Tradeoffs in Network Complexity

Tradeoffs in Network Complexity

- Defining Complexity
- Measuring Complexity
- Complexity Tradeoff
- Fast Reroute as an Example
- Whither Complexity?

Defining Complexity

Network Complexity Index

- Breaks a network up into communities
- The interaction of the communities is calculated to provide a network complexity index
- Based on
  - Number of nodes
  - Degree of nodes
  - Number of edges
  - Rate of change in nodes and edges

$$B(N) = \text{Max } j,\ X[j] \geq j$$

…where j ε [1, …, p]. B(N) is a well-known statistic called the H-index and is used commonly in citation analysis [10]. The formulation above of the H-index in terms of a maximization problem is due to Glanzel [11].

A Network Complexity Index for Networks of Networks; Stuart Bailey and Robert L. Grossman

NetComplex

- Computes complexity from state and function
- Pits distributed against centralized algorithms
- Makes note of aggregation and other mechanisms

$$c_{s\leftarrow x} = w_v u_{s\leftarrow x} + w_t \sum_{y \in T_{s\leftarrow x}} \max(c_y, \varepsilon) + c_x$$

Our metric assigns equal importance to value and transport dependencies. However, depending on the system environment, this may not be the best choice…

Our metric treats all input or transport states as equally important. However, sometime certain input or transport states are more important (for correctness, robustness, etc.) than others.

Our metric treats all inputs as independent which might result in over-counting dependencies from correlated inputs.

http://www.eecs.berkeley.edu/~sylvia/papers/netcomp.pdf

Design Intent

Router 1

1. interface GigabitEthernet 1/1 — *Define an interface*
2. ip address 10.1.0.1 255.255.255.252
3. !
4. router eigrp 10
5. distribute-list prefix TO-SAT out GigabitEthernet1/1 — *Install a route filter on an interface*
6. !
7. ip prefix-list TO-SAT seq 5 permit 192.168.1.0/24 — *Define a route filter*

Router 2

8. interface FastEthernet1/1
9. ip address 192.168.1.1 255.255.255.0 — *Define a subnet*
10. !

Definition of an object

Reference to an object

A referential link

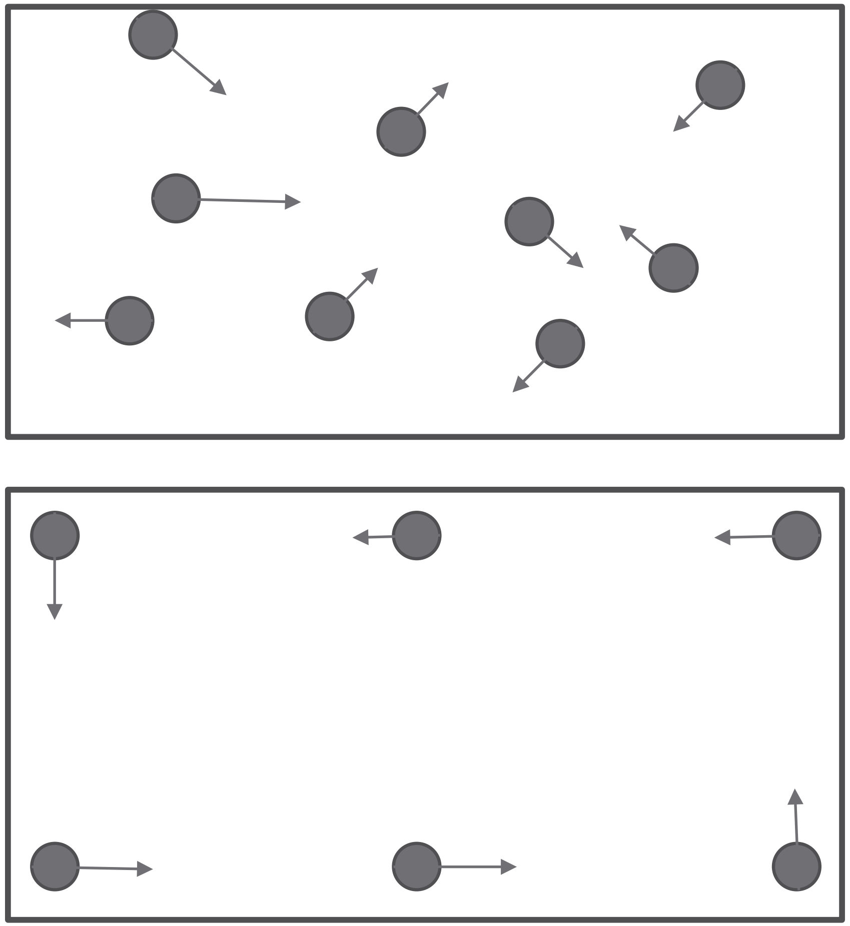
http://www.ietf.org/proceedings/85/slides/slides-85-ncrg-0.pdf

