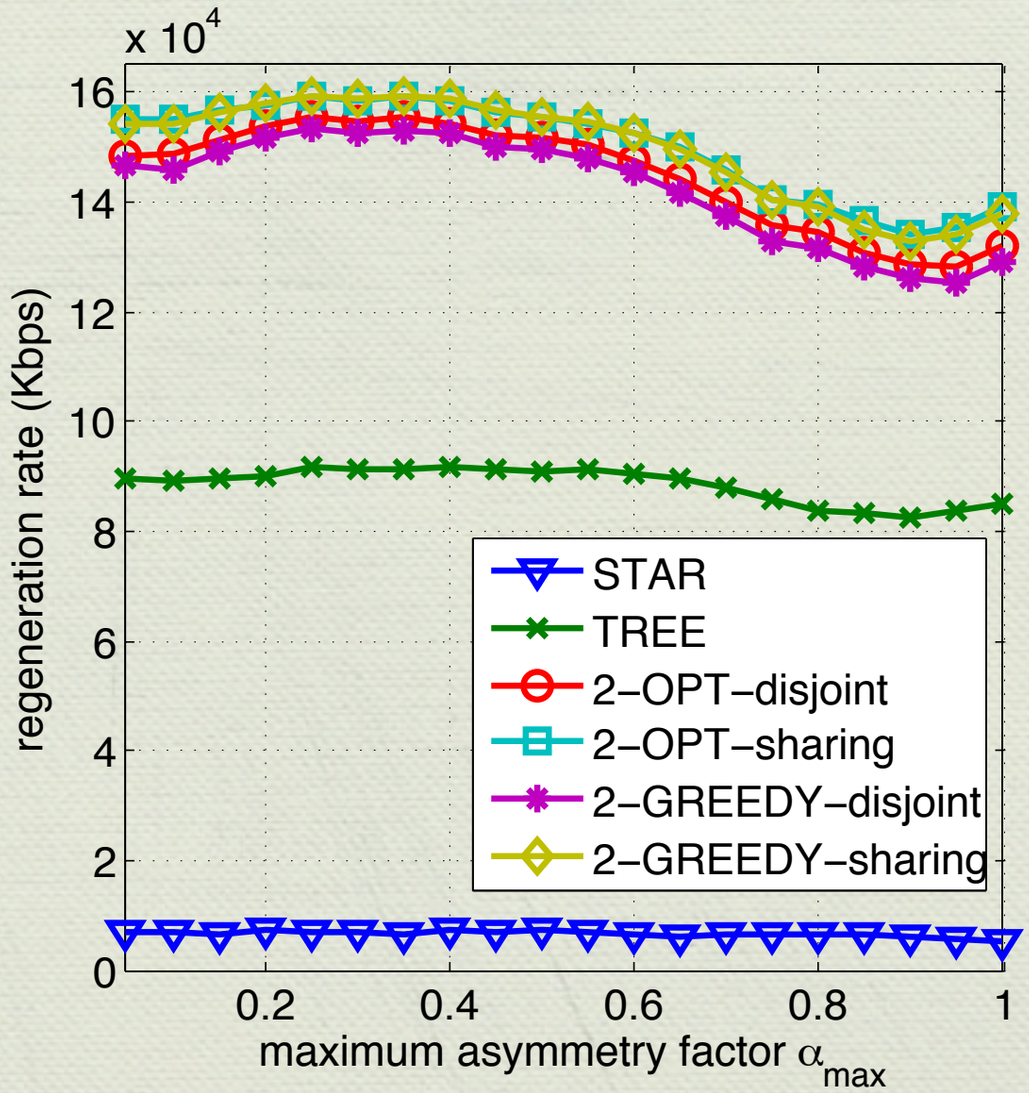
- **number of providers (i.e., k)**
  - regeneration rates increases with k except for STAR
  - OPT algorithms perform a bit better than GREEDY algorithms
  - 2 parallel trees achieve almost twice regeneration rate of TREE
  - more parallel trees lead to higher regeneration rate
  - for smaller k, use TREE or 2 parallel trees

