

Internet Infrastructure for Efficient Overlays

John Jannotti

MIT - LCS

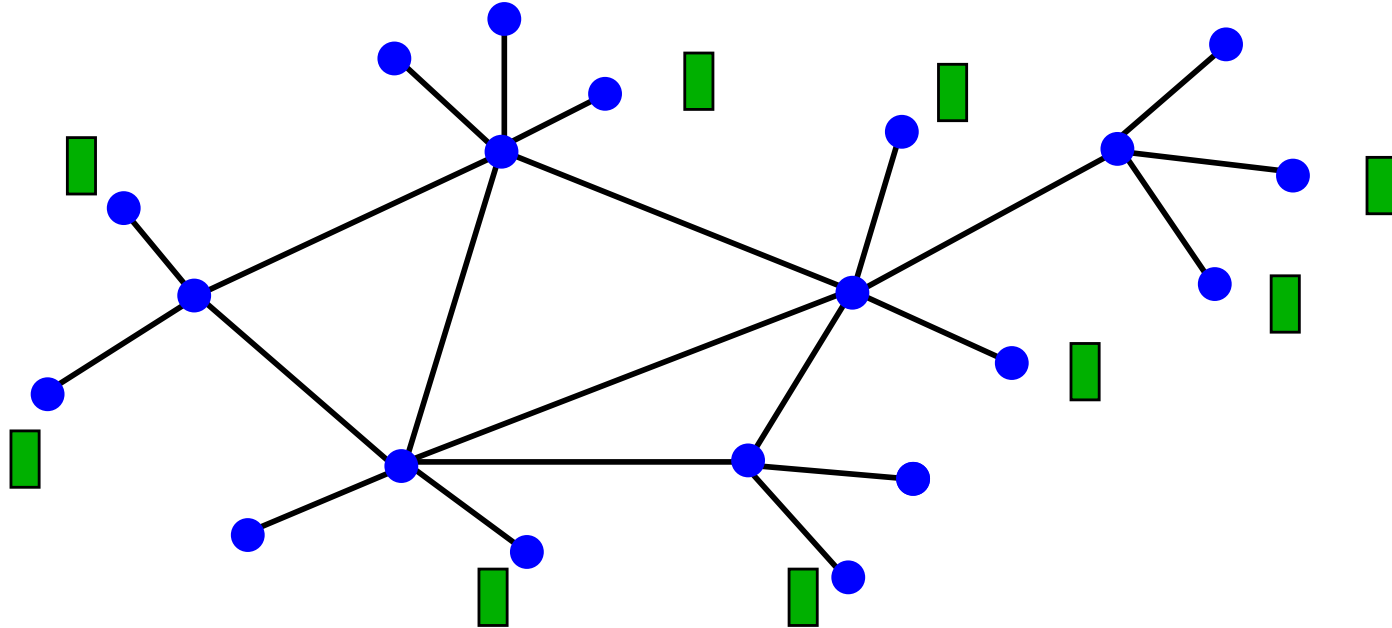
The Problem

People want to offer new applications:

- ▷ Websites that broadcast TV to thousands
- ▷ Stock tickers that work when links fail
- ▷ High-bandwidth teleconferencing

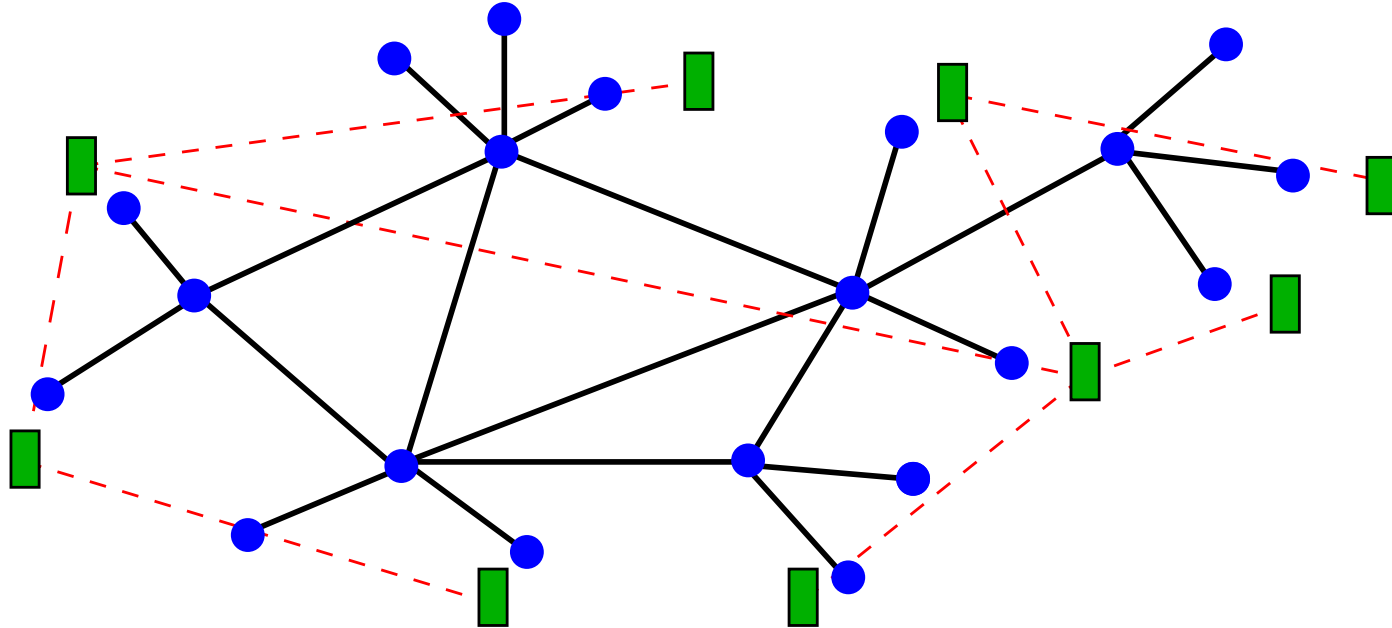
but the Internet lacks the support required.

Evolution of Overlays



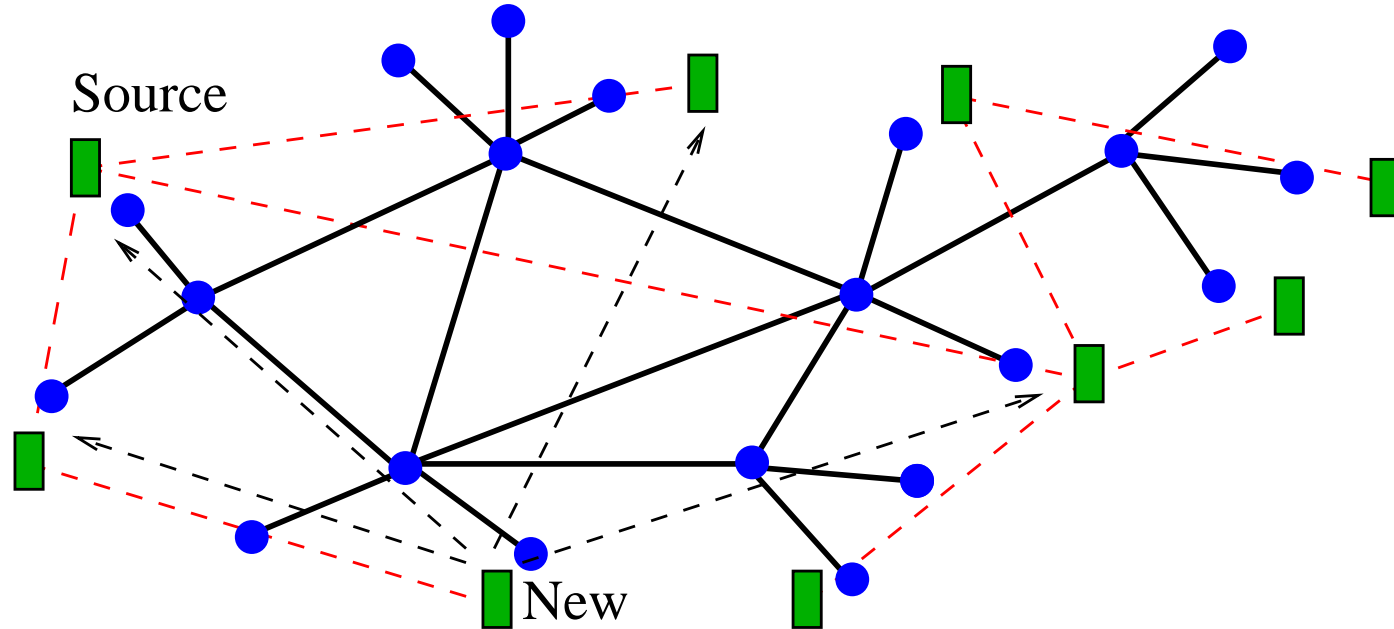
- ▷ Spread computers around

Evolution of Overlays



- ▷ Spread computers around
- ▷ Form a virtual topology

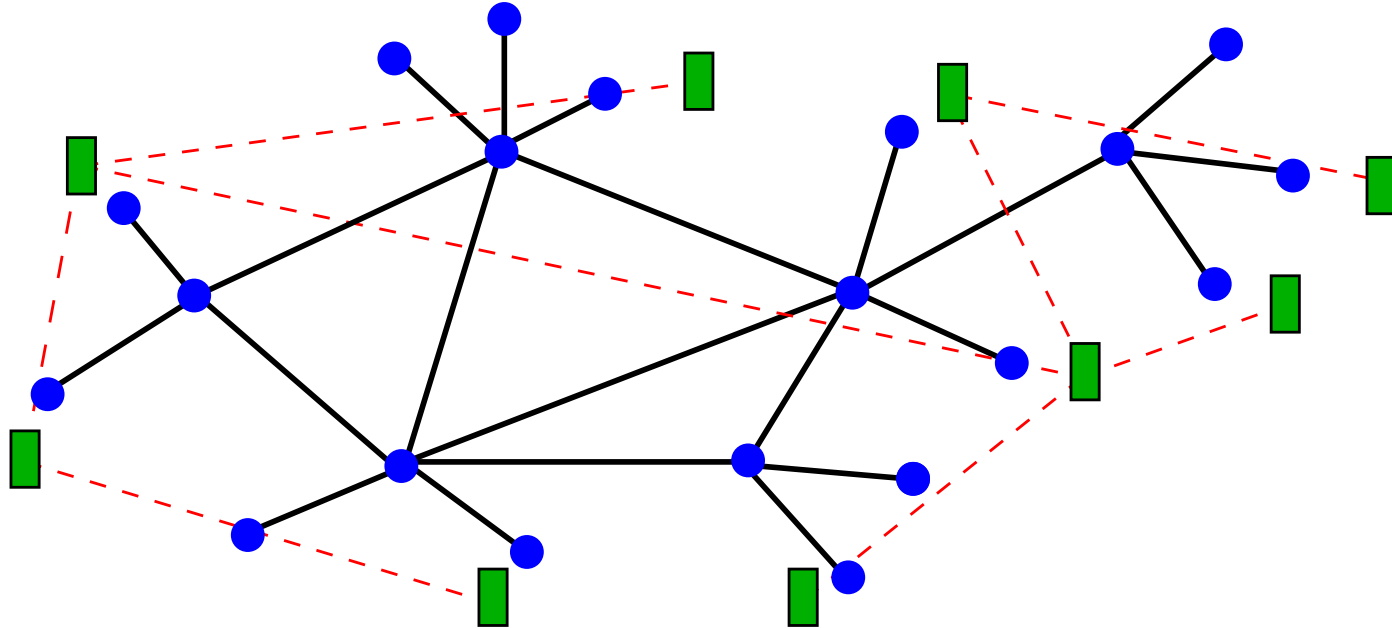
An Example: Overcast



Application Level Multicast

- ▷ Members join by running bandwidth experiments
- ▷ Data streams through the distribution tree
- ▷ Data is cached at each step
- ▷ Commercialized as a Content Distribution Network

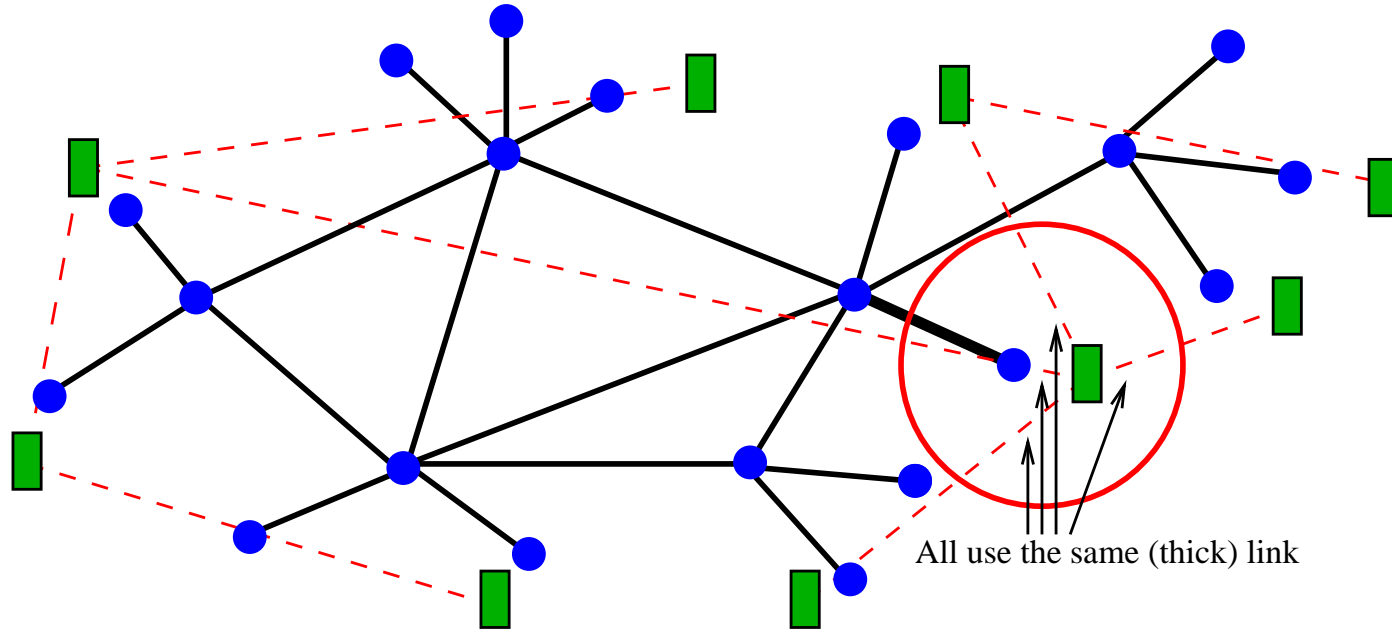
Overlays Allow New Services



- ▷ Allow innovation without consensus
- ▷ Keep functionality in end systems

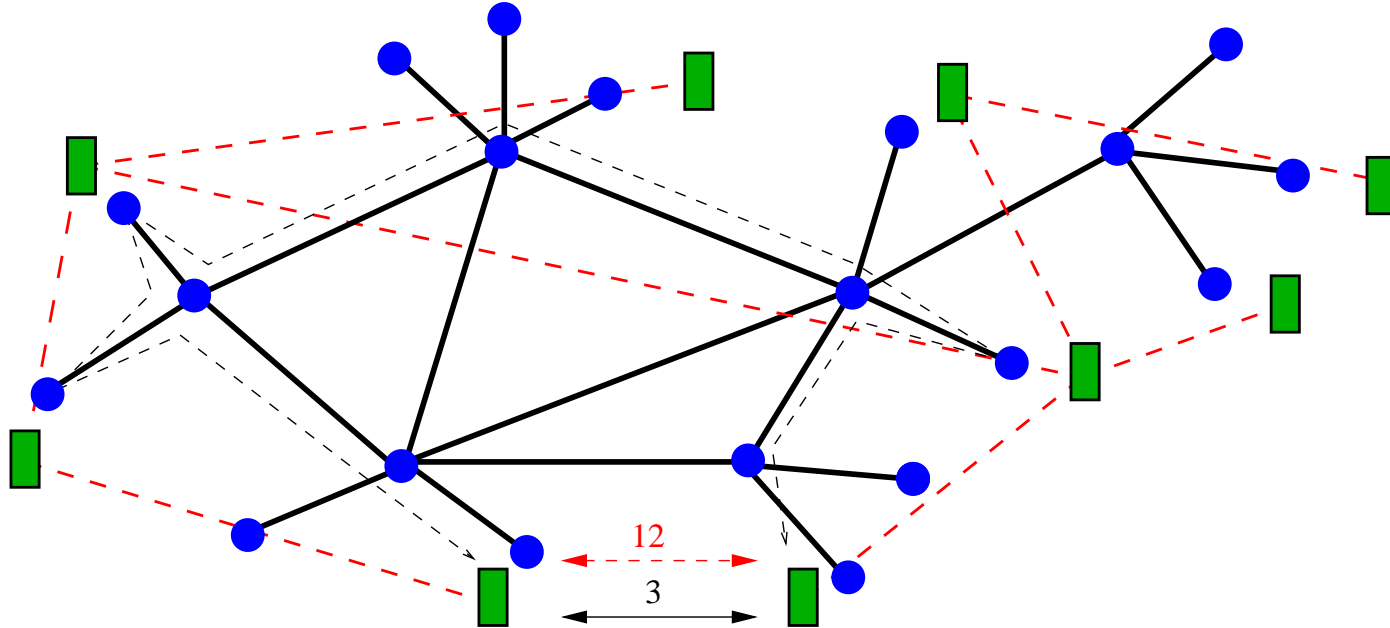
Akamai, ALMI, Chord, Dynabone, End System Multicast,
FastForward, Freenet, i3, Overcast, Pastry, RON, Yoid...

Overlays Are Inefficient



- ▷ Overlays increase network *stress*

Overlays Are Inefficient



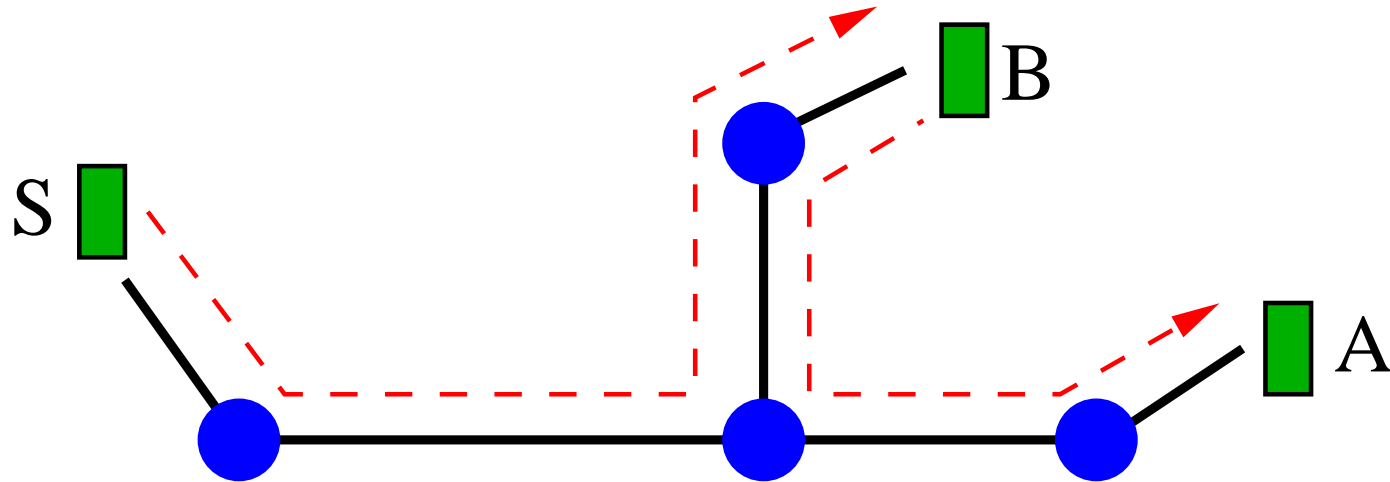
- ▷ Overlays increase network *stress*
- ▷ Overlays *stretch* latency

Minimal Router Support

Network support should be:

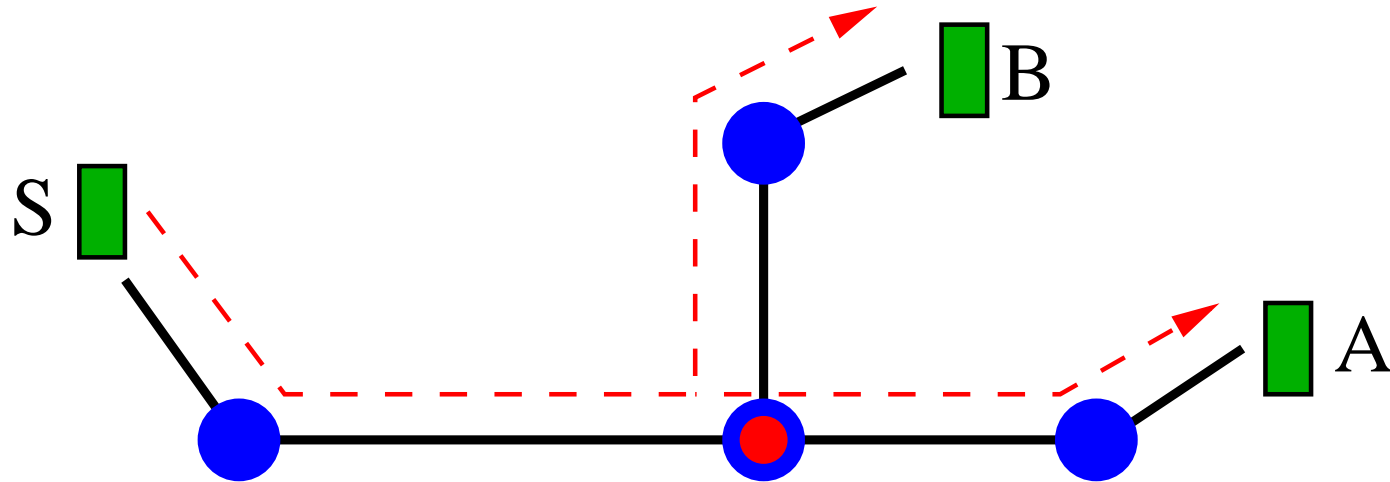
- ▷ Incrementally deployable - benefits to early adopters
- ▷ Flexible - address multiple application domains
- ▷ Simple - complexity in routers is unacceptable
- ▷ Precise - arbitrary power is slow and dangerous

Packet Reflection



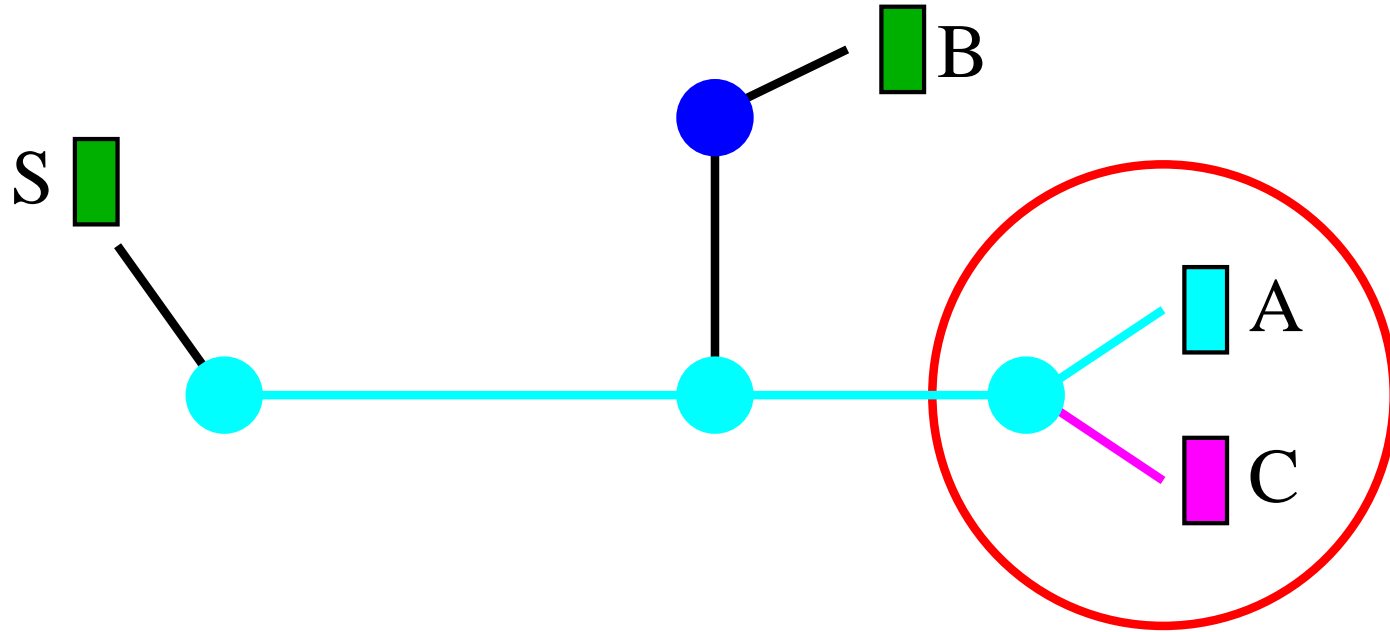
- ▷ $Reflect(S \rightarrow B, \{(B \rightarrow A)\})$
- ▷ The duplication happens in the network