Where We Are

- There are some *(good!)* tools out there
  - This research will only get better over time
- Each one focuses on a single part of the overall problem
  - Control plane state
  - Configuration complexity
- Each one attempts to provide an absolute measure in one specific area
- But systemic complexity *isn't* absolute
  - Complexity in one system interacts with complexity in other systems

Network Complexity is Organized

- Organized complexity is different than disorganized complexity
- *They are all problems which involve dealing simultaneously with a sizable number of factors which are interrelated into an organic whole. They are all, in the language here proposed, problems of organized complexity. –Weaver, 1948*



Where We Are

- Statistics will be of limited use in this realm
  - Statistics will tell you if there is information (Shannon), but not what that information means
- We must interact with *intent*
  - Network design *intends* to solve specific problems
  - How can you measure intent?

Complexity Tradeoffs

Complexity verses the Problem

- Harder problems tend to require more complex solutions
  - Complexity has no meaning outside the context of the problem being solved
  - Nail verses screw verses screw+glue
- How many balloons fit in a bag?

Complexity verses the Toolset

- More complexity can be managed with better tools
  - If your only tool is a hammer…
- But we need to figure in the cost of the tool
  - Nail guns are harder to maintain than hammers
  - Sonic screwdrivers are notorious for breaking at just the wrong moment