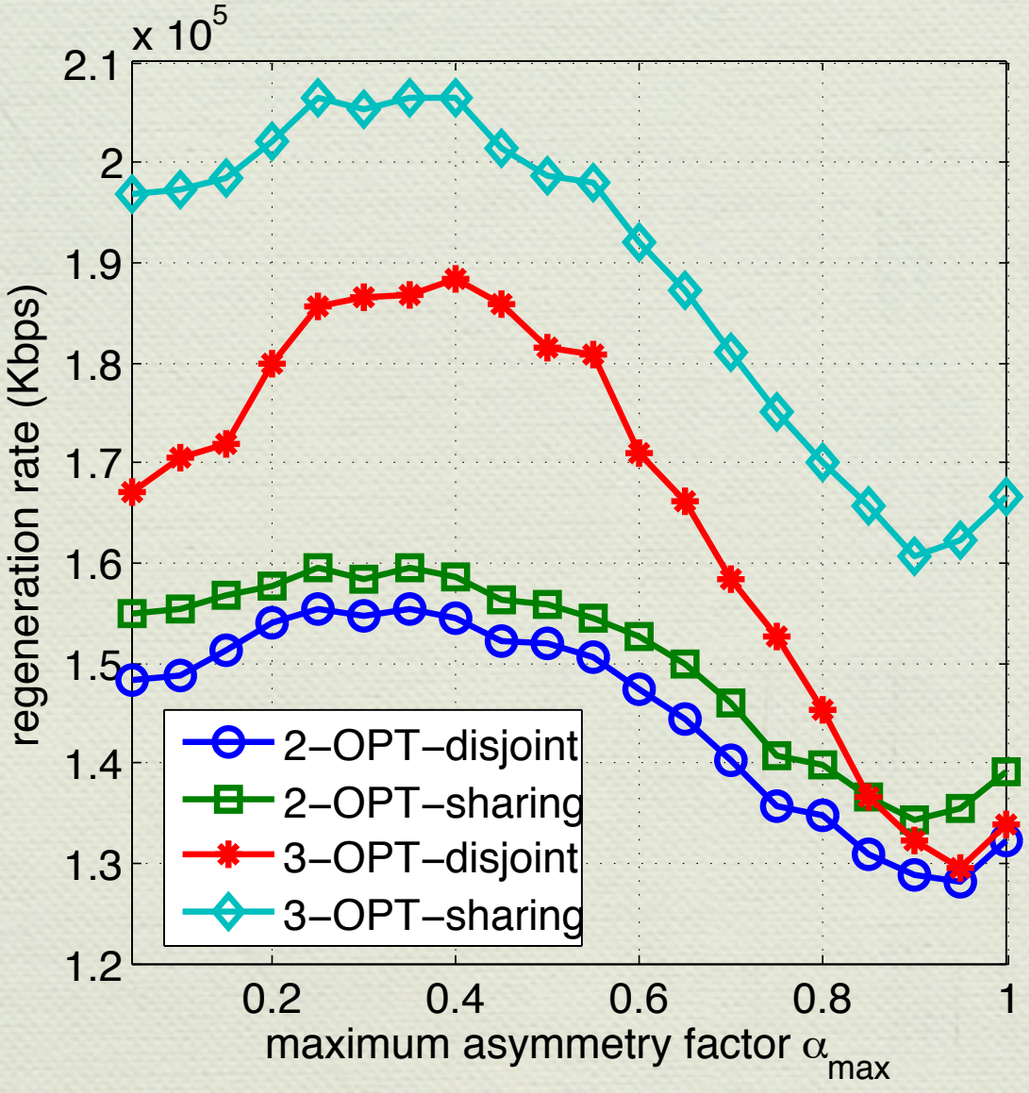




asymmetry factor

- asymmetry factor can influence the performance of constructing algorithms
- edge-disjoint algorithms can be more influenced than edge-sharing algorithms