Packet Reflection



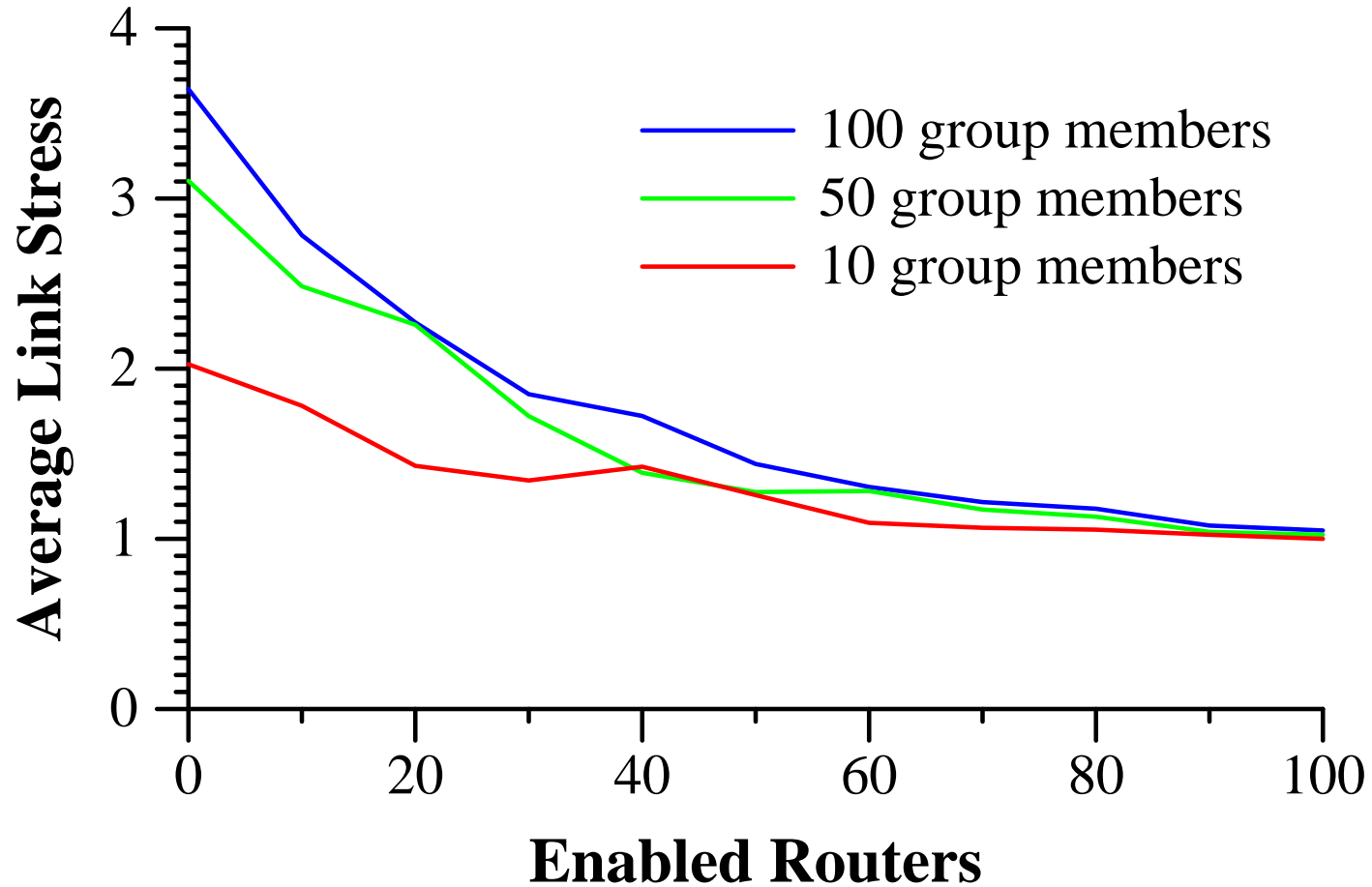
- ▷ $Reflect(S \rightarrow B, \{(B \rightarrow A)\})$
- ▷ The duplication happens in the network

Path Painting



- ▷ $Paint(S)$
- ▷ A and C learn about each other

A Performance Preview



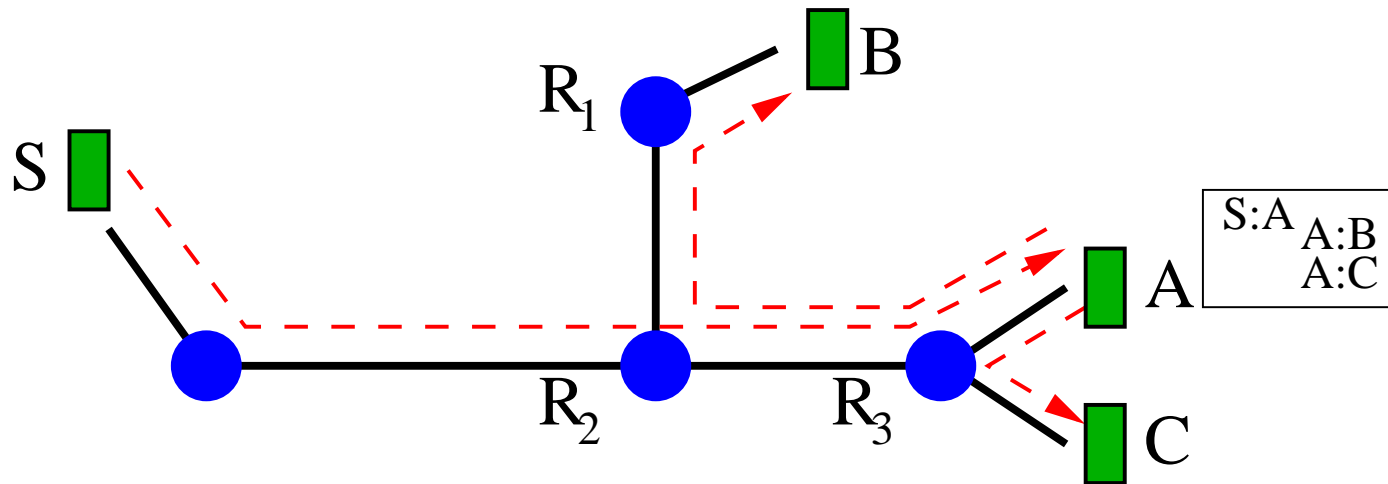
Stress is reduced drastically, even at low deployment levels

Outline

- ▷ Packet Reflection
- ▷ Path Painting
- ▷ Application Level Multicast
- ▷ Evaluation
- ▷ Related Work / Conclusion

Packet Reflection

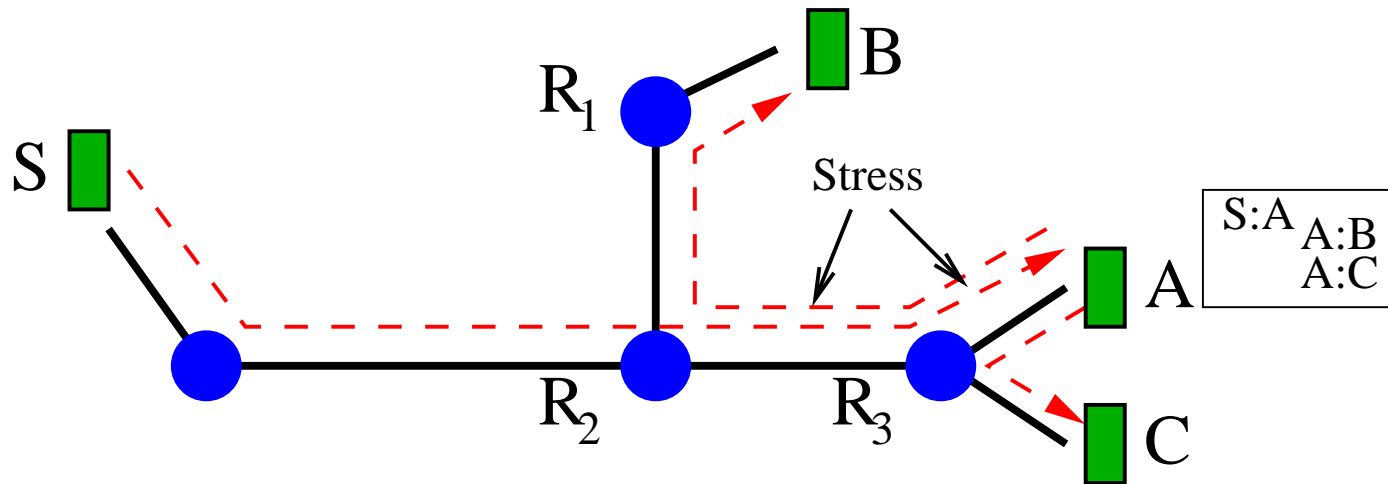
Push Routing and Duplication into Routers



- ▷ Overlays route and duplicate packets in end-hosts.

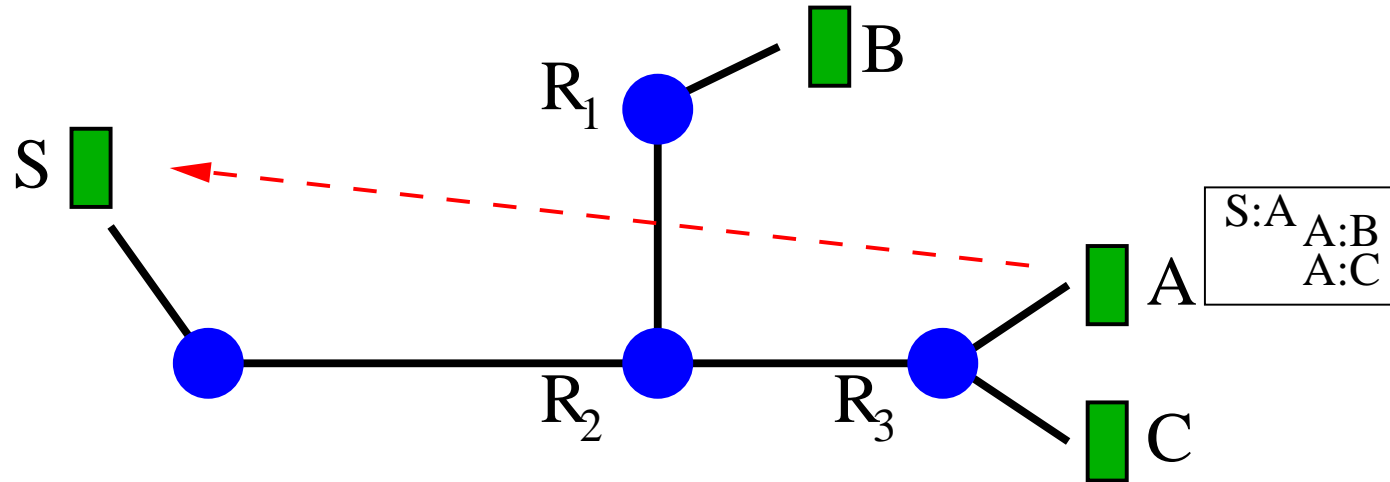
Packet Reflection

Push Routing and Duplication into Routers



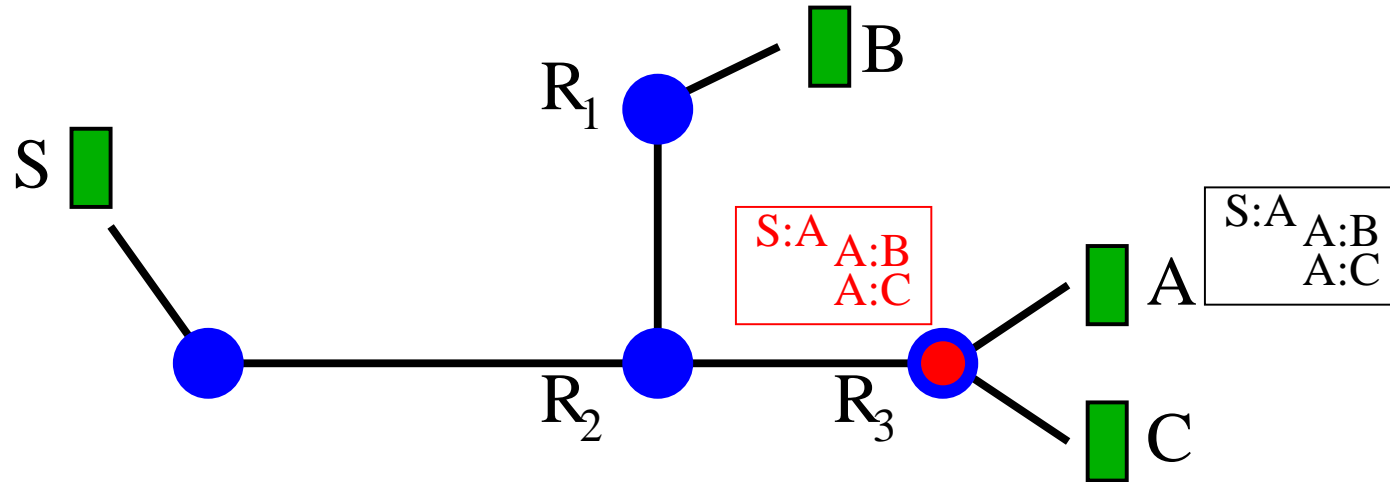
- ▷ Overlays route and duplicate packets in end-hosts.
- ▷ Two links transmit the “same” packets multiple times.
- ▷ $Reflect(S \rightarrow A, \{A \rightarrow B, A \rightarrow C\})$

Tracing Reflection Requests



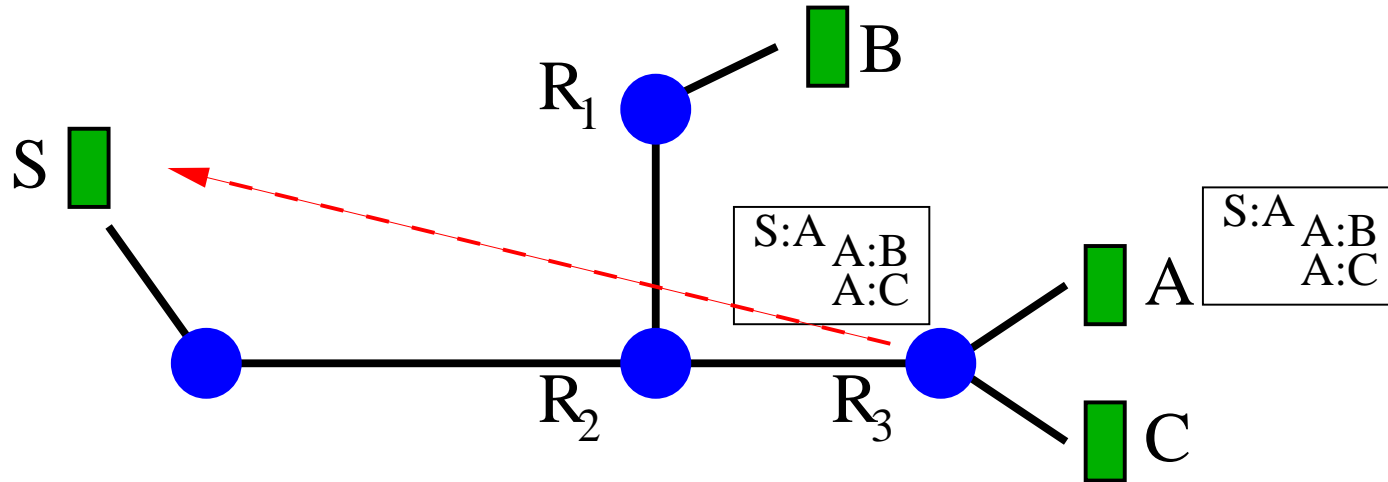
▷ $A \rightarrow S : Reflect(S \rightarrow A, \{A \rightarrow B, A \rightarrow C\})$

Tracing Reflection Requests



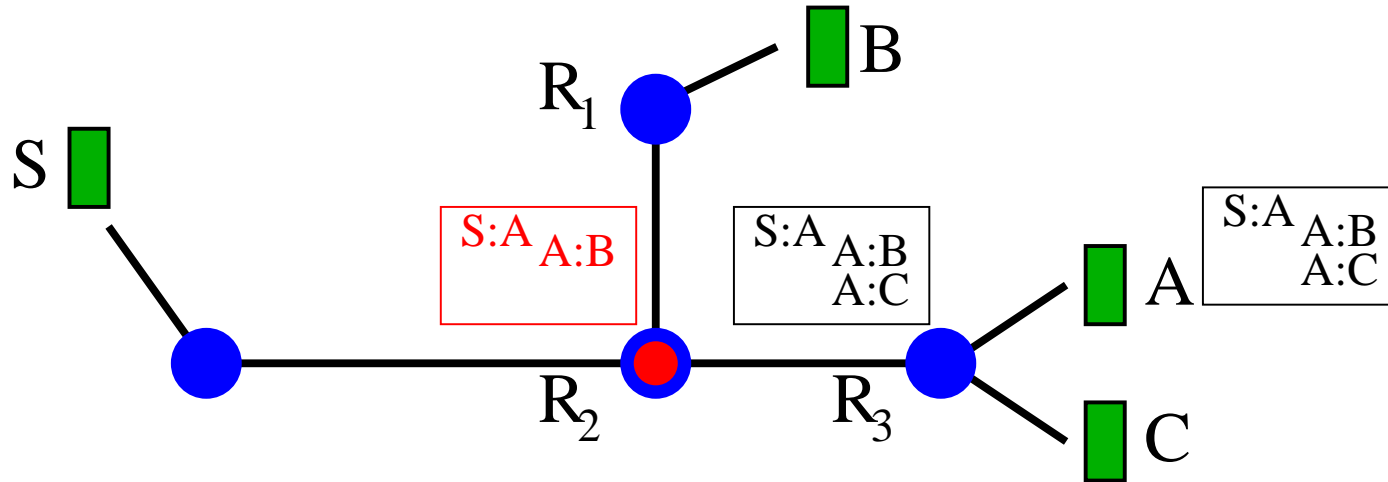
- ▷ $A \rightarrow S : Reflect(S \rightarrow A, \{A \rightarrow B, A \rightarrow C\})$
- ▷ Intercepted by R_3

Tracing Reflection Requests



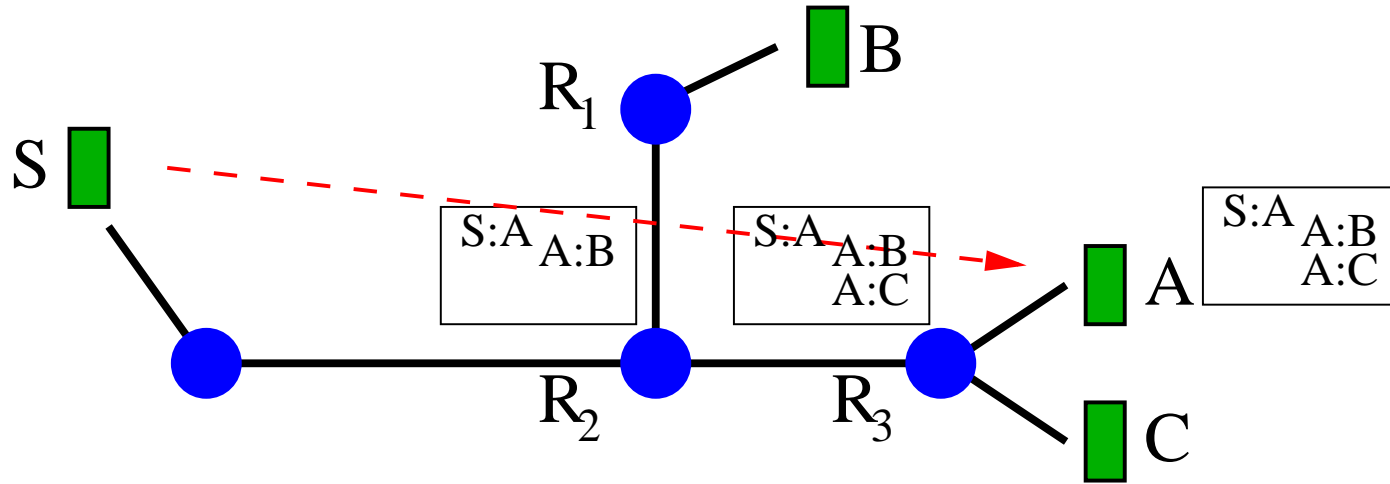
- ▷ $A \rightarrow S : Reflect(S \rightarrow A, \{A \rightarrow B, A \rightarrow C\})$
- ▷ Intercepted by R_3
- ▷ $R_3 \rightarrow S : Reflect(S \rightarrow A, \{A \rightarrow B\})$

Tracing Reflection Requests



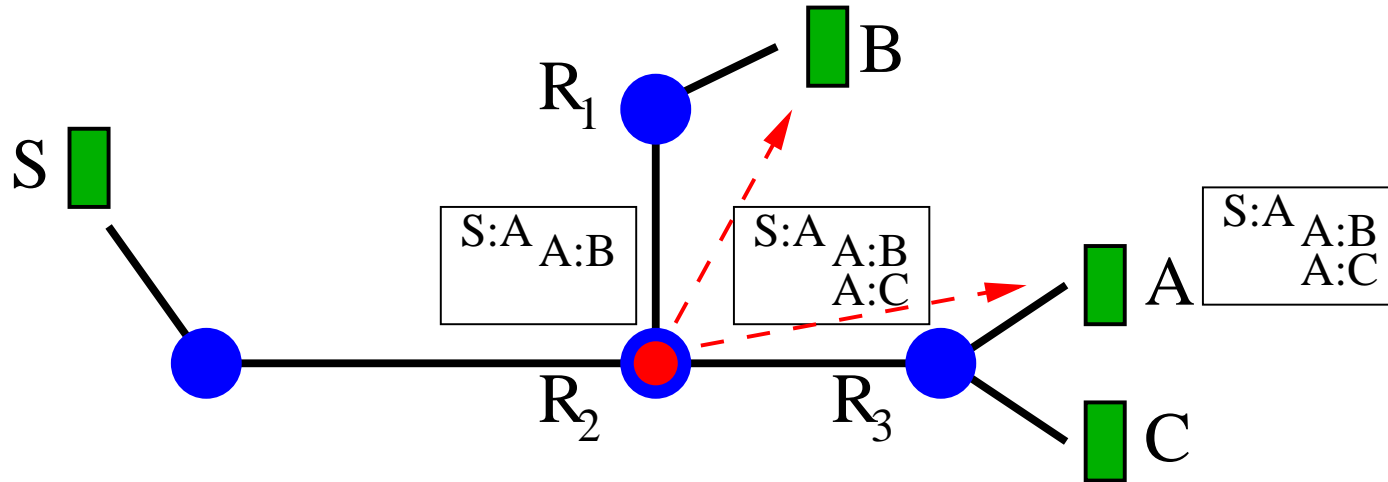
- ▷ $A \rightarrow S : Reflect(S \rightarrow A, \{A \rightarrow B, A \rightarrow C\})$
- ▷ Intercepted by R_3
- ▷ $R_3 \rightarrow S : Reflect(S \rightarrow A, \{A \rightarrow B\})$
- ▷ Intercepted by R_2