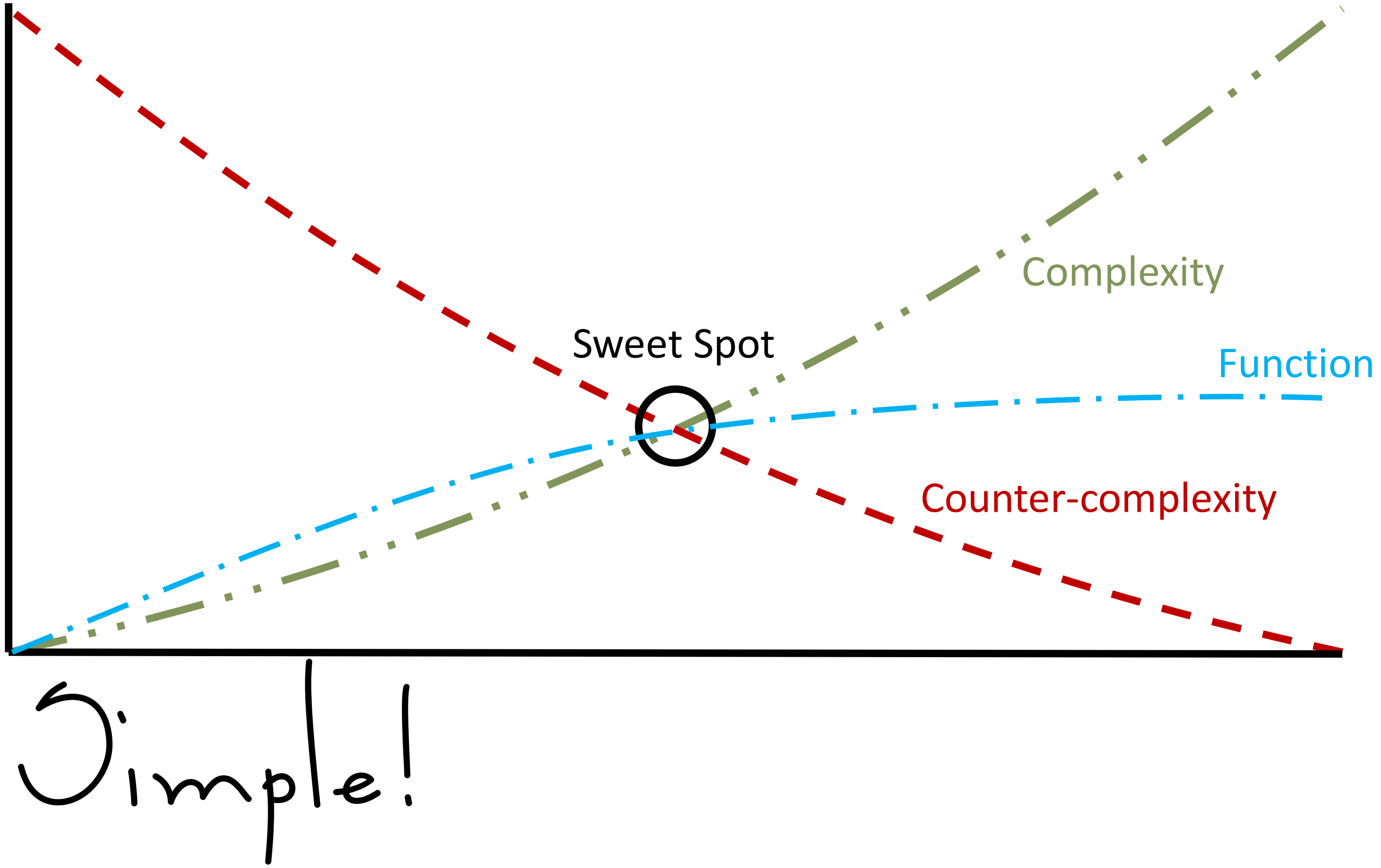
Complexity verses Skill Set

- Things that are complex for one person might not be for another…
  - This isn't a (just) matter of intelligence, it's also a matter of focus and training

Complexity verses Complexity

- Complexity comes in pairs
  - *It is easier to move a problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it.*
  - *It is always possible to add another level of indirection.*
  - RFC1925
- Decreasing complexity in one part of the system will (almost always) increase complexity in another

The Complexity Graph



The Point

- You can never reach some other desirable goal without increasing complexity
  - Decreasing complexity in one place will (nearly) always increase it in another
  - Decreasing complexity in one place will often lead to suboptimal behavior in another
  - Increasing service levels or solving hard problems will almost always increase complexity

You don't have to have a point, to have a point…

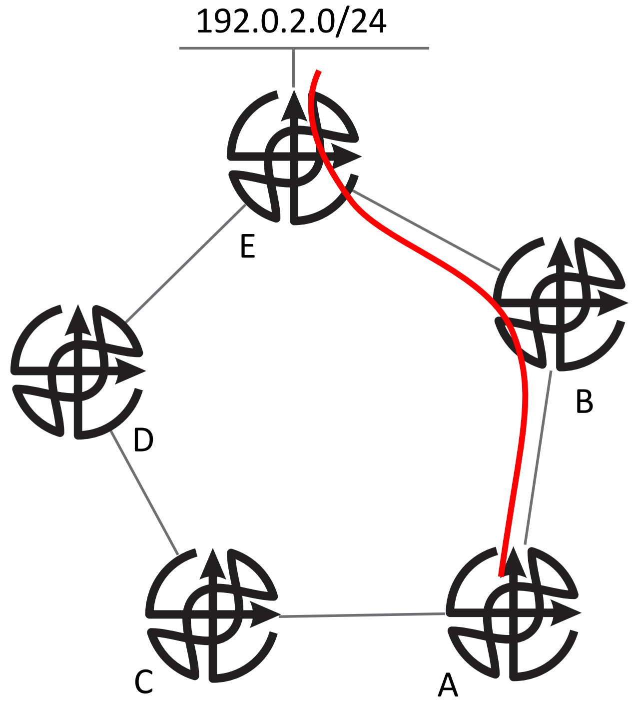
The Goal

- Bad questions
  - How complex is this?
  - Will this scale?
- Good questions
  - Where will adding this new thing increase complexity?
  - If I reduce complexity here, where will I increase it?
  - If I reduce complexity here, where will suboptimal behavior show up?
- Complexity at the system level is about *tradeoffs,* not *absolutes*