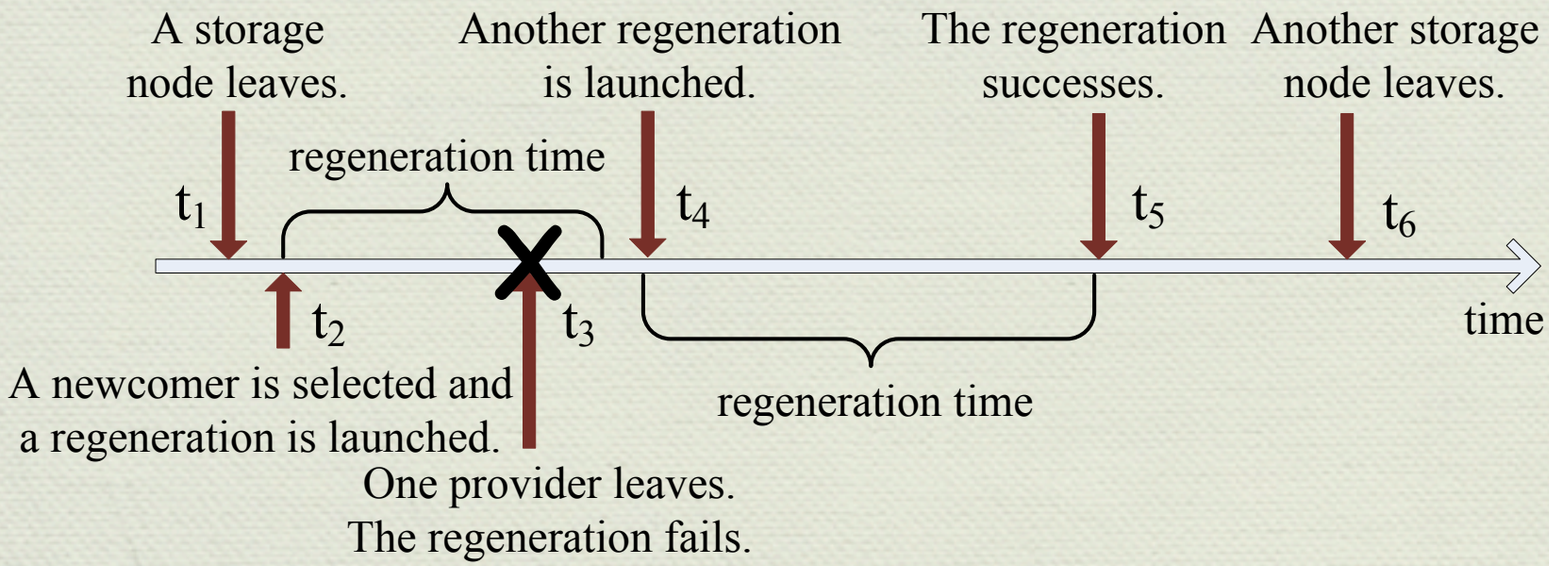
Dynamic network environment

Dynamic network environment

- **Node behaviors in PlanetLab detected by Lee *et al.***
  - select 100 nodes from the trace file
  - assign the real available bandwidth to each link between any pair of these nodes

