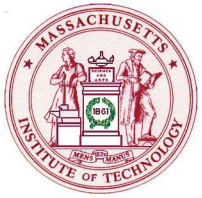
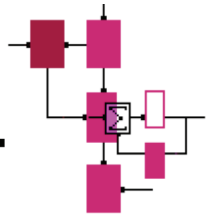


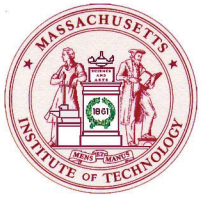


Routing in Data Networks

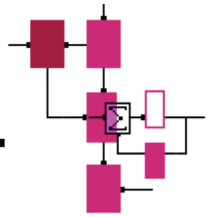
Muriel Medard

EECS

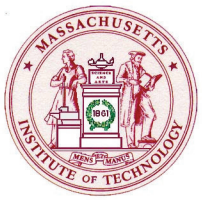
LIDS



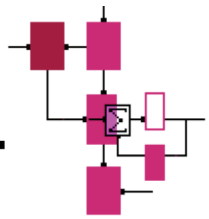
Routing



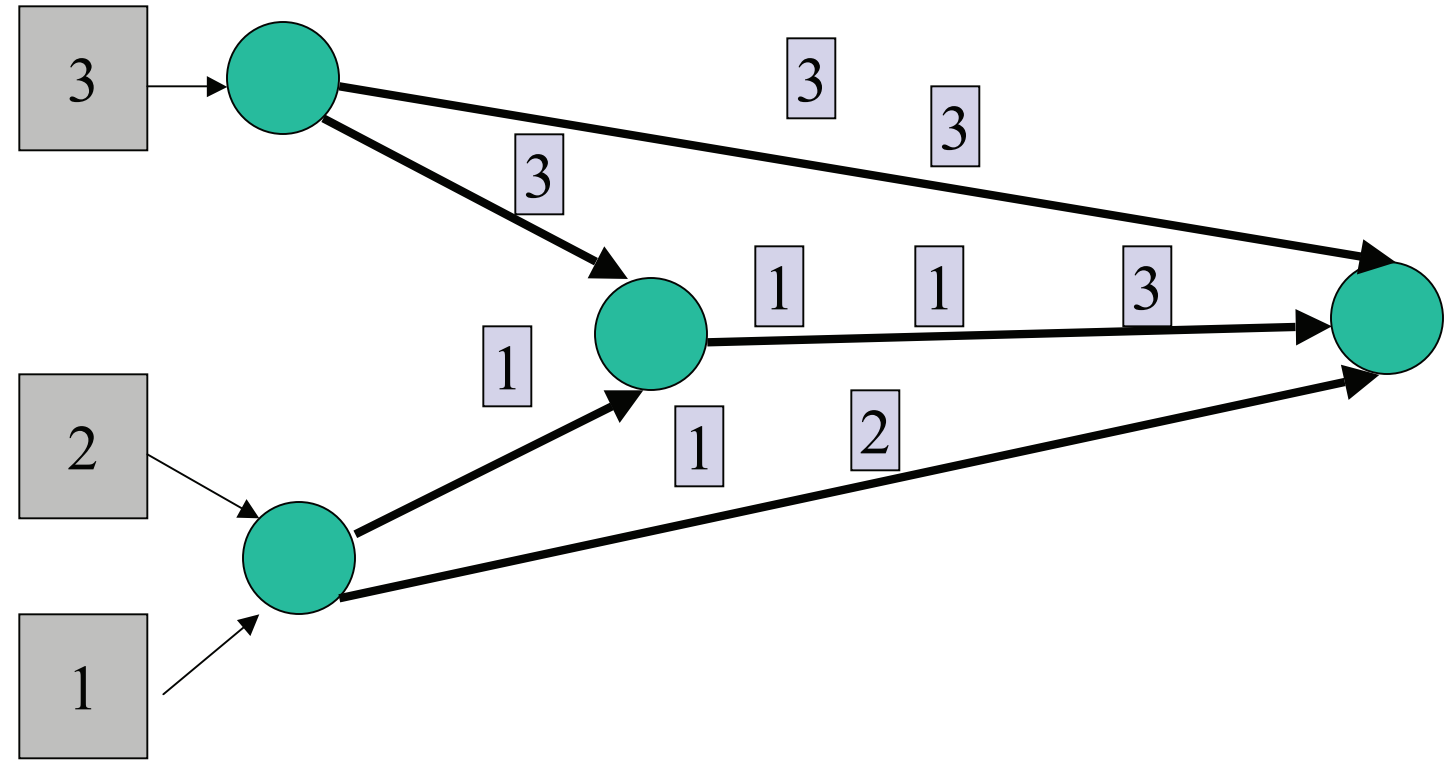
- Introduction
- Routing for shortest path
  - Dijsktra
  - Bellman-Ford
  - spanning trees
- Optimal routing based on flows
- First derivative length
- Optimal routing characterization
- Flow deviation: Frank-Wolfe

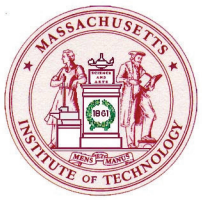


Routing functions

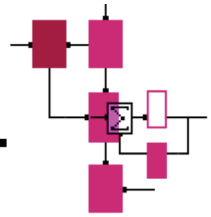


- Datagram

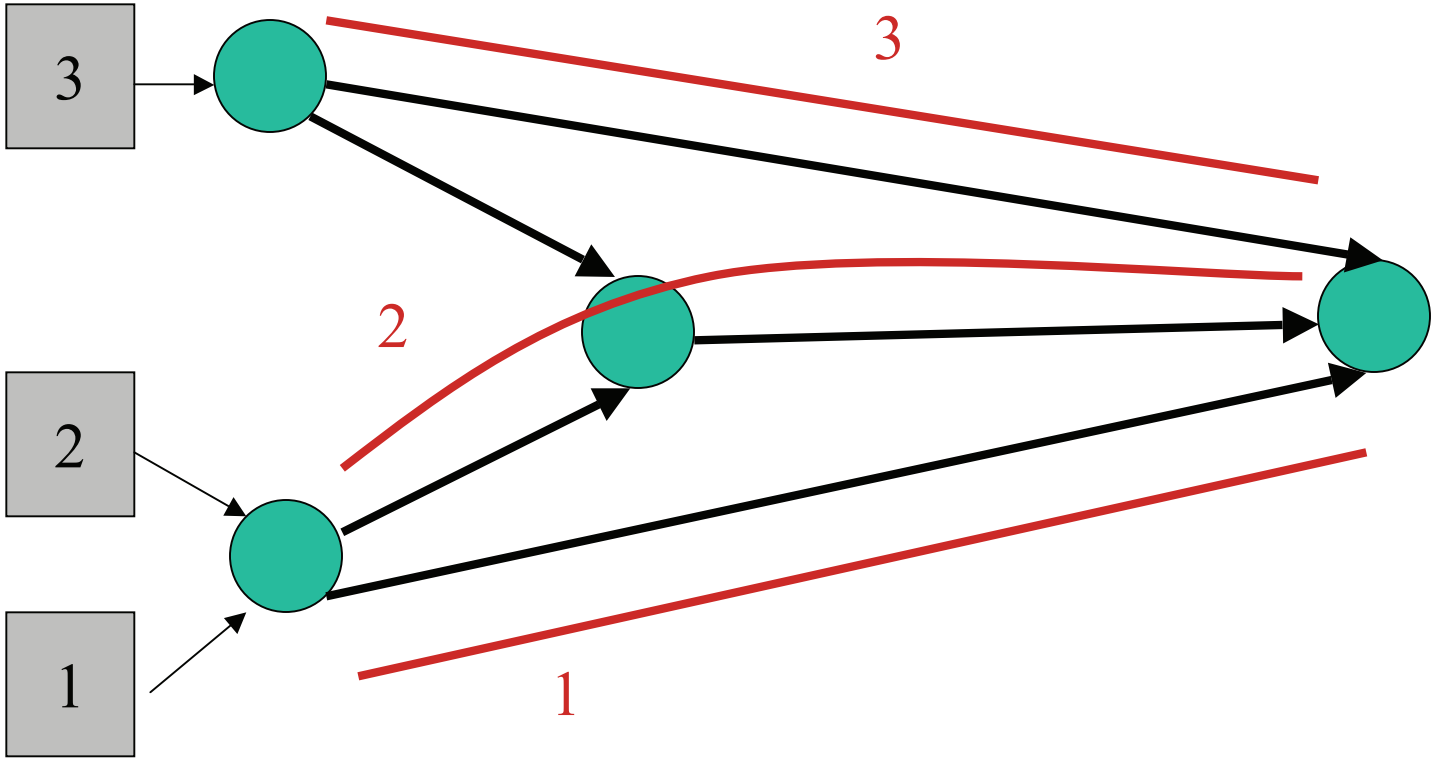


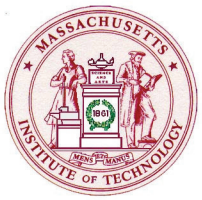


Routing functions

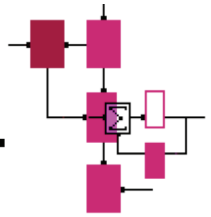


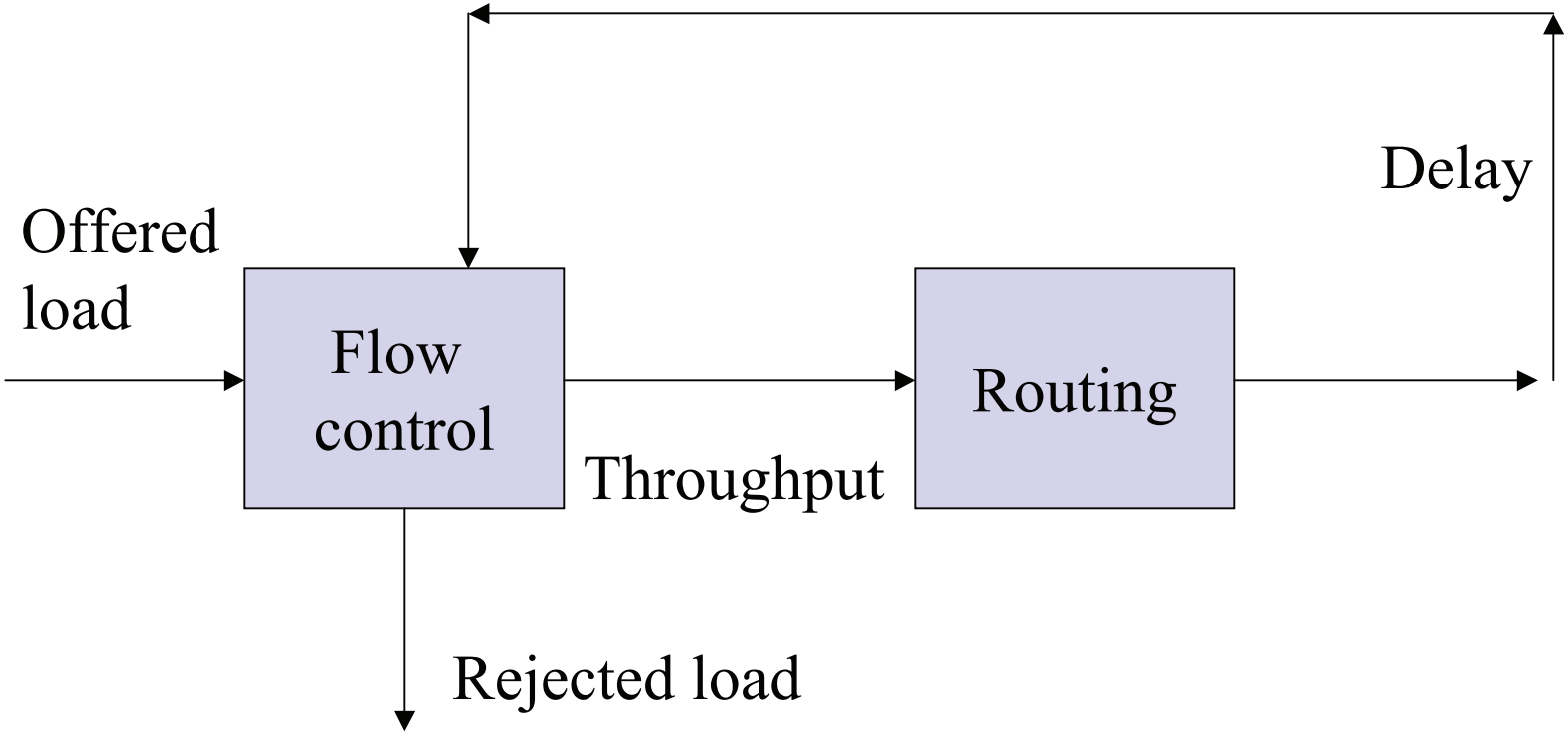
- Circuit or virtual circuit: routing decisions are made at time circuit is established



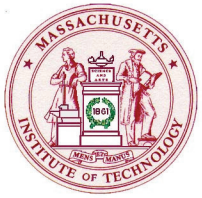


What is the purpose of routing?

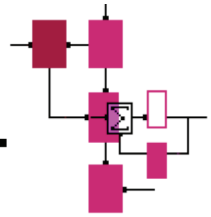




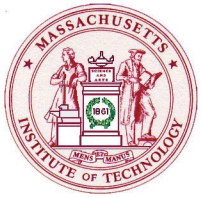
Routing affects delay, hence congestion and eventually throughput (we will see this in queueing)



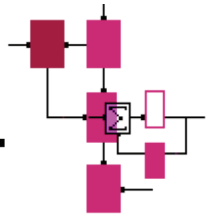
Routing methods



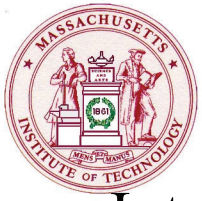
- Centralized: all routing decisions at a single node
- Distributed: nodes share computation
- Static: routes are fixed for origin/destination pairs (OD pairs)
- Adaptive or Dynamic: respond to perceived changes in traffic input pattern



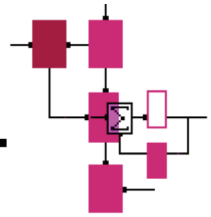
Common strategies



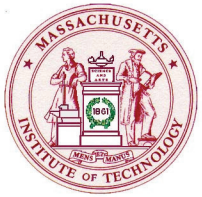
- Communicate to all points:
  - flooding, all nods talk to all nodes on all links
  - multicast, spanning trees
- Point-to-point: a variety of metrics may be used, most common is shortest path
- Optimal routing: optimization problem in terms of commodity flow



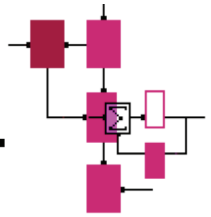
Shortest Paths



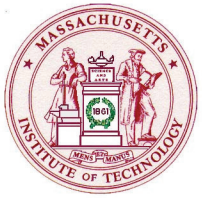
- Interior gateway protocol
- Option 1 (routing information protocol (RIP)):
  - vector distance protocol: each gateway propagates a list of the networks it can reach and the distance to each network
  - gateways use the list to compute new routes, then propagate their list of reachable networks
- Option 2 (open shortest path first (OSPF)):
  - link-state protocol: each gateway propagates status of its individual connections to networks
  - protocol delivers each link state message to all other participating gateways
  - if new link state information arrives, then gateway recomputes next-hop along shortest path to each destination



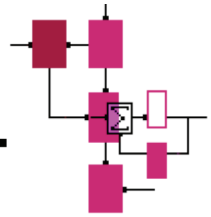
OSPF



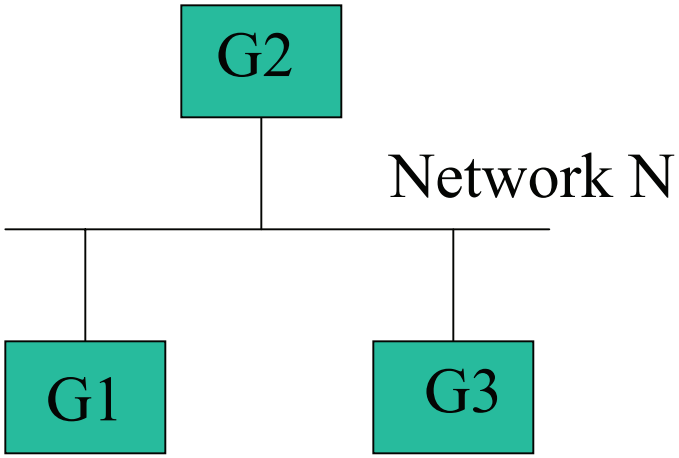
- OSPF has each gateway maintain a topology graph
- Each node is either a gateway or a network
- If a physical connection exists between two objects in an internet, the OSPF graph contains a pair of directed edges between the nodes representing the objects
- Note: gateways engage in active propagation of routing information while hosts acquire routing information passively and never propagate it

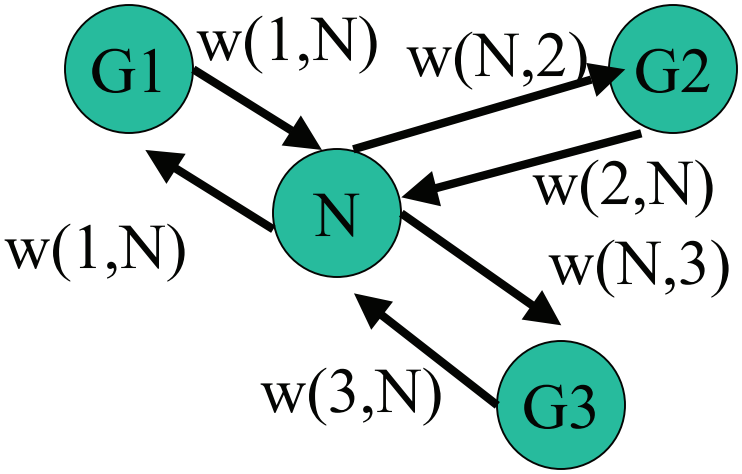


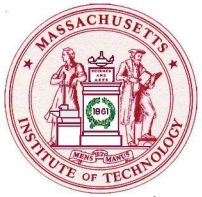
OSPF



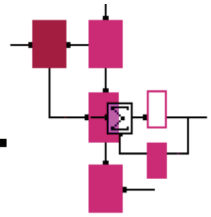
- Weights can be asymmetric: w(i,j) need not be equal to w(j,i)
- All weights are positive
- Weights are assigned by the network manager



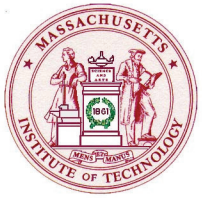




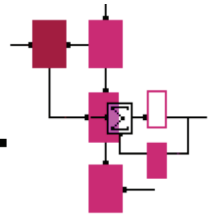
Shortest Path Algorithms



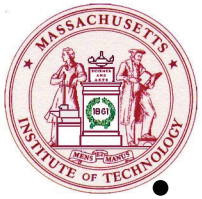
- Shortest path between two nodes: length = weight
- Directed graphs (digraphs) (recall that MSTs were on undirected graphs), edges are called arcs and have a direction $(i,j) \neq (j,i)$
- Shortest path problem: a directed path from A to B is a sequence of distinct nodes A, n_1, n_2, …, n_k, B, where (A, n_1), (n_1, n_2), …, (n_k, B) are directed arcs - find the shortest such path
- Variants of the problem: find shortest path from an origin to all nodes or from all nodes to an origin
- Assumption: all cycles have non-negative length
- Three main algorithms:
  - Dijsktra
  - Bellman-Ford
  - Floyd-Warshall