Fast Reroute as an Example

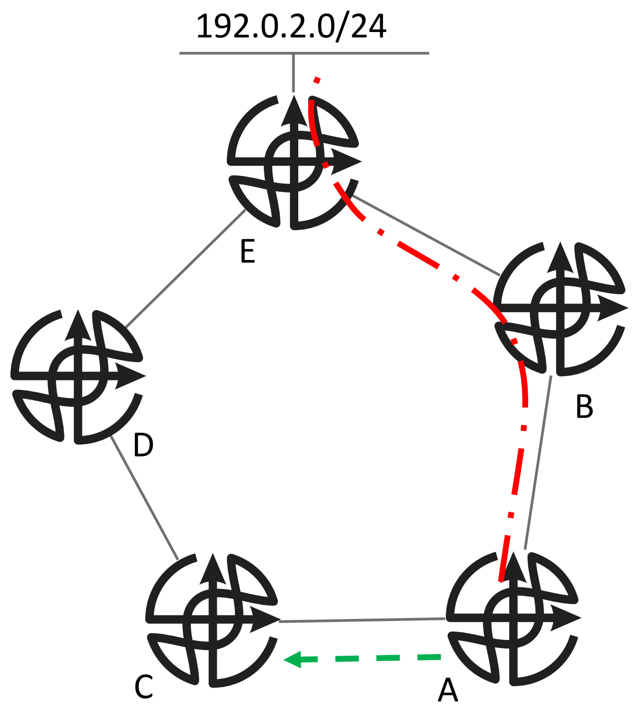
Precompute

- Router A uses the path through B as its primary path to 192.0.2.0/24
- There is a path through C, but this path is blocked by the control plane
  - If A forwards traffic towards 192.0.2.0/24 to C, there is at least some chance that traffic will be reflected back to A, forming a routing loop
- We would like to be able to use C as an alternate path in the case of a link failure along A->B->E



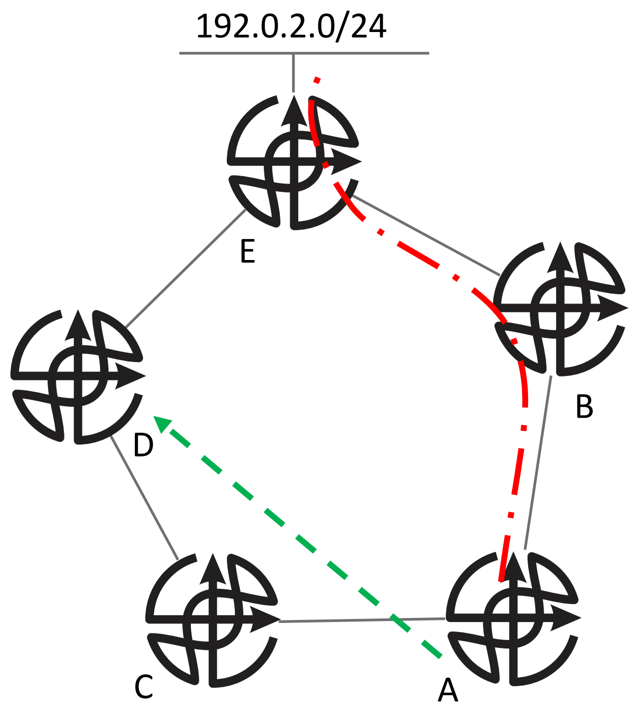
Precompute: LFAs

- Loop Free Alternates (LFAs)
  - A can compute the cost from C to determine if traffic forwarded to 192.0.2.0/24 will, in fact, be looped back to A
  - If not, then A can install the path through C as a backup path
- Gains
  - Faster convergence
- Costs
  - Additional computation at A *(almost nil)*
  - Designing the network with LFAs in mind