Tracing the Data Path



▷ $S \rightarrow A$

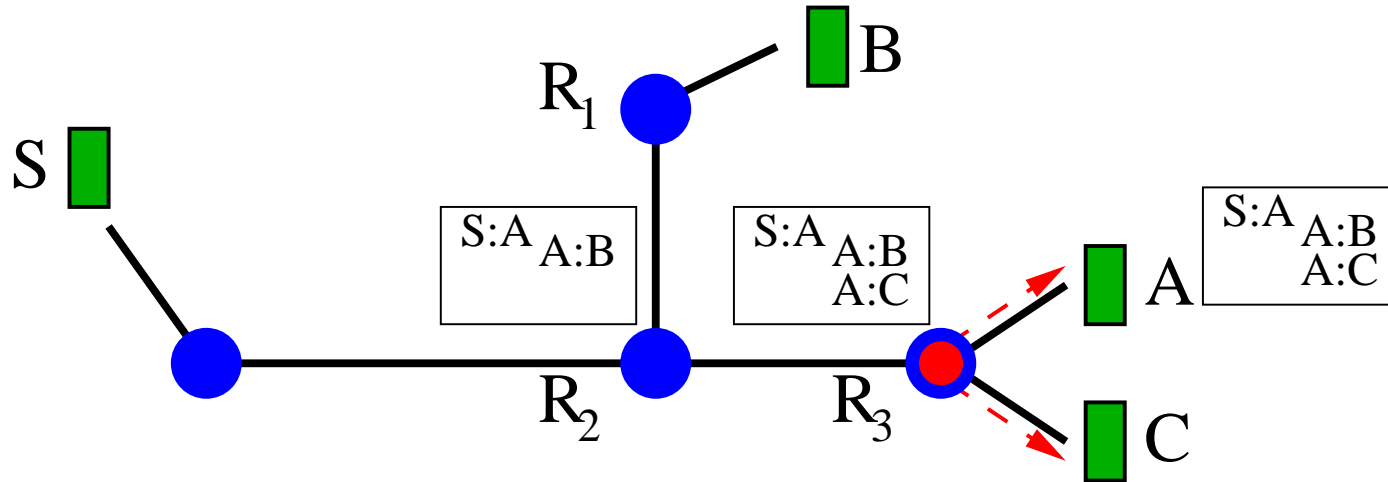
Tracing the Data Path



▷ $S \rightarrow A$

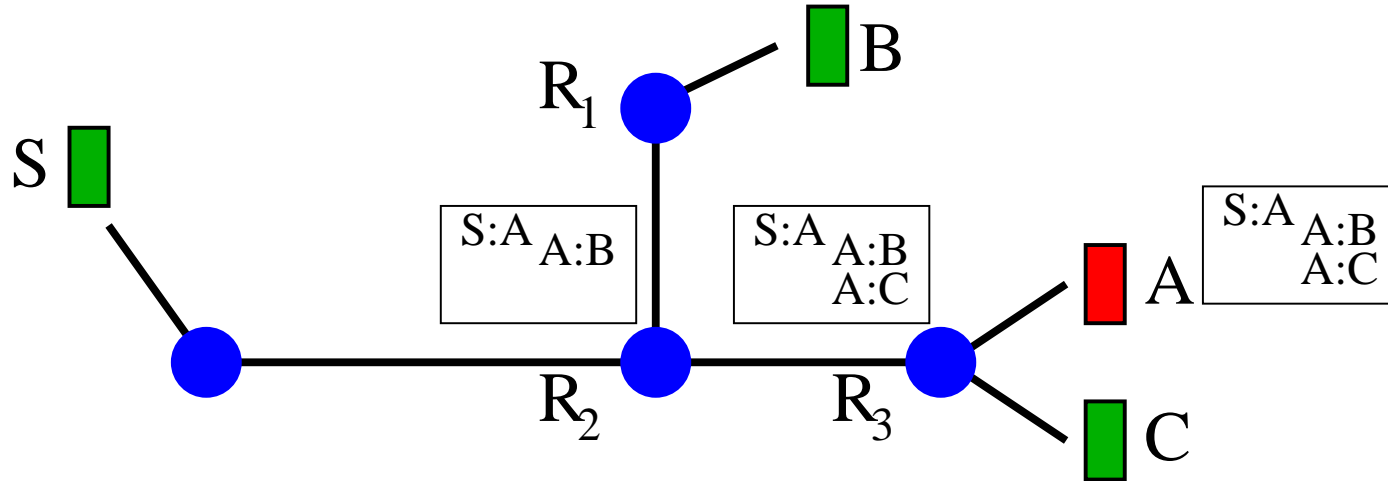
▷ R_2 emits $A \rightarrow B$ and forwards the original

Tracing the Data Path



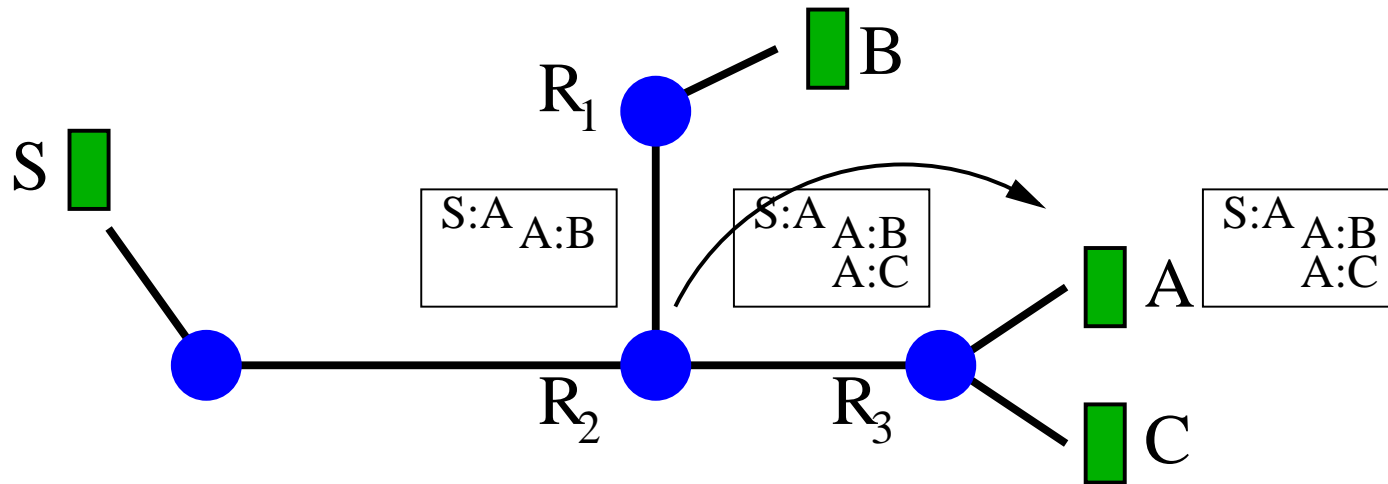
- ▷ $S \rightarrow A$
- ▷ R_2 emits $A \rightarrow B$ and forwards the original
- ▷ R_3 emits $A \rightarrow C$ and forwards the original

Tracing the Data Path



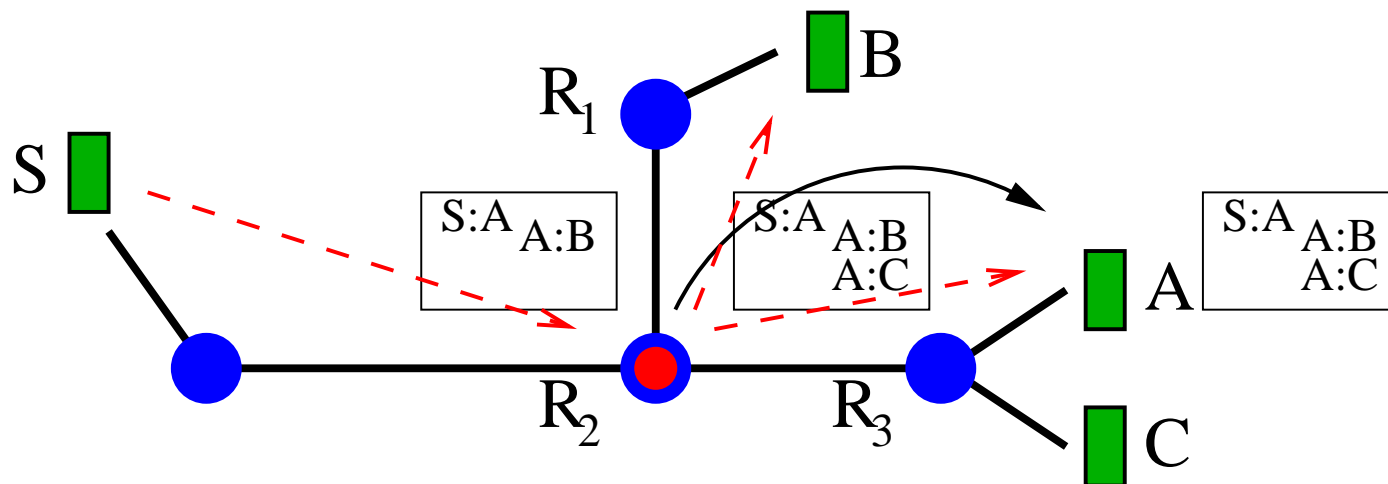
- ▷ $S \rightarrow A$
- ▷ R_2 emits $A \rightarrow B$ and forwards the original
- ▷ R_3 emits $A \rightarrow C$ and forwards the original
- ▷ A does nothing

Route asymmetry



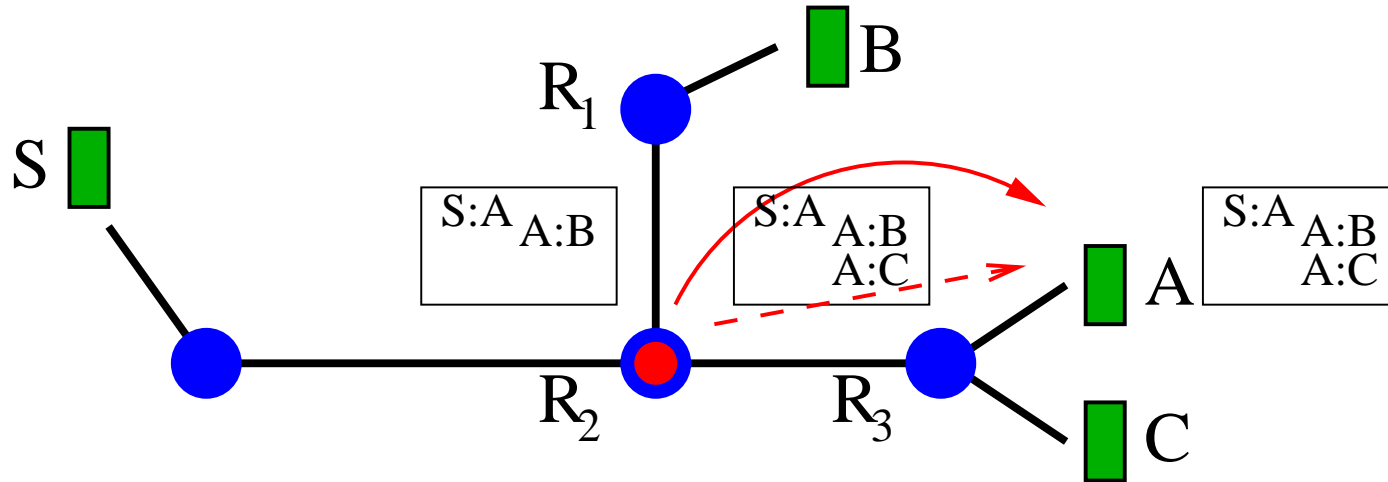
- ▷ A new, asymmetric link is introduced

Route asymmetry



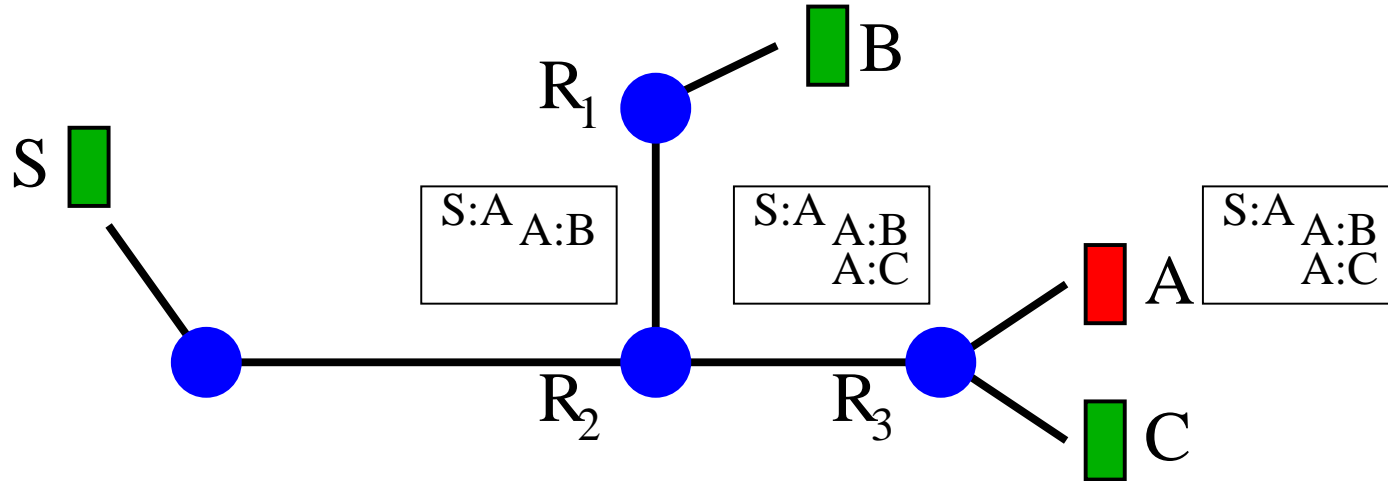
- ▷ A new, asymmetric link is introduced
- ▷ R_2 performs duplication

Route asymmetry



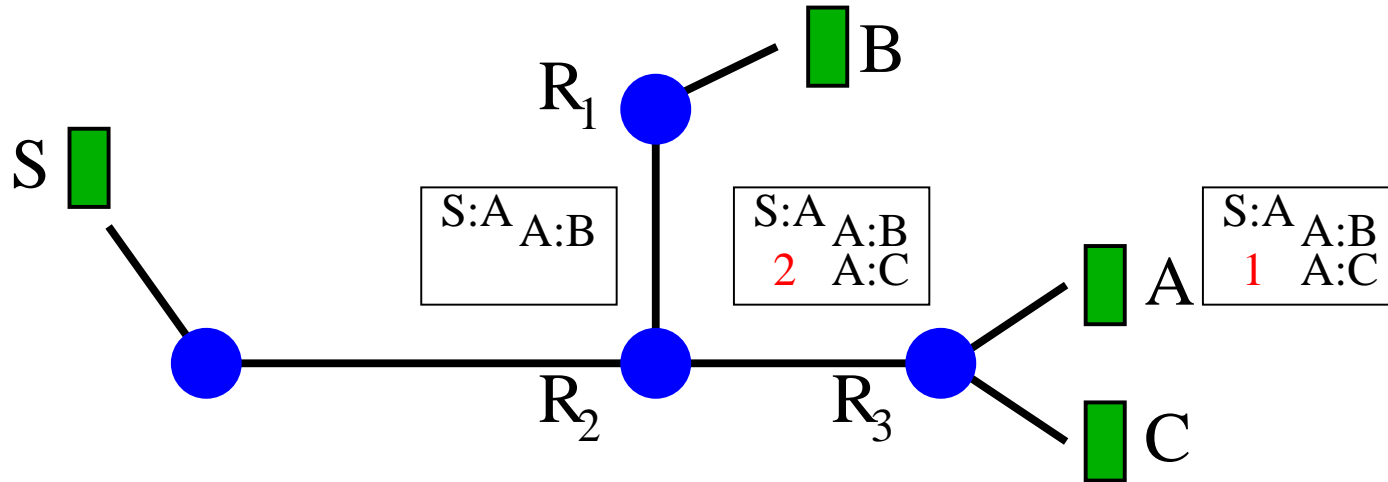
- ▷ A new, asymmetric link is introduced
- ▷ R_2 performs duplication
- ▷ R_3 is skipped

Route asymmetry



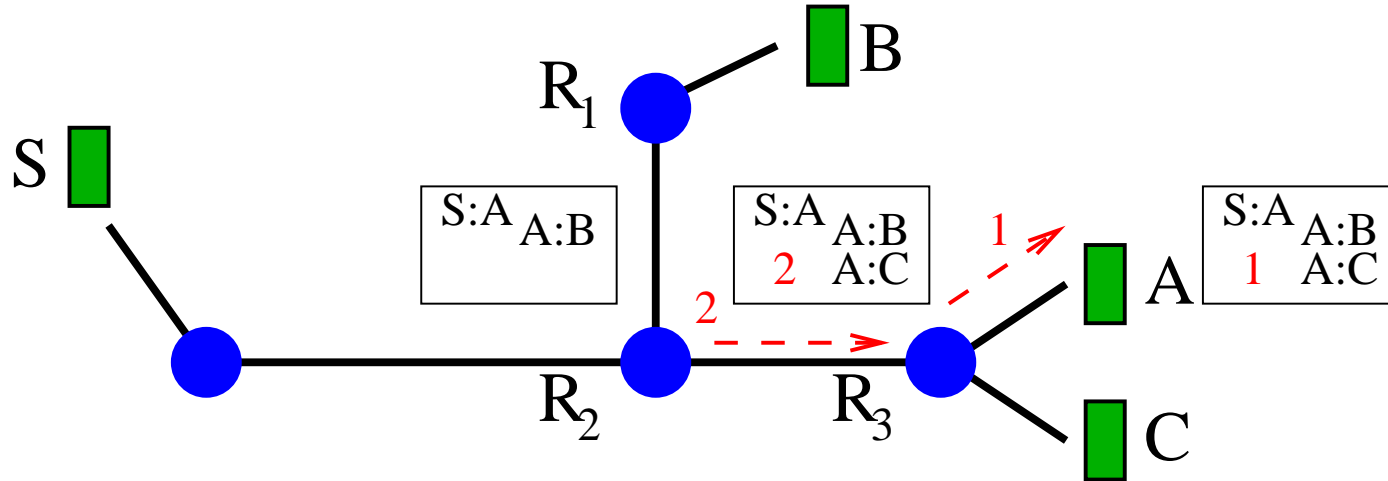
- ▷ A new, asymmetric link is introduced
- ▷ R_2 performs duplication
- ▷ R_3 is skipped
- ▷ A (erroneously) does nothing

Tags confirm reflection



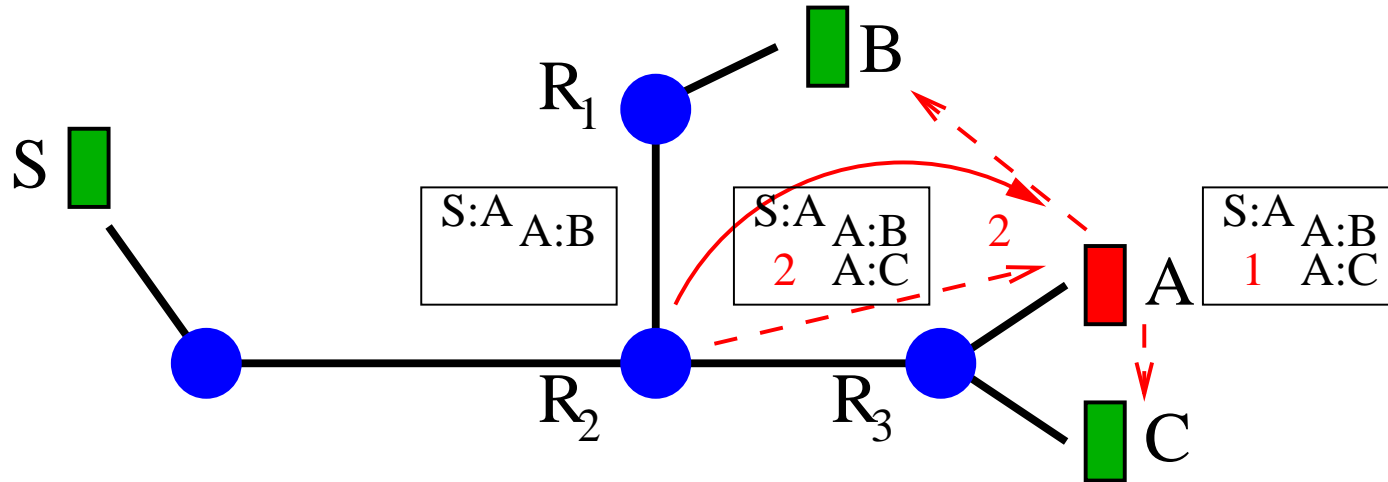
- ▷ Reflect requests include success tags

Tags confirm reflection



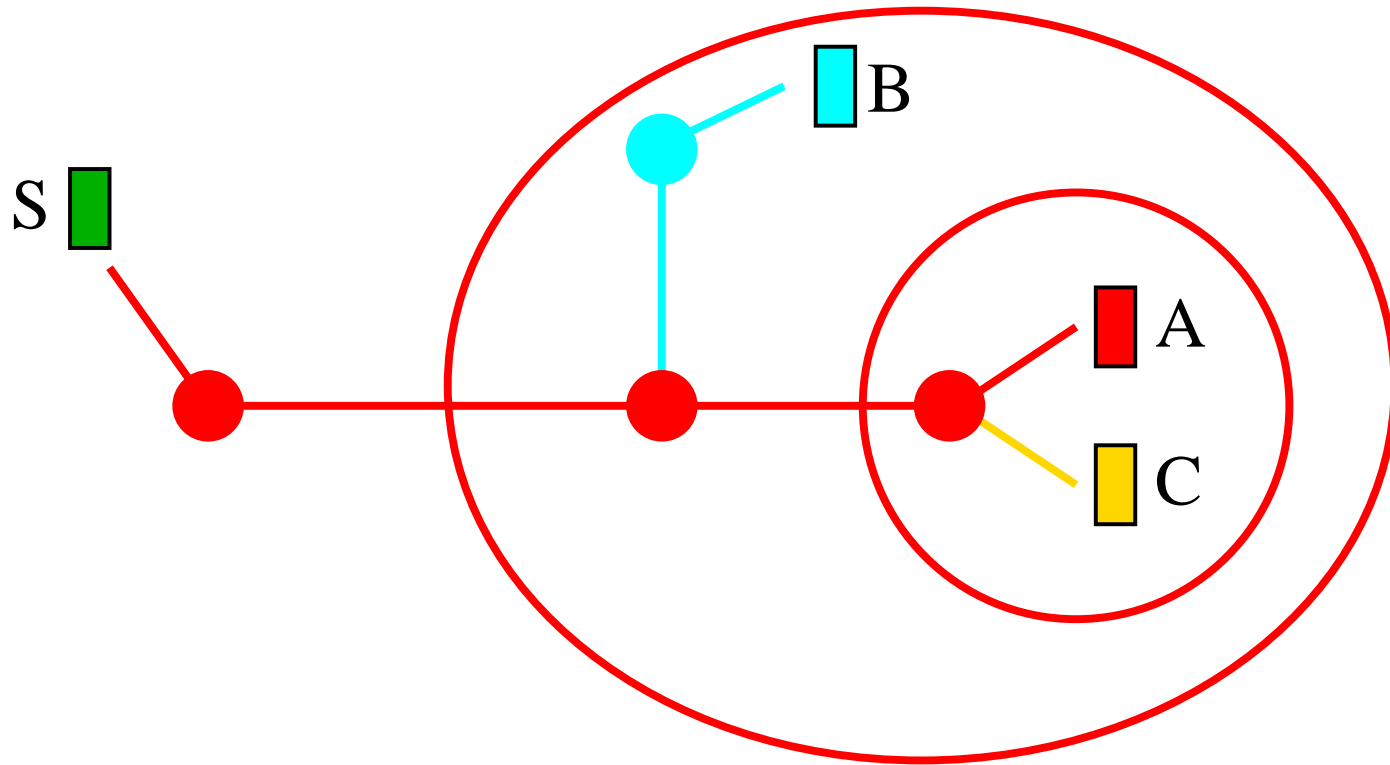
- ▷ Reflect requests include success tags
- ▷ When reflecting, a router tags the original packet