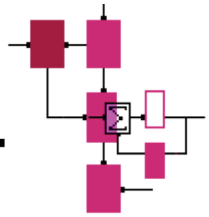
Dijkstra's algorithm



- Assume all lengths are non-negative, length d(i,j) for arc (i,j)
- Shortest path from all nodes to node 1
- P = {1} to start out with - P is the set of permanently labeled nodes
- The node added at each step is the closest to node 1 out of all the nodes not yet in P
- D(j) is the estimate of the shortest path length from j to 1
- Initially:
  - D(1) = 0
  - D(j) = d(j,1) if (j,1) exists and ∞ if it does not exist

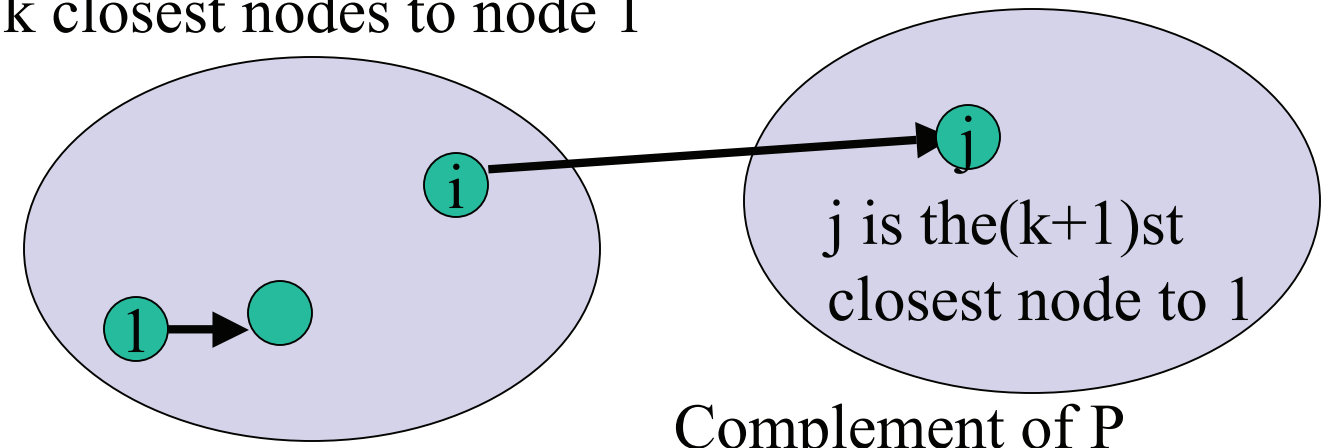


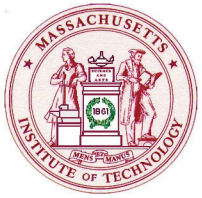
Dijkstra's algorithm



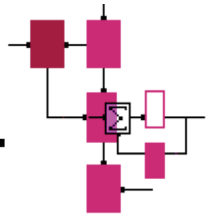
- Step 1:
  - find node i not in P with minimum D(i)
  - add i to P
  - if P contains all nodes, stop
- Step 2:
  - for all j not in P, update D(j) = min[D(j), D(i) + d(j,i)]
  - goto 1

P: set of k closest nodes to node 1

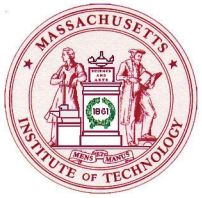




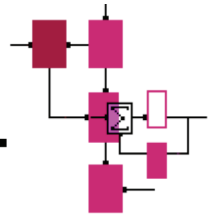
Dijkstra



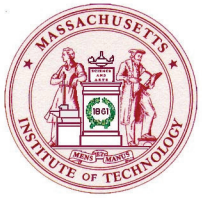
- At the beginning of each step 1,
  - D(j) is the shortest distance from j to 1 using nodes in 1 only (except j may possibly be outside P)
  - D(i) ≤ D(j) when i is on p and j is not in P
- How long does is take to run?
  - We go through step 1and 2 roughly N times, where N is the number of nodes
  - we go through roughly N computations at each step 2
  - time complexity is thus $O(N^2)$
  - Note: to be more precise, we would have to take into account the maximum size d of the lengths and the time complexity would depend on log(d)



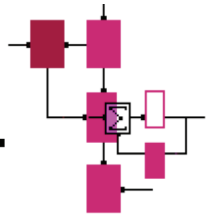
Bellman-Ford



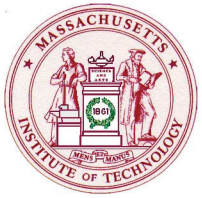
- Allows negative lengths, but not negative cycles
- B-F works at looking at negative lengths from every node to node 1
- If arc (i,j) does not exist, we set d(i,j) to infinity
- We look at walks: consider the shortest walk from node i to 1 after at most h arcs
- Algorithm:
  - $D^{h+1}(i) = \min_{\text{over all } j}[d(i,j) + D^h(i)]$ for all i other than 1
  - we terminate when $D^{h+1}(i) = D^h(i)$
- The $D^{h+1}(i)$ are the lengths of the shortest path from i to 1 with no more than h arcs in it



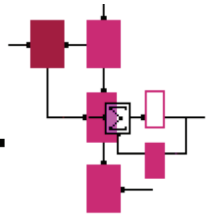
Bellman-Ford



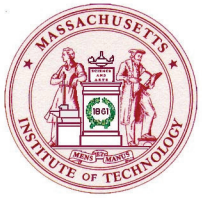
- Let us show this by induction
  - $D^1(i) = d(i,1)$for every i other than 1, since one hop corresponds to having a single arc
  - now suppose this holds for some h, let us show it for h+1: we assume that for all $k \leq h$, $D^k(i)$ is the length of the shortest walk from i to 1 with k arcs or fewer
  - $\min_{\text{over all } j}[d(i,j) + D^h(i)]$ allows up to h+1 arcs, but $D^h(i)$ would have fewer than h arcs, so $\min[D^h(i), \min_{\text{over all } j}[d(i,j) + D^h(i)]] = D^{h+1}(i)$
- Time complexity: A, where A is the number of arcs, for at most N-1 nodes (note: A can be up to $(N-1)^2$)
- In practice, B-F still often performs better than Dijkstra



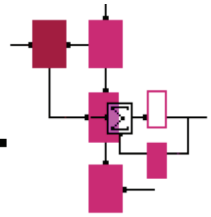
Floyd-Warshall



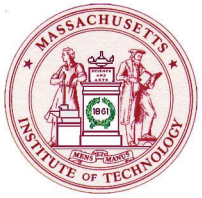
- Find the shortest path between all pairs of nodes together
- $D^0(i,j) = d(i,j)$
- $D^{n+1}(i,j) = \min[D^n(i,j), D^n(i,(n+1)) + D^n((n+1),j)]$
- Claim: $D^n(i,j)$ is the shortest path from i to j using only nodes 1 to n for intermediate nodes
- By induction:
  - for n = 0, it clearly holds
  - assume that the claim holds up to n
  - the shortest path from i to j using nodes 1 through n+1 either:
    - does not contain node n+1, so it is the same as for the path using nodes 1 through n
    - does contain node n+1, then the path is i …. n+1 …j



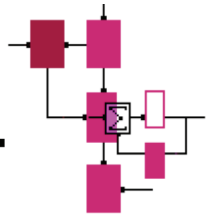
Distributed Asynchronous B-F



- The algorithms we investigated work well when we have a single centralized entity doing all the computation - what happens when we have a network that is operating in a distributed and asynchronous fashion?
- Let us call N(i) the set of nodes that are neighbors of node i
- At every time t, every node i other than 1 has available :
  - $D^i_j(t)$: estimate of shortest distance of each neighbor node j in N(i) which was last communicated to node i
  - $D^i(t)$: estimate of the shortest distance of node i which was last computed at node i using B-F



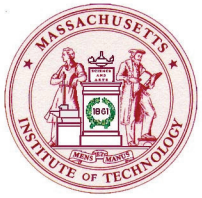
Distributed Asynchronous B-F



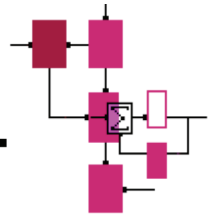
- $D^1(t) = 0$ at all times
- Each node i has available link lengths d(i,j) for all j in N(i)
- Distance estimates change only at time $t_0, t_1, .., t_m$, where t_m becomes infinitely large at m becomes infinitely large
- At these times:
  - $D^i(t) = \min_{j \text{ in } N(i)}[d(i,j) + D^i_j(t)]$, but leaves estimate $D^i_j(t)$ for all j in N(i) unchanged

OR – node i receives from one or more neighbors their D^j, which becomes D^i_j (all other D^i_j are unchanged)

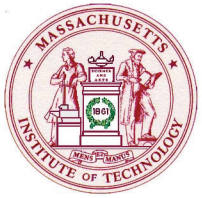
OR – node i is idle



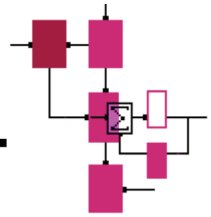
Distributed Asynchronous B-F



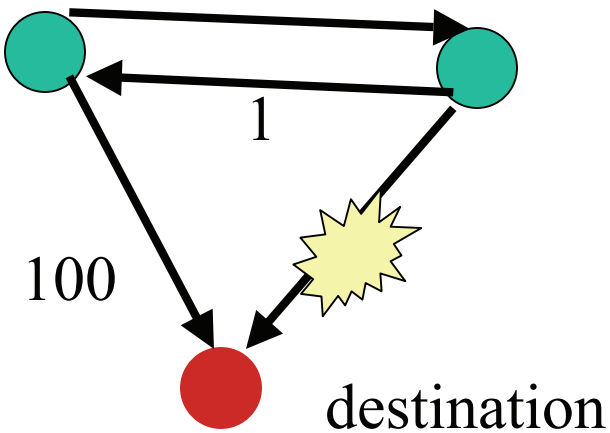
- Assumptions:
  - if there is a link (i,j), there is also a link (j,i)
  - no negative length cycles
  - nodes never stop updating estimates and receiving updated estimates
  - old distance information is eventually purged
  - distances are fixed
- Under those conditions:for any initial $D^i_j(t0)$, $D^i(t)$, for some t_m, eventually all values $D^i(t) = D^i$ for all t greater than t_m

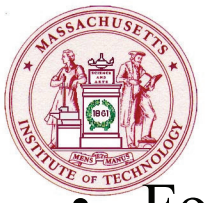


Failure recovery

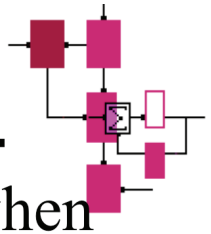


- Often asynchronous distributed Bellman-Ford works even when there are changes, including failures
- However, the algorithm may take a long time to recover from a failure that is located on a shortest path, particularly if the alternate path is much longer than the original path (bad news phenomenon)

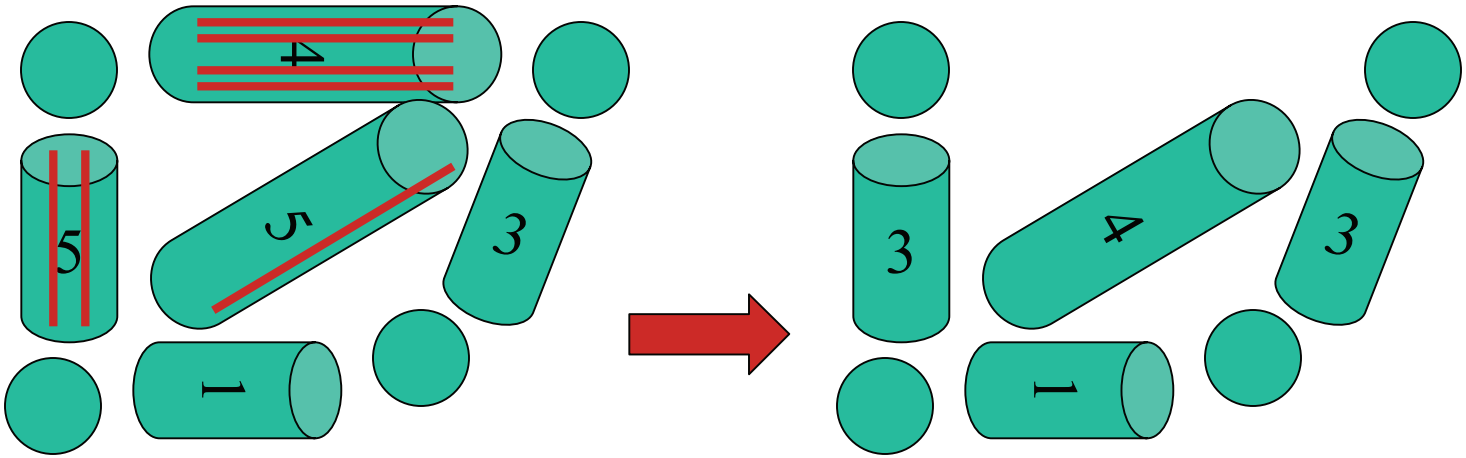




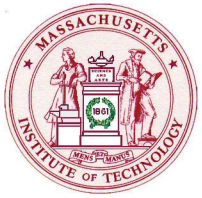
Optimal routing based on flows



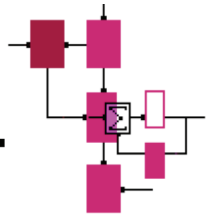
- For path selection, we have considered minimum cost when we have a given price is paid per path
- What happens when we add capacity considerations?
- If we have circuits or virtual circuits, then we can create a topology that takes into account the presence of other users



- If we have a probabilistic description of traffic progression, then we could use dynamic programming

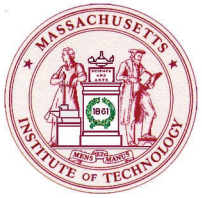


Routing based on flows

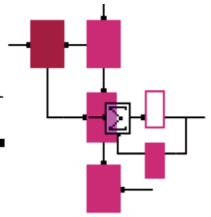


- When we have packets or fine granularity virtual circuits (VCs), we can assume that we have roughly fluid flows
- We will try to optimize a cost function related to the flow F(i,j) (arrival rate in terms of of what we may be considering) on the link (i,j)
- A reasonable metric is $\sum_{(i,j)} D(F(i,j))$

  where D is some monotonically increasing function that grows very sharply when F(i,j) approaches the link capacity
- These type of models are called flow models - they look only at mean flow, they do not consider higher moments



Example based on Kleinrock's approximation



$$\lambda(i,j) = \sum_{\text{all paths p traversing link (i,j)}} \lambda(p)$$

$\mu(i,j)$ = service rate on link (i, j)

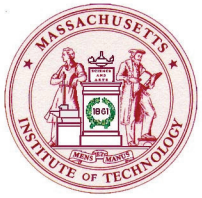
N(i, j) = average number of packets on link (i, j)

- Assume all queues behave like M/M/1 with arrival rate λ(i,j), service rate μ(i,j), and service/propagation delay d(i,j)
- Then

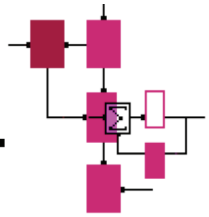
$$N_{i,j} = \frac{\lambda_{i,j}}{\mu_{i,j} - \lambda_{i,j}} + \lambda_{i,j} d_{i,j}$$

Increases sharply when we approach link capacity

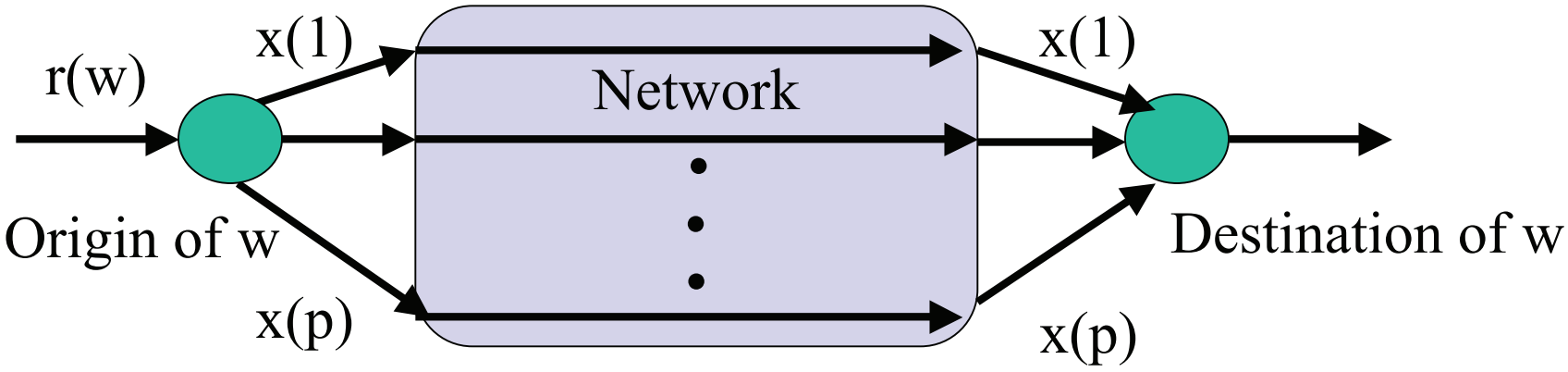
$$D(F(i,j)) = \frac{F(i,j)}{C(i,j) - F(i,j)} + d(i,j)\,F(i,j)$$

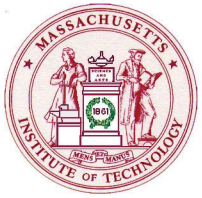


Optimal flow routing problem

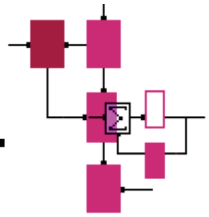


- For every OD pair w=(i,j), we are given the arrival rate r(w)
- We denote:
  - W: the set of all OD pairs w
  - P(w): a set of paths available for routing for OD pair w (possibly all paths)
  - x(p): the flow of path p (in units per second)