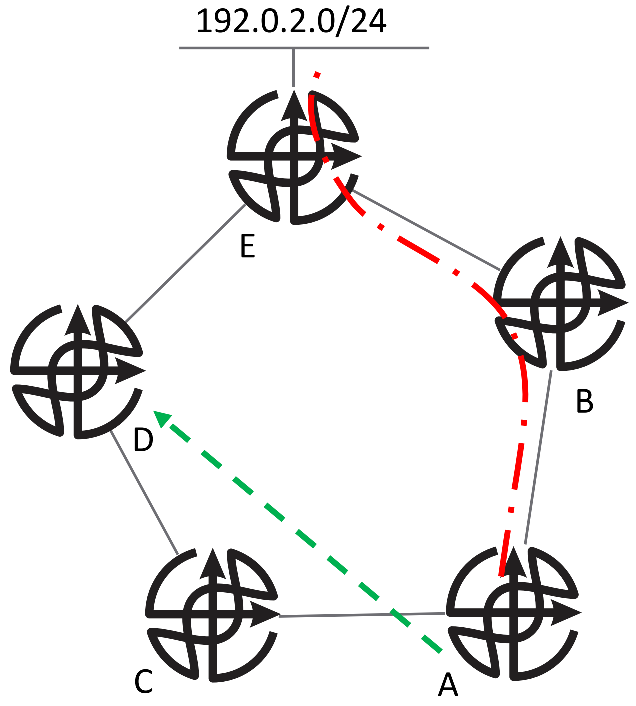
Precompute: Tunneled LFAs

- Tunnel into Q
  - A can compute the first hop beyond C where traffic destined to 192.0.2.0/24 will not loop back
  - A then dynamically builds a tunnel through C to this point and installs the tunnel interface as a backup route
  - There are a number of ways to do this
    - NotVIA, MRT, Remote LFA, etc.
    - Different computation and tunneling mechanisms, but the general theory of operation is the same



Precompute: Tunneled LFAs

- Gains
  - Relaxed network design rules (rings are okay)
  - Eliminates microloops
  - Faster convergence
- Costs
  - Additional computation at A *(almost nil)*
  - Some form of dynamic tunnel
  - Additional control plane state
  - Designing the network with alternate paths in mind
    - These mechanisms don't support every possible topology (but more than LFAs)
    - Thinking about alternate traffic patterns to project link overload, QoS requirements, etc.



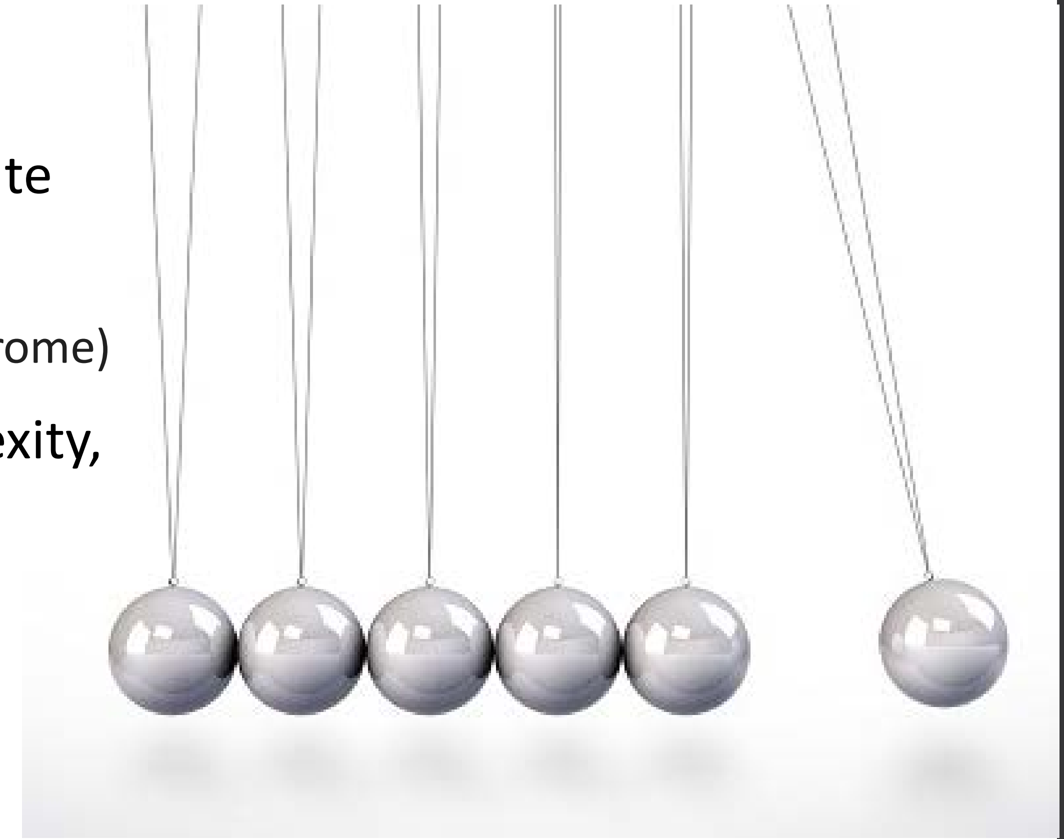
Whither Complexity?