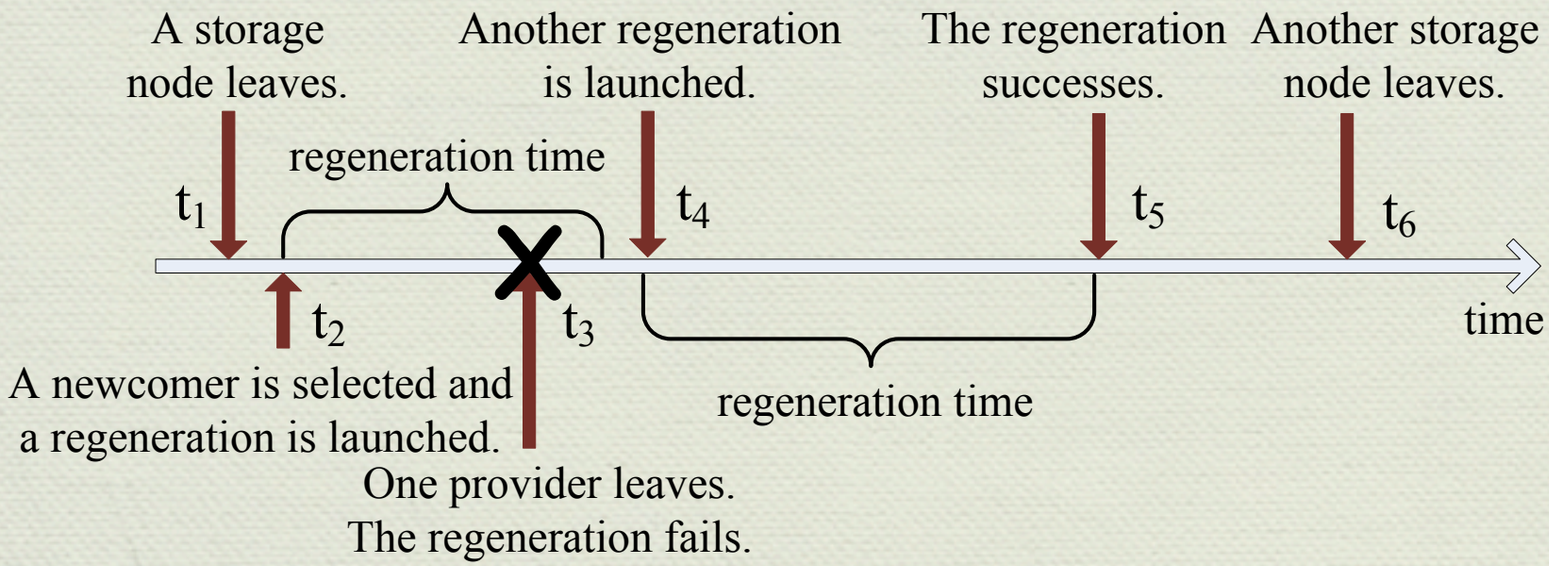
Dynamic network environment

- **Node behaviors in PlanetLab detected by Lee *et al.***
  - select 100 nodes from the trace file
  - assign the real available bandwidth to each link between any pair of these nodes
- **Event-driven simulator**



Dynamic network environment

- **Node behaviors in PlanetLab detected by Lee *et al.***
  - select 100 nodes from the trace file
  - assign the real available bandwidth to each link between any pair of these nodes
- **Event-driven simulator**

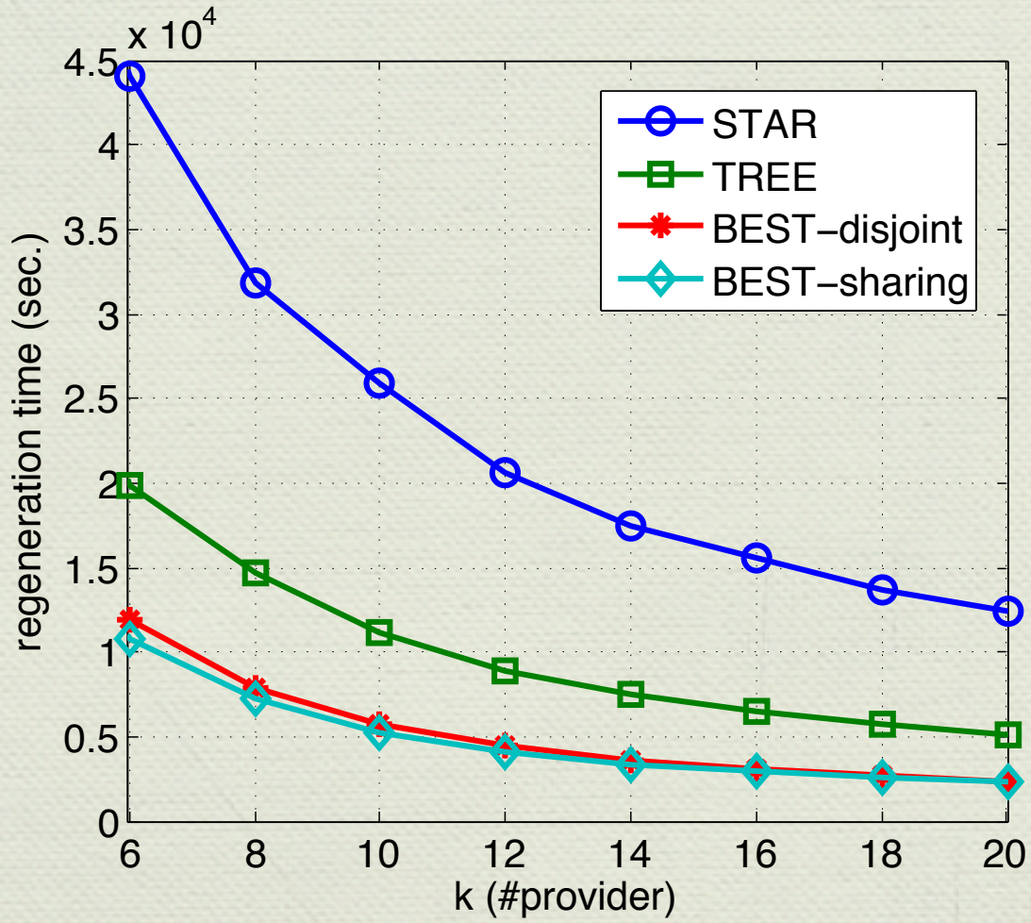


- **Metrics**
  - regeneration time, successful probability of regeneration, file availability,