Problem statement



- We want to choose the flows so that we minimize

$$\sum_{(i,j)} D(F(i,j))$$

subject to the constraints

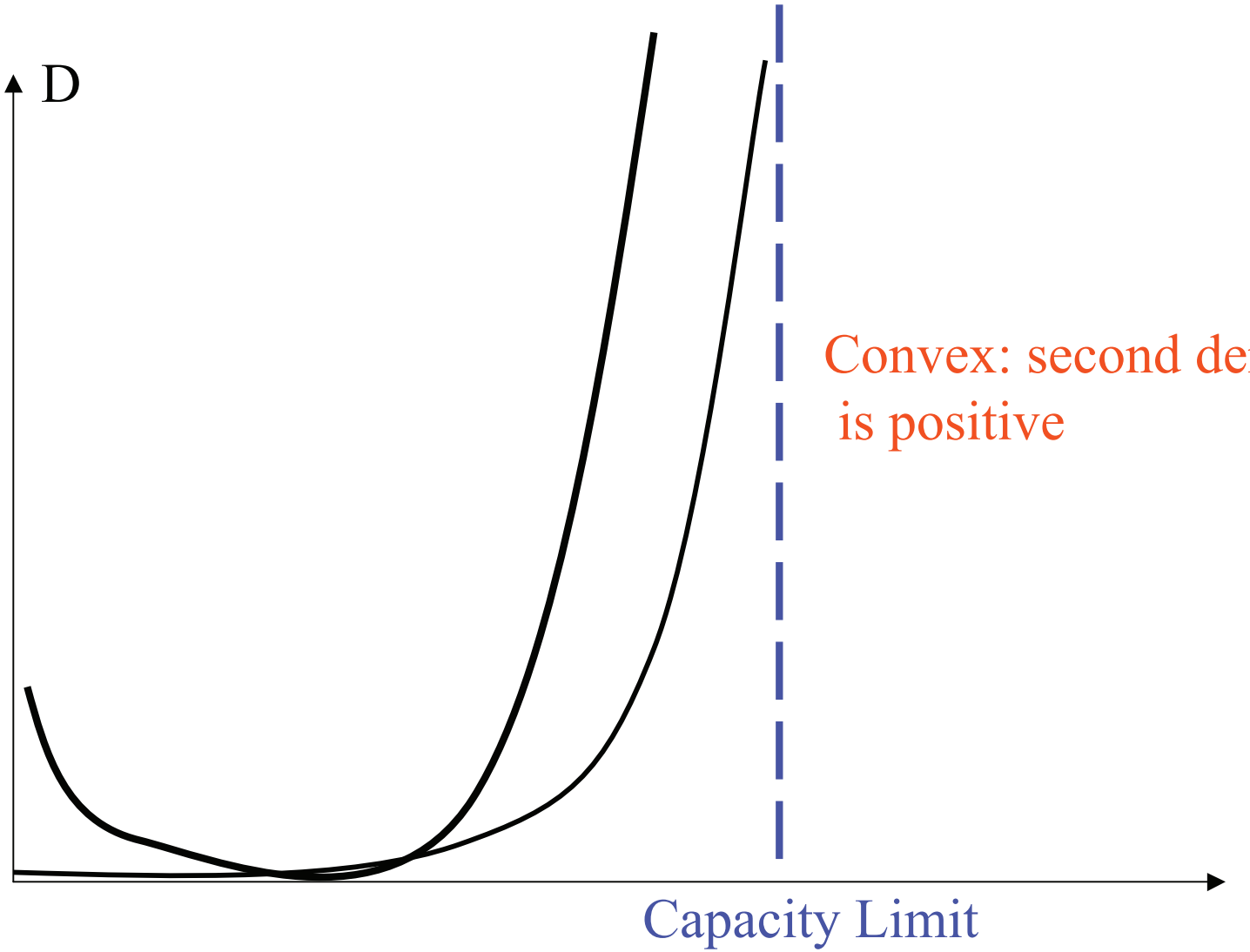
$$F(i,j) = \sum_{\text{all paths p containing } (i,j)} x(p)$$

$$\sum_{p \text{ in } P(w)} x(p) = r(w) \text{ for al OD pairs w in W}$$

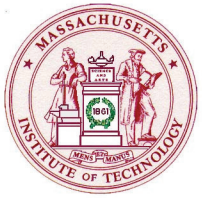
$$x(p) \geq 0$$

we assume that D is convex and that its first and second derivatives exist

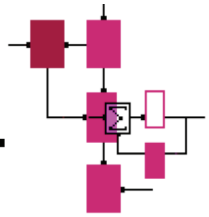
Examples of D



Convex: second derivative is positive



Flow problem

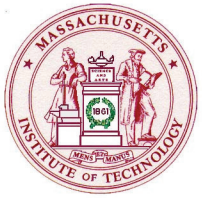


- We can perform a substitution in the following manner: consider the cost

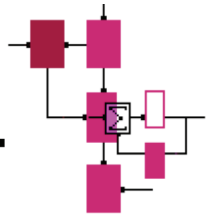
$$D(x) = \sum_{(i,j)} D\left(\sum_{\text{all paths p containing (i,j)}} x(p) \right)$$

consider the partial derivatives

$$\frac{\partial D(x)}{\partial x(p)} = \sum_{\text{all links (i,j) on path p}} \underbrace{D'\left(\sum_{\text{all paths p containing (i,j)}} x(p) \right)}_{\text{Depends on (i,j) only}}$$



First derivative lengths

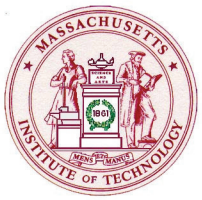


- Length interpretation:

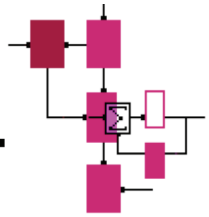
Define $D'_{i,j} = D'\left(\sum_{\text{all paths p containing (i,j)}} x(p)\right)$ to be the first derivative length of the link (i, j)

Note : the first derivative length depends on x crucially

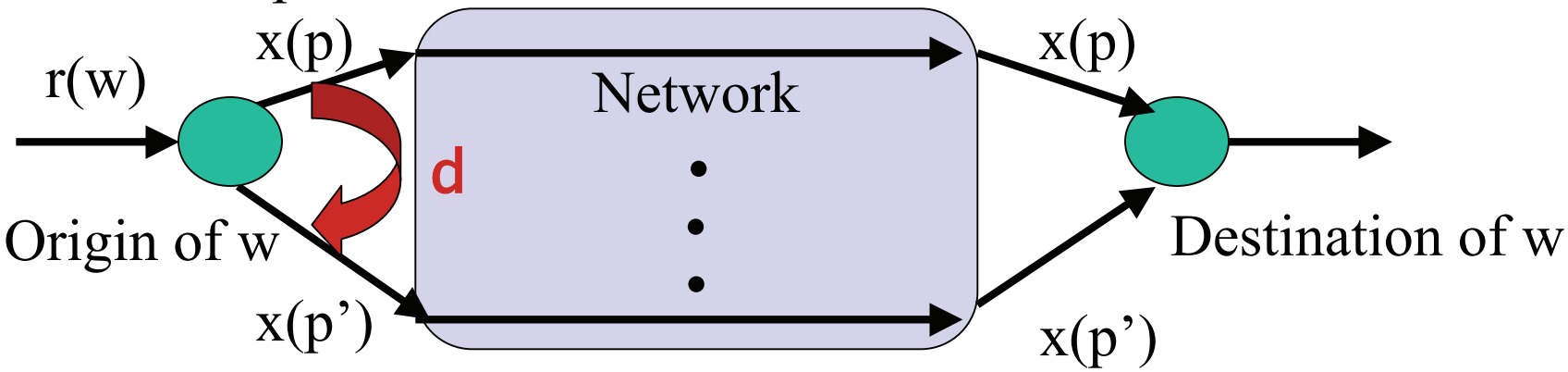
The derivative of D with respect to x(p) is thus the sum of the first derivative lengths of the links on the path p



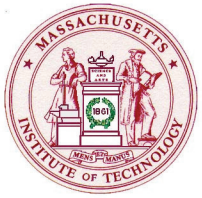
Characterization of optimal flows



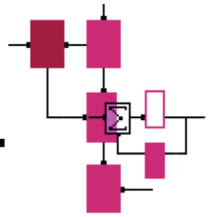
- A set of path flows x is optimal if the cost cannot be improved by making a feasible change of flow
- Consider shifting an increment d from path p to path p' of the OD pair w



- What is the effect of this transfer on the cost (assuming feasibility is maintained?)



Change in cost



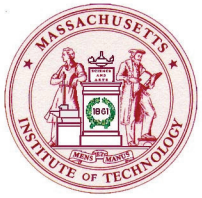
- Convexity means that a decrease is possible only if there is a local decrease direction - no local minima that are non global
- Let us use the fact that D has first and second derivatives
- A Taylor series expansion approach yields that the difference in cost due to the infinitesimal transfer of flow must be

$$-\frac{\partial D(x)}{\partial x(p)}\delta - \frac{\partial^2 D(x)}{\partial^2 x(p)}\delta^2 + \frac{\partial D(x)}{\partial x(p')}\delta + \frac{\partial^2 D(x)}{\partial^2 x(p')}\delta^2 + o(\delta^3)$$

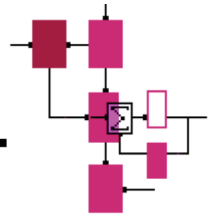
- To a first order approximation, the change in cost is

$$-\frac{\partial D(x)}{\partial x(p)}\delta + \frac{\partial D(x)}{\partial x(p')}\delta$$

Approximately linear in flow change



Characterization of optimal flows



- Cost change due to shifting traffic from p to p' must be detrimental in cost in the solution was optimal
- So

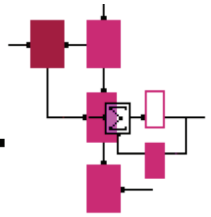
$$\frac{\partial D(x)}{\partial x(p)} \leq \frac{\partial D(x)}{\partial x(p')}$$

- This condition is the same as rewriting for all OD pairs that for the optimal flow x*,

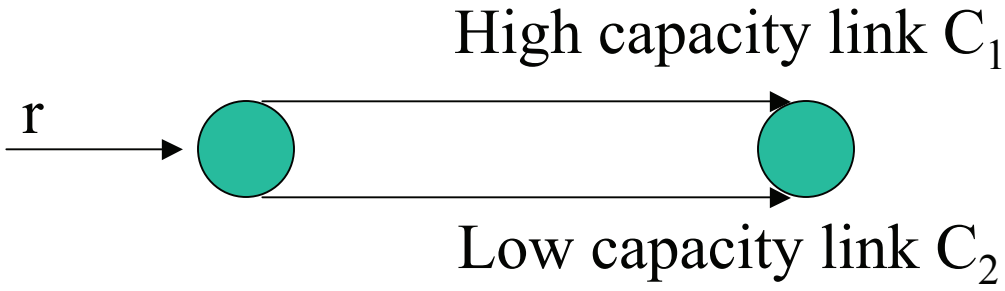
$$x^*(p) > 0 \text{ only if } \frac{\partial D(x^*)}{\partial x(p)} \leq \frac{\partial D(x^*)}{\partial x(p')} \text{ for all } p' \text{ in } P(w)$$

- So optimal path flows are positive only on paths that are shortest with respect to first derivative lengths

Solutions



- The link lengths depend on the link flows
- The first derivative lengths depend on the unknown optimal path flows, so the problem is harder than a shortest path problem
- Sometimes we can solve the problem using a closed form solution, but generally more involved methods may be necessary



$$\text{cost } D(x(i)) = \frac{x(i)}{C_i - x(i)}$$

$C_1 \geq C_2$, so two cases are possible

Solution example

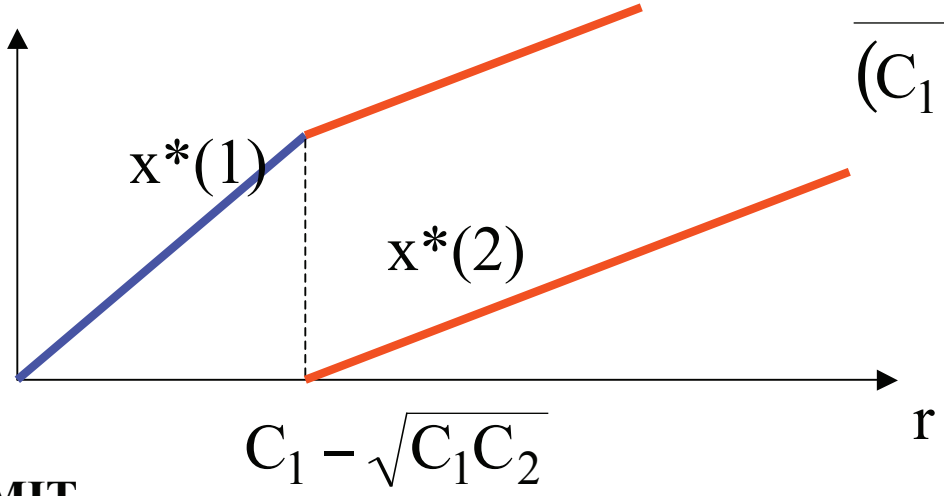
- Case 1: all the flow goes along 1

$$\frac{\partial D(x^*)}{\partial x(p)} \le \frac{\partial D(x^*)}{\partial x(p')} \text{ for all } p' \text{ in } P(w)$$

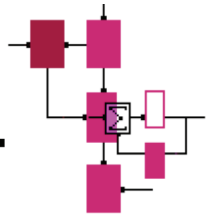
$$\text{gives } \frac{C_1}{(C_1 - r)^2} \le \frac{1}{C_2} \Rightarrow r \le C_1 - \sqrt{C_1 C_2}$$

- Case 2 : Flow goes along both paths

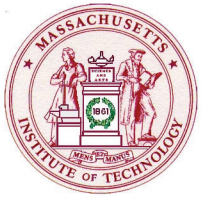
$$\frac{C_1}{(C_1 - x(1))^2} \le \frac{C_2}{(C_2 - x(2))^2}$$



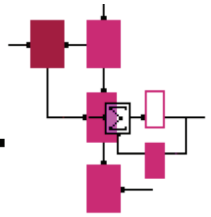
Solution methods



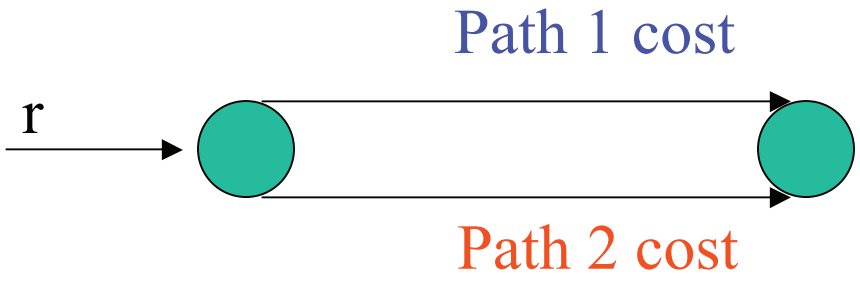
- In general finding solutions may be difficult and different types of methods may need to be applied
- Reduce cost while maintaining feasibility
- A common theme in the methods for improving solutions is to perturb current solution in a way that is feasible and reduces cost
- Why not just use minimum first derivative lengths (MFDLs)?
- Lengths are dependent on flow values x, so search is in general difficult and can lead to instability

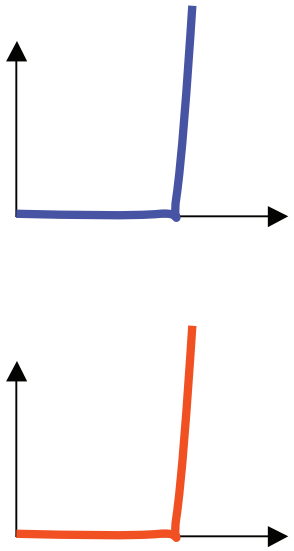


Search problems

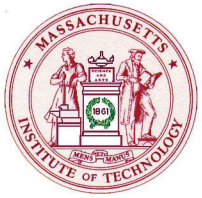


- Two path example with steep costs

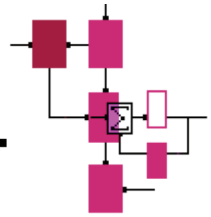




- Paths alternate having zero cost, flow is thrown from one path to another unless step size is chosen accurately and we are in the right neighborhood



General flow deviation



- General formulation:

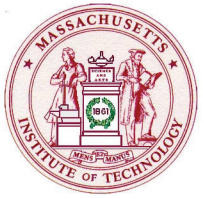
Feasibility : $\sum_{p \text{ in } P(w)} \Delta x(p) = 0$ for all w in W

overall flow conservation

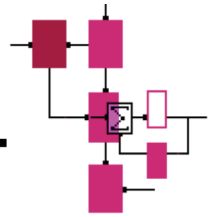
Descent direction : $\sum_{w \text{ in } W} \sum_{p \text{ in } P(w)} \frac{\partial D(x)}{\partial x(p)} \Delta x(p) \leq 0$

cost must have local reduction possibilities if any global reduction opportunities exist

(use first derivative linear cost approximation)



Frank-Wolfe



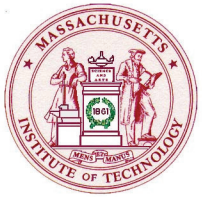
- An example of flow deviation method
- Linearly mix the current solution with the solution using only minimum first derivative lengths (MFDLs)
- Start at some x(1), x(2), …, x(n) and find MFDL
- Let $\overline{x(1)}, \overline{x(2)}, \ldots, \overline{x(n)}$ be the solution when all the flow is sent along paths with MFDL
- We set convex combination

$x(p) := x(p) + \alpha\,(\overline{x(p)} - x(p))$ for all p in P(w), all w in W

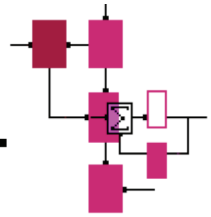
α is selected so that it minimizes the cost for the x(p) assignment above

α is the same for all paths

Problem: very slow in general



Slow convergence of Frank-Wolfe



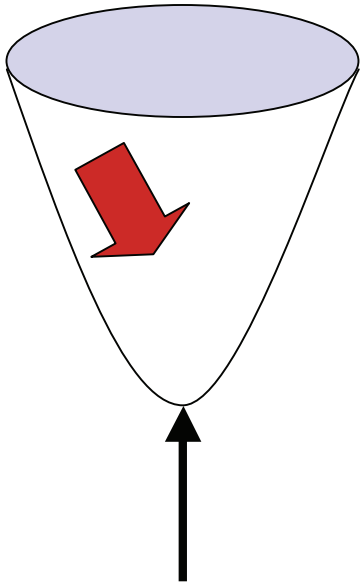
- Let us revisit the two path example
- Suppose we should be in case 1 (all flow along first path), but that we currently have some small flow along path 2 also
- How do we converge to a solution in which we get rid of the flow along path 2?

$$\text{MFDL for path 2}: \frac{C_2}{(C_2 - x(2))^2}$$

- Suppose that we have very large C_2 - then the step taken by F-W is very small, and the behavior is dominated by C_2 rather than by the changing flow
- While each step yields an improvement, the convergence is very slow

Steepest descent

Gradient of some function f : $\nabla f(\mathrm{x}) = \begin{bmatrix} \frac{\partial f(\mathrm{x})}{\partial \mathrm{x}(1)} \\ \dots \\ \frac{\partial f(\mathrm{x})}{\partial \mathrm{x}(n)} \end{bmatrix}$

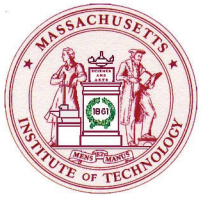


Descent direction

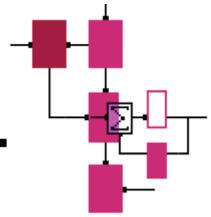
Optimum point

Hessian : $\nabla^2 f(\mathrm{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathrm{x})}{\partial \mathrm{x}(1)^2} \dots & \frac{\partial^2 f(\mathrm{x})}{\partial \mathrm{x}(1)\partial \mathrm{x}(n)} \\ \dots & \dots \\ \frac{\partial^2 f(\mathrm{x})}{\partial \mathrm{x}(n)\partial \mathrm{x}(1)} \dots & \frac{\partial^2 f(\mathrm{x})}{\partial \mathrm{x}(n)^2} \end{bmatrix}$

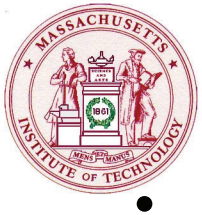
Convex function: the Hessian is positive semidefinite



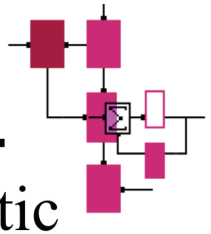
Outline



- Rerouting
- Path-based recovery
- Rings
- Beyond rings
- Beyond rerouting : codes



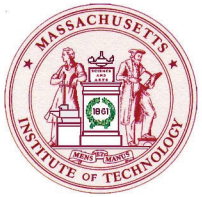
Rerouting



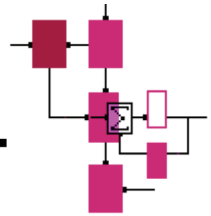
- We have considered how to route when we have a static network, but we must also consider how to react when we have changes, in particular when we need to avoid a location because of failures or because of congestion

