802.1aq Link State Protocol and Loop Prevention

Dave Allan dallan@nortel.com

Jerome Chiabaut chiabaut@nortel.com
Nigel Bragg nbragg@nortel.com
Don Fedyk dwfedyk@nortel.com

NORTEL

From aq-farkas-loop-prevention-1107

Ingress checking

> Frames not arriving on the shortest path from the Source Bridge are discarded
> Makes the tree directed
> Good for loop prevention in most cases
> Transient loops may appear
> Ingress filtering has to be modified
> **Spreading of frames confined to the traffic trapped in loop when it closed**
  - **Loop's only valid source is itself**

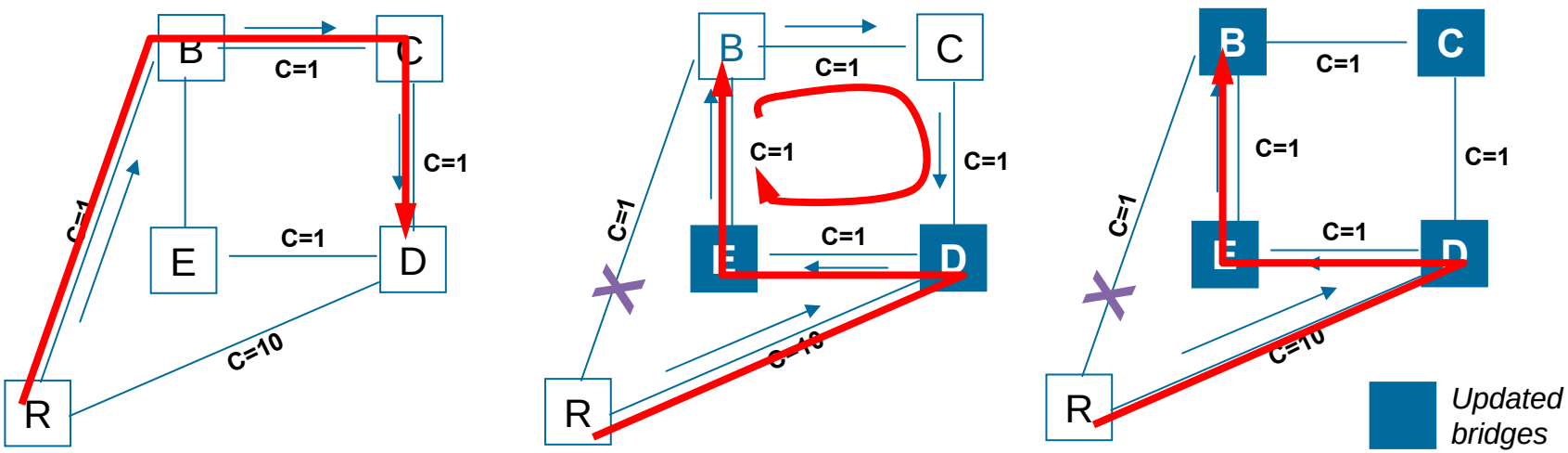
TTL (hop-count)

> Looped frames are discarded after a while
> Spreading of multicast frames lasts shorter but not eliminated
  - **because the duration of looping of an individual frame is bounded,**
  - **however, traffic can continue to enter loop after it has formed**
> New field in header

Multicast without Ingress Check or consistent tie-breaking

B&D are leaves on multicast tree rooted at R

Tie breaking of BCD vs. BED is not consistent



This discussion focuses exclusively on looping of **multicast** frames :

> this has more serious consequences and suppression is more challenging than the looping of unicast frames.

The combination of promiscuous receipt of multicast frames, and inconsistent tie-breaking means a ***single failure*** can produce a loop

> The amount of traffic looping is **Σ** offered load for duration of the loop

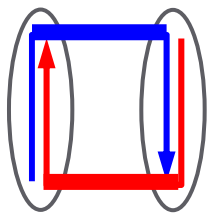
Minimum Complexity of a loop with Ingress Check

> The minimum number of simultaneous topology changes to produce a loop with Ingress Check & consistent tie breaking is two

> If an individual node is not allowed to have two different views of the network simultaneously, a loop produced by two topology changes will have at least four nodes in it