- **Select the number of parallel regeneration trees adaptively**
  - such that the optimal regeneration rate is achieved
  - BEST-disjoint and BEST-sharing

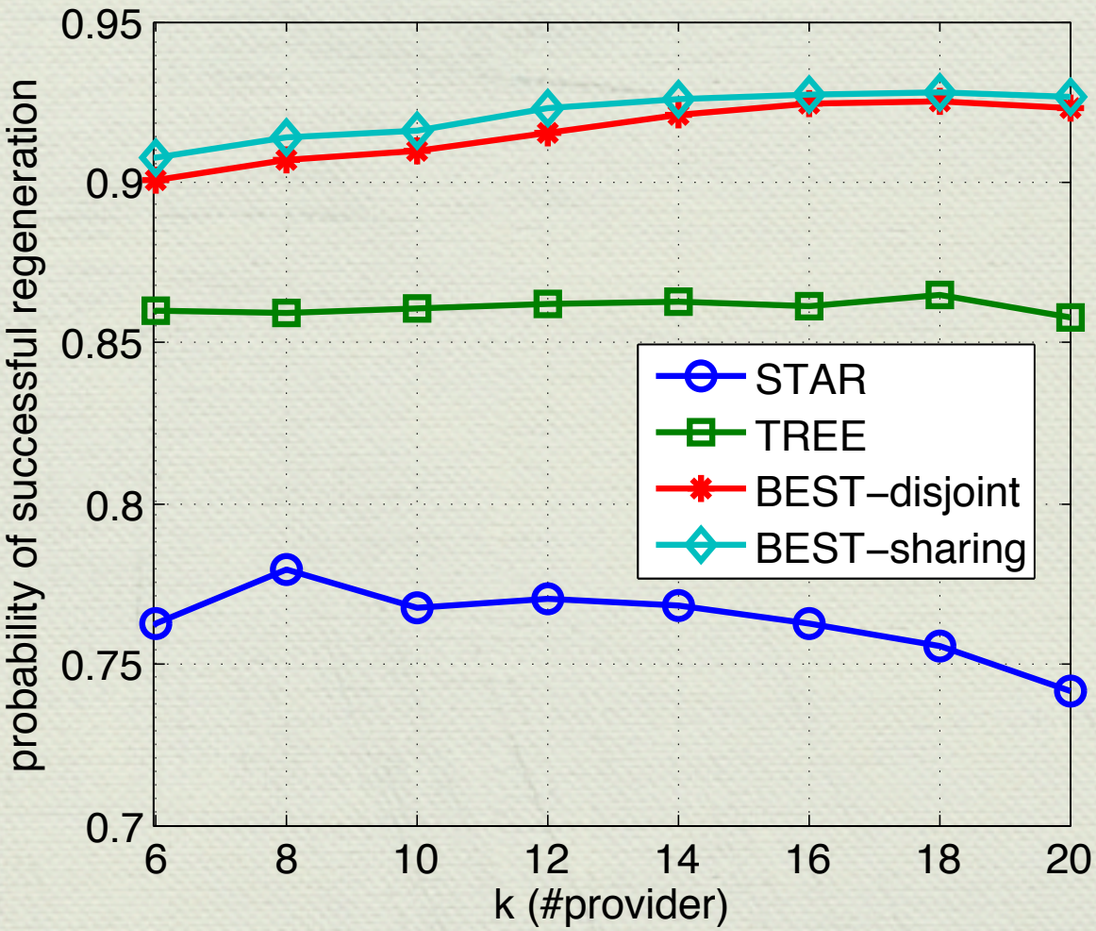
- **Select the number of parallel regeneration trees adaptively**
  - such that the optimal regeneration rate is achieved
  - BEST-disjoint and BEST-sharing
- **Settings**
  - 10^7 seconds; 100 nodes;
  - 500GB data with redundancy rate of 1.5
  - at most 5 parallel regeneration trees