- Preplanned:
  - fast (ms to ns)
  - typically a large portion of the whole network is involved in re-routing
  - traditionally combines self-healing rings (SHRs) and diversity protection (DP) => constrains topology
  - hard-wired
  - all excess capacity is preplanned

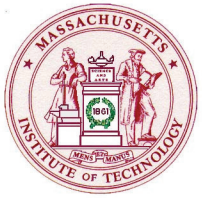
- Dynamic:
  - slow (s to mn)
  - typically localized and distributed
  - well-suited to mesh networks => more flexibility in topology
  - software approach
  - uses real-time availability of spare capacity



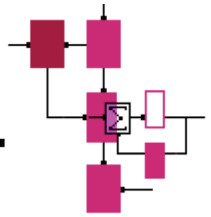
Example of rerouting in the IP world



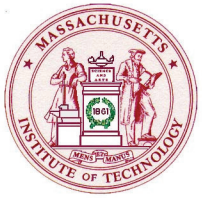
- Internet control message protocol (ICMP)
- Gateway generates ICMP error message, for instance for congestion
- ICMP redirect: “ipdirect” specifies a pointer to a buffer in which there is a packet, an interface number, pointer to a new route
- How do we get new route?
  - First: check the interface is other than the one over which the packet arrives
  - Second: run “rtget” (route get) to compute route to machine that sent datagram, returns a pointer to a structure describing the route
- If the failure or congestion is temporary, we may use flow control instead of a new route



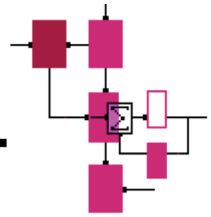
Rerouting for ATM



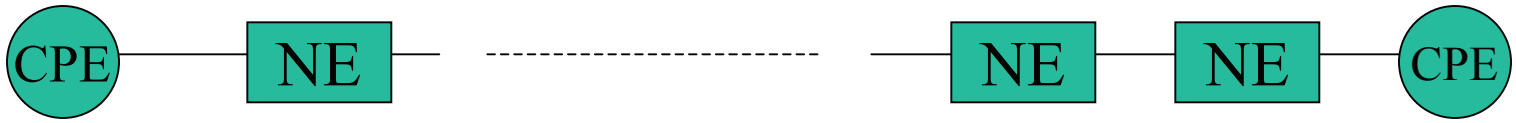
- ATM is part datagram, part circuit oriented, so recovery methods span many different types
- Dynamic methods release connections and then seek ways of re-establishing them: not necessarily per VP or VC approach
  - private network to network interface (PNNI) crankback
  - distributed restoration algorithms (DRAs)
- Circuit-oriented methods often have preplanned component and work on a per VC, VP basis
  - dedicated shared VPs, VCs or soft VPs, VCs



PNNI self-healing

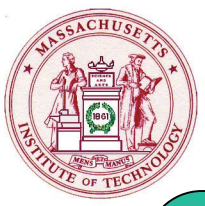


- PNNI is how ATM switches talk to each other
- Around failure or congestion area, initiate crankback
- End equipment (CPE: customer premise equipment) initiates a new connection
- In phase 2 PNNI, automatic call rerouting, freeing up CPEs from having to instigate new calls, the ATM setup message includes a request for a fault-tolerant connection

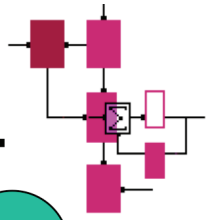


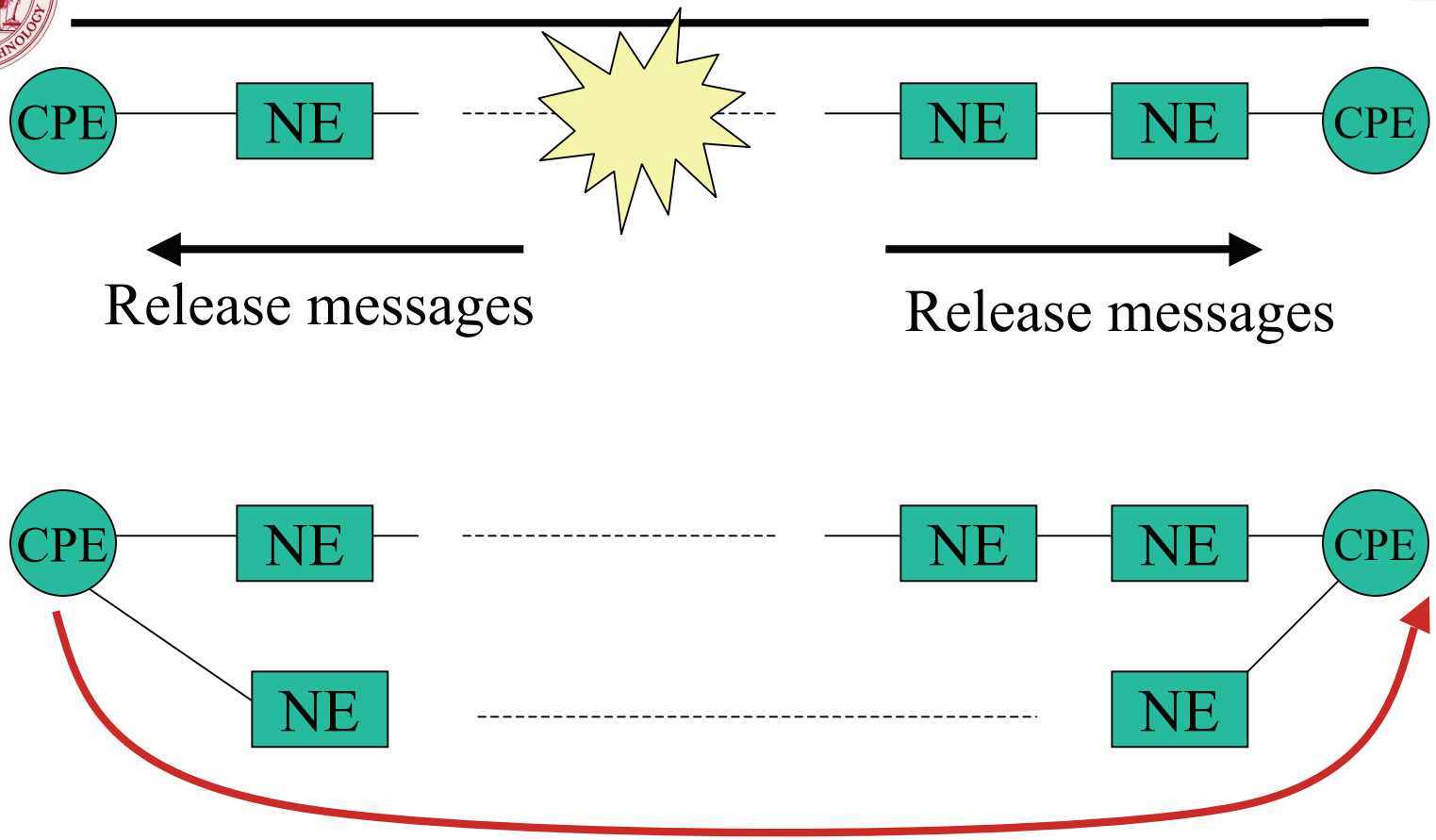
Network element

Before failure



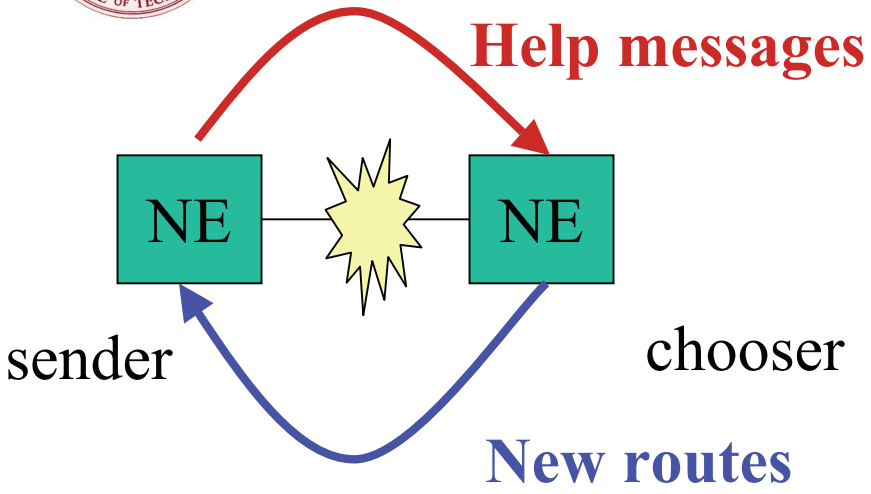
Connection re-establishment

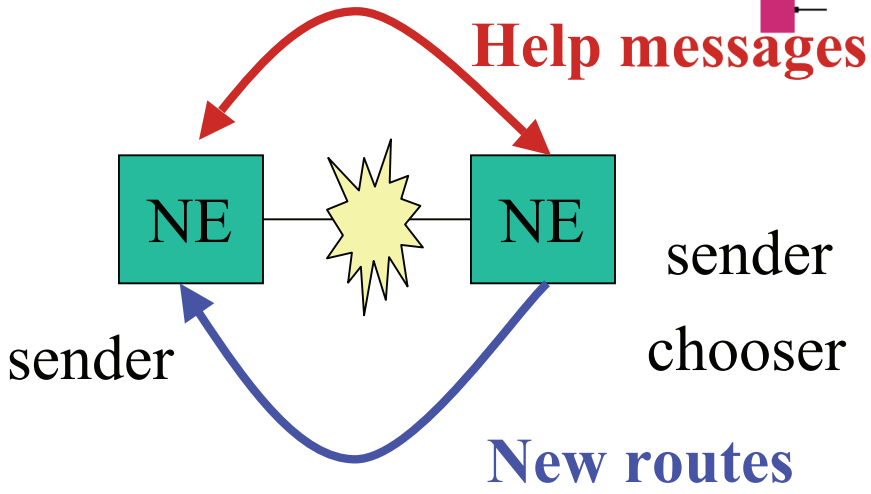


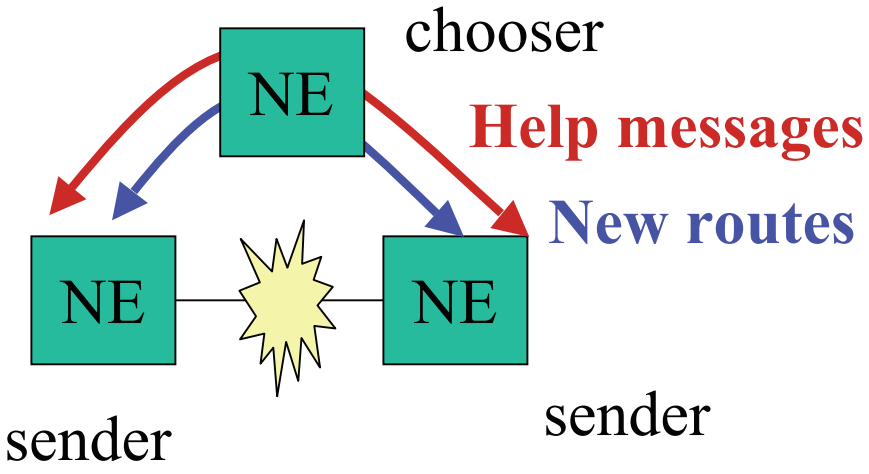


Issue: the congestion may cascade, giving unstable conditions, which cause an ATM storm

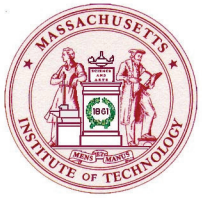
DRAs



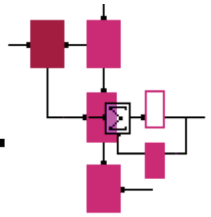




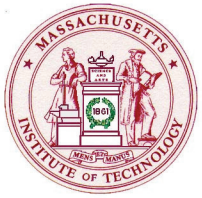
The DRAs have at least one end node transmit help messages to some nodes around them, usually within a certain hop radius, and new routes, possible splitting flows, are selected and used



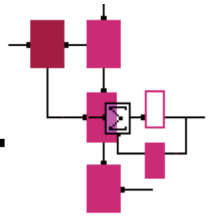
Circuit-oriented methods



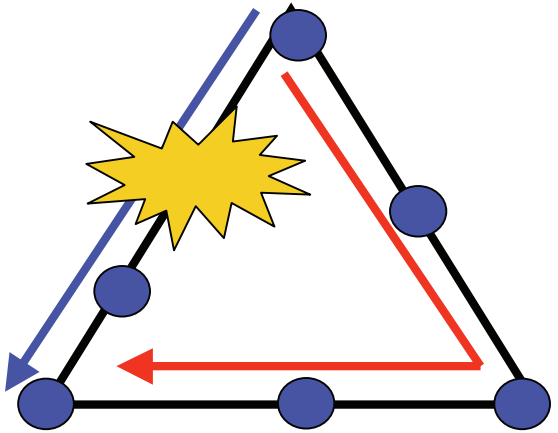
- Circuit-oriented methods seek to replace a route with another one, whether end-to-end or over some portion that is affected by a failure
- Several issues arise:
  - How do we perform recovery in a bandwidth-efficient manner
  - How does recovery interface with network management
  - What sort of granularity do we need
  - What happens when a node rather than a link fails



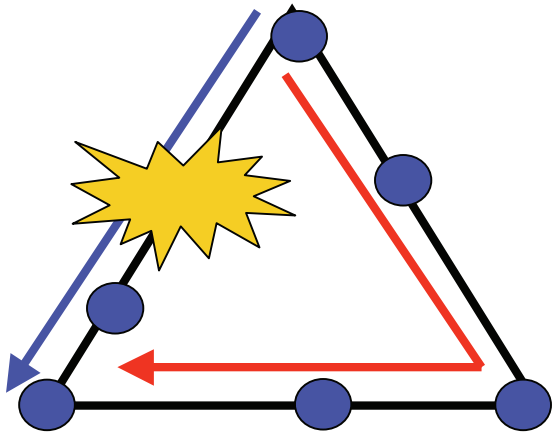
Path-based Methods



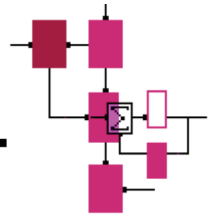
- Live back-up
  - backup bandwidth is dedicated
  - only receiver is involved
  - fast but bandwidth inefficient



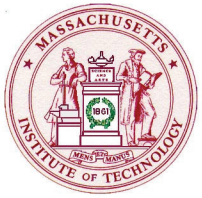
- Failure triggered back-up
  - backup bandwidth is shared
  - sender and receiver are involved
  - slow but bandwidth efficient



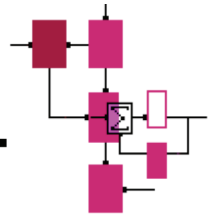
Link/Node Restoration

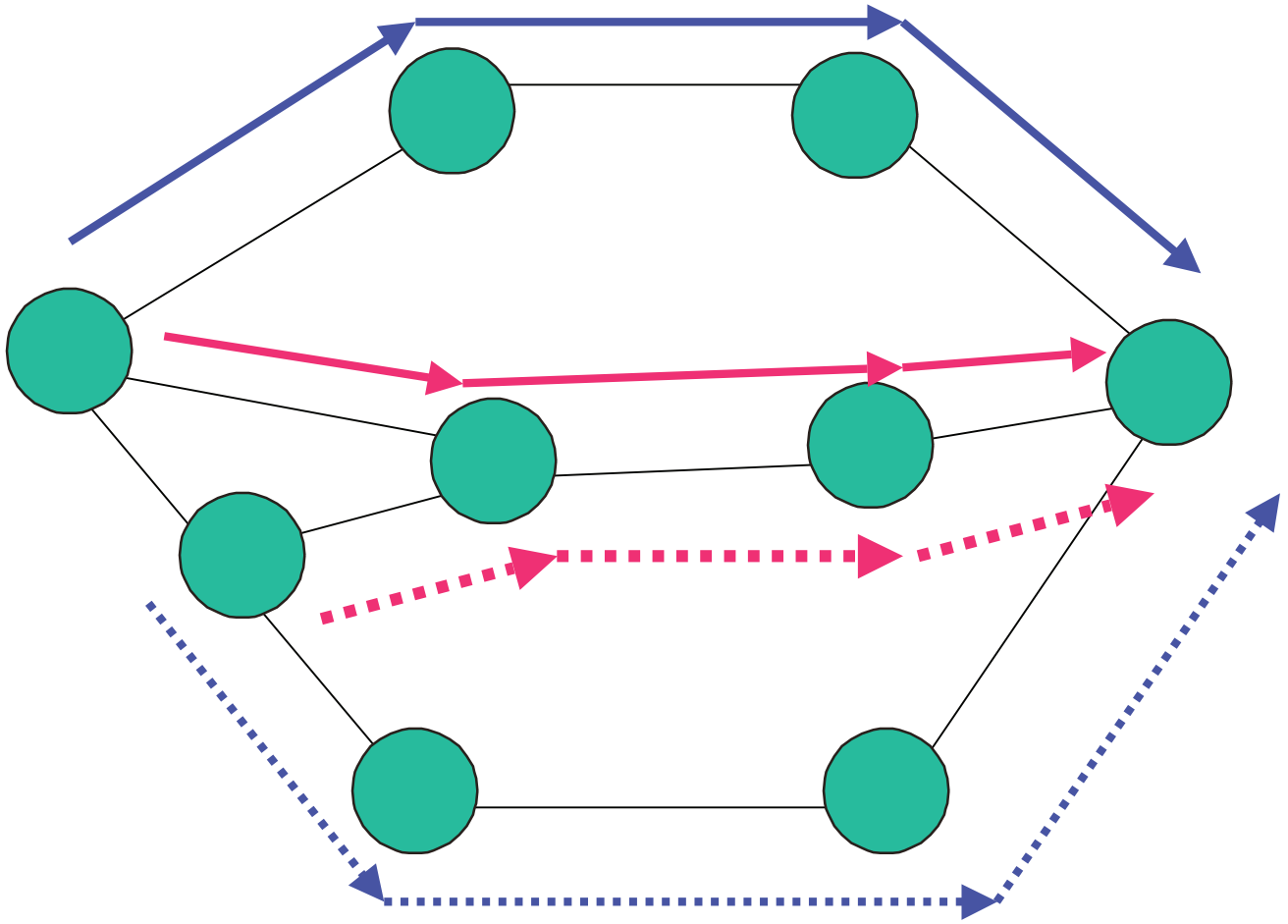


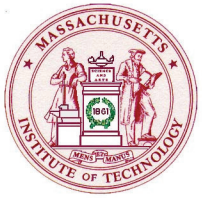
- Consider a link or the links going through a node as a path (or paths)
- Recover those paths in a preplanned manner
- Network now performs recovery locally and independently of actual connections
- Speed and capacity efficiency are between those of the path based methods