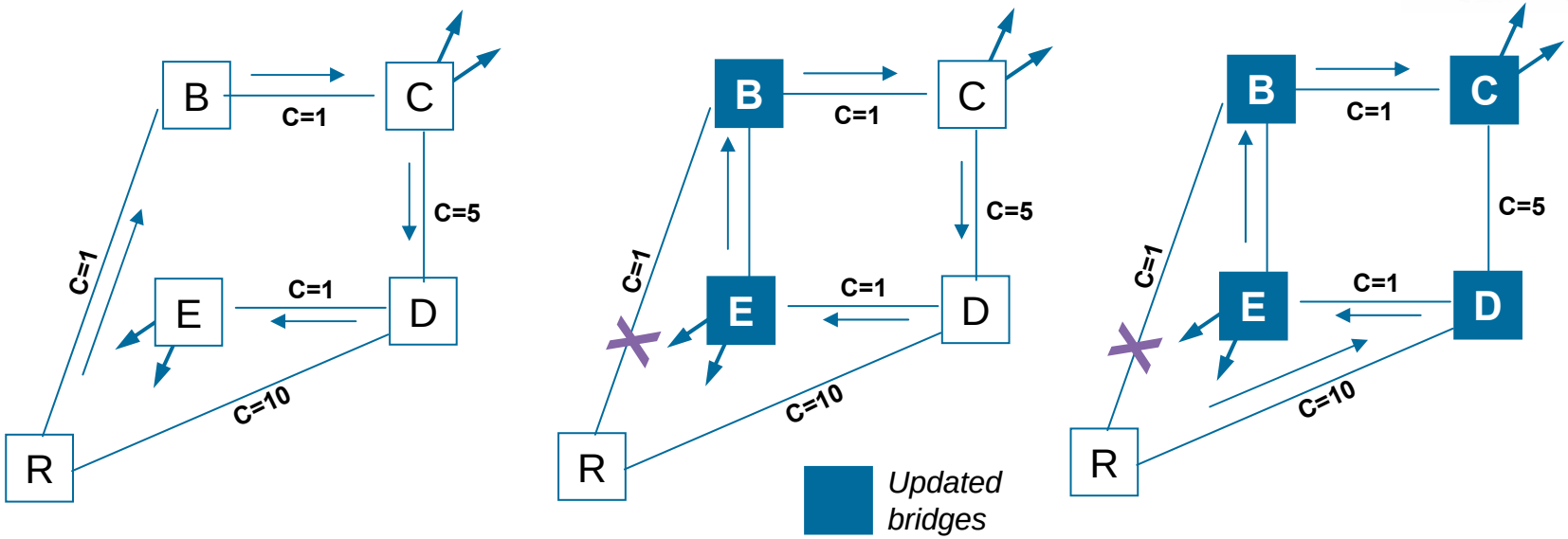
> Why?

- Two or more paths must be stitched together to form a loop
  - Without loss of generality, assume two, non-looping, paths
- These paths can't agree all the way around the loop
  - There must be a break in each of them since they are acyclic
- There must be section of the loop which is unique to each path
  - This section closes the loop when added to the other path
- These sections must be joined by common sections:
  - section of path1 → common section → section of path2 → common section
- The two common sections are distinct and each has a node at each end



Transient loop example (min. complexity)

(from aq-farkas-loop-prevention-1107)



Generalized description:

Minimum of two ~simultaneous topology changes,

one to close the loop,

one to relocate the shortest path from the loop to the root

A minimum of two sets of a minimum of two nodes each,

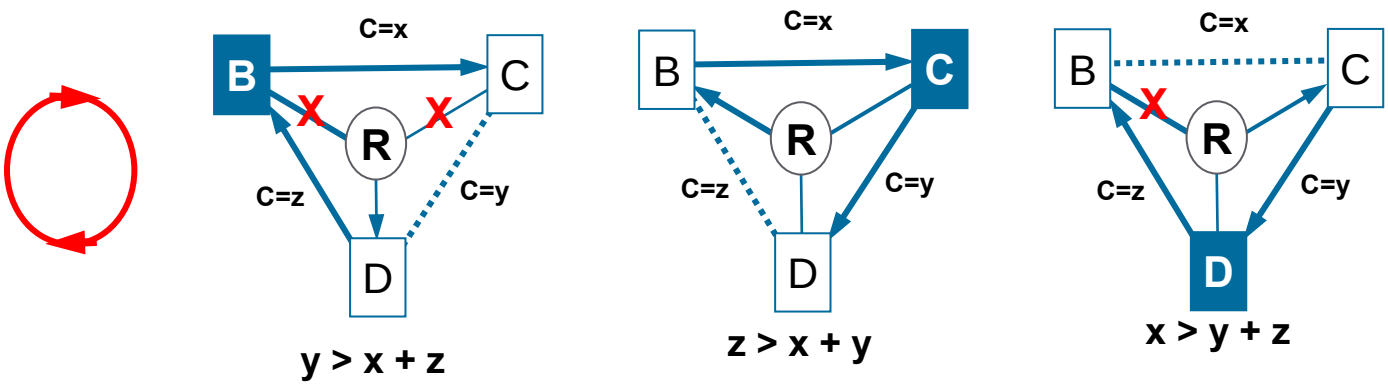
FIB in one set of nodes (C&D) corresponds to a previous graph state,