- **Select the number of parallel regeneration trees adaptively**
  - such that the optimal regeneration rate is achieved
  - BEST-disjoint and BEST-sharing
- **Settings**
  - 10^7 seconds; 100 nodes;
  - 500GB data with redundancy rate of 1.5
  - at most 5 parallel regeneration trees
- **Regeneration time**
  - 58% and 55% of regeneration time is saved by BEST-disjoint and BEST-sharing, respectively



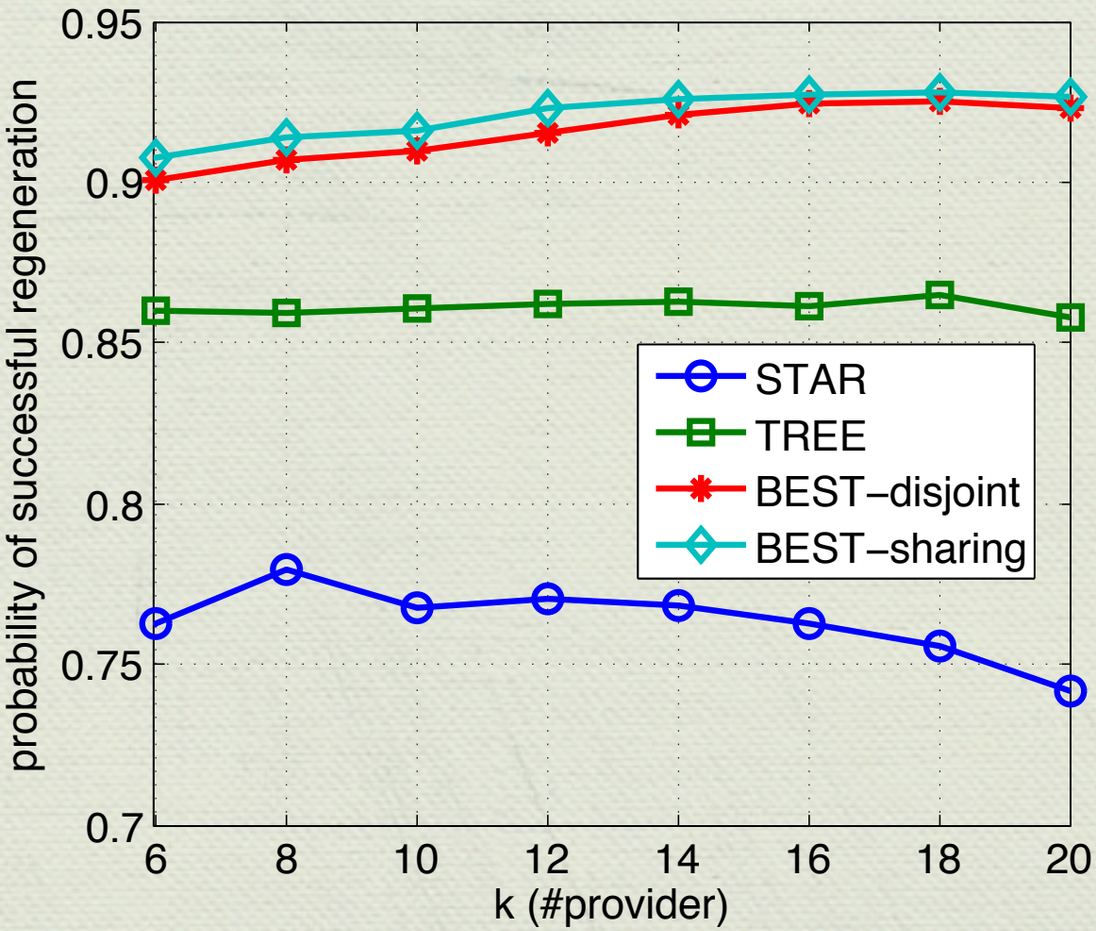
Successful probability of regeneration

- more than 90% for BEST-disjoint and BEST-sharing
- only about 75% and 85% for STAR and TREE, respectively