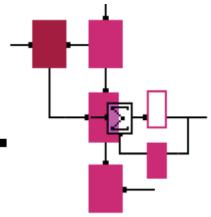
Path Protection Allows Sharing



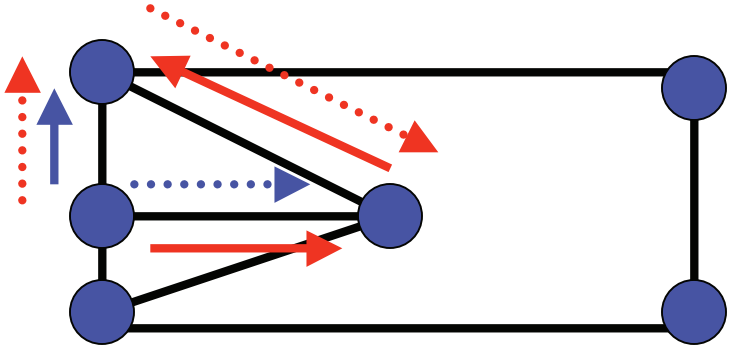


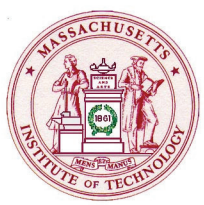


Path Rerouting on Redundant Graphs

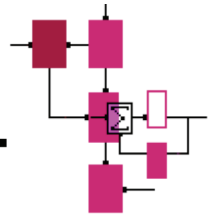


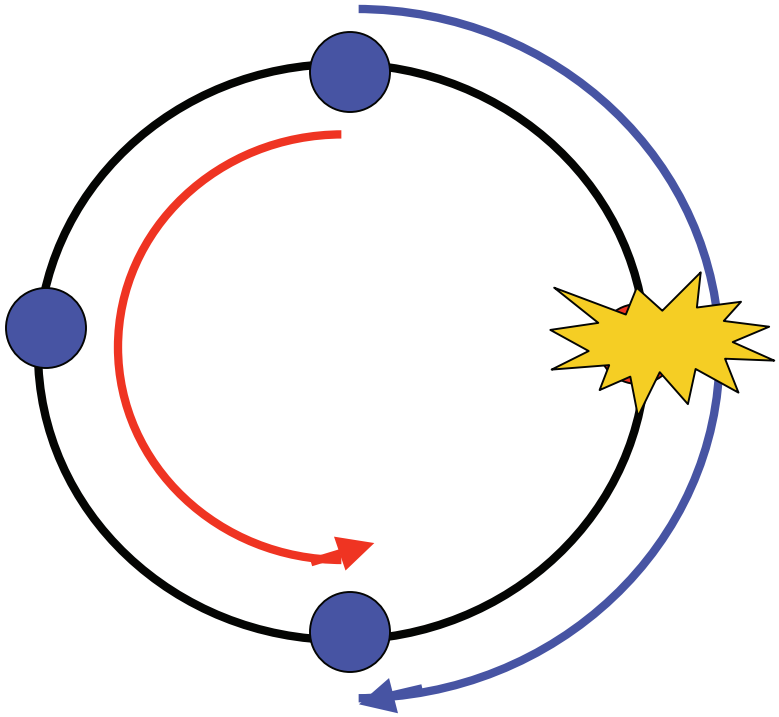
- Minimum topological redundancy:
  - 2-edge connected for resilience against link failure
  - 2-vertex connected for resilience against node failure
- Menger's theorem states that we can find two edge (vertex) disjoint paths between any pair of nodes

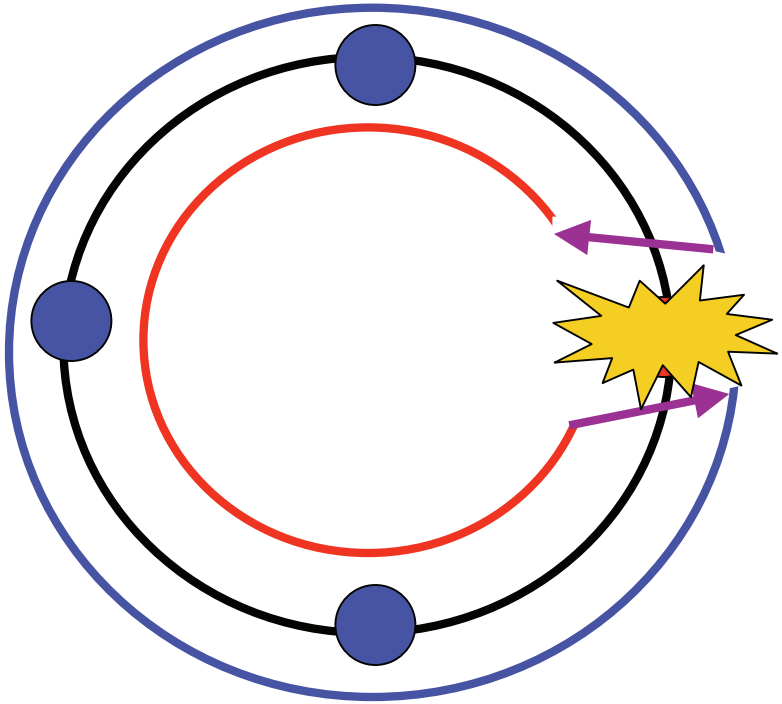




Rings: Path Rerouting, Link/Node Rerouting

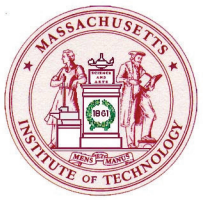




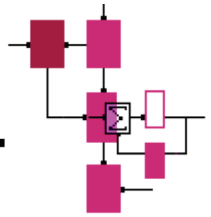


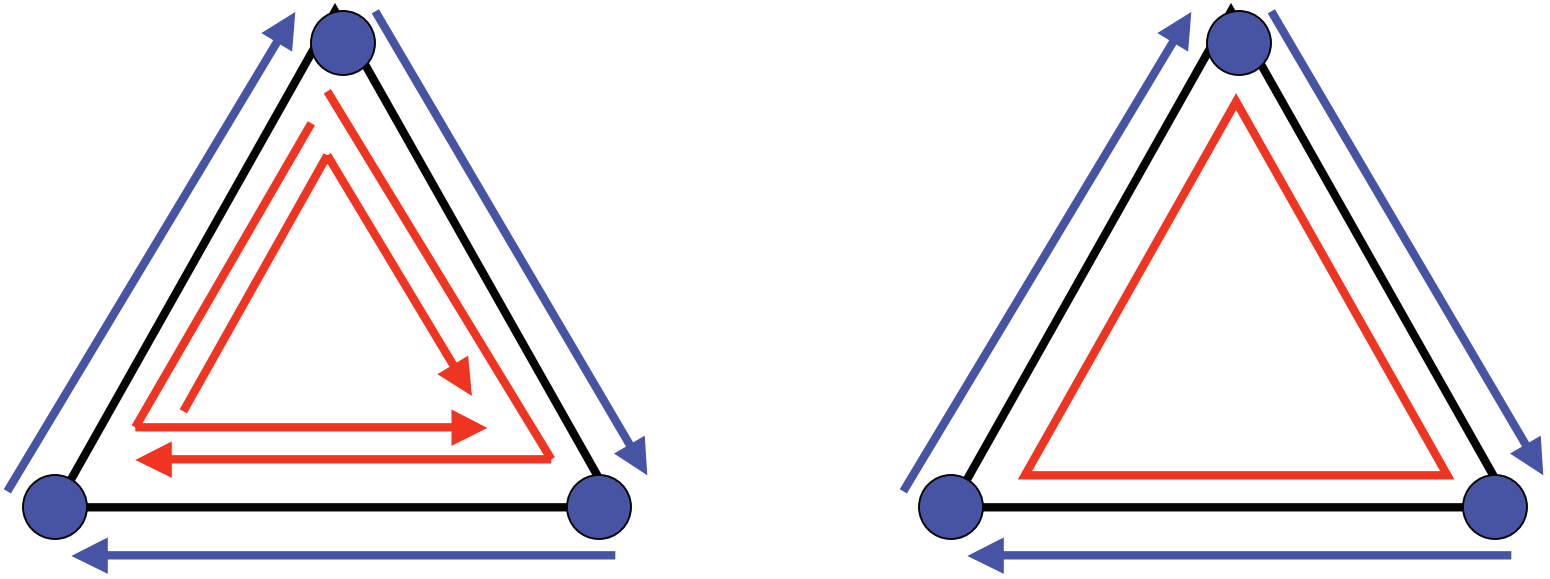
UPSR: automatic path switching

BLSR: link/node rerouting



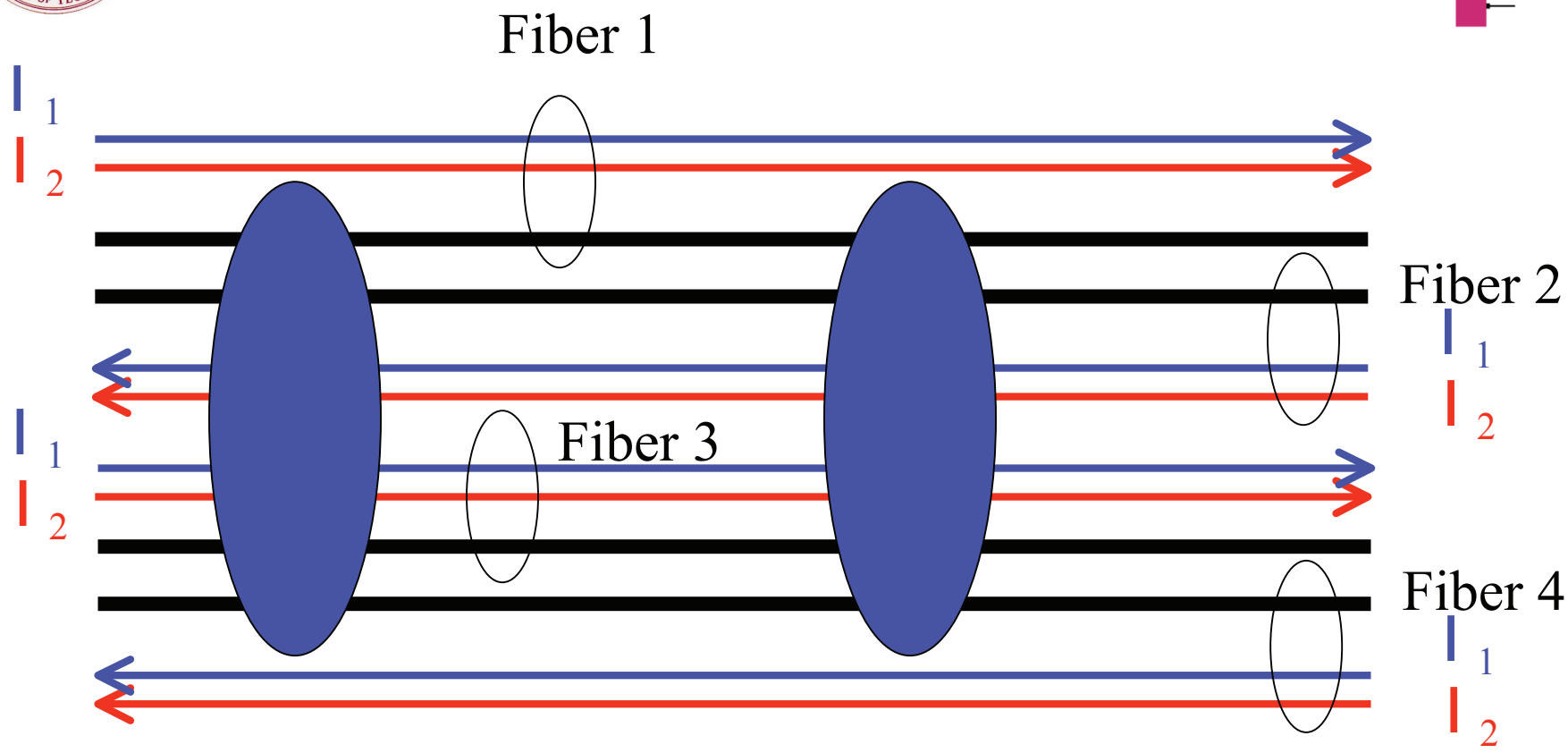
Path Rerouting Requires more Bandwidth



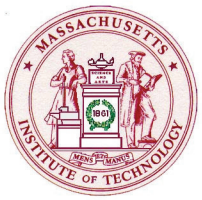


Path rerouting requires one backup path per primary path
Link/node rerouting allows sharing of resources

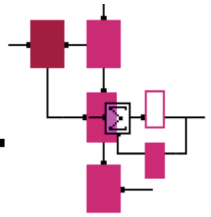
Traditional Fiber-based Loop-back



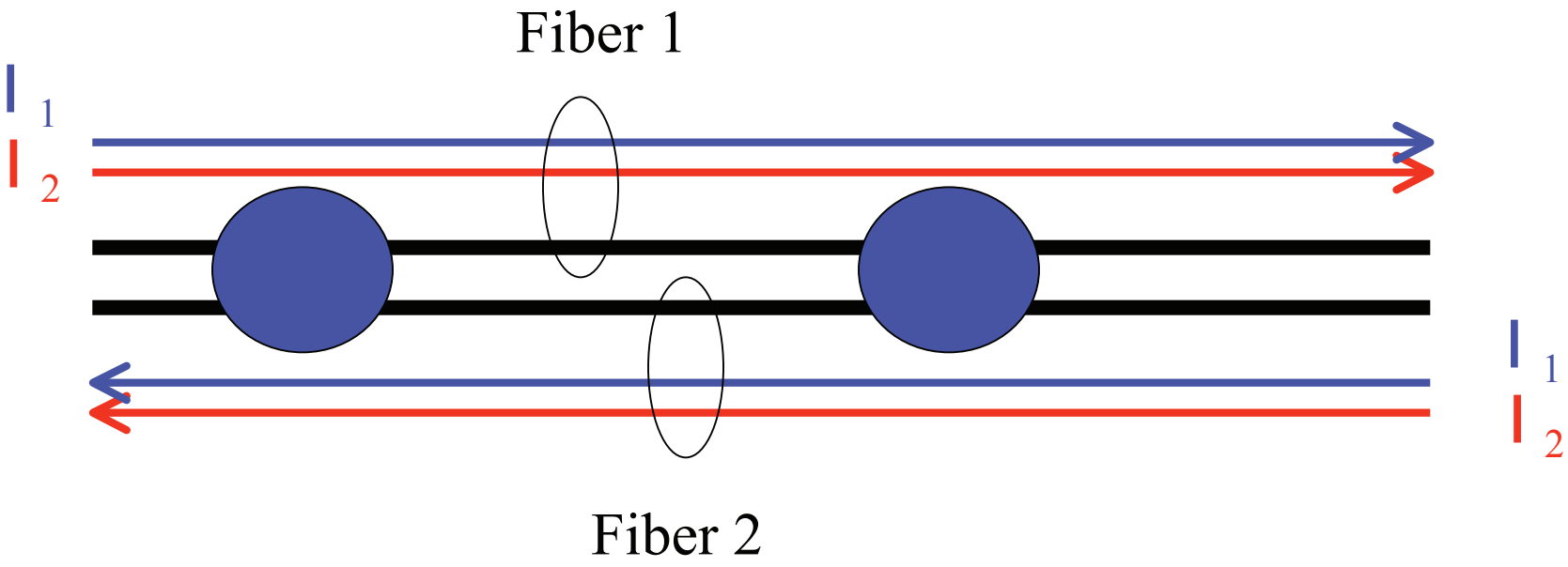
Primary traffic is carried by Fiber 1 and by Fiber 2. Backup is provided by Fiber 3 for Fiber 1 and by Fiber 4 for Fiber 2.



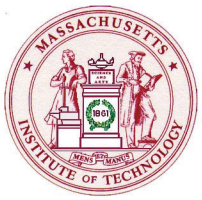
WDM-based Loop-back



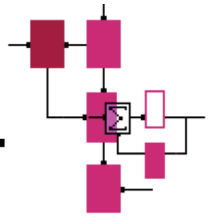
Two-fiber system

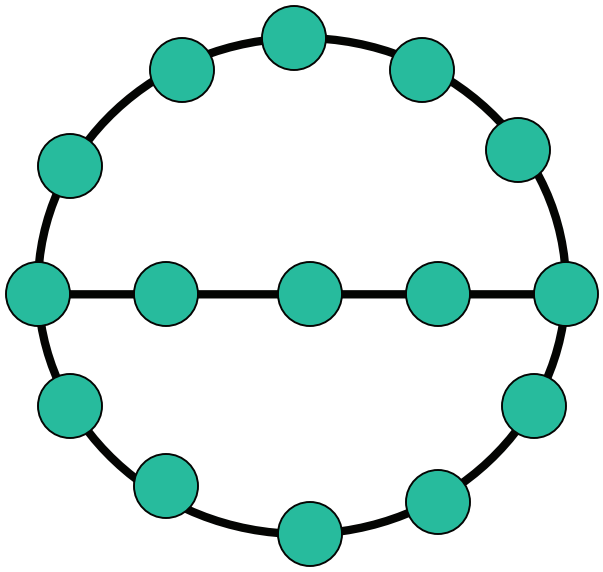


Primary traffic is carried by Fiber 1 on λ_1 and by Fiber 2 on λ_2.
Backup is provided by on λ_1 on Fiber 2 for λ_1 on Fiber 1.
λ_2 on Fiber 2 is backed up by λ_2 on Fiber 1.

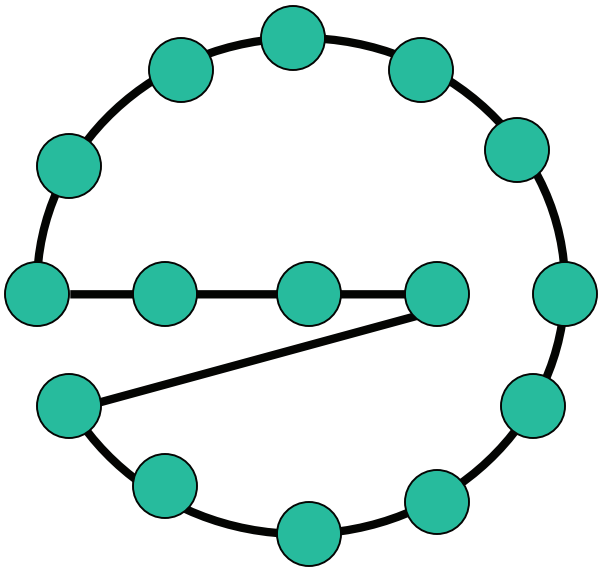


Why rings may be expensive

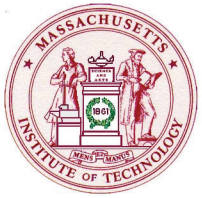




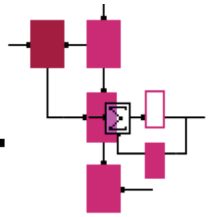
Minimum fiber length



Minimum fiber length for a ring

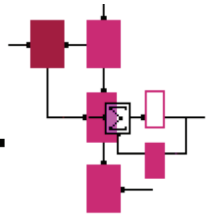


Covers of Rings

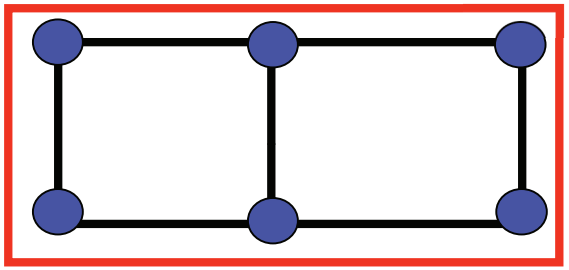


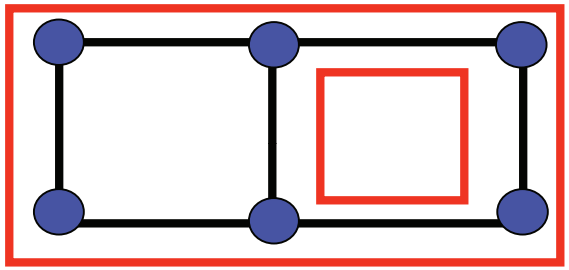
- Usual method for applying preplanned recovery to mesh networks
- UPSR style covers: minimum cycle cover is NP complete problem (conjectured by Itai, Lipton, Papadimitriou and Rodeh, shown by Thomassen)
- BLSR style covers: double cycle cover conjecture (see Jaeger, applied to restoration by Ellinas, Stern and Hailemariam)
- Hierarchical Rings (Shi and Fonseka)
- Is there a fundamental reason for rings?