FIB in one set of nodes (B&E) corresponds to a new graph state

More on loops

> This four-node loop example may be generalised to a loop of arbitrary size :
  - by adding “padding” nodes into the sections;
  - each node must share the topology view of its neighbours for the loop to form (previous or new graph state)

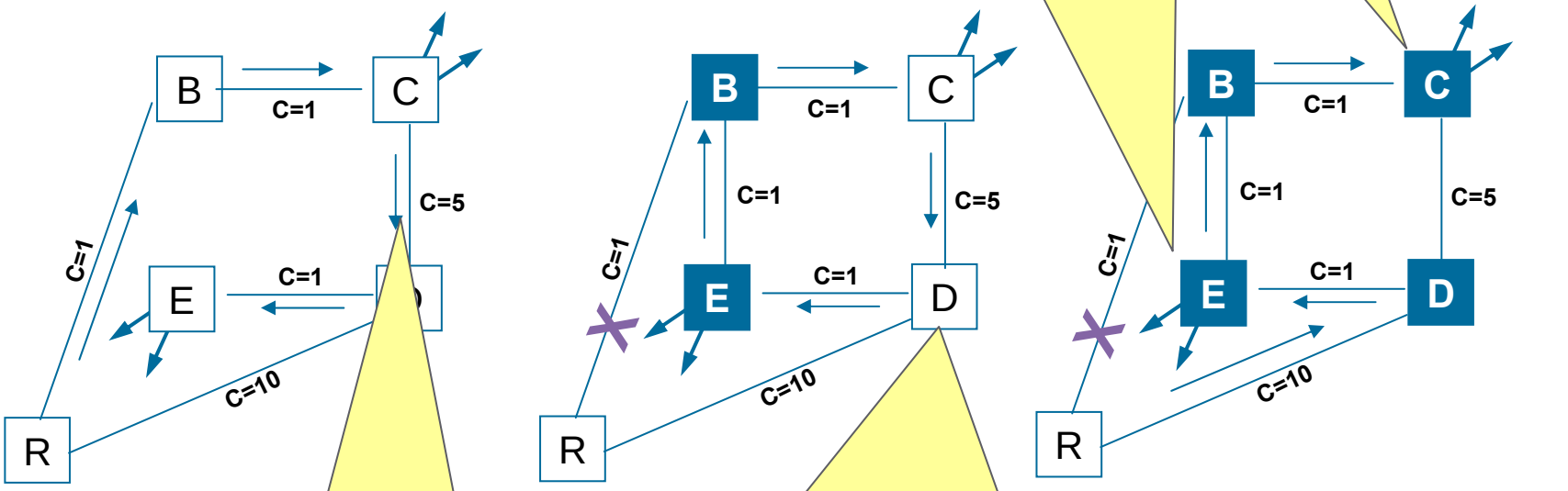
> Forming a three-node loop requires that each node has both a different view of the topology, and an inconsistent view of the link metrics round the loop :