Tags confirm reflection



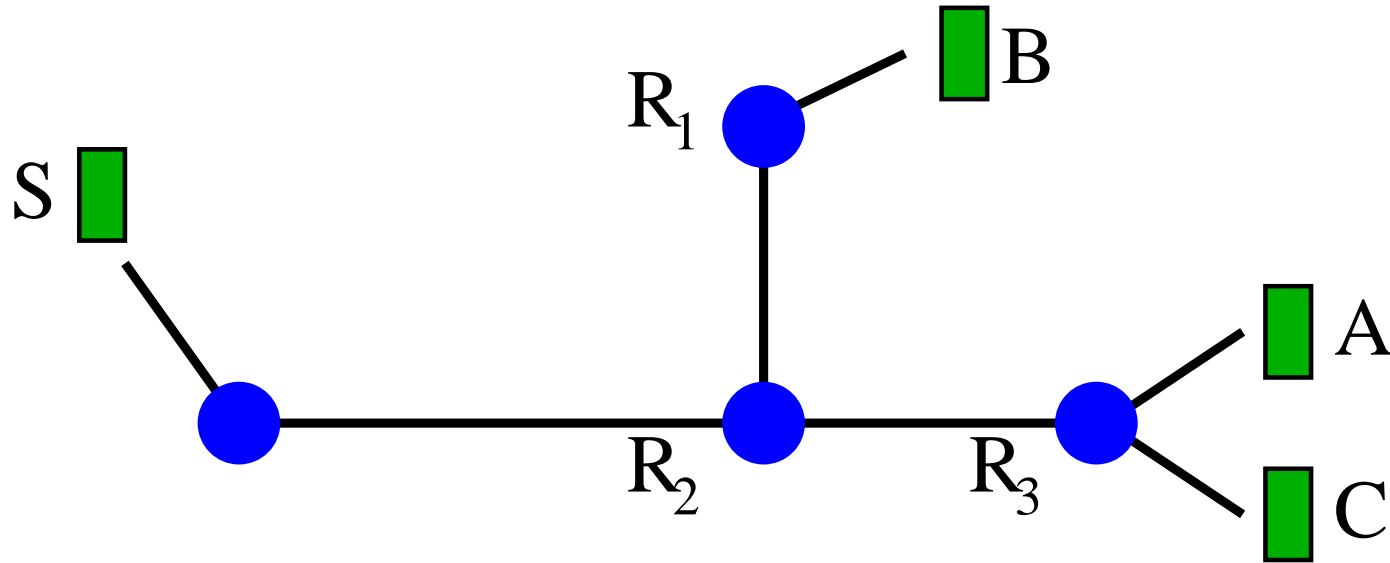
- ▷ Reflect requests include success tags
- ▷ When reflecting, a router tags the original packet
- ▷ If tag wrong, duplicate anyway

Path Painting



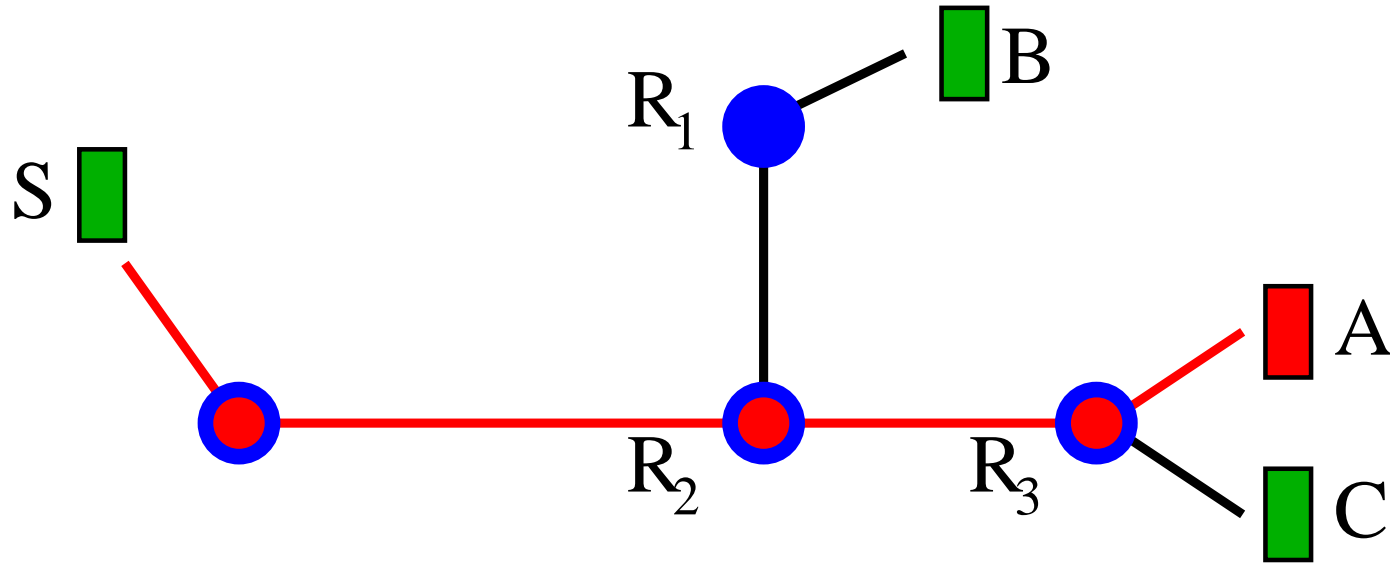
- ▷ Nodes “behind” a router should aggregate
- ▷ Aggregation at any scale
- ▷ With reflection (and some assumptions), stress = 1

Tracing Path Painting



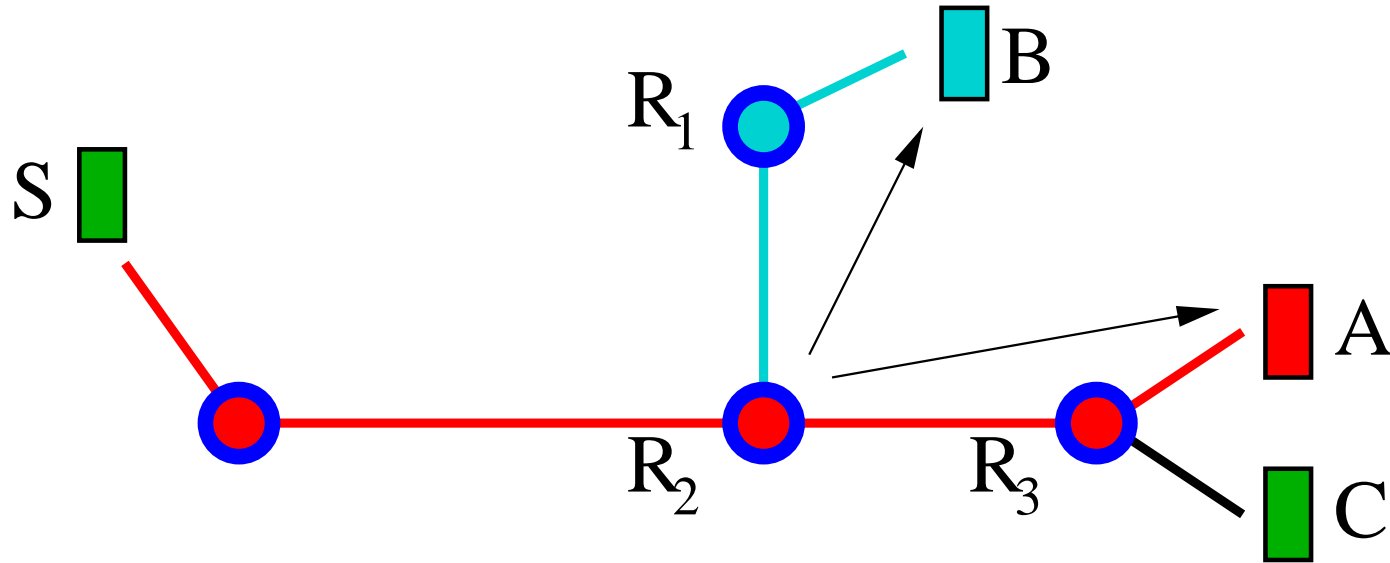
▷ A , B , and C want to form an overlay with S

Tracing Path Painting



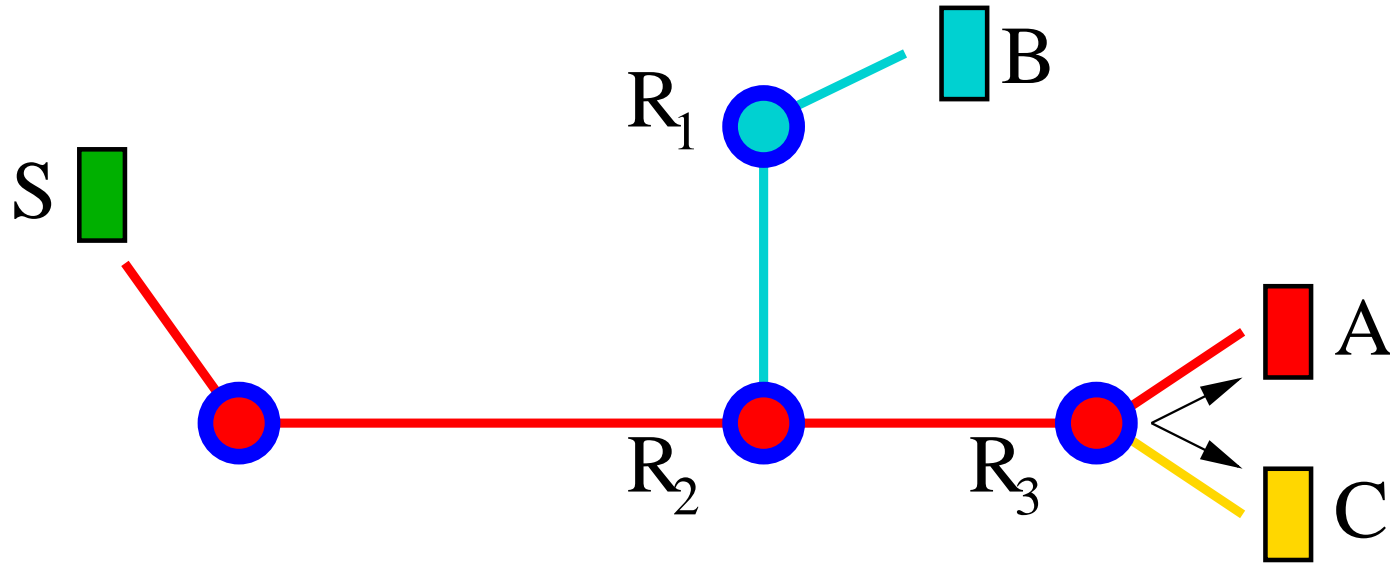
- ▷ A , B , and C want to form an overlay with S
- ▷ $A \rightarrow S : Paint(S)$ (reaches S)

Tracing Path Painting



- ▷ A , B , and C want to form an overlay with S
- ▷ $A \rightarrow S : Paint(S)$ (reaches S)
- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)

Tracing Path Painting



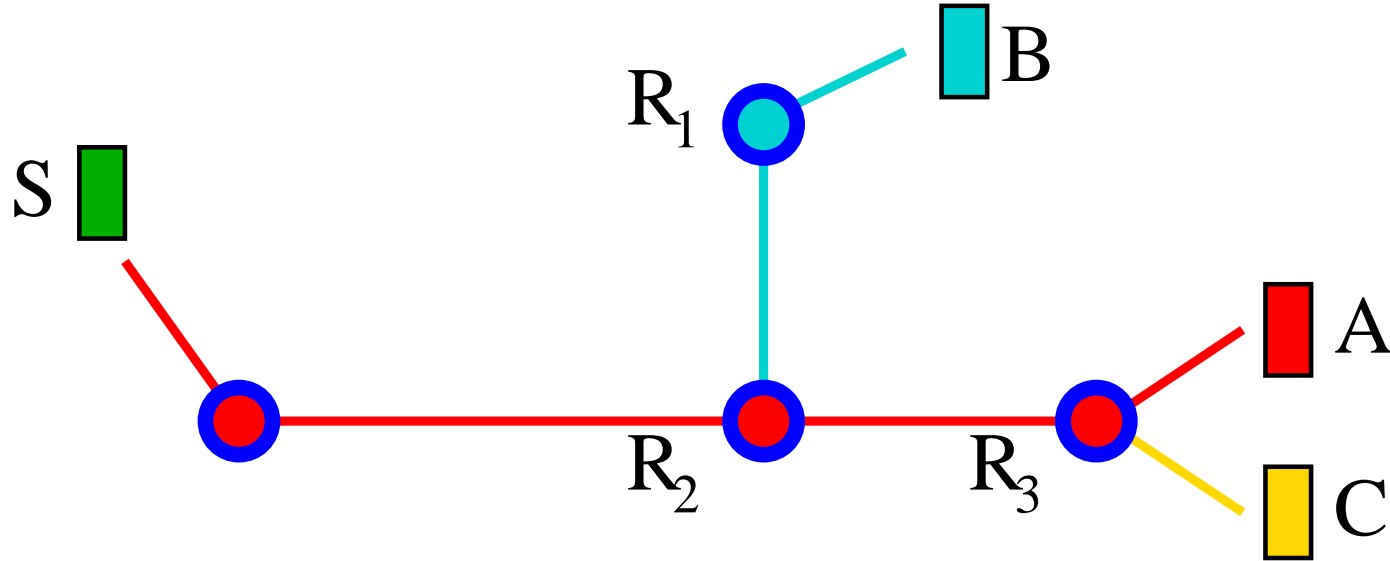
- ▷ A , B , and C want to form an overlay with S
- ▷ $A \rightarrow S : Paint(S)$ (reaches S)
- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)
- ▷ $C \rightarrow S : Paint(S)$ (stops at R_3)

Painting Details

Paint(A, concede, ignore)

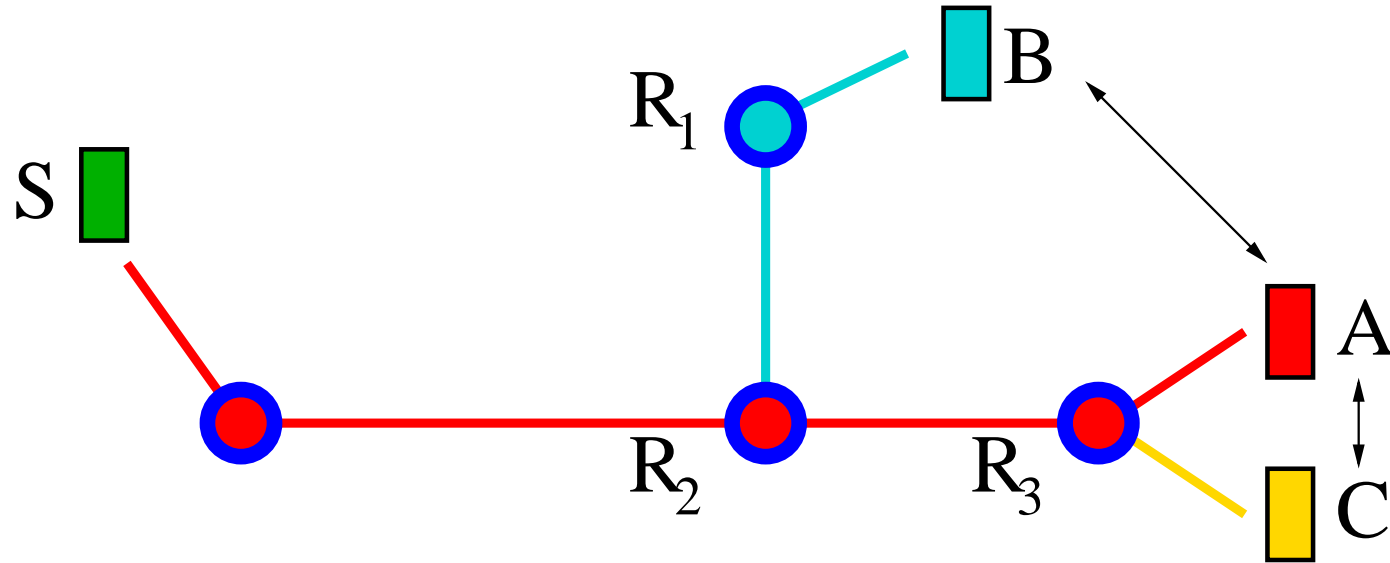
- ▷ Application Control - *concede* can choose the “winner”
- ▷ Admission Control - *ignore* list can skip “bad” colors

Application Level Multicast



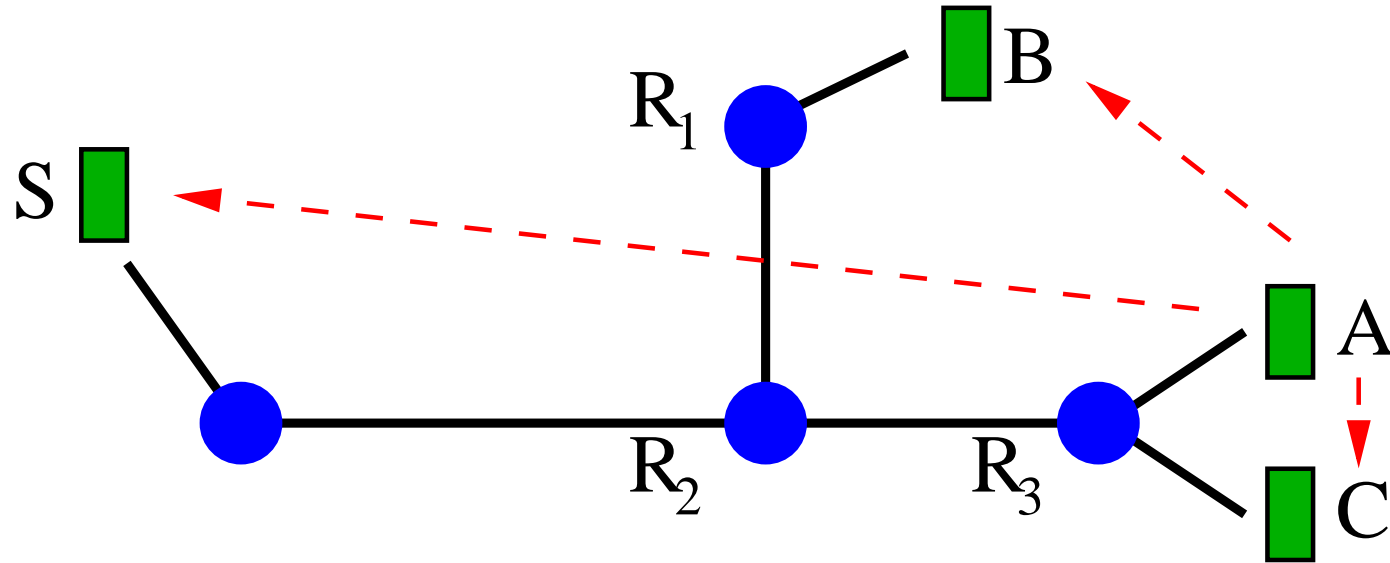
▷ A, B, C all $Paint(S)$

Application Level Multicast



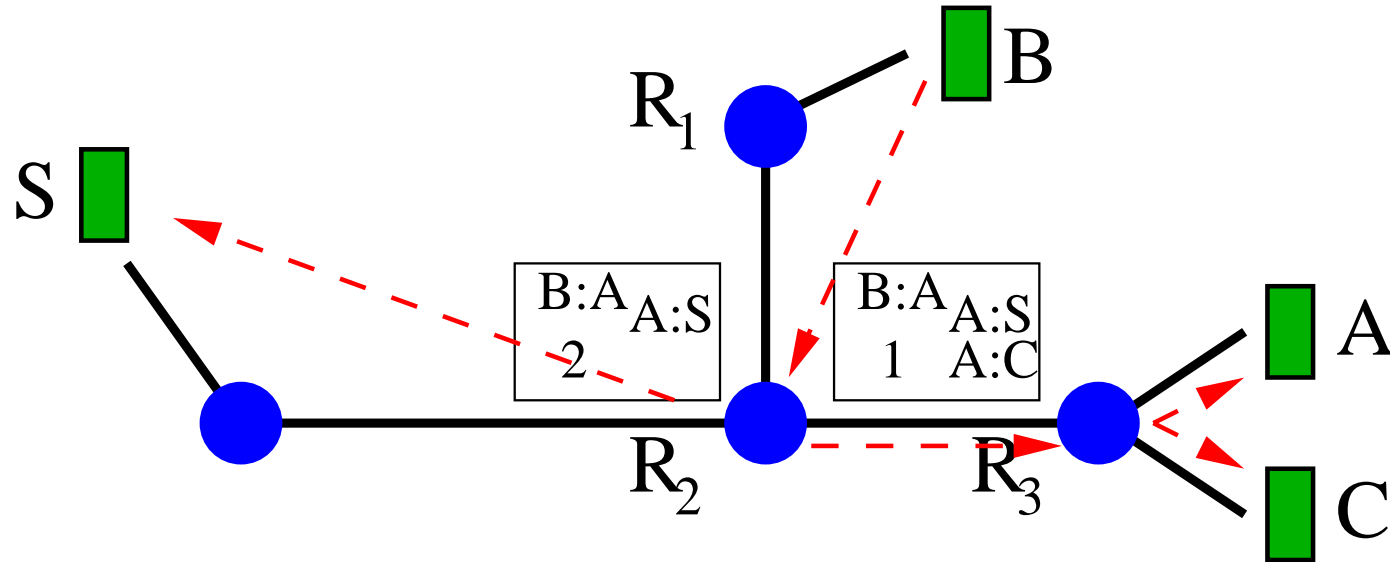
- ▷ A, B, C all $Paint(S)$
- ▷ A becomes parent of B and C

Application Level Multicast



- ▷ A, B, C all $Paint(S)$
- ▷ A becomes parent of B and C
- ▷ A sends a *Reflect* request to each neighbor (S, B, C)

Application Level Multicast



- ▷ A, B, C all $Paint(S)$
- ▷ A becomes parent of B and C
- ▷ A sends a *Reflect* request to each neighbor (S, B, C)
- ▷ To send, emit to all neighbors (B is shown)

Easy Extension

Many multicast semantics are simple extensions

- ▷ Admittance Control - Negotiate during *Paint*.

Easy Extension

Many multicast semantics are simple extensions

- ▷ Admittance Control - Negotiate during *Paint*.
- ▷ Single source - Only *Reflect* your upstream neighbor.

Easy Extension

Many multicast semantics are simple extensions

- ▷ Admittance Control - Negotiate during *Paint*.
- ▷ Single source - Only *Reflect* your upstream neighbor.
- ▷ Heterogeneity - Transcode on congested links.

Easy Extension

Many multicast semantics are simple extensions

- ▷ Admittance Control - Negotiate during *Paint*.
- ▷ Single source - Only *Reflect* your upstream neighbor.
- ▷ Heterogeneity - Transcode on congested links.
- ▷ Reliability - Contact parent for lost frames.

Easy Extension

Many multicast semantics are simple extensions

- ▷ Admittance Control - Negotiate during *Paint*.
- ▷ Single source - Only *Reflect* your upstream neighbor.
- ▷ Heterogeneity - Transcode on congested links.
- ▷ Reliability - Contact parent for lost frames.
- ▷ Time shifting - Cache data in overlay nodes.