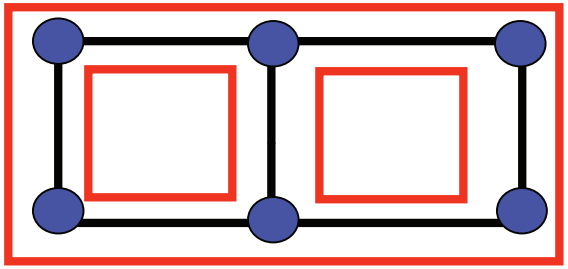
Ring-based Methods for Link Restoration

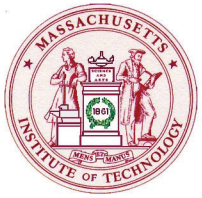


- Covering nodes with rings
- Cover each link with at least one ring
- Cover each link with exactly two rings
- Ring cover drawbacks:
  - onerous backup
  - limited choices of covers

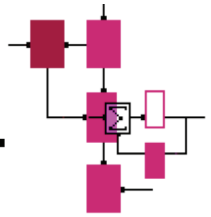


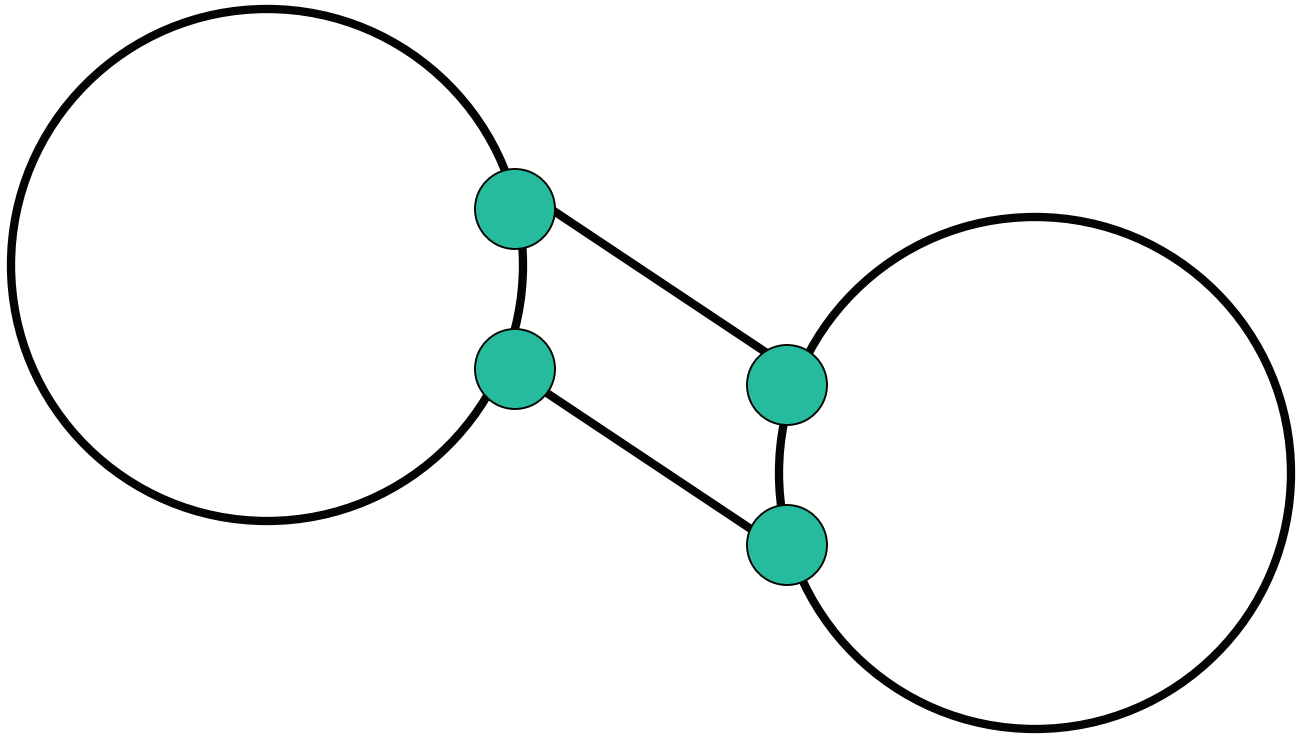




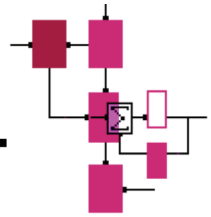


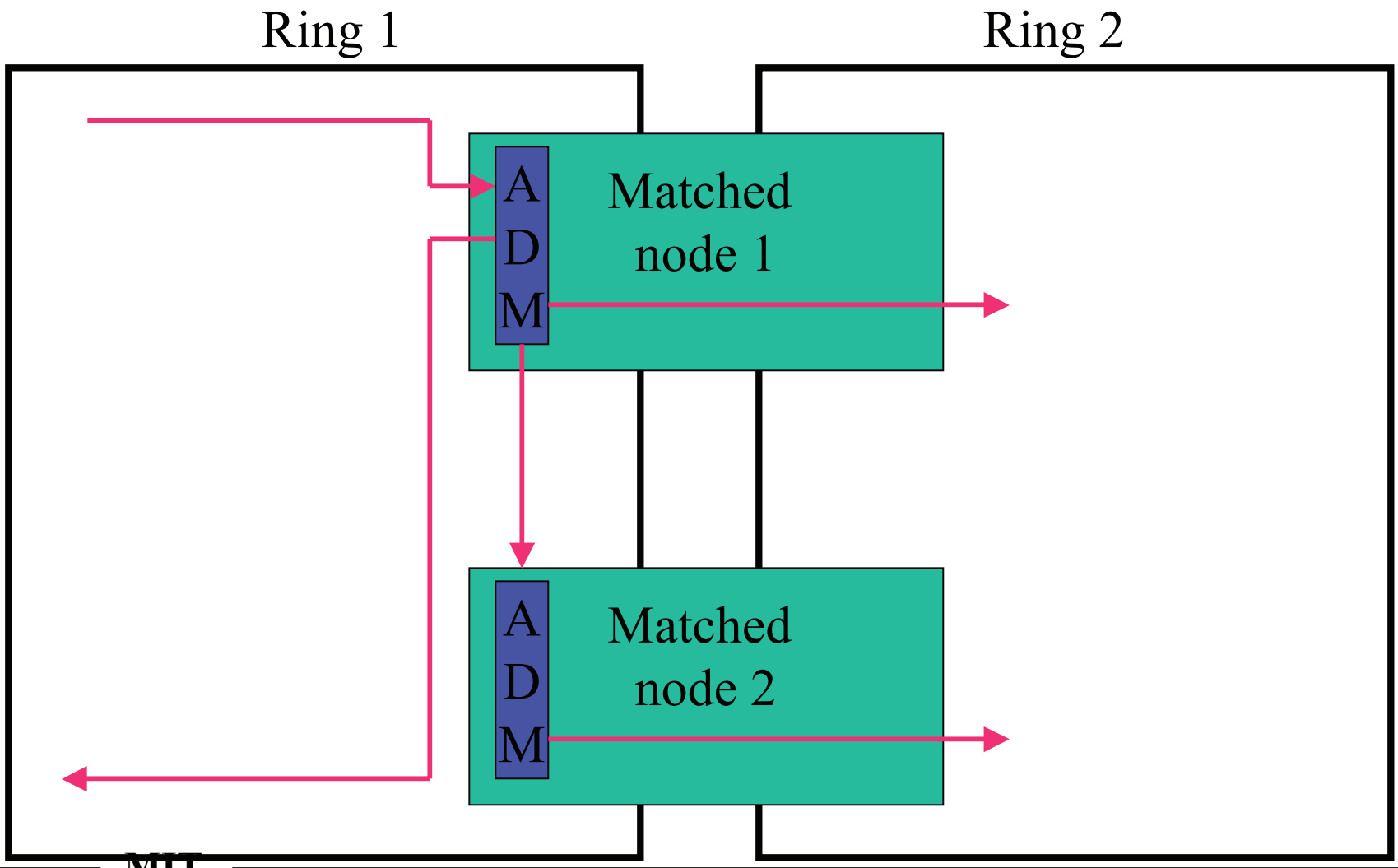
Rings and Diversity Protection





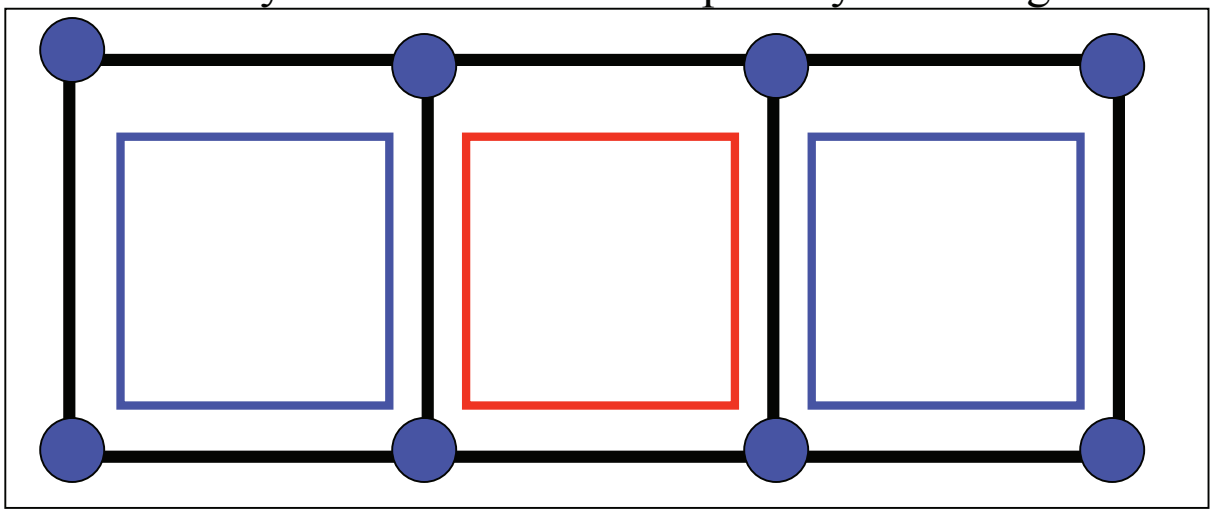
Ring Covers and Node Recovery



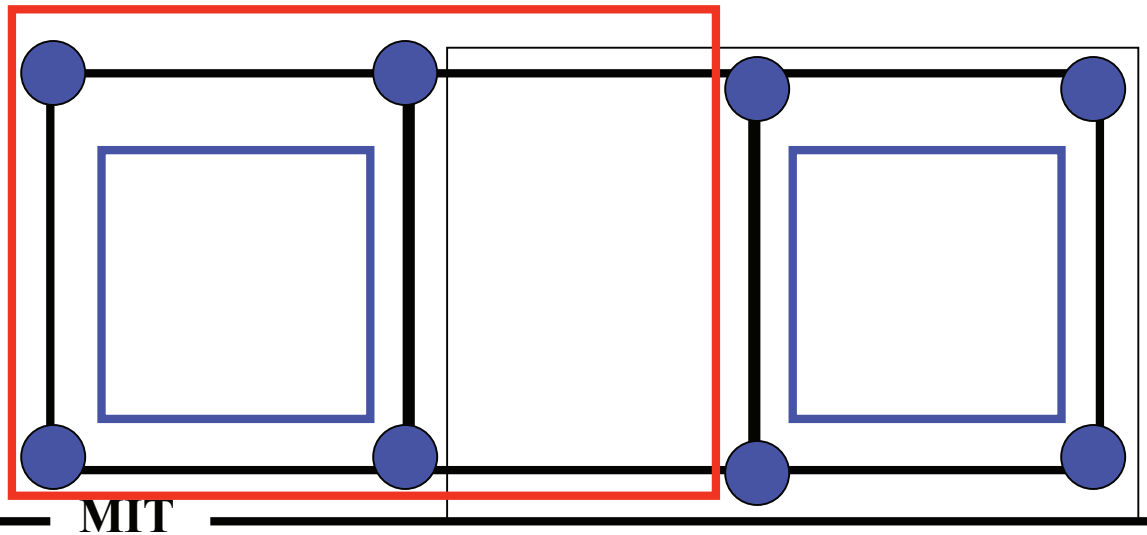


Double Cycle Covers

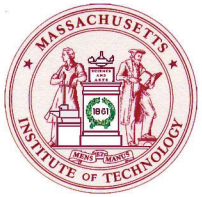
On a two-fiber system, can double cycle covers offer WDM-based recovery? We need to select a primary wavelength.



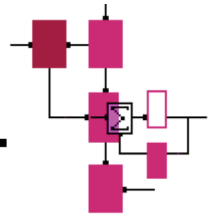
Outer ring: no wavelength assignment possible



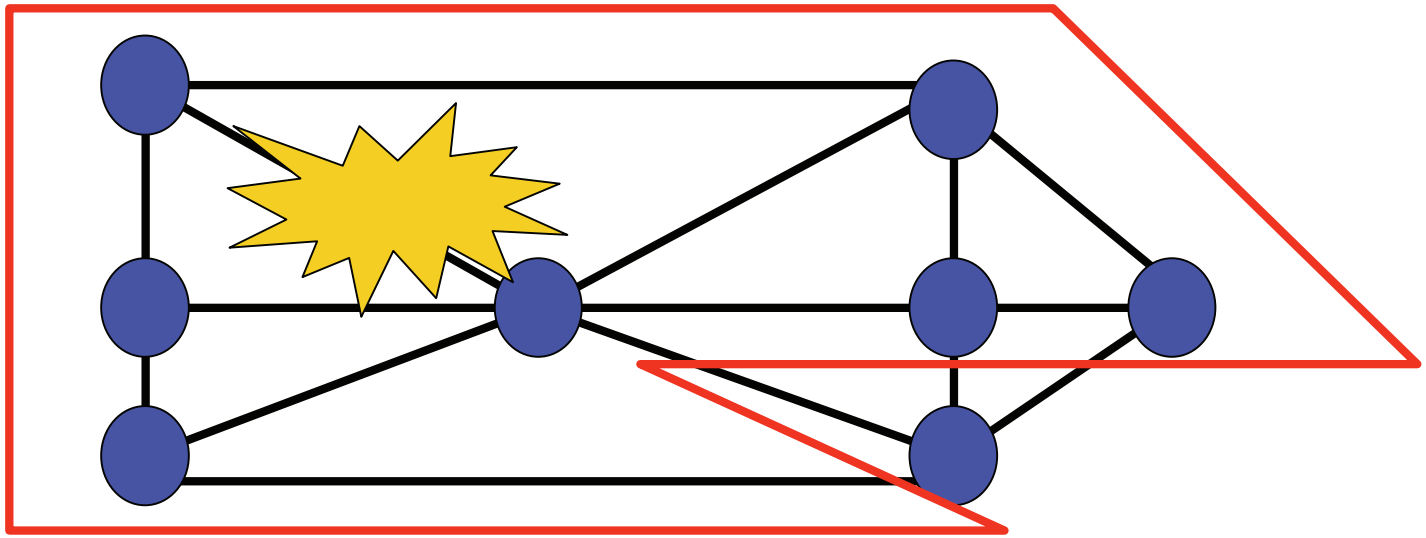
Right-most ring: no primary wavelength assignment possible



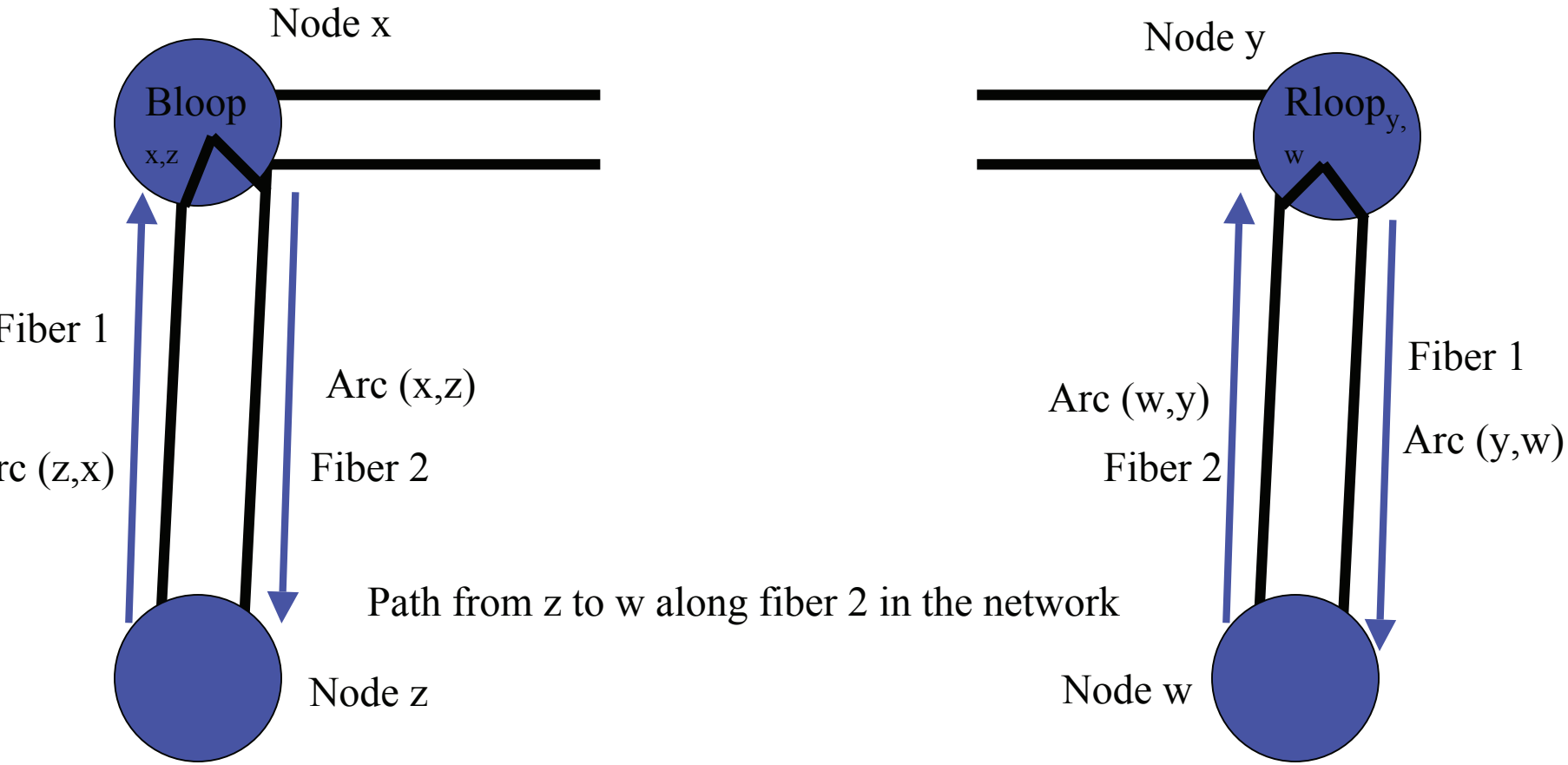
Hybrid Approach: p-cycle

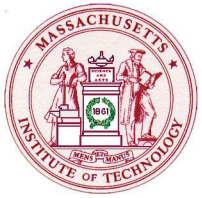


- Cycle covering nodes
- Links on cycle are recovered as on ring
- Links off cycle are recovered by using either section of cycle around failed link