Summing these inequalities :- **x + y + z > 2 . (x + y + z) Hmm . . .**

We conclude that the 4-node loop is a useful model to work with

It is not just four nodes and two views

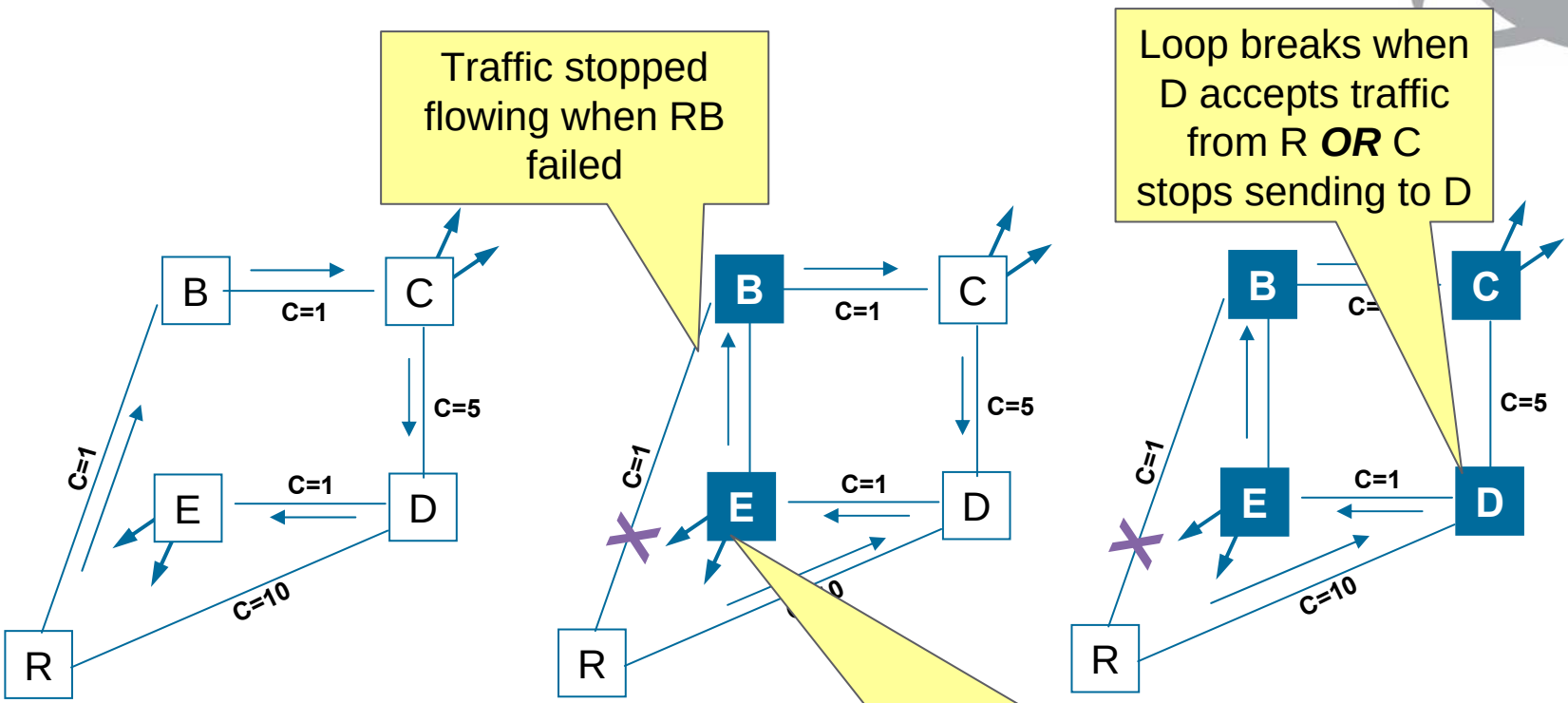


Ingress to each portion of the loop needs to be a branch on the multicast tree

Σ of metrics of the "slow to converge" portion of the network needs to be greater than the rest of the loop, CD > DE+EB+BC

The point of ingress from the root (the shortest path to the root) needs to shift from the last node in the "fast converging" section of the loop to last node in the "slow converging" section (viewed from the POV of direction of looping)