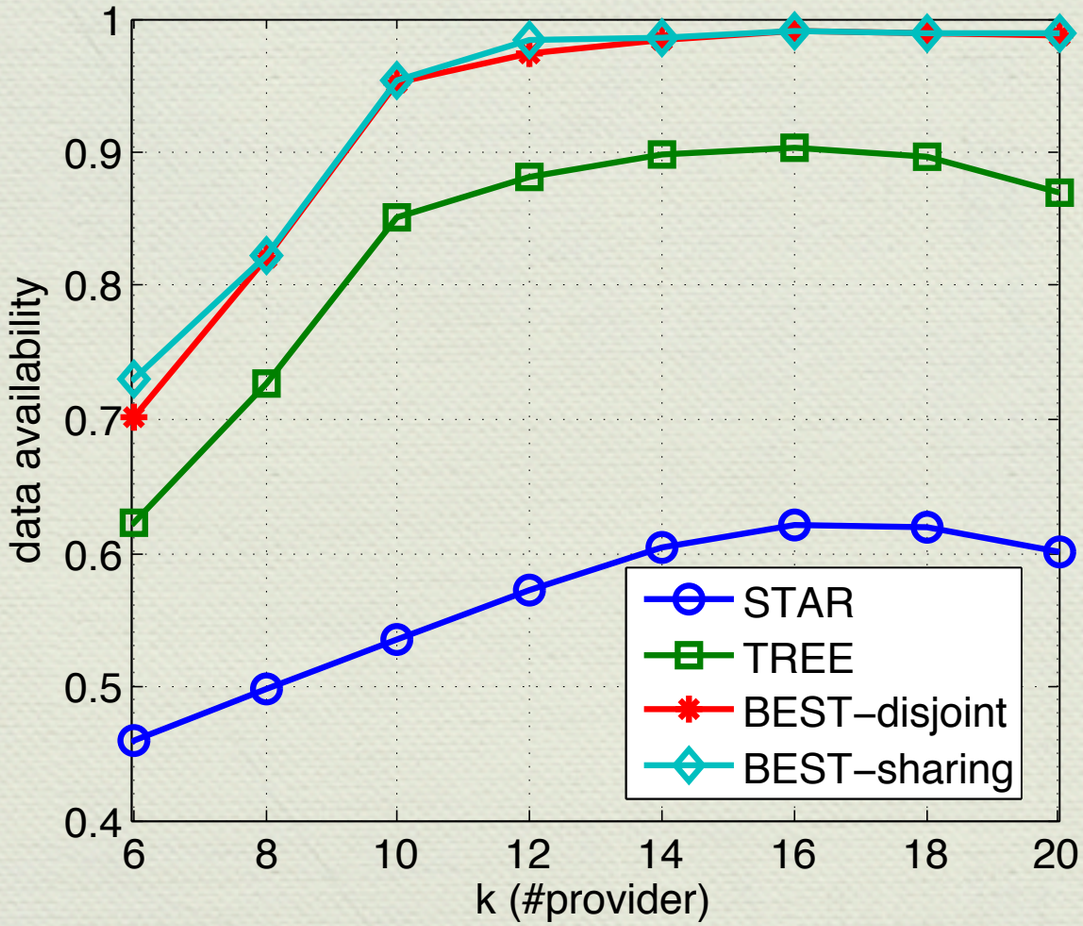
Successful probability of regeneration

- more than 90% for BEST-disjoint and BEST-sharing
- only about 75% and 85% for STAR and TREE, respectively



Data availability

- unacceptable for STAR and TREE
- BEST-disjoint and BEST-sharing achieve more than 98% of availability



Conclusions

Conclusions

- **Discuss the construction of regeneration in the network with asymmetric links**
  - analysis of regeneration rate

Conclusions

- **Discuss the construction of regeneration in the network with asymmetric links**
  - analysis of regeneration rate
- **Construct multiple parallel edge-disjoint and edge-sharing regeneration trees**
  - polynomial algorithms

Conclusions

- **Discuss the construction of regeneration in the network with asymmetric links**
  - analysis of regeneration rate
- **Construct multiple parallel edge-disjoint and edge-sharing regeneration trees**
  - polynomial algorithms
- **Static and dynamic simulations**
  - reduce the regeneration time significantly such that data can be regarded as almost available in a dynamic network environment with node joins and departures.
  - edge-sharing algorithms perform better than the edge-disjoint algorithms, both on the available bandwidth and on the resistance to the high asymmetry factor.

Thank you!

For more information:
http://sonic.fudan.edu.cn/junli/