Designed for Flexibility

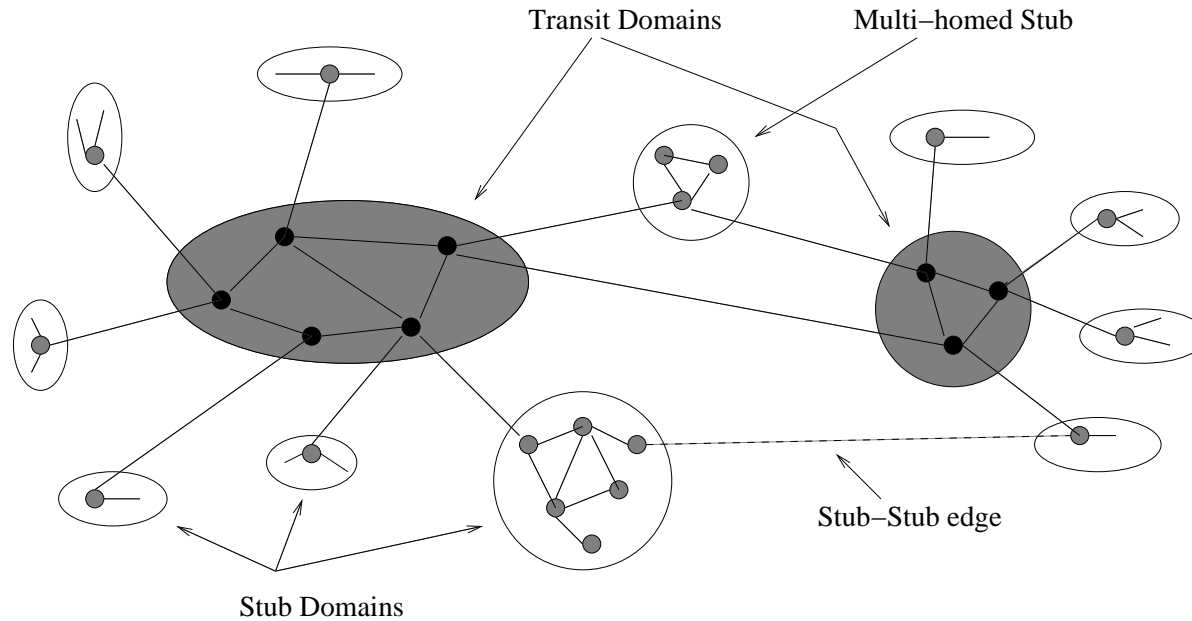
Primitives support other types of overlays:

- ▷ RON, *i3*, Mobile IP - use reflection to reduce latency
- ▷ P2P, CDNs - use painting to find nearby members, reflection for transport

But not every overlay can use both:

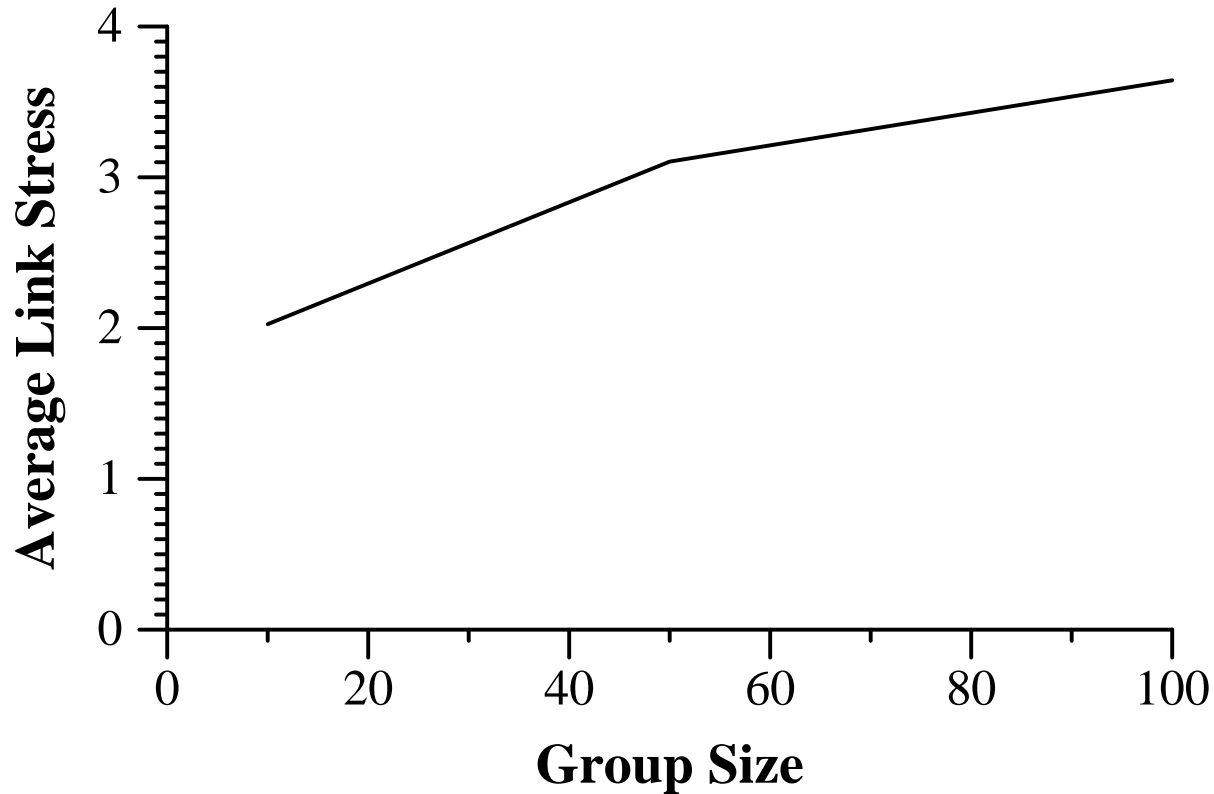
- ▷ RON, Mobile IP - topology determined by application: no paint
- ▷ DHT - lookups involve unpredictable traffic: no reflection

Experimental Setup



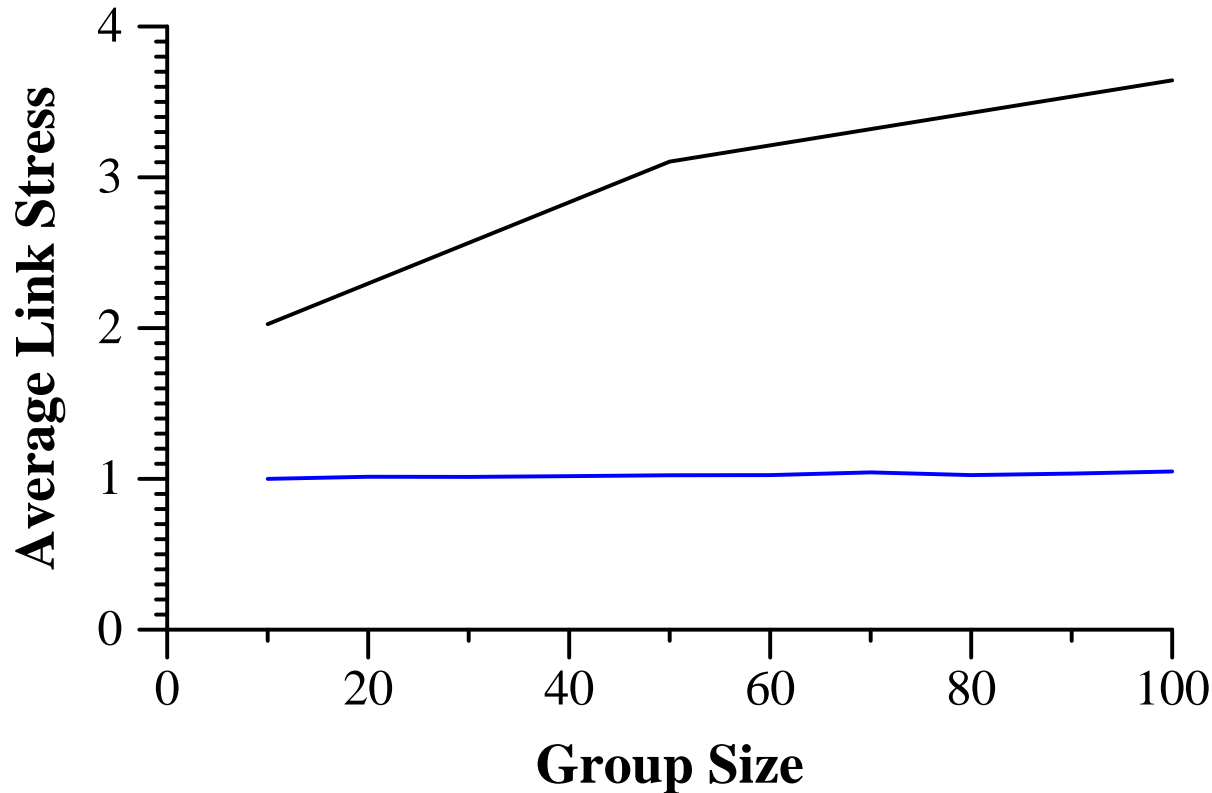
- ▷ ALM (Single Source).
- ▷ ns Network Simulator (3 new OTcl classes)
- ▷ Georgia Tech Topology Models (transit-stub)
- ▷ 100 router networks

Reflection reduces stress



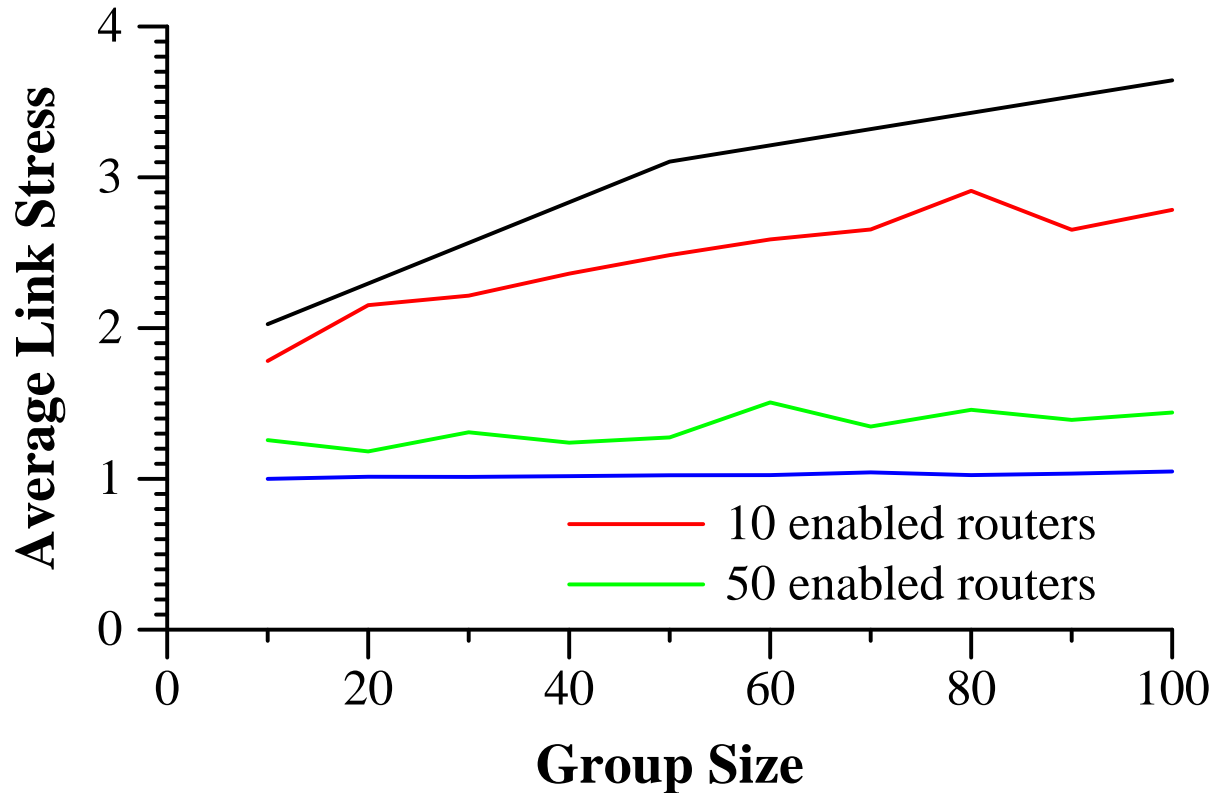
- ▷ Iterated Unicast causes high stress

Reflection reduces stress



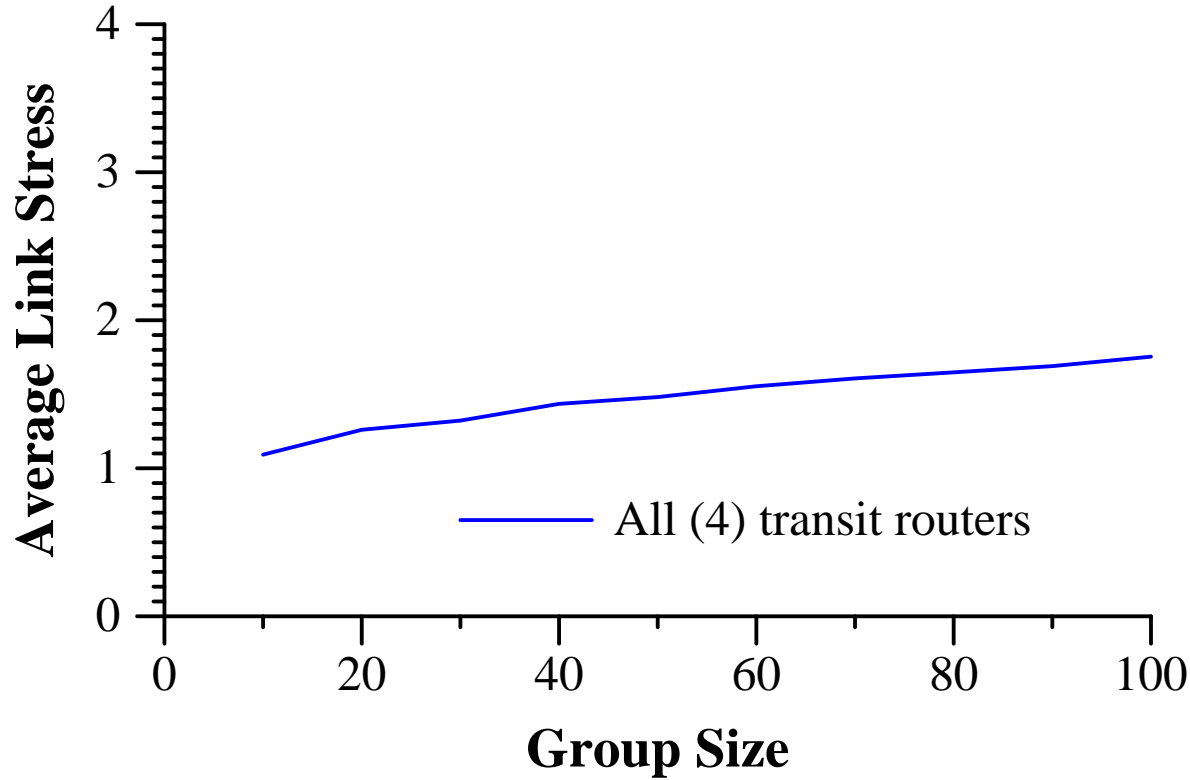
- ▷ Iterated Unicast causes high stress
- ▷ Complete deployment essentially eliminates stress

Reflection reduces stress



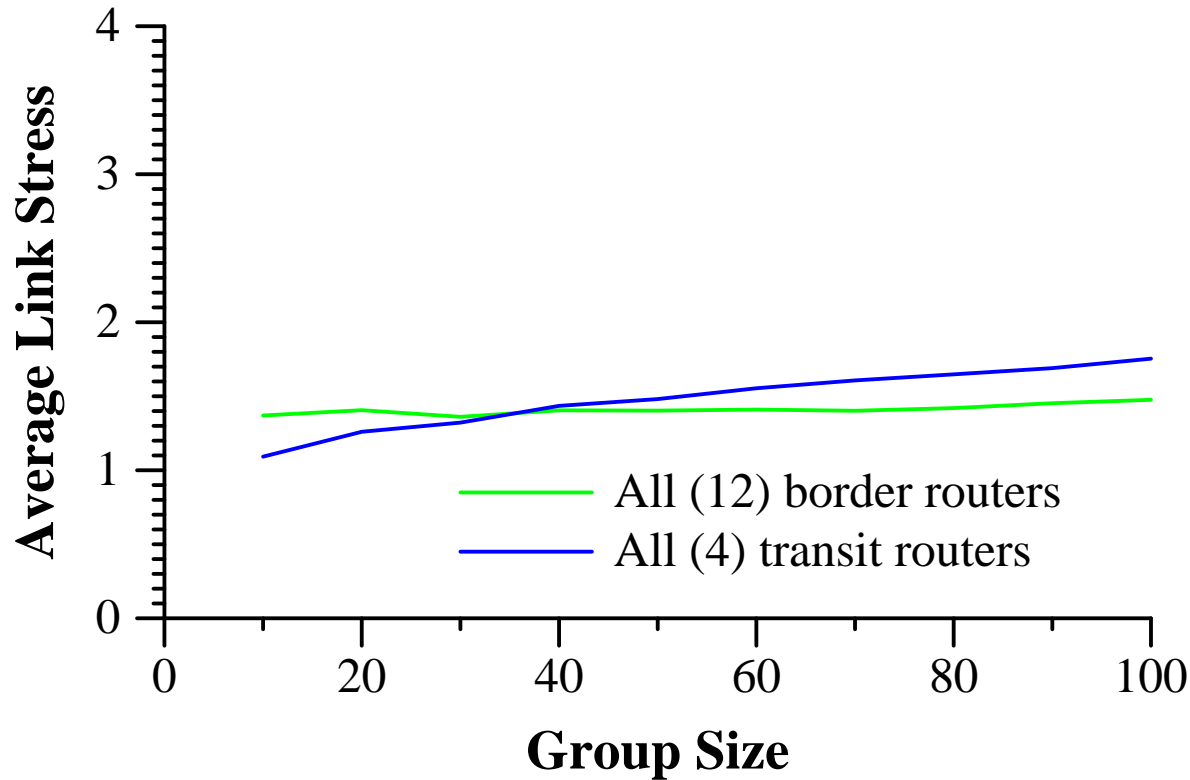
- ▷ Iterated Unicast causes high stress
- ▷ Complete deployment essentially eliminates stress
- ▷ Incremental deployment is also effective

Intelligent deployment helps more



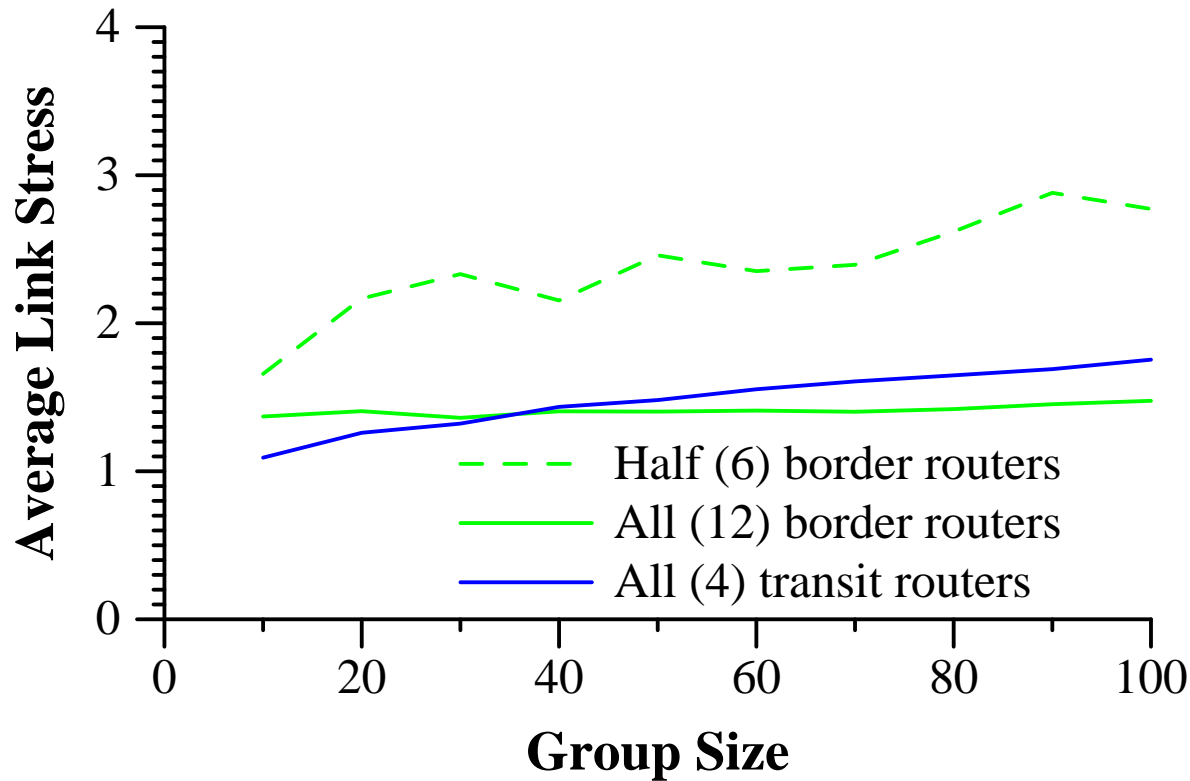
▷ All (4) transit routers

Intelligent deployment helps more



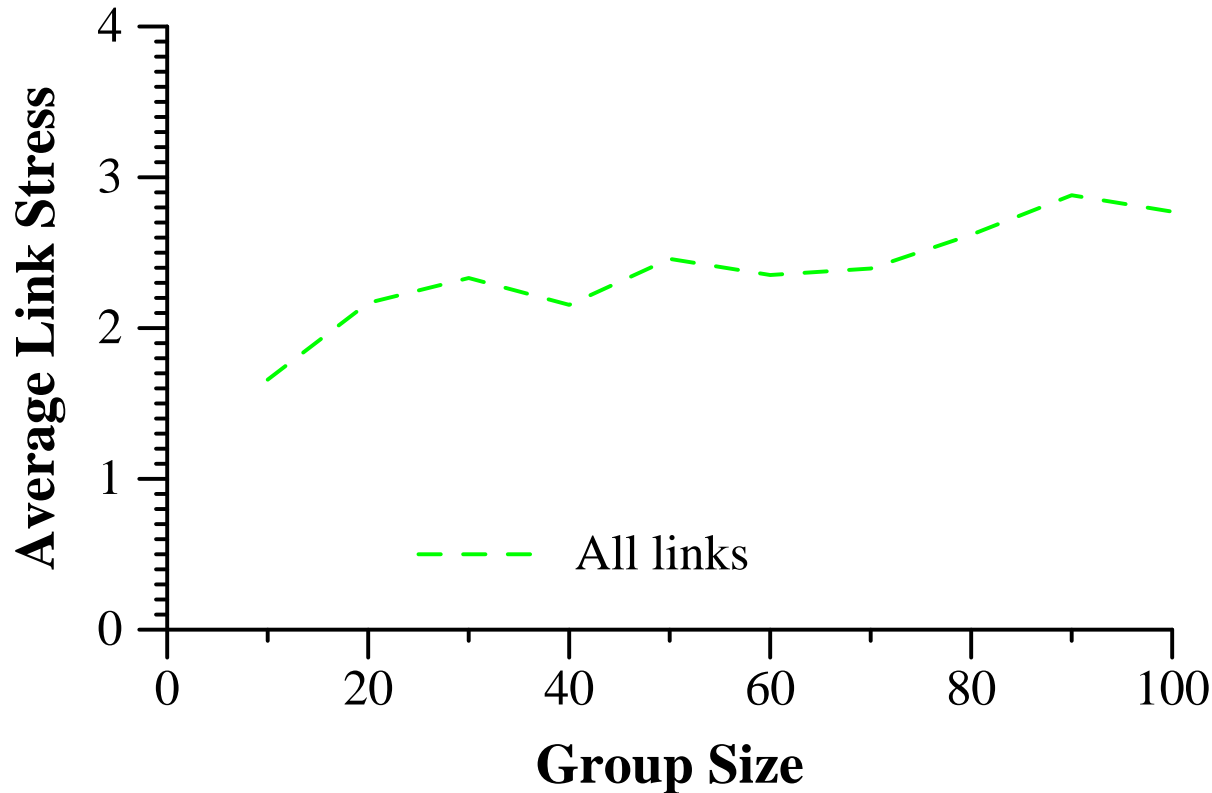
- ▷ All (4) transit routers
- ▷ All (12) border routers