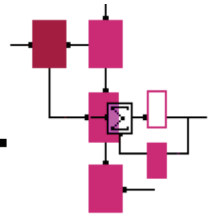


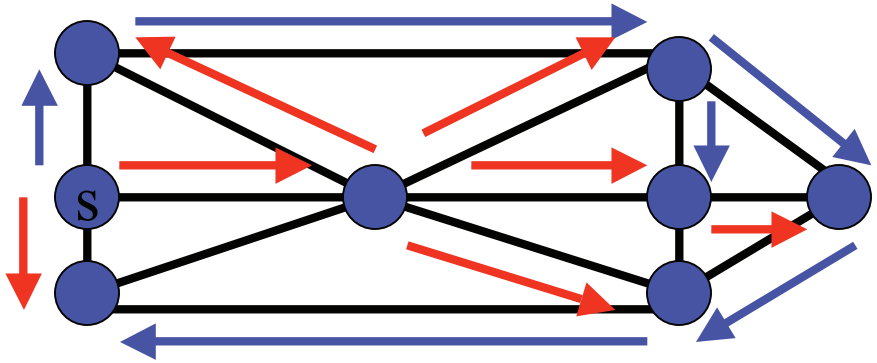
WDM Loopback Rerouting



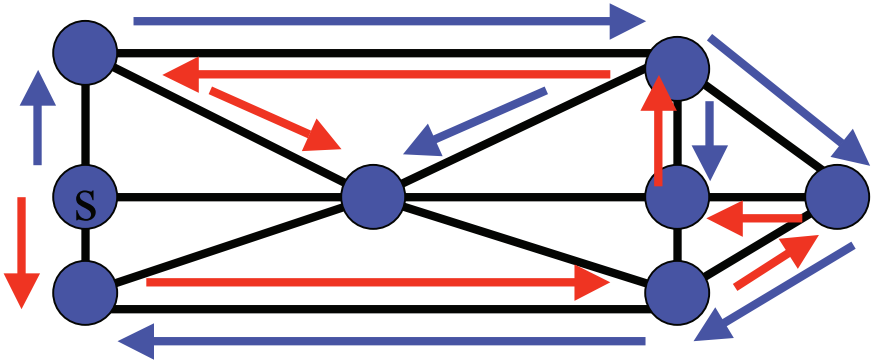


Redundancy with Trees

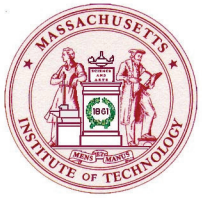




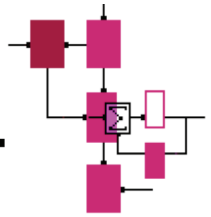
Link-disjoint spanning trees (Yener, Offek, Yung):
not applicable to node failures



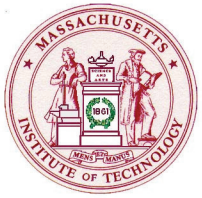
Arc-disjoint spanning trees (Tarjan; Shiloah):
not applicable to duplex link or node failures



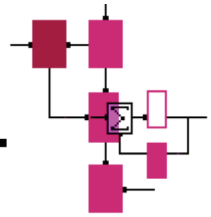
Advantages of Trees



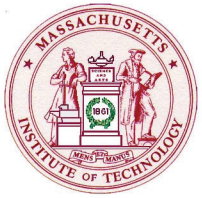
- Trees are good for many applications:
  - multicast and incast applications
  - hierarchical network management
- The problem of minimum cost multicast tree is NP-complete (Steiner tree problem)
- How to construct 2 trees which allow path rerouting for link and node failure:
  - algorithm by Itai and Rodeh based upon a labeling by Tarjan and Even



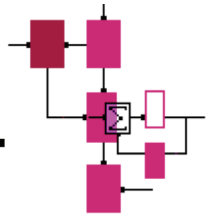
Beyond rerouting



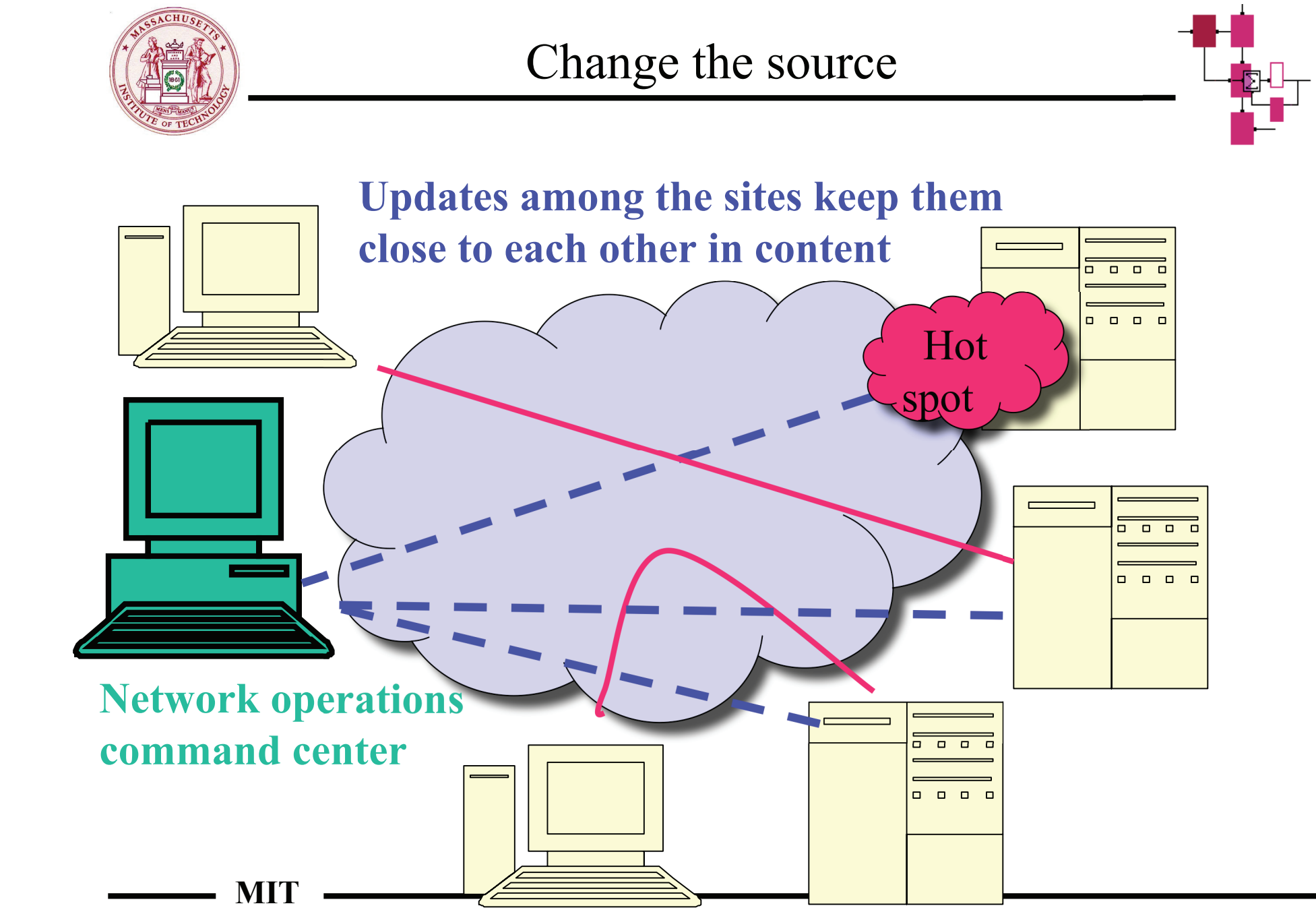
- In rerouting, the source and the destination remain the same, but we change the route between them
- Often rerouting is needed because of congestion rather than because of outright failures and rerouting may not be very useful if we have portions of the network around the origin or destination backed up
- Two new approaches go beyond rerouting:
  - Change one of the nodes to be in a less congested portion of the network
  - Do not reroute, but instead make use of randomness in the packet losses to code over packet losses

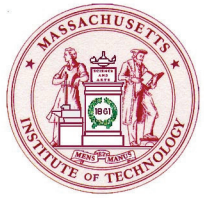


Change the source

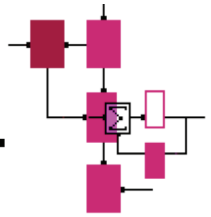


- If there is heavy demand at one location, then there will be delay in that location
- Change to another site that has better characteristics for a particular user's location
- Several sites located at the edge of the network can support a particular request
- A central controller keeps a weathermap of the Internet and assigns the best location for requests – this reduces the load on the individual sites
- Every appears to be request is served as though it was served from a single location

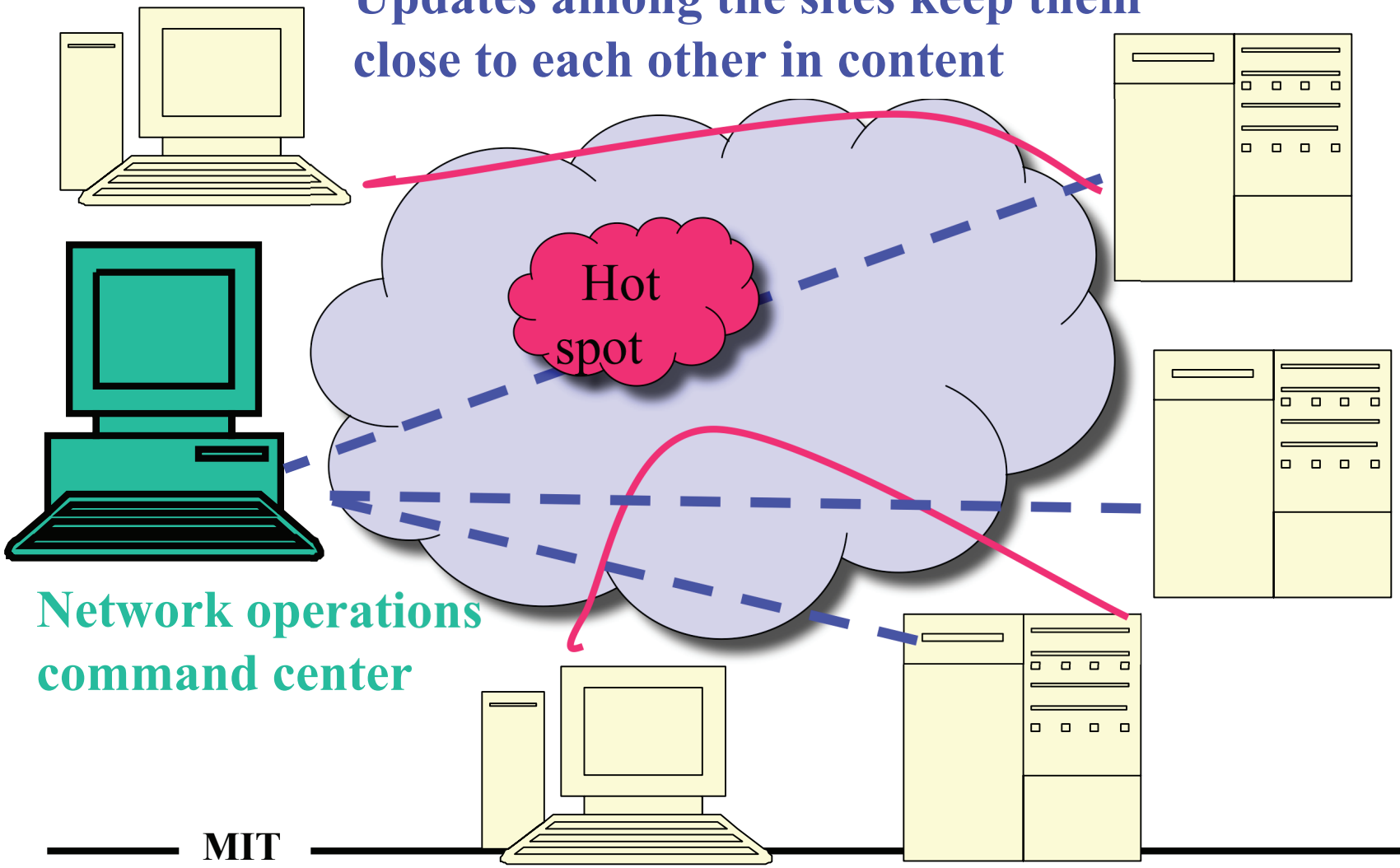

Change the source
Updates among the sites keep them close to each other in content
Hot spot
Network operations command center
MIT



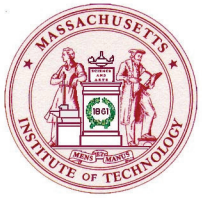
Change the source



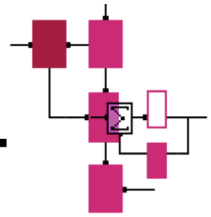
Updates among the sites keep them close to each other in content



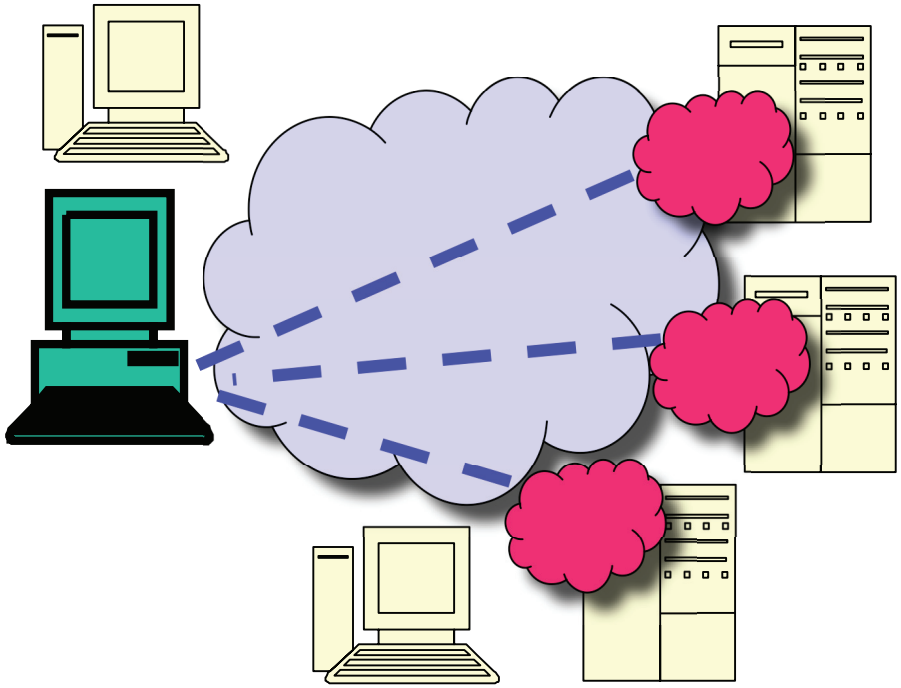
Network operations command center

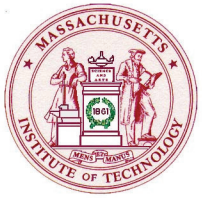


Issues with changing the source

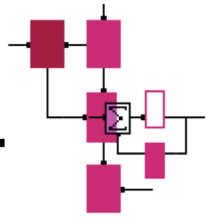


- Under certain conditions, there will be several hot spots in several parts of the network because of excessive traffic, even if the application that is being accessed in not in particularly heavy demand

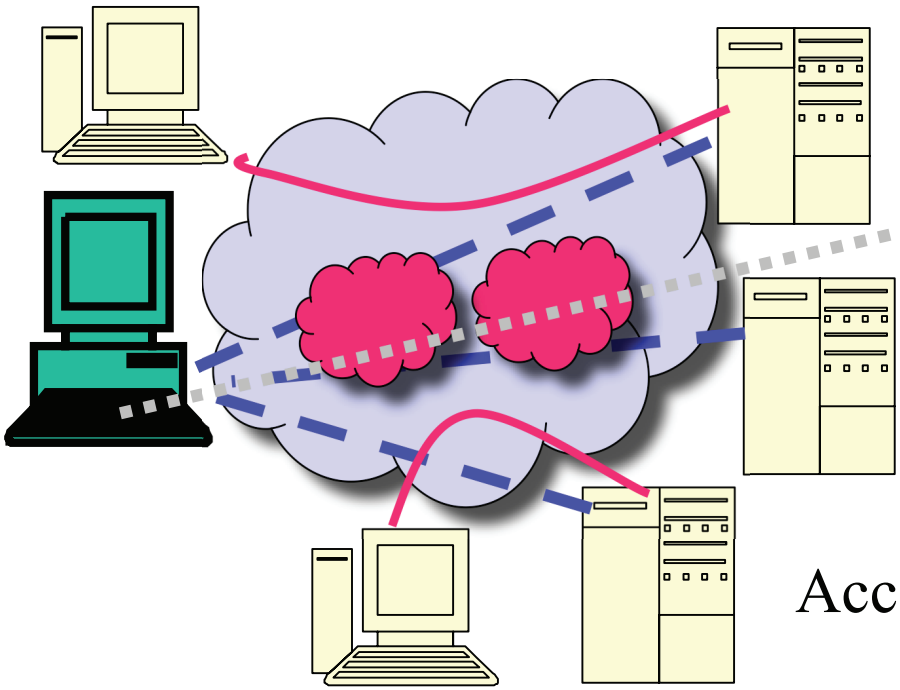




Issues with changing the source



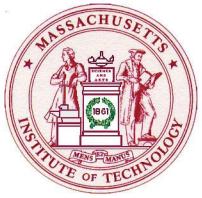
- The updates among sites need to be regular for interactive services – for instance sale of tickets, brokerage services, have to be done even if there are hotspots



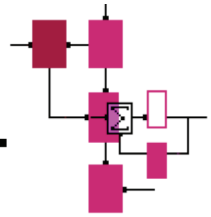
Accessing account 12345

Difficulty getting across the Network from one site to another to update information

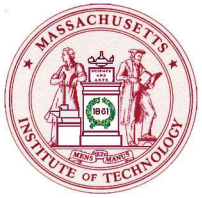
Accessing account 12345



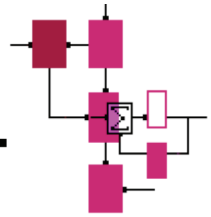
Coding across the network



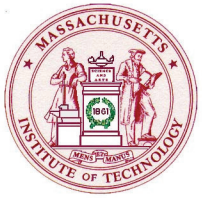
- Routing diversity to average out the loss of packets over the network
- Access several mirror sites rather than single one
- The data is then coded across packets in order to withstand the loss of packets without incurring the loss of all packets
- Rather than select the “best” route, we want routes to be diverse enough that congestion in one location will not bring down a whole stream
- This may be done with traditional Reed-Solomon erasure codes or with Tornado codes



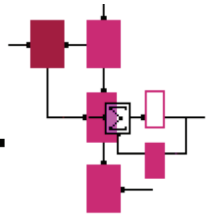
Tornado codes



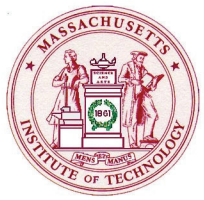
- Use a computationally inexpensive forward erasure-correction code
- For every k packets, you create parity and stop after receiving any distinct k out of (k+l) packets to reconstruct the original data.
- Packets are scheduled to minimize the number of duplicate packets received before getting k distinct ones
- Encoding/decoding time is O((l+k)P) for a P-sized packet
- For other codes such as Reed-Solomon, typically O(lkP)
- See ***A Digital Fountain Approach to Reliable Distribution of Bulk Data*** by John Byers, Michael Luby, Michael Mitzenmacher, Ashutosh Rege