Whither Complexity?

- Will we ever have a single number that tells us how complex a network is?
  - No…
  - But we *will* have a bunch of numbers that help us characterize specific parts
- Will we ever have something we can point to that will mathematically prove, “this is complex,” “that won’t scale,” etc.?
  - No…
  - But we can understand what complexity looks like so we can “see” elegance more clearly

Whither Complexity?

- One useful result would be a more realistic view of network design and operation
- We’re caught on multiple pendulums
  - Centralize! Decentralize!
  - Layer protocols! Reduce protocol count!
- Most of these swings relate to our absolute view of complexity
  - There *must* be a better solution!
  - Let’s go try that over there! (shiny thing syndrome)
- If we could gain a realistic view of complexity, we might be able to see how to at least reduce the frequency and amplitude…



Whither Complexity?

- One useful result would be a more realistic view of network design and operation
- We’re caught on multiple pendulums
  - Centralize! Decentralize!
  - Layer protocols! Reduce protocol count!
- Most of these swings relate to our absolute view of complexity
  - This is so complex – there *must* be a better solution!
  - Let’s go try that over there! (shiny thing syndrome)
- If we could gain a realistic view of complexity, we might be able to see how to at least reduce the frequency and amplitude of these pendulum swings…

One Way Forward

- Measurements within a framework
  - Understand the system as a whole
  - Think about how to measure each point
  - Think about how to compare, or weigh, each pair of points
- Document the tradeoffs we find in real life
  - Helps guide the work of developing measurements
  - Helps build a “body of knowledge” that will drive the state of the art in network design forward

Efforts to Measure & Describe

- Network Complexity Working Group (NCRG)
  - IRTF working group
  - Trying to find ways to describe and measure complexity
  - Gathering papers in the network complexity space on networkcomplexity.org
- draft-irtf-ncrg-network-design-complexity-00.txt
  - Within NCRG
  - Parallel to this presentation

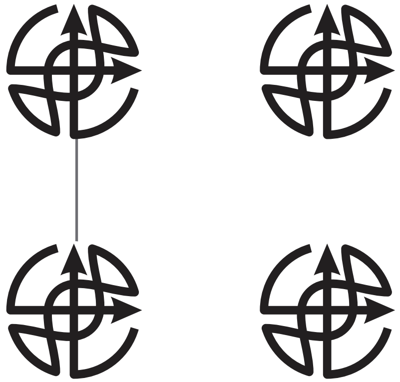
The End!

Fast Reroute as an Example

Additional Slides

Fast Detection

- Interaction with the Control Plane
  - If we can detect failures faster than the control plane can react, we can build a supported feedback loop that overwhelms the control plane, resulting in a general failure
- Solutions?
  - We can exponentially back off notifications
  - We can notify on down immediately, and up much more slowly
  - Both of these increase policy, increase MTTR, etc.