Intelligent deployment helps more



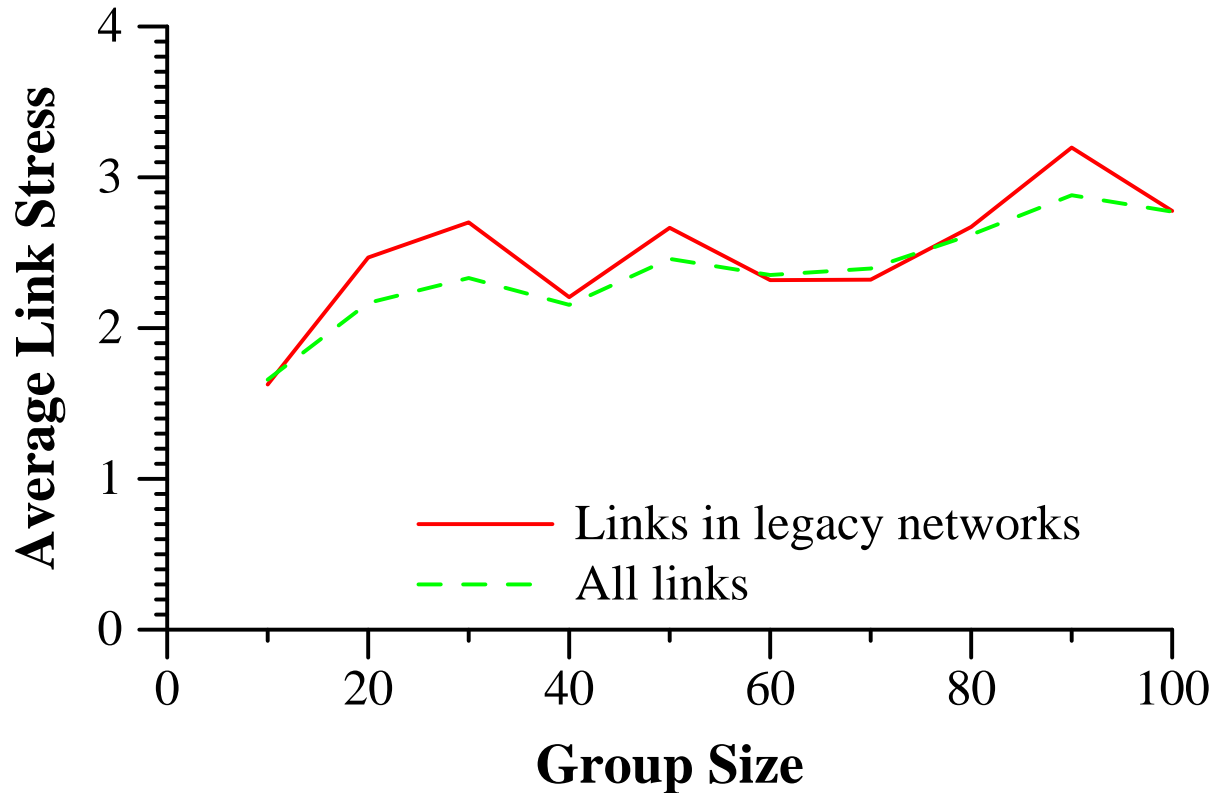
- ▷ All (4) transit routers
- ▷ All (12) border routers
- ▷ Half (6) of the border routers

Local deployment has local benefits



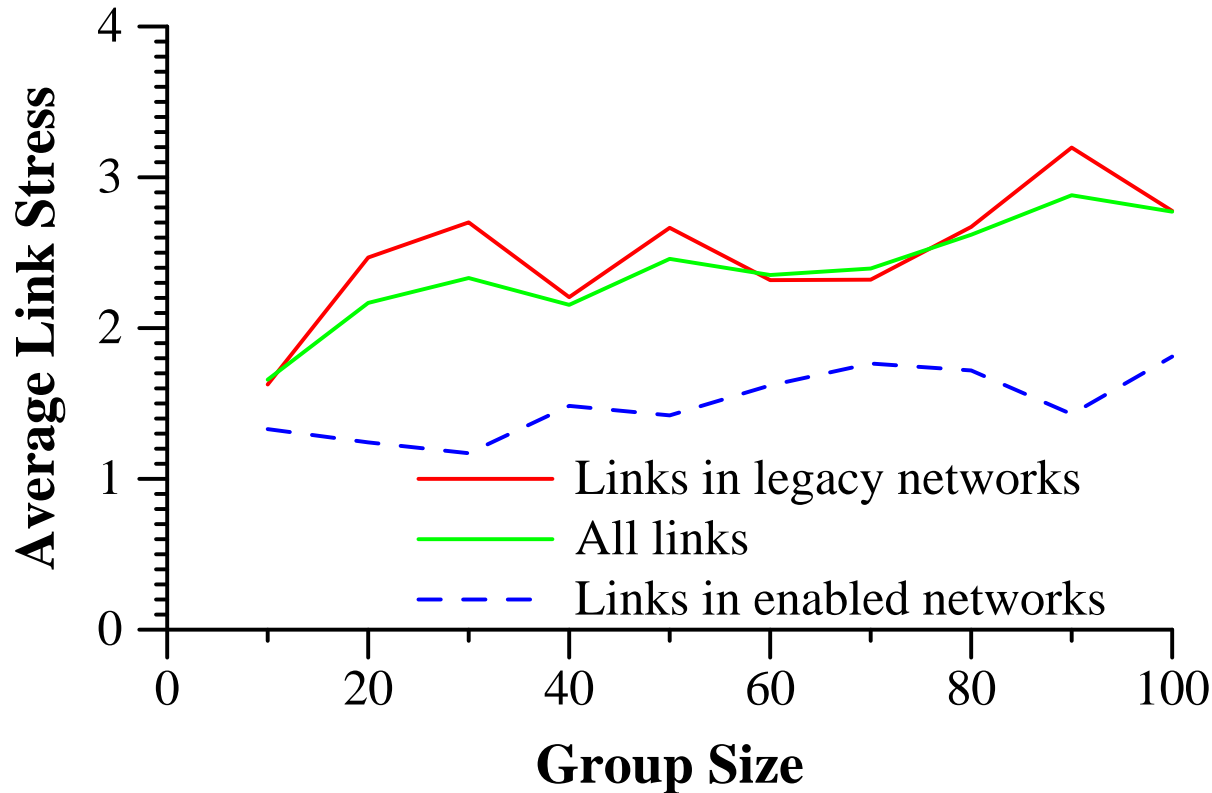
▷ Look closer at the last experiment...

Local deployment has local benefits



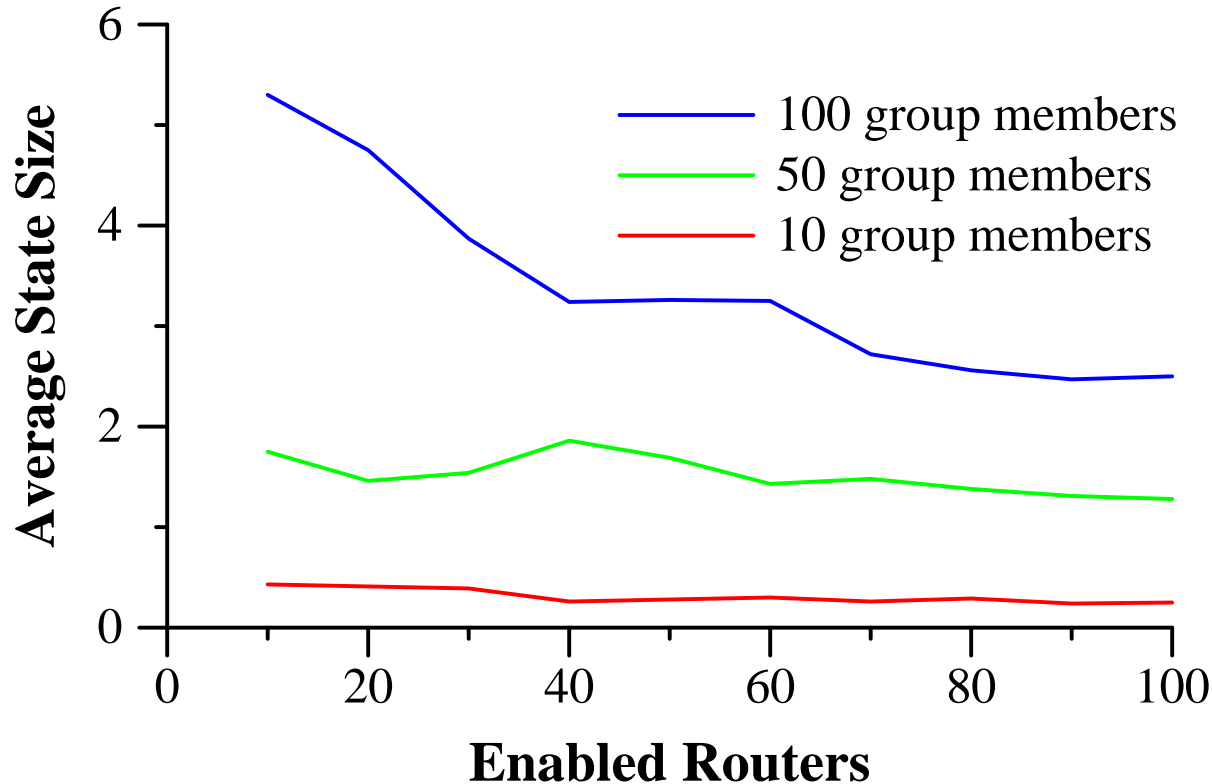
- ▷ Look closer at the last experiment...
- ▷ Networks without enabled routers see little gain

Local deployment has local benefits



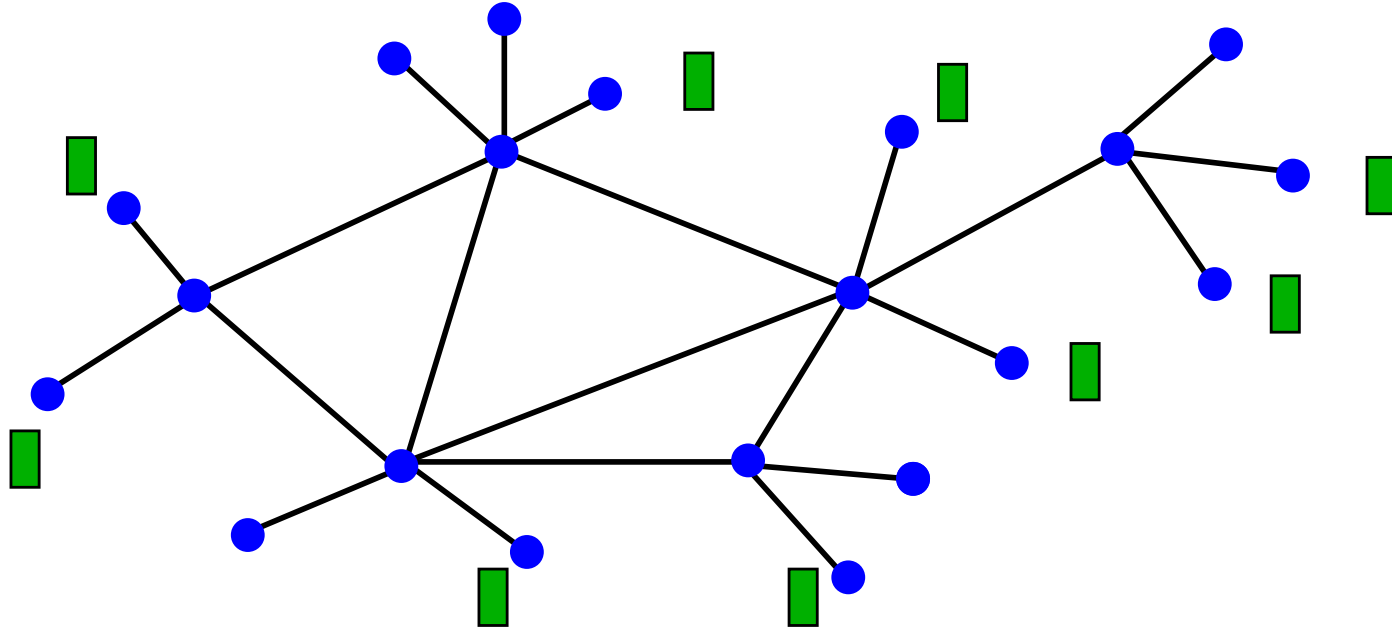
- ▷ Look closer at the last experiment...
- ▷ Networks without enabled routers see little gain
- ▷ Stress is reduced locally with enabled border routers

State requirements



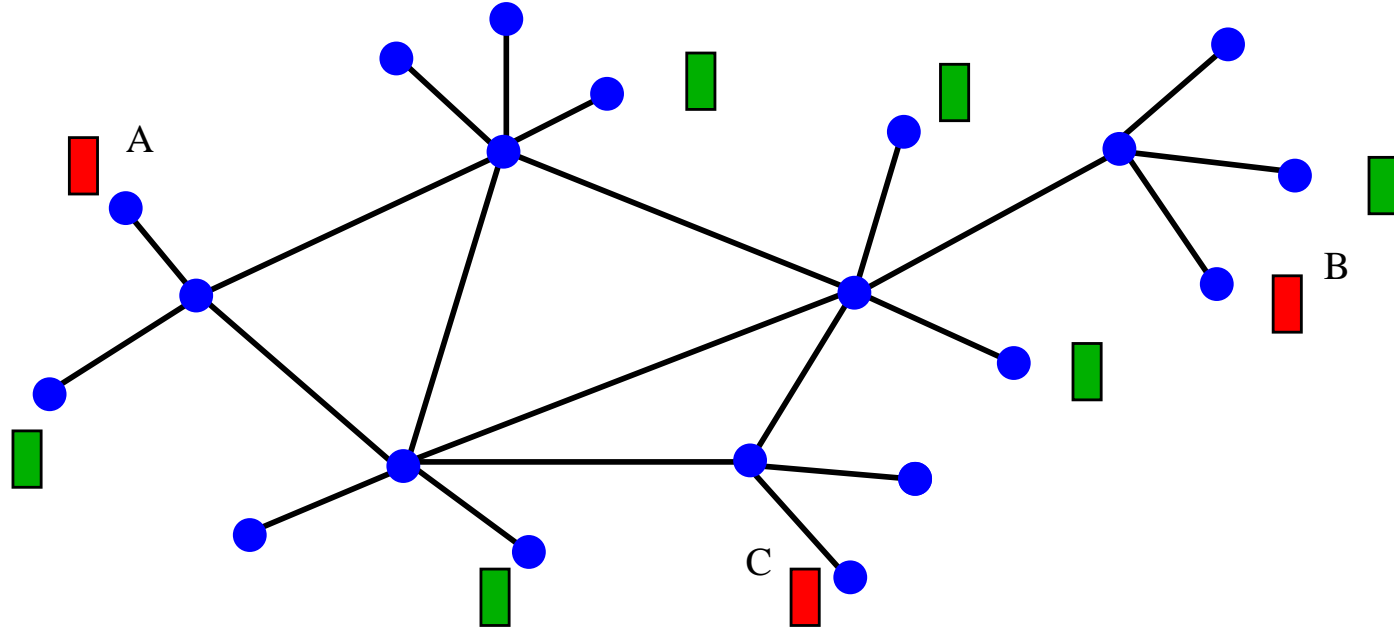
- ▷ State size is number of reflection table entries
- ▷ With the exception of 100 node groups, fairly constant
- ▷ Routers may choose to limit their state

Two-hop Experiments



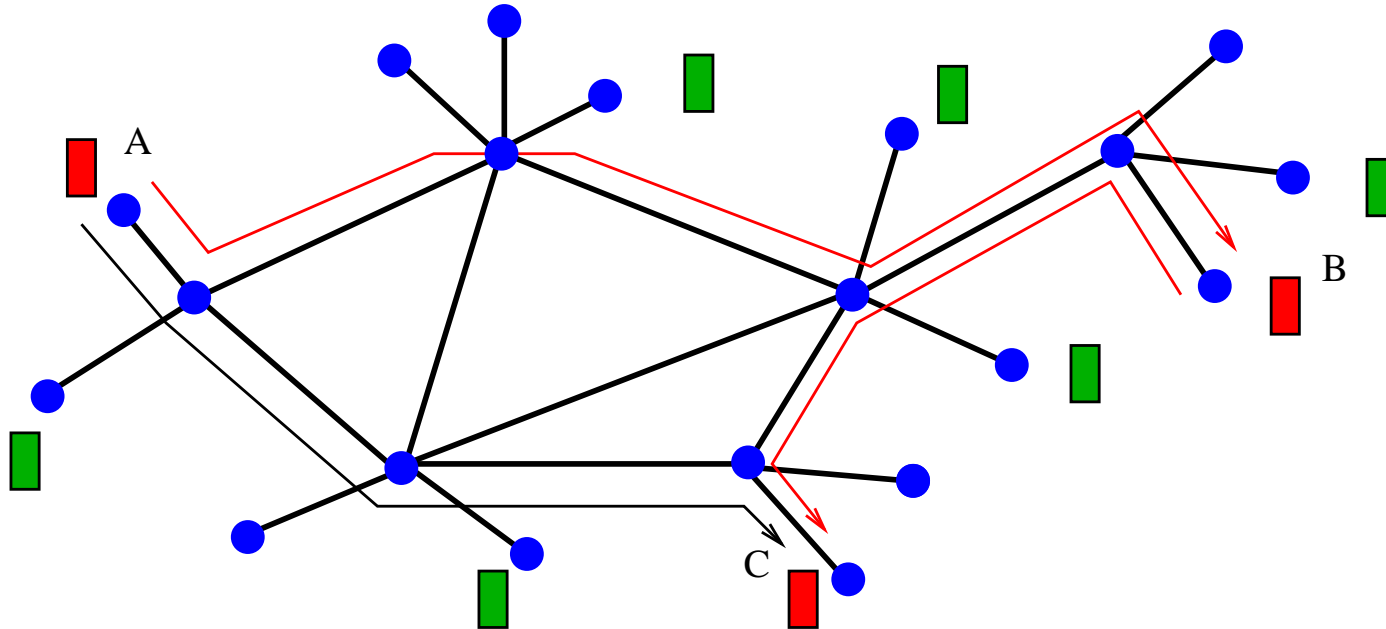
- ▷ Same topologies (100 routers, 100 end-hosts)

Two-hop Experiments



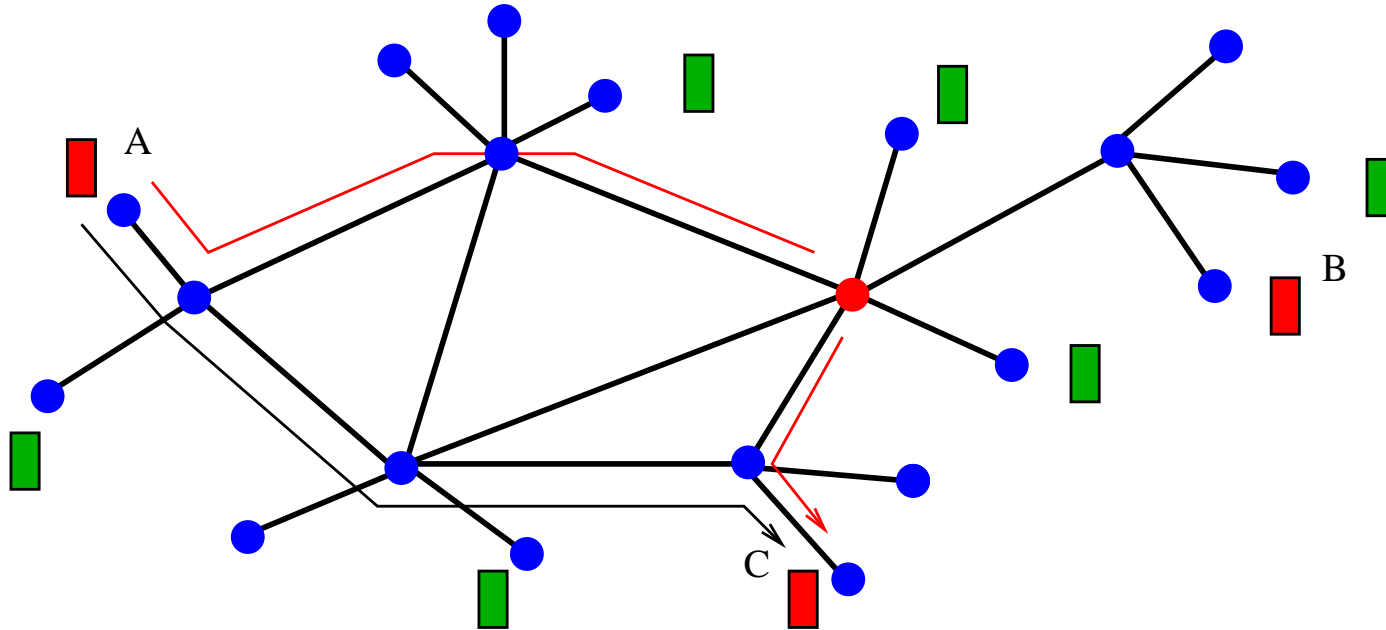
- ▷ Same topologies (100 routers, 100 end-hosts)
- ▷ Choose three nodes, A , B , and C

Two-hop Experiments



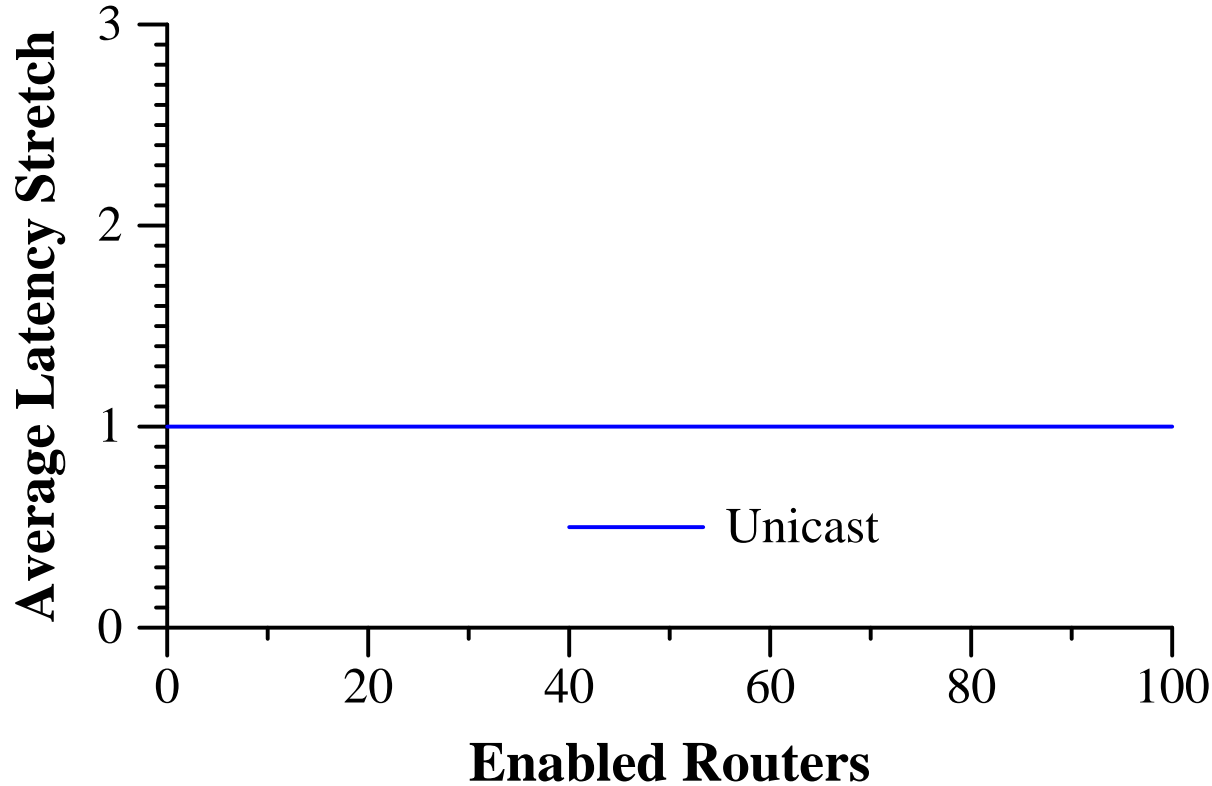
- ▷ Same topologies (100 routers, 100 end-hosts)
- ▷ Choose three nodes, A , B , and C
- ▷ Compare direct routes to two-hop routes