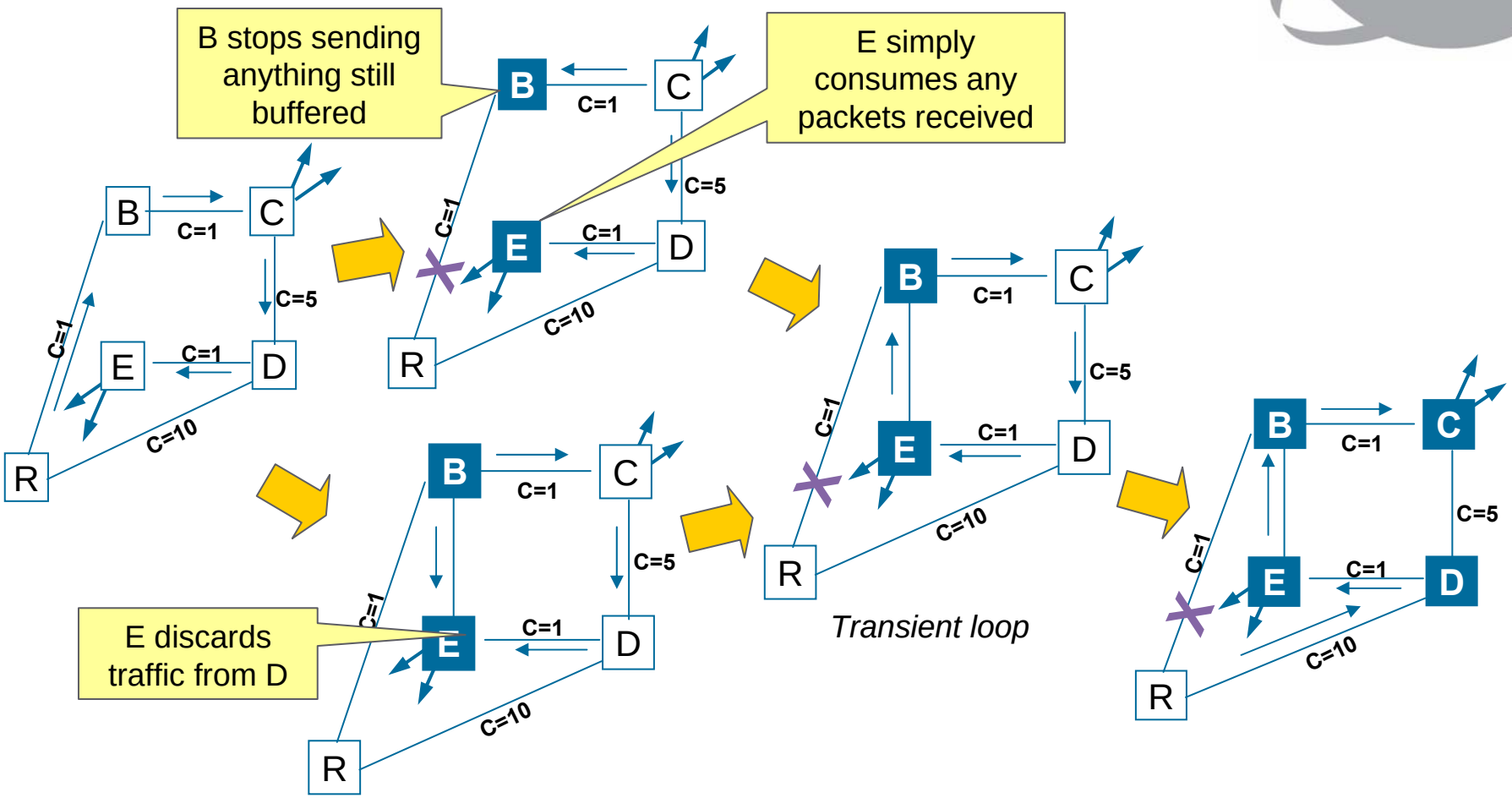
How much traffic was trapped in the loop?



Traffic will only be trapped if loop RTT time > detection + notification + convergence time for B&E for both RB and EB link state changes

Loop did not close until minimum of:
detection time of RB failure
+ propagation time from B to E
+ convergence time
(assuming that the EB link becomes available as the RB failure is detected)

How much traffic was trapped in the loop?



Any convergence on intermediate state topology introduces either Ingress Check blocking or no downstream FIB entry, further diminishing the quantity of traffic that can be trapped

Concern

- We agree transient loop cases can exist
  - But they have to be carefully contrived under Ingress Check and even when they exist, they may be empty
- How do we make the network robust without over engineering for contrived corner cases?
- Most synchronization proposals introduce complexity, delay and additional state, and so may not be an improvement over the status quo ?
  - Neighbor checking
  - Ordered convergence
- It does not seem sensible to penalise the response to **all** topology changes, especially when we can further mitigate the impact of corner cases

General Direction

> It is multiple near simultaneous events that produce loops

> We need to narrow the window of what constitutes "simultaneous"

> We can define that window as the delta between installation of loop closure state and installation of loop blocking state

- Minimizing the variability in convergence times across the network is a key technique in minimizing the duration of the window of vulnerability