Fast Detection

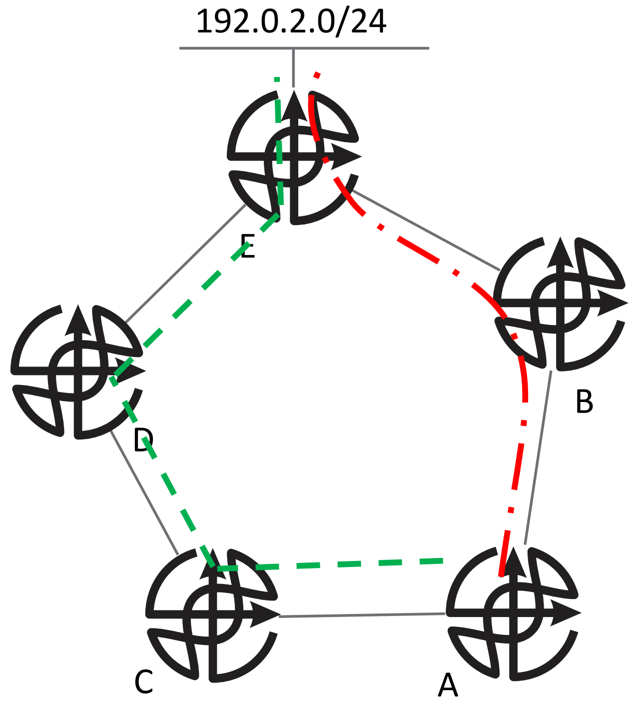
- A number of different systems have been devised over the years to detect link and device failure
  - BFD, fast hellos, etc.
- Using these techniques, we can get failure detection into the 10’s of ms
- What are the complexity tradeoffs?

Fast Detection

- False Positives
  - Dropped packets can cause a an apparent failure where no failure exists
  - We can exponentially backoff failure reports…
    - But we must manage these backoffs on a per link or situation basis
    - Policy dispersion, anyone?
  - We can hold a link down for specific periods of time after a failure
    - When there is no failure, this just exacerbates the effect of the false positive
    - When there is a failure, this makes the MTTR longer
- Any solution adds policy (and complexity) to the control plane

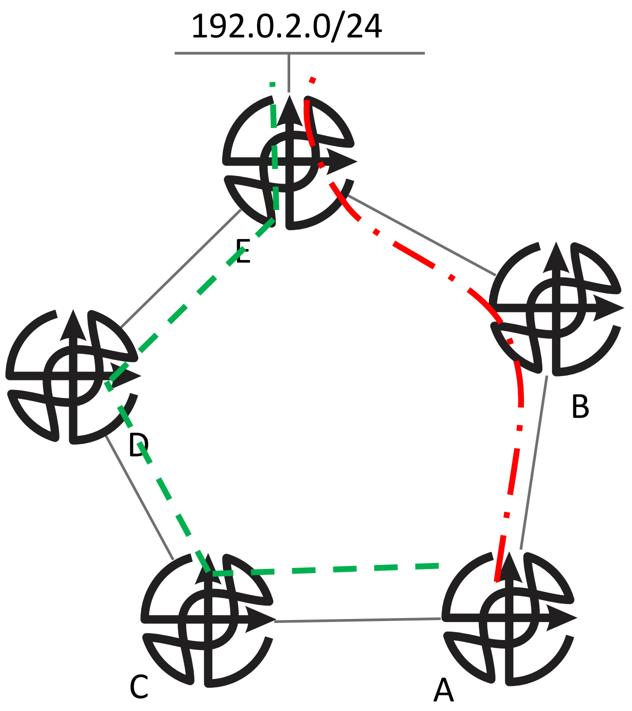
Precompute: Edge-to-Edge Tunnels

- Edge-to-Edge Tunnels
  - A can compute the best path to 192.0.2.0/24
    - This first path would normally be a tunnel, such as MPLS
  - A can then precompute a second path, tunneled edge-to-edge, to 192.0.2.0/24
  - If the primary path fails, A places traffic on the backup tunnel
- The normal way to do this is MPLS
  - On networks that are already using MPLS to transport edge-to-edge traffic