Two-hop Experiments



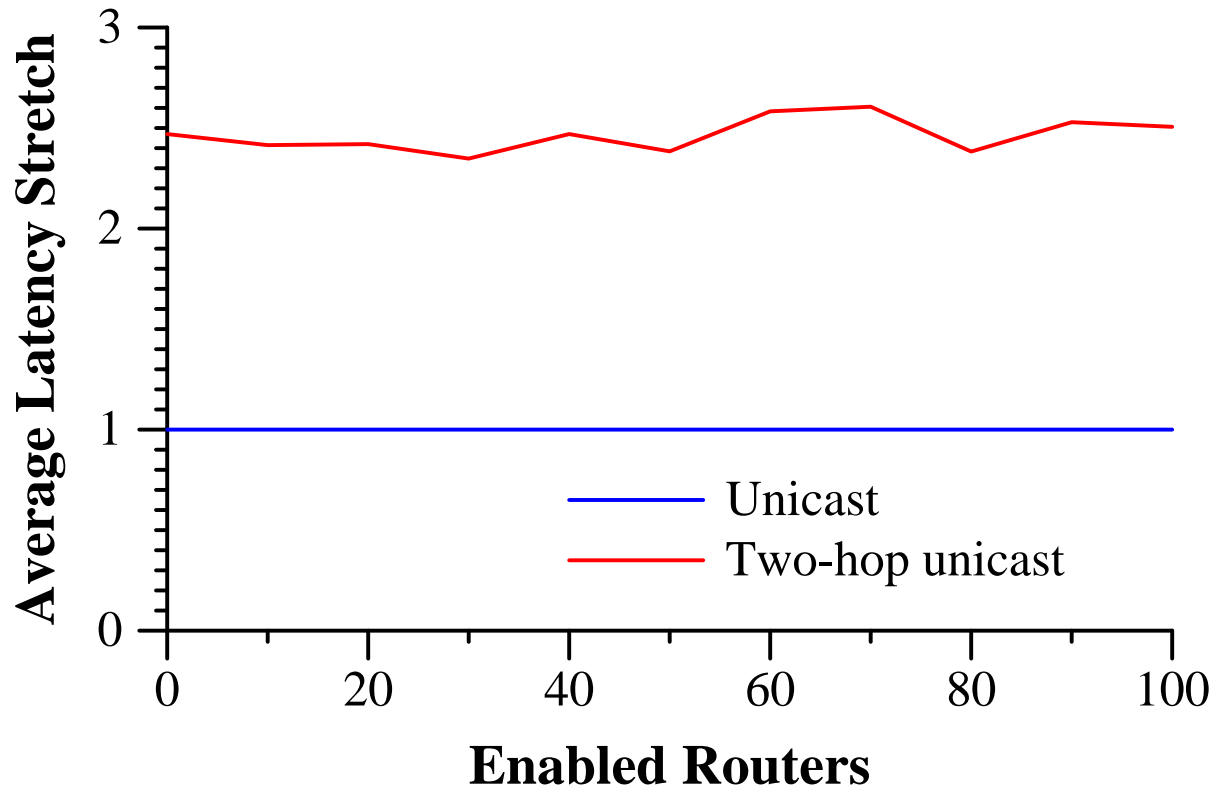
- ▷ Same topologies (100 routers, 100 end-hosts)
- ▷ Choose three nodes, A , B , and C
- ▷ Compare direct routes to two-hop routes
- ▷ Reflect

Optimizing two-hop routes (stretch)



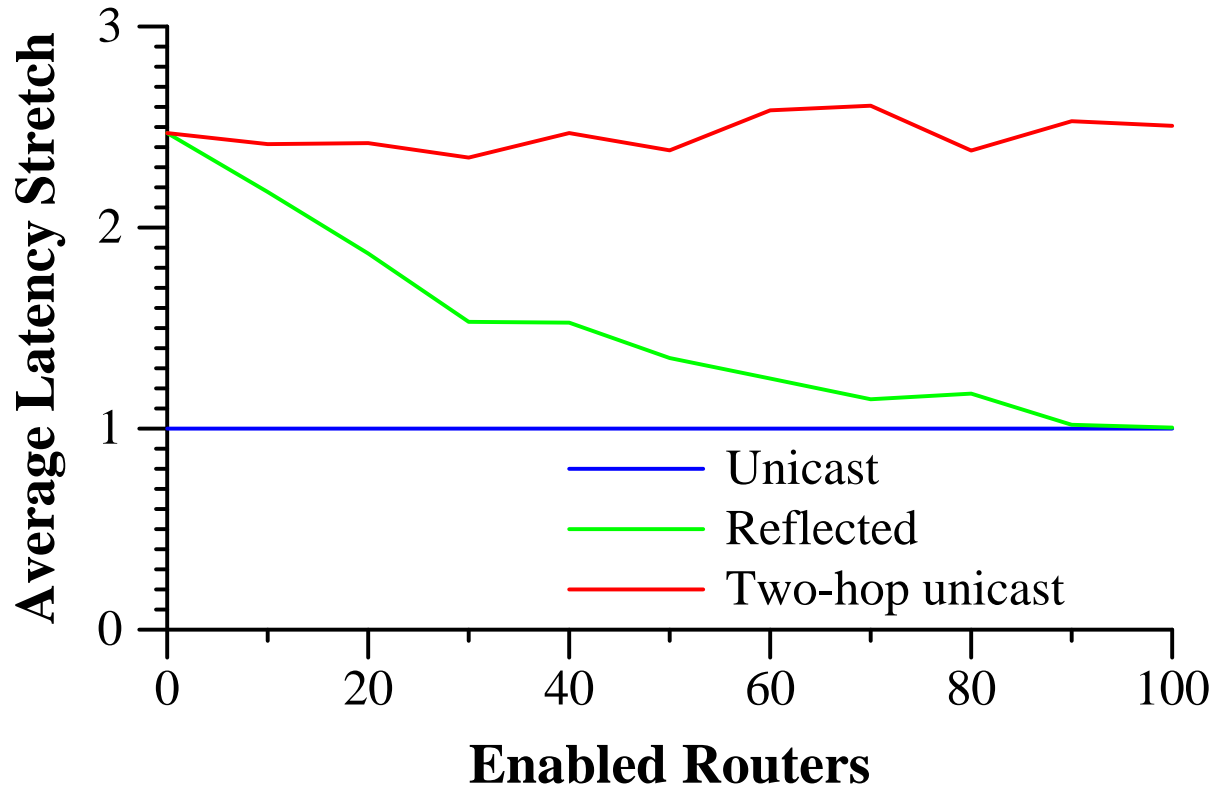
- ▷ Direct unicast (normalized to 1.0)

Optimizing two-hop routes (stretch)



- ▷ Direct unicast (normalized to 1.0)
- ▷ Two-hop routes: stretch is about 2.5

Optimizing two-hop routes (stretch)



- ▷ Direct unicast (normalized to 1.0)
- ▷ Two-hop routes: stretch is about 2.5
- ▷ Optimized with reflection

Related Work

Application-specific router extensions

- ▷ IP Multicast
- ▷ SSM, Express - service model
- ▷ Replier, PGM, BCFS - reliability primitives
- ▷ Reunite [Stoica00] - incremental deployment, unicast hops

Generic router extensions

- ▷ Active Networks
- ▷ Ephemeral state probes, lightweight processing modules [CGW01,02]

Conclusions

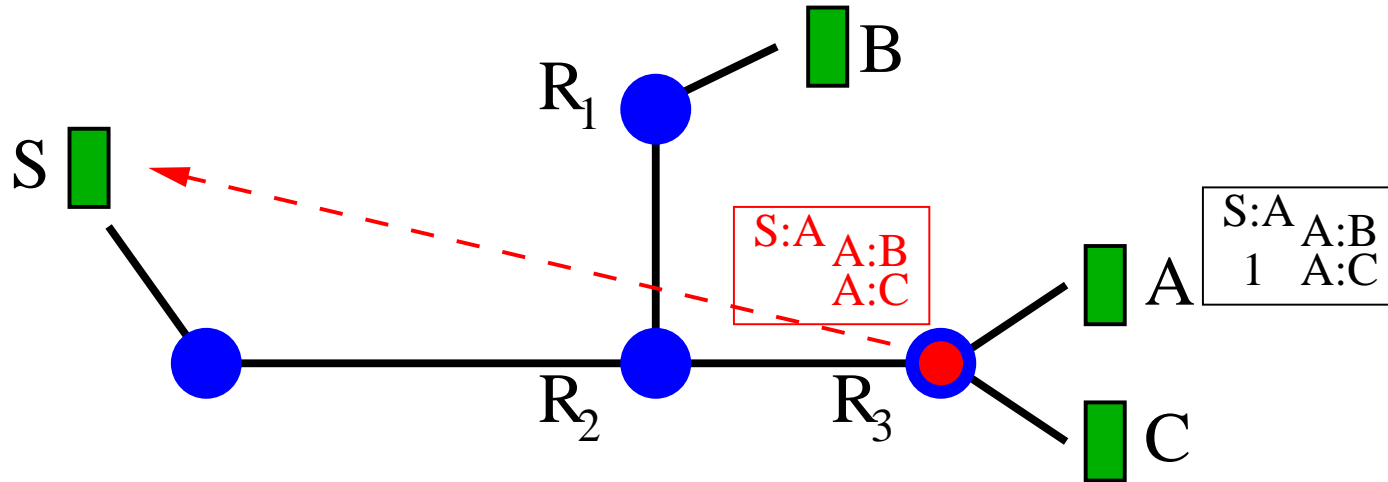
Overlay networks offer flexibility, but currently come at too high of a price.

But a minimal set of router extensions can

- ▷ reduce stress and stretch in various overlay applications
- ▷ be incrementally deployed
- ▷ support a wide variety of applications

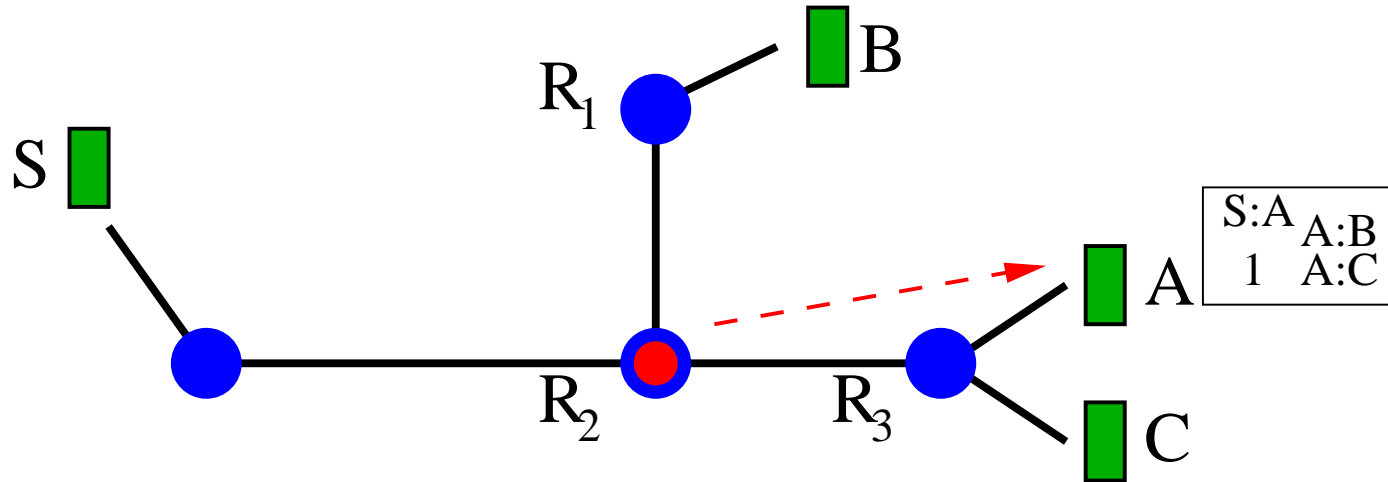
Virtualization is a powerful idea. Networks should support it.

Anatomy of a Single Request



▷ $R_3 \rightarrow S : ASK(S \rightarrow A, 1, \{A \rightarrow B, A \rightarrow C\})$

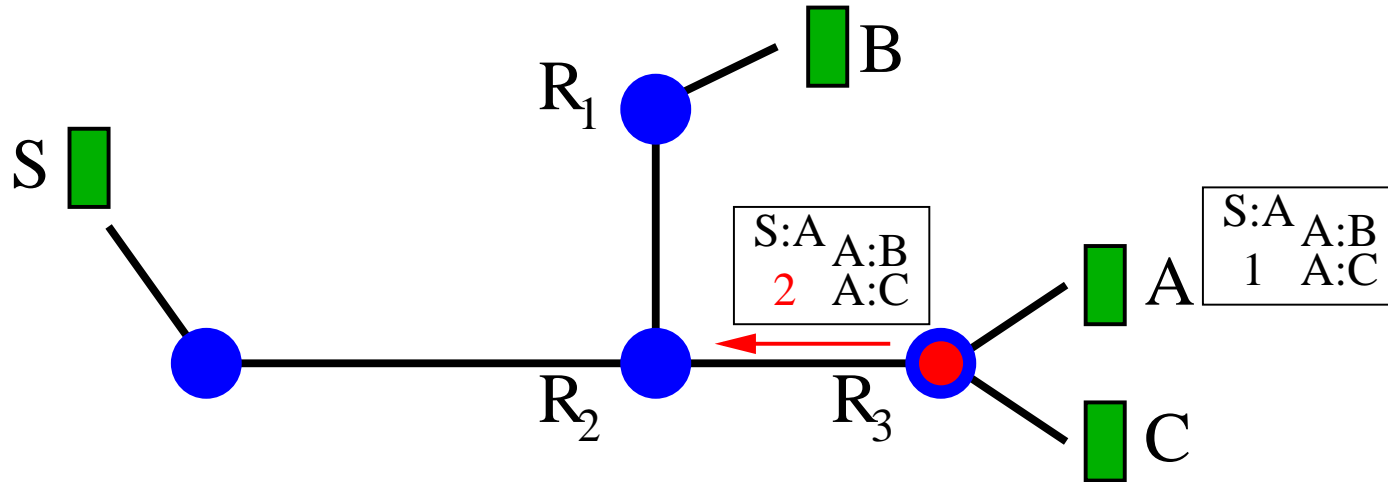
Anatomy of a Single Request



▷ $R_3 \rightarrow S : ASK(S \rightarrow A, 1, \{A \rightarrow B, A \rightarrow C\})$

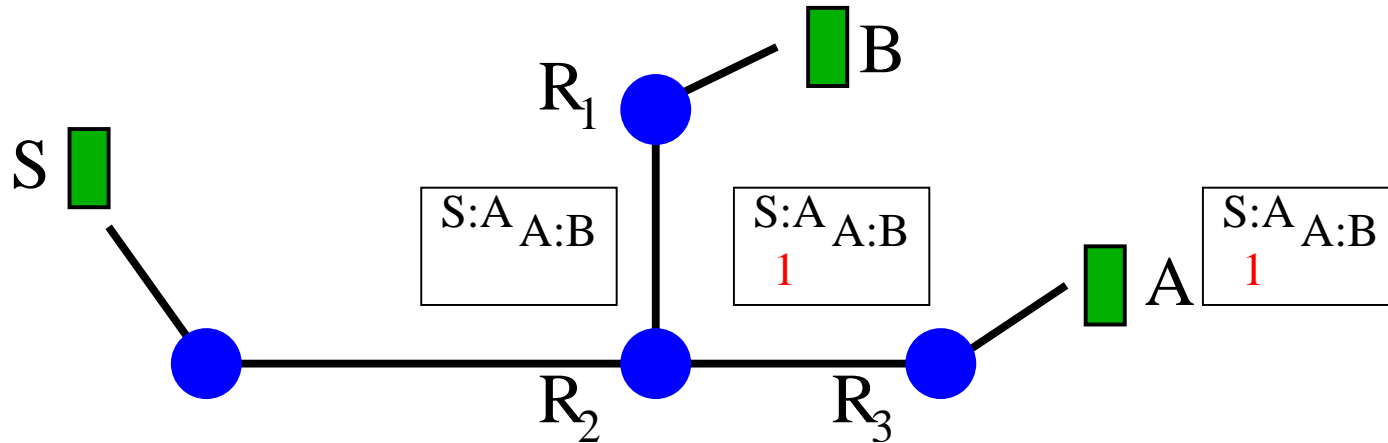
▷ $R_2 \rightarrow A : OFFER(S \rightarrow A, 1, \{A \rightarrow B\}, 81723)$

Anatomy of a Single Request



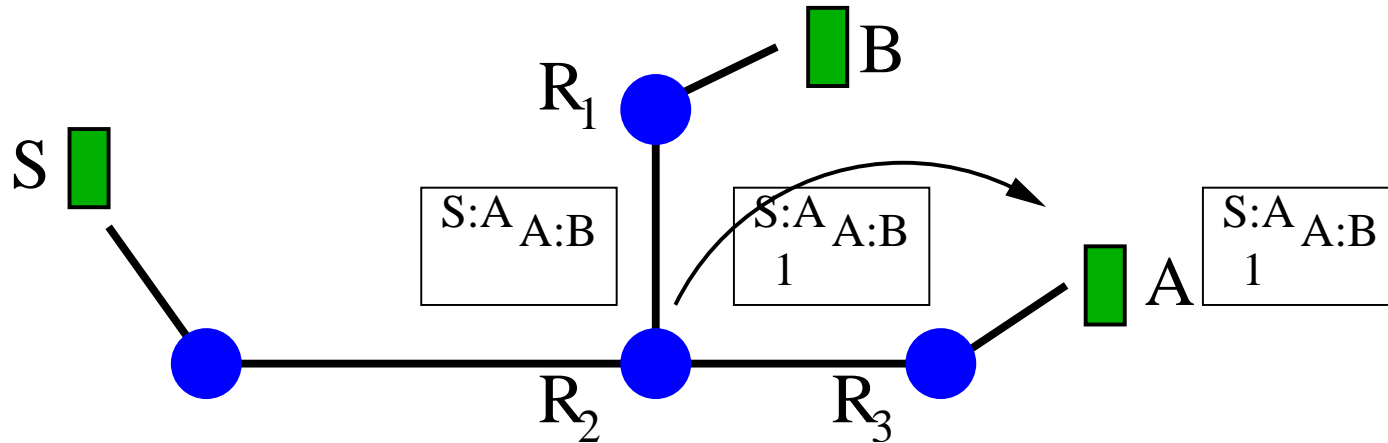
- ▷ $R_3 \rightarrow S : ASK(S \rightarrow A, 1, \{A \rightarrow B, A \rightarrow C\})$
- ▷ $R_2 \rightarrow A : OFFER(S \rightarrow A, 1, \{A \rightarrow B\}, 81723)$
- ▷ $R_3 \rightarrow R_2 : DEMAND(S \rightarrow A, 2, \{A \rightarrow B\}, 81723)$

Tags need not always change



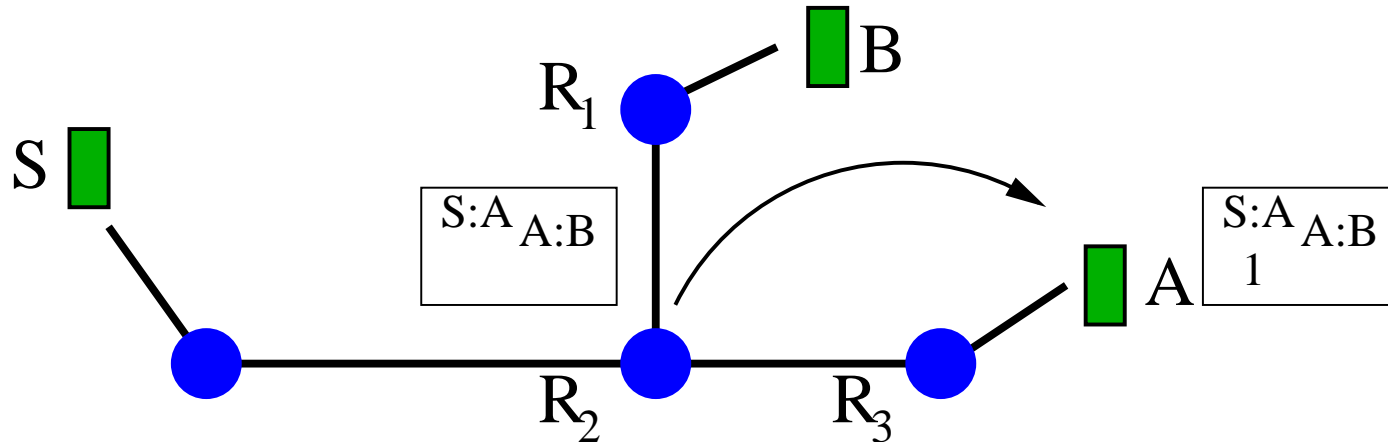
- ▷ If the request is unchanged, neither is the tag

Tags need not always change



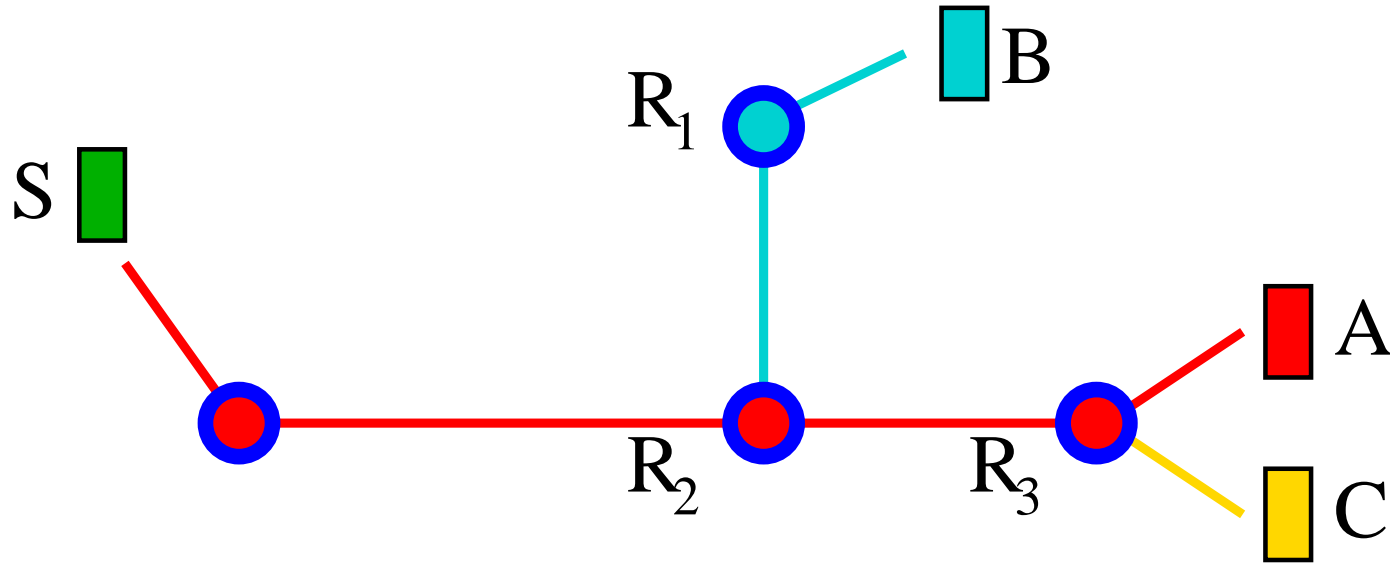
- ▷ If the request is unchanged, neither is the tag
- ▷ Unchanged tags permit network “surprises”