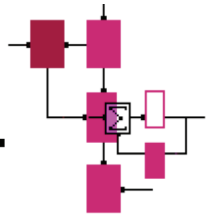
The digital fountain approach

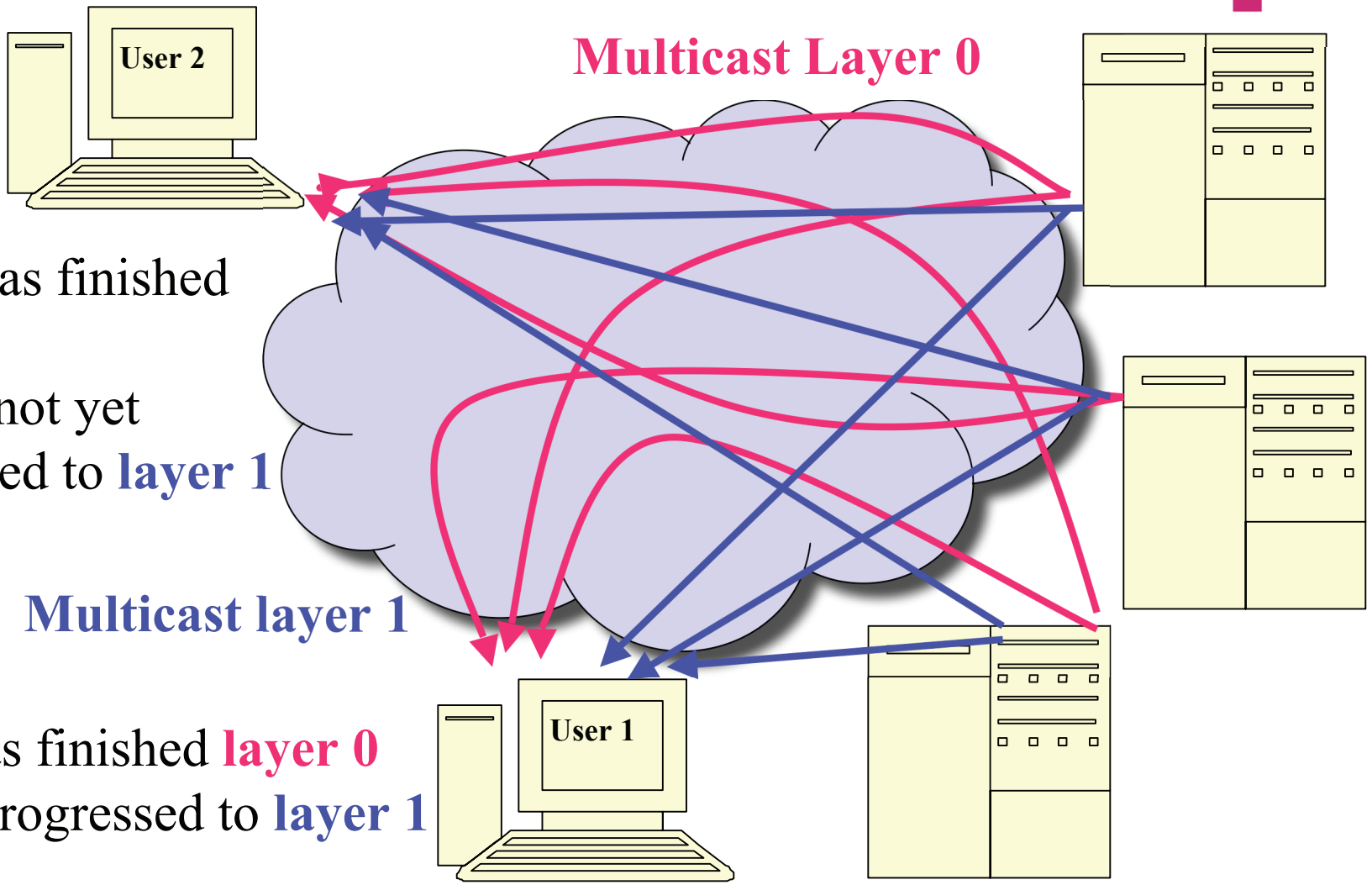


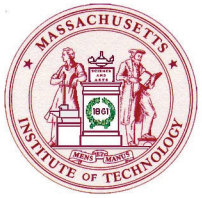
- Idea: have users tune in whenever they want, and receive data according to the bandwidth that is available at their location in the network – “fountain” because the data stream is always on
- Create multicast layers: each layer has twice the bandwidth of the lower layer (think of progressively better resolution on images, for instance), except for the first two layers
- If receiver stays at same layer throughout, and packet loss rate is low enough, then receiver can reconstruct source data before receiving any duplicate packets : "One-level property"
- Receivers can only subscribe to higher layer after seeing a*synchronization point* (SP) in their own layer
- The frequency of SPs is inversely proportional to layer BW,



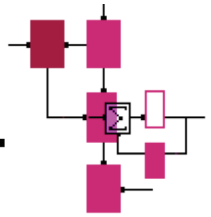
Digital fountain





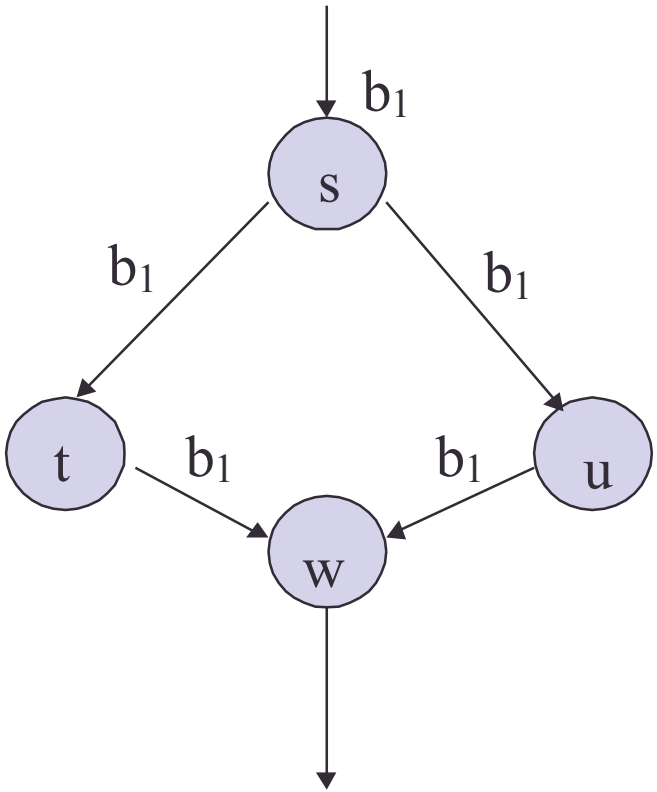


Issues



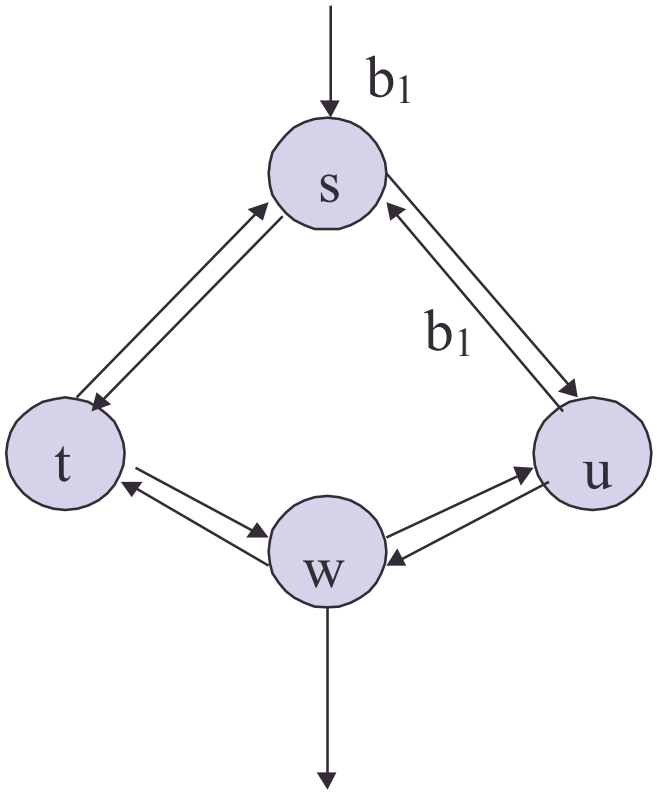
- The open-loop multicast aspect is attractive – no central node keeping track of the network
- The coding seems to be fast
- Many questions remain:
  - How much delay is induced because of the SPs and how does this compare to whatever coding gains may be obtained?
  - What is the effect on the higher layer of these delays?
  - How much network congestion is generated by having the fountain on at all times and at all points?
  - How do we do proper interleaving of the data and how do codes for such interleaving interact with Tornado codes?

Rerouting as a code



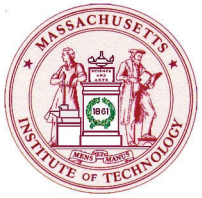
$\overline{s} \cdot d_{t,w} + s \cdot d_{u,w} = b_1$

a. Live path protection

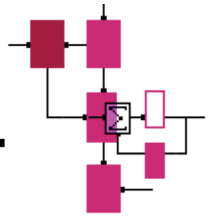


$d_{t,w} + d_{u,w} = b_1$

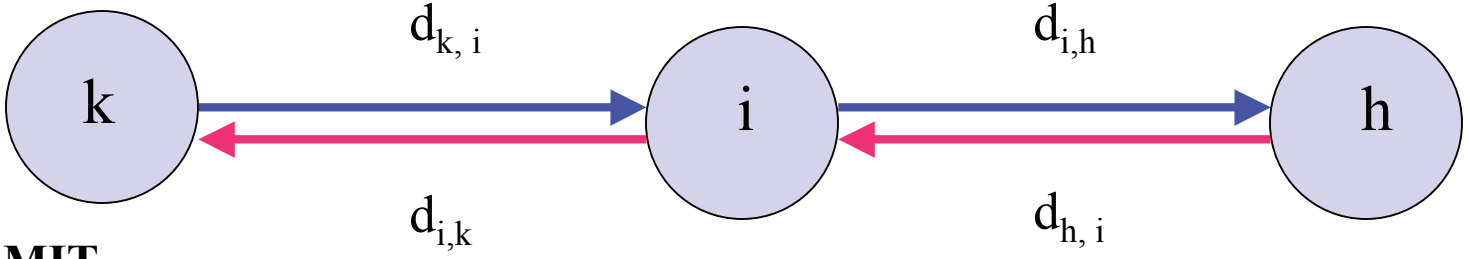
b. Link recovery

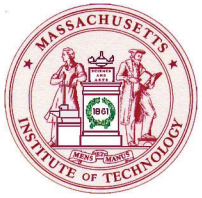


Rerouting as a code

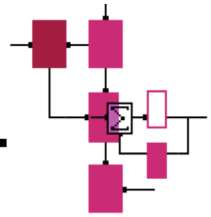


- Live path protection: we have an extra supervisory signal s =1 when the primary path is live, s =0 otherwise
- Failure-triggered path protection: the backup signal is multiplied by $\bar{s}$
- Link recovery:
  - $d_{i,h} = d_{k,i} + d_{h,i}$ for the primary link (i, h) emanating from i, where (k, i) is the primary link into i and (h, i) is the secondary link into i
  - for secondary link emanating from i, the code is $d_{i,k} = d_{i,h} \cdot s_{i,h} + d_{k,i}$

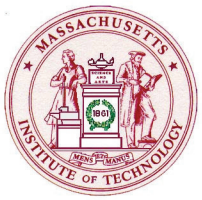




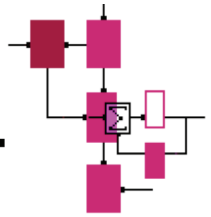
Codes and routes



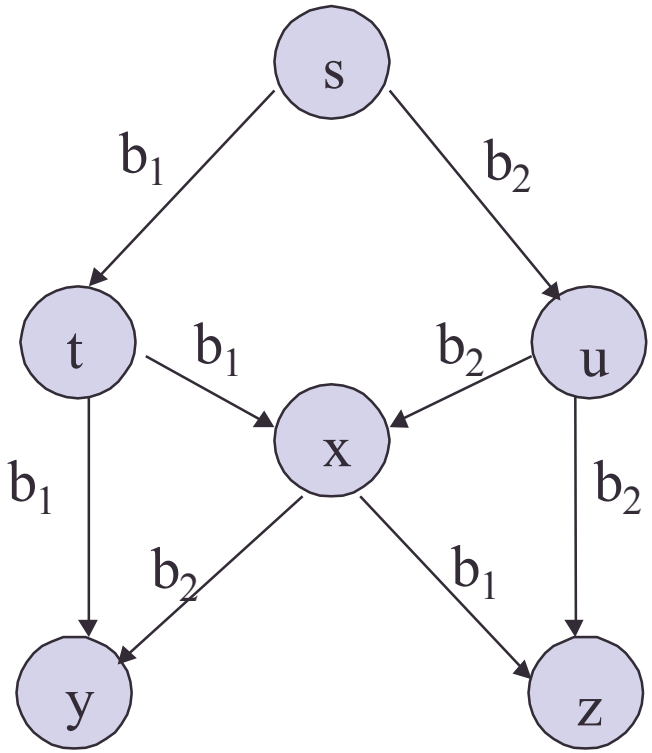
- In effect, every routing and rerouting scheme can be mapped to some type of code, which may involve the presence of a network management component
- Thus, removing the restrictions of routing can only improve performance - can we actively make use of this generality?

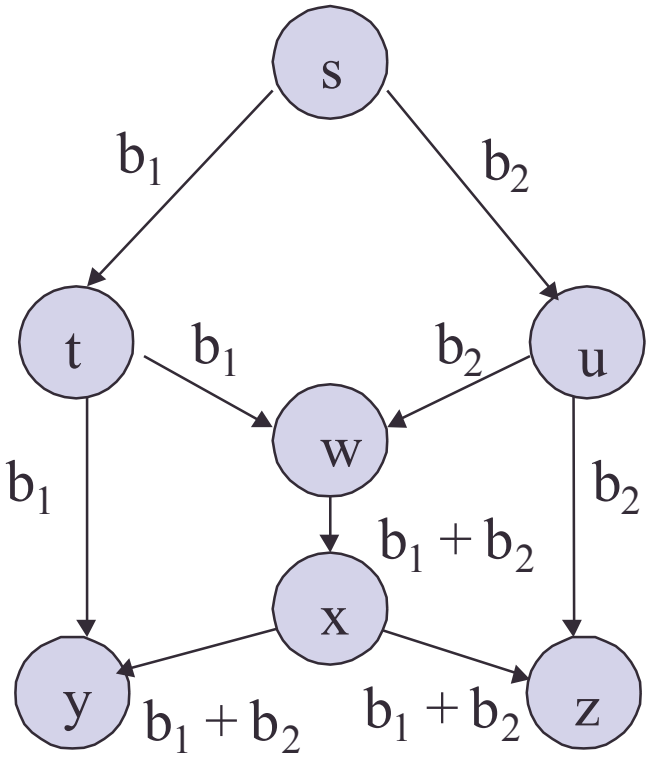


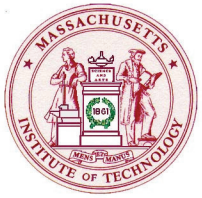
Network coding



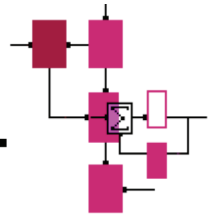
- The canonical example



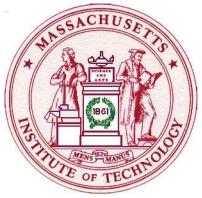




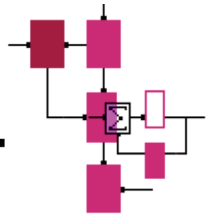
Network coding vs. Coding for networks



- The source-based approaches consider the networks as in effect channels with ergodic erasures or errors, and code over them, attempting to reduce excessive redundancy
- The data is expanded, not combined to adapt to topology and capacity
- Underlying coding for networks, traditional routing problems remain, which yield the virtual channel over which coding takes place
- Network coding subsumes all functions of routing - algebraic data manipulation and forwarding are fused

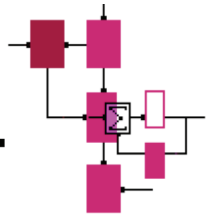


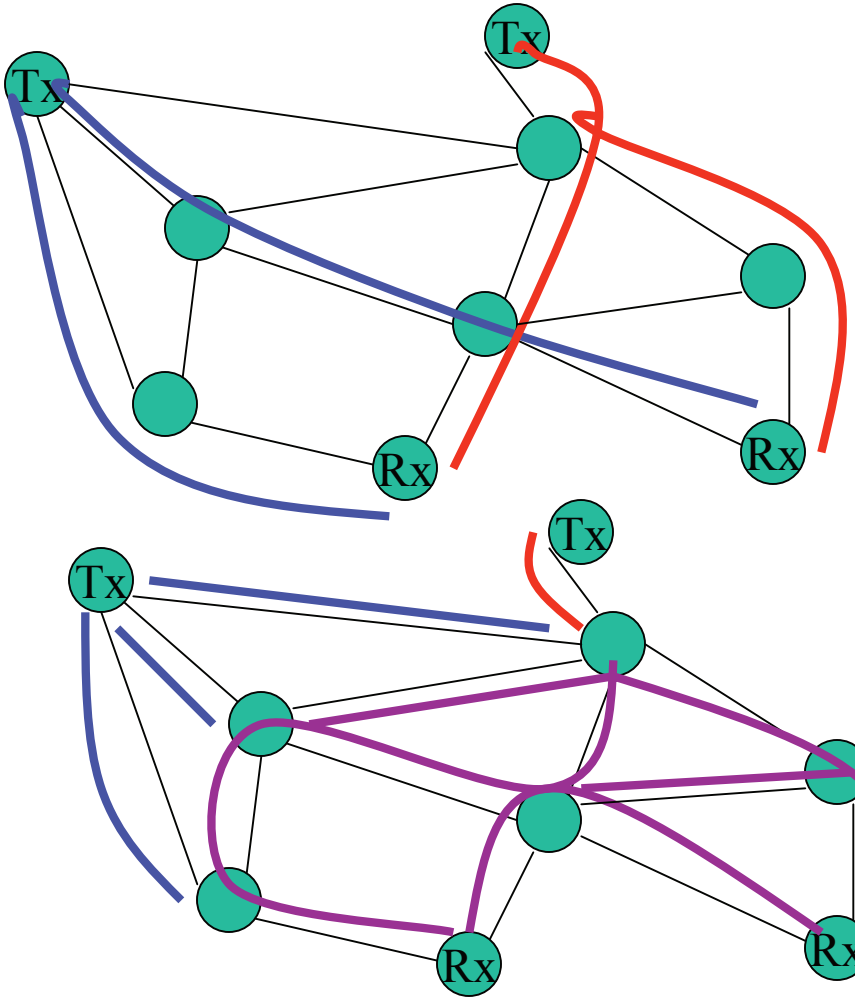
Reliable multicast using network codes



- Nodes in networks can ***randomly*** select the codes to mix traffic in the network
- This choice is entirely ***distributed*** without any coordination among nodes
- There is not need for any node to have information about the network topology, there are no routing tables
- For any general multi-input multicast network, a ***single*** randomly chosen code can recover from ***all recoverable failures*** - this is not possible with routing (Ho, Koetter, Medard, Karger, Effros, 2003)
- Network recovery in multicast networks can always be done solely by the receivers, using the knowledge of the composite effect of the network (by sending of a canonical basis)

Reliable multicast using network codes versus routing





Routing: single points of failure require re-routing

Coding: network is optimally used with a single code under all conditions