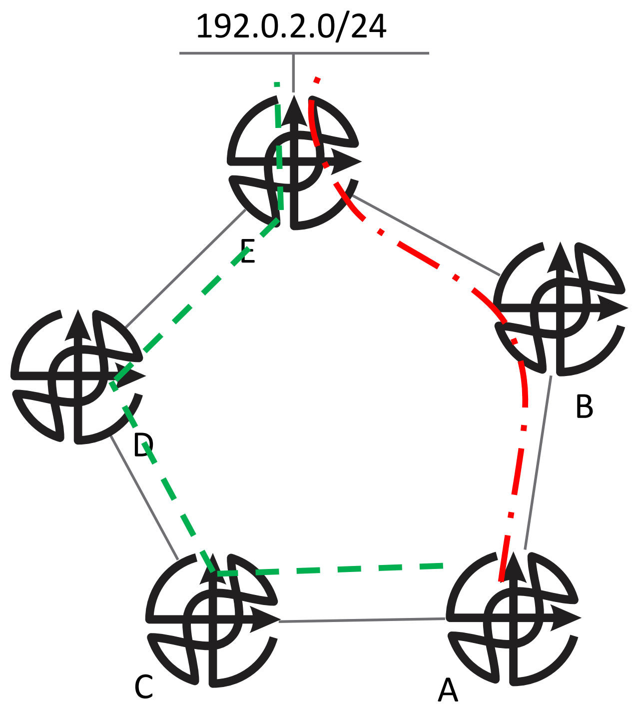
Precompute: Edge-to-Edge Tunnels

- Complexity Costs
  - Additional processing at edge nodes
    - To compute alternate paths – *almost nil*
  - Some form of dynamic tunnels
    - For networks already running edge-to-edge paths, *nil*
  - Additional state
    - Requires the flooding of alternate end points to protect against tunnel head and tail failure
    - Requires additional forwarding state in edge (and sometimes core) devices
  - Open end points on every device in the network
    - For networks already running edge-to-edge paths, *nil*



Precompute: Edge-to-Edge Tunnels

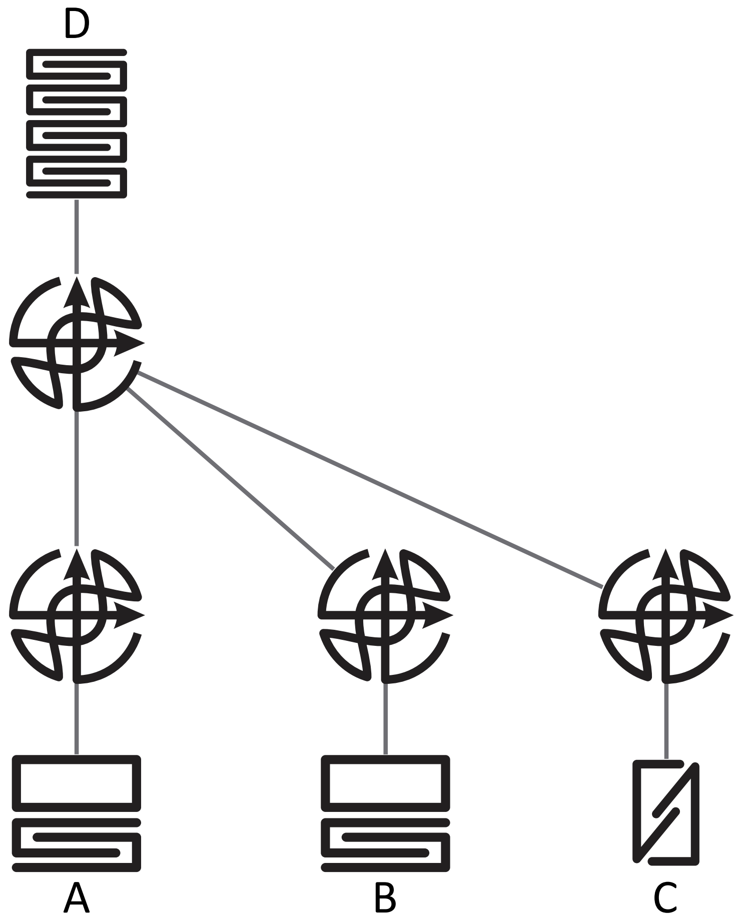
- Complexity Costs (continued)
  - Designing the topology for alternate paths
    - Any two connected topology will do
      - There are no topology restrictions – actually reduces design complexity
    - Link overload and quality of service issues
      - For networks already using edge-to-edge tunnels for traffic engineering, this is probably *close to nil*
  - Additional management complexity
    - Overlay control plane deployed throughout network
    - Troubleshooting complexity, etc.



Policy Dispersion Example