Sources of variability of convergence

> Network size
- Manifests itself in order of complexity of computation
- Manifests itself in delays in synchronization of link state databases due to LSP propagation time

> Compute capability of individual nodes
- N^2.logN amplifies the impact of deltas in compute power

> Quality of implementation
- Unstable software
- Poor algorithm implementation
- Slow transfer of FIB from control plane to bridging components
- Intermediate states in FIB installation

> Operational attributes
- Inconsistent application of hold off timers

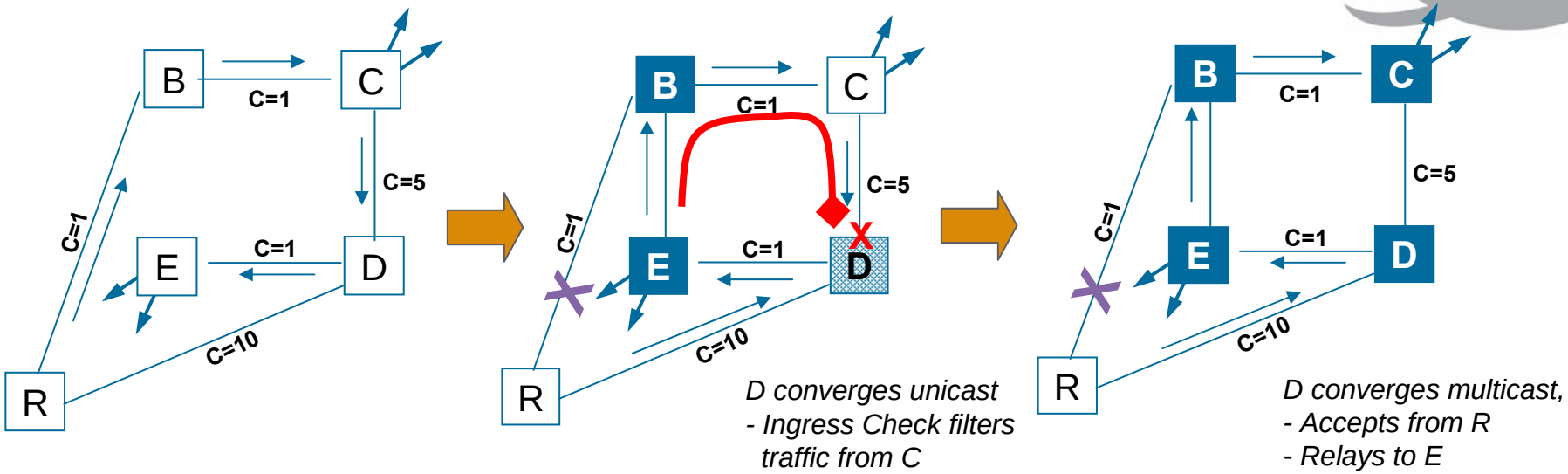
Minimizing variability

> Do "**break before make**" by first converging unicast and installing unicast MACs in the FIB (immediately enabling SA-based Ingress Check) ***before*** performing multicast computation
  - The complexity of computing unicast shortest path is N.logN
  - The complexity of computing multicast paths is N^2.logN

> Result is:
  - Minimal time to installing Ingress Check loop blocking state in FIBs
  - Differences in compute power of nodes is mitigated
  - Impact of network size on compute complexity and synchronization is mitigated
  - Many aspects of quality of implementation are mitigated

Loop blocking can be computed orders of magnitude faster than loop closing

Applied to the example…



Both B & E must learn of ***both*** topology changes (RB & EB), and converge both unicast ***and*** multicast before:

1. D learns of either change and converges unicast
2. C learns of EB change only and converges unicast and multicast
3. C learns of RB change only and converges unicast
   > with only RB failed, C accepts from D for a transient loop to exist

Summary

> It requires multiple simultaneous topology changes combined with variability of convergence to produce a transient loop
  - Simultaneous being defined as “within a narrow window”
  - Bounded by notification time plus unicast convergence time of loop breakers

> A transient loop frequently will be empty
  - Cut off from the source and drained before it actually closed

> “break before make” reduces the window size within which multiple events can cause a transient loop
  - Loop breaking computed much faster than loop closure
  - Unicast FIB installed before multicast FIB for SA based ingress check

> “break before make” reduces the duration of existence of a transient loop
  - No multi-hop delay in ordered convergence or neighbor checks