Tags need not always change



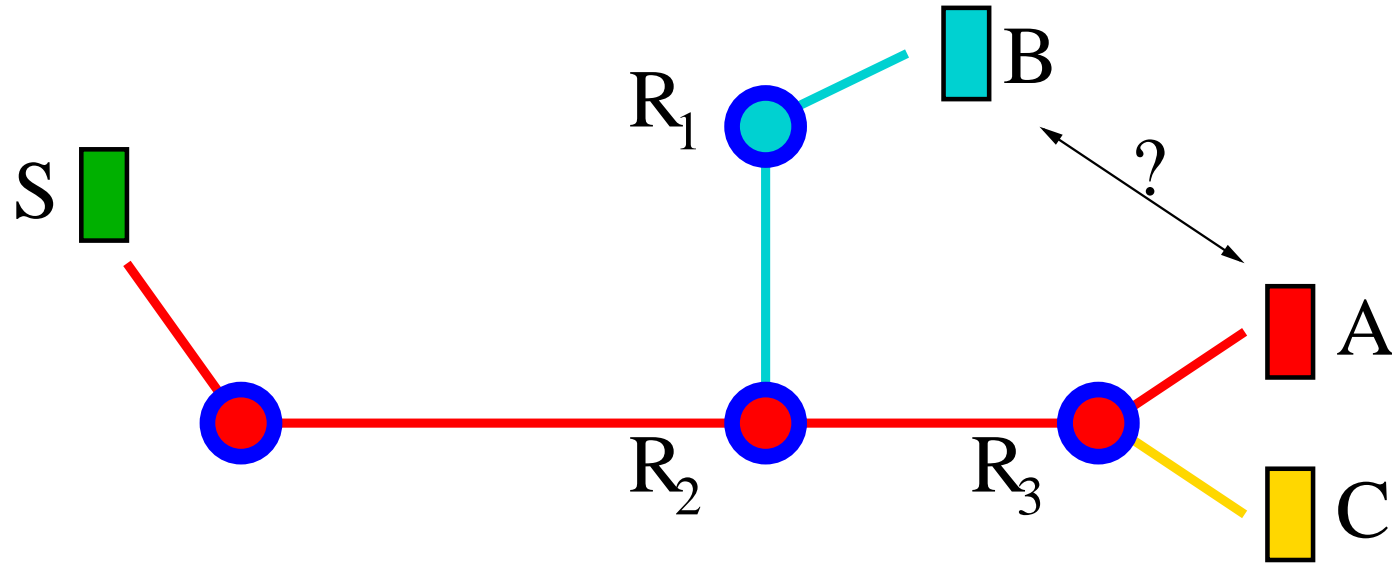
- ▷ If the request is unchanged, neither is the tag
- ▷ Unchanged tags permit network “surprises”
- ▷ R_3 could throw out its table

Overlay nodes retain control: concede



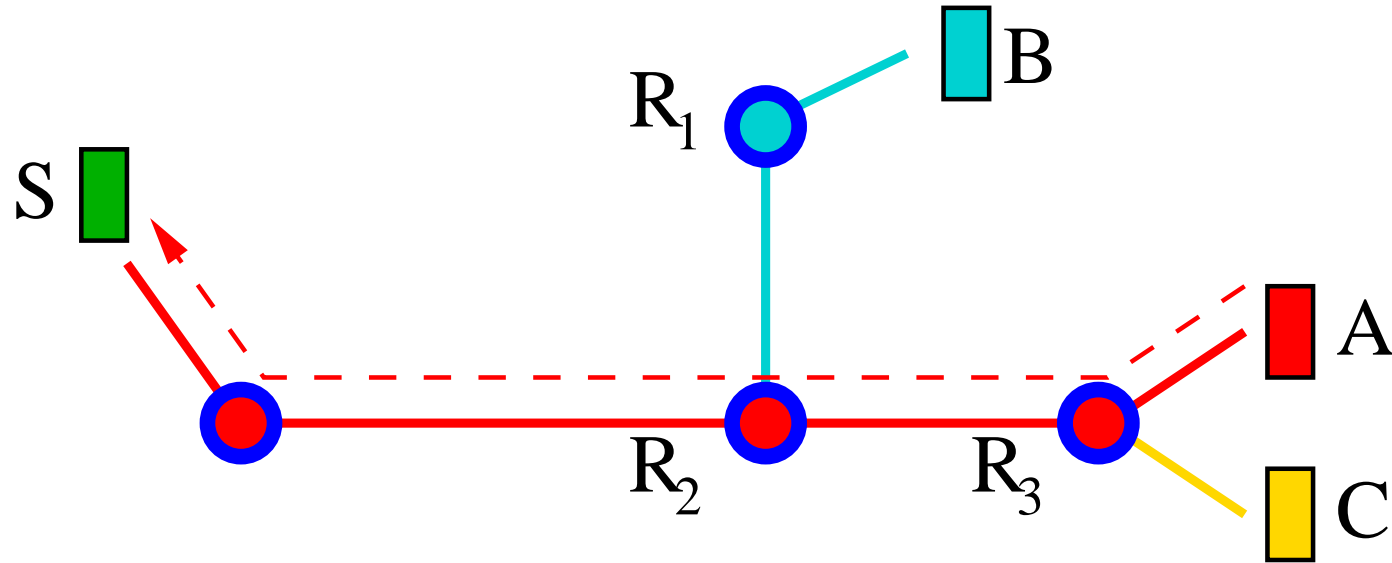
▷ Topology given by paint

Overlay nodes retain control: concede



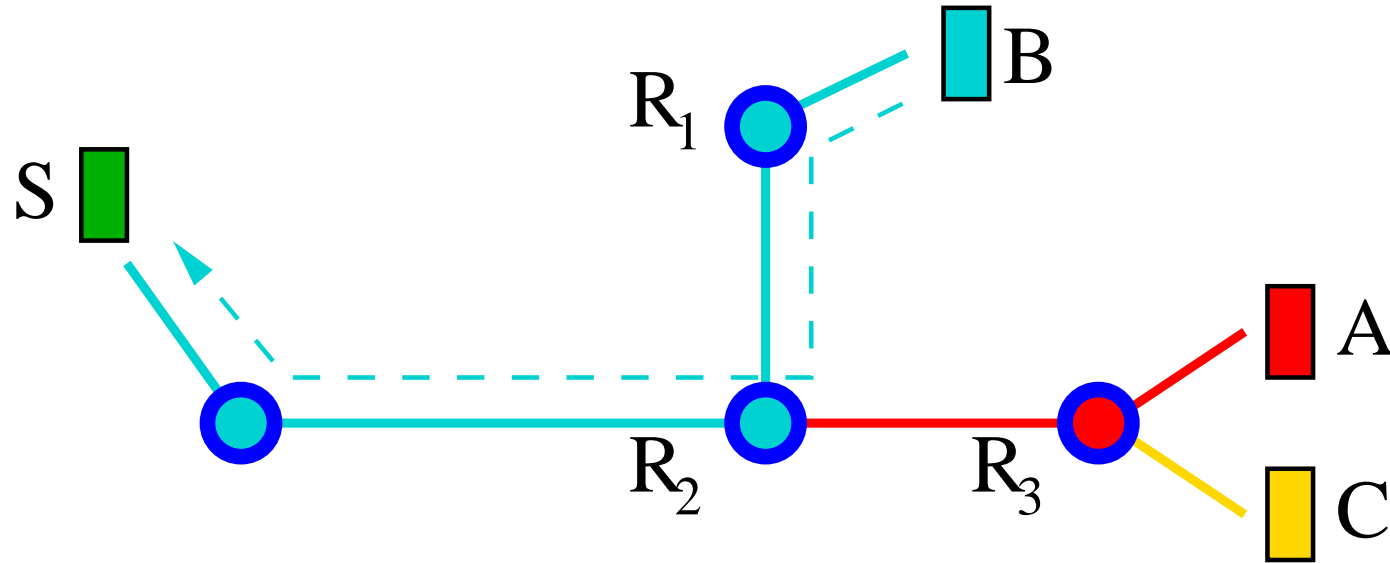
- ▷ Topology given by paint
- ▷ A and B decide B 's paint should continue

Overlay nodes retain control: concede



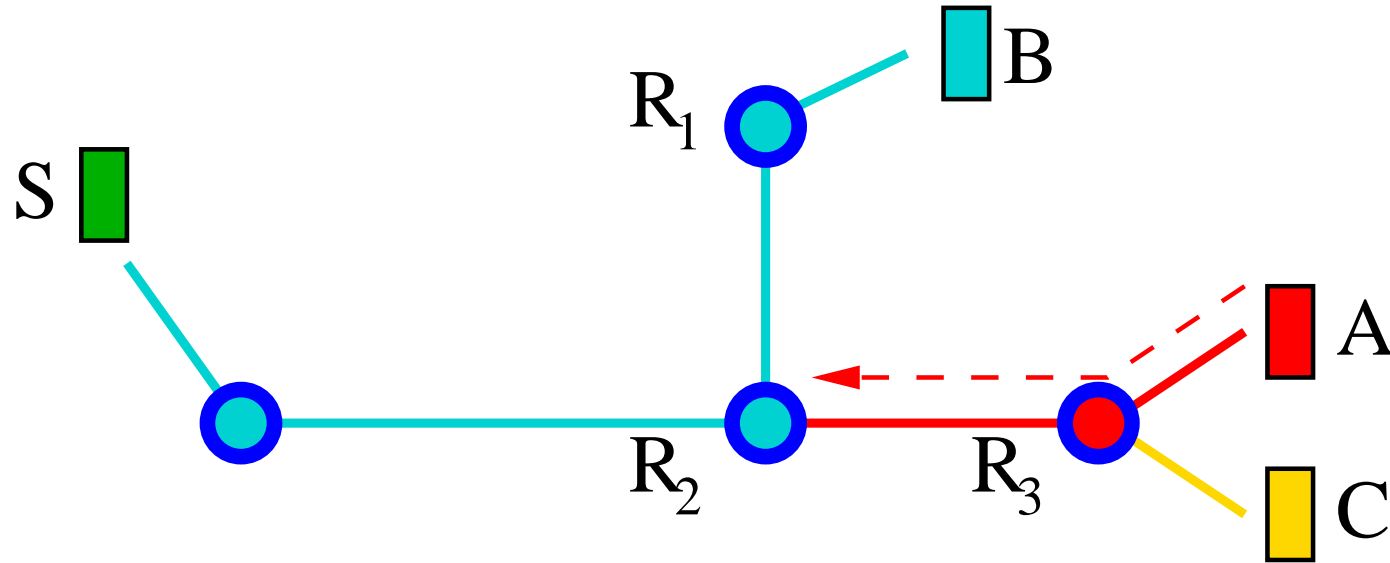
- ▷ Topology given by paint
- ▷ A and B decide B 's paint should continue
- ▷ $A \rightarrow S : \text{Paint}(S, \text{concede} = B)$ (stops at S)

Overlay nodes retain control: concede



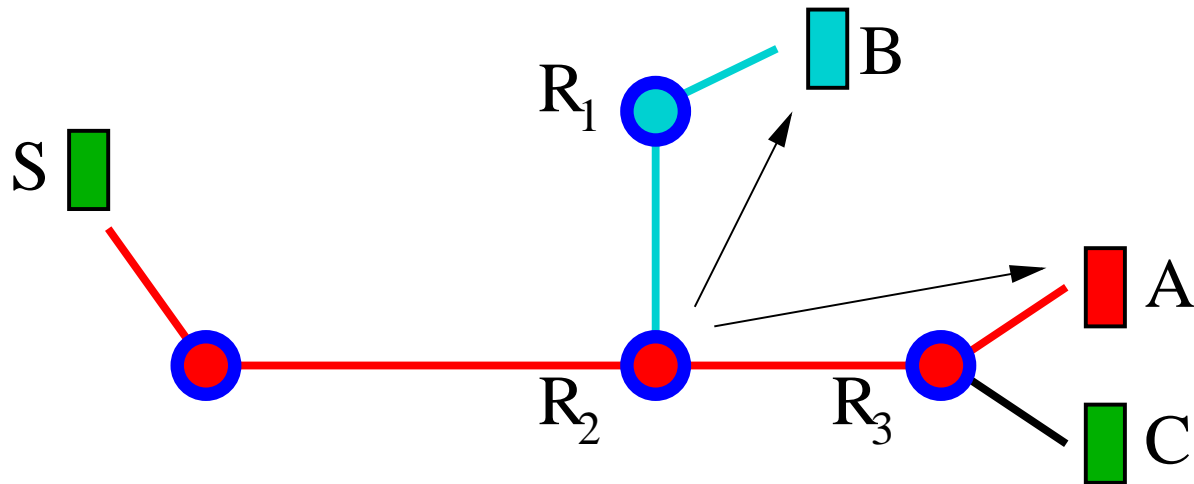
- ▷ Topology given by paint
- ▷ A and B decide B 's paint should continue
- ▷ $A \rightarrow S : Paint(S, concede = B)$ (stops at S)
- ▷ $B \rightarrow S : Paint(S)$ (stops at S)

Overlay nodes retain control: concede



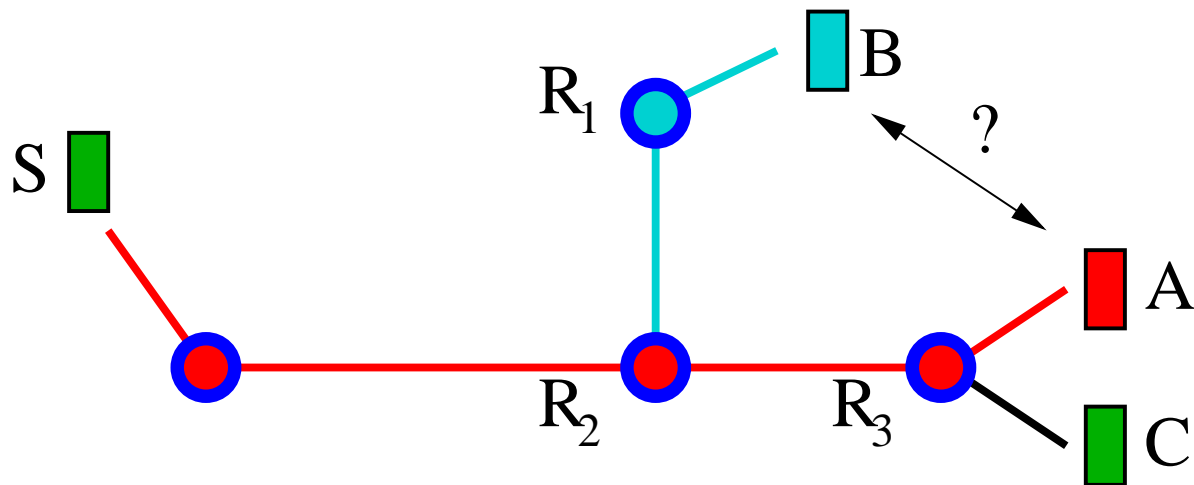
- ▷ Topology given by paint
- ▷ A and B decide B 's paint should continue
- ▷ $A \rightarrow S : \text{Paint}(S, \text{concede} = B)$ (stops at S)
- ▷ $B \rightarrow S : \text{Paint}(S)$ (stops at S)
- ▷ $A \rightarrow S : \text{Paint}(S, \text{concede} = B)$ (stops at R_2)

Admission control: ignoring A



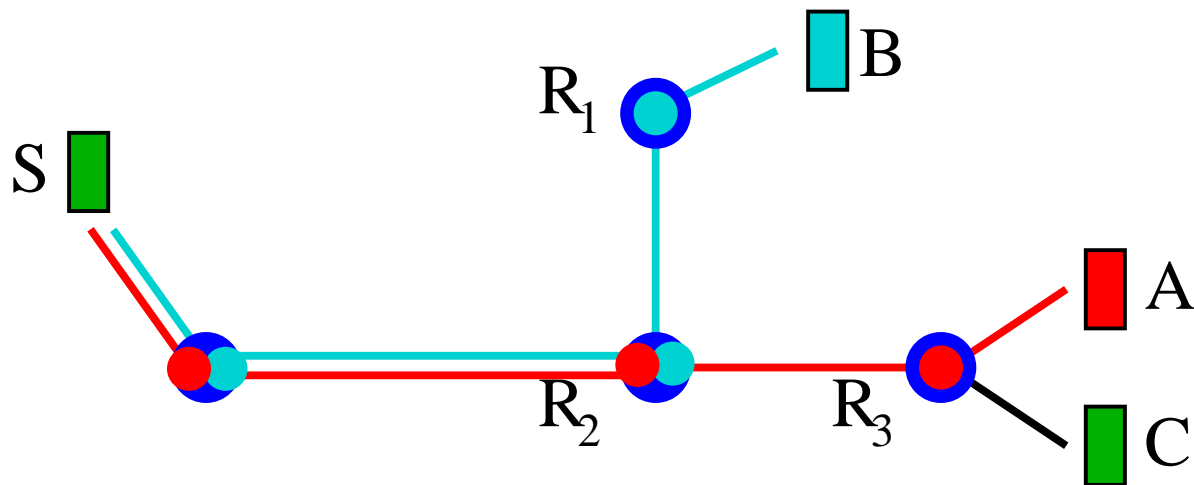
▷ $B \rightarrow S : Paint(S)$ (stops at R_2)

Admission control: ignoring A



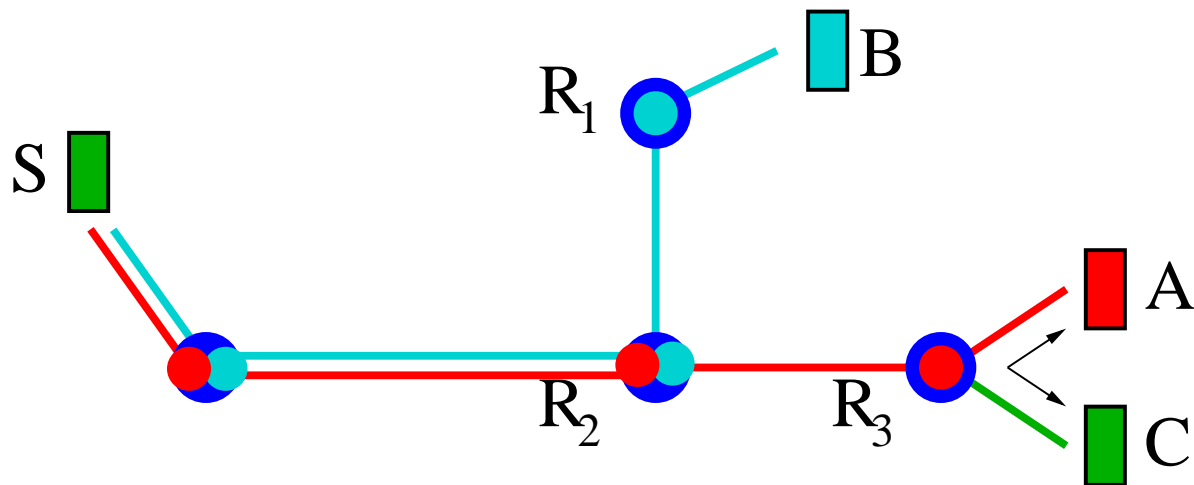
- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)
- ▷ B is unwilling to accept A

Admission control: ignoring A



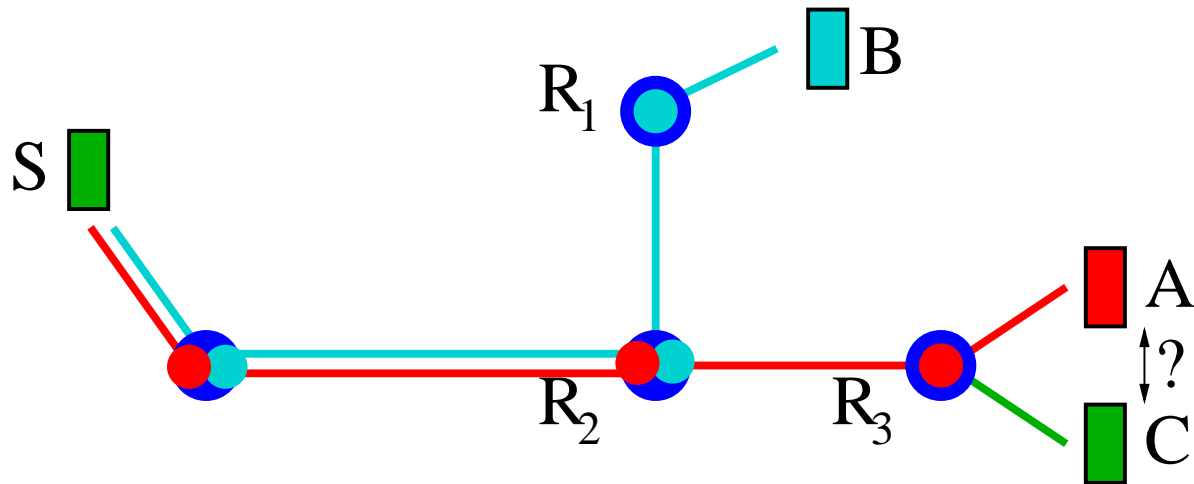
- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)
- ▷ B is unwilling to accept A
- ▷ $B \rightarrow S : Paint(S, ignore = A)$ (stops at S)

Admission control: ignoring A



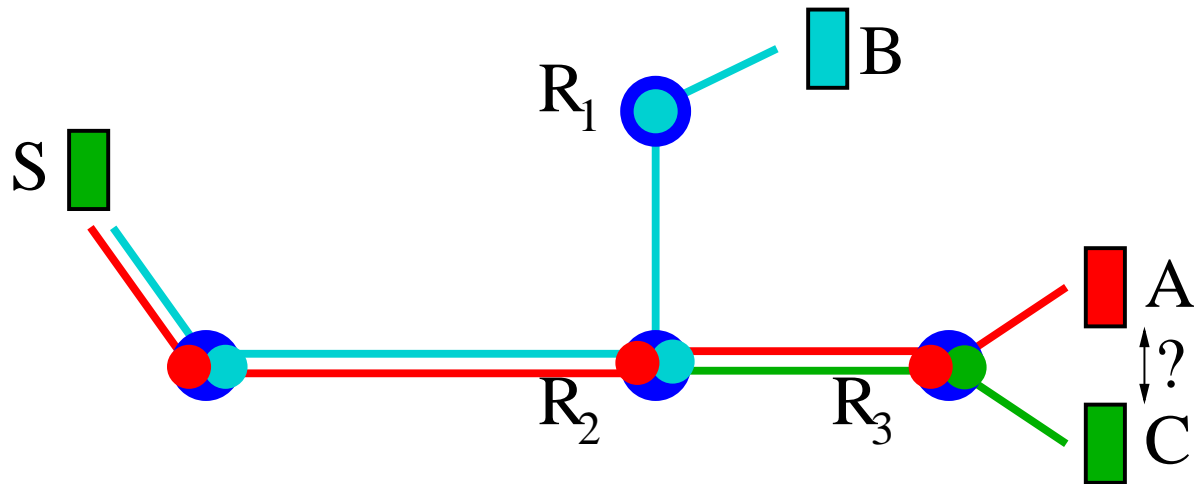
- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)
- ▷ B is unwilling to accept A
- ▷ $B \rightarrow S : Paint(S, ignore = A)$ (stops at S)
- ▷ $C \rightarrow S : Paint(S)$ (stops at R_3)

Admission control: ignoring A



- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)
- ▷ B is unwilling to accept A
- ▷ $B \rightarrow S : Paint(S, ignore = A)$ (stops at S)
- ▷ $C \rightarrow S : Paint(S)$ (stops at R_3)
- ▷ C is unwilling to accept A

Admission control: ignoring A



- ▷ $B \rightarrow S : Paint(S)$ (stops at R_2)
- ▷ B is unwilling to accept A
- ▷ $B \rightarrow S : Paint(S, ignore = A)$ (stops at S)
- ▷ $C \rightarrow S : Paint(S)$ (stops at R_3)
- ▷ C is unwilling to accept A
- ▷ $C \rightarrow S : Paint(S, ignore = A)$ (stops at R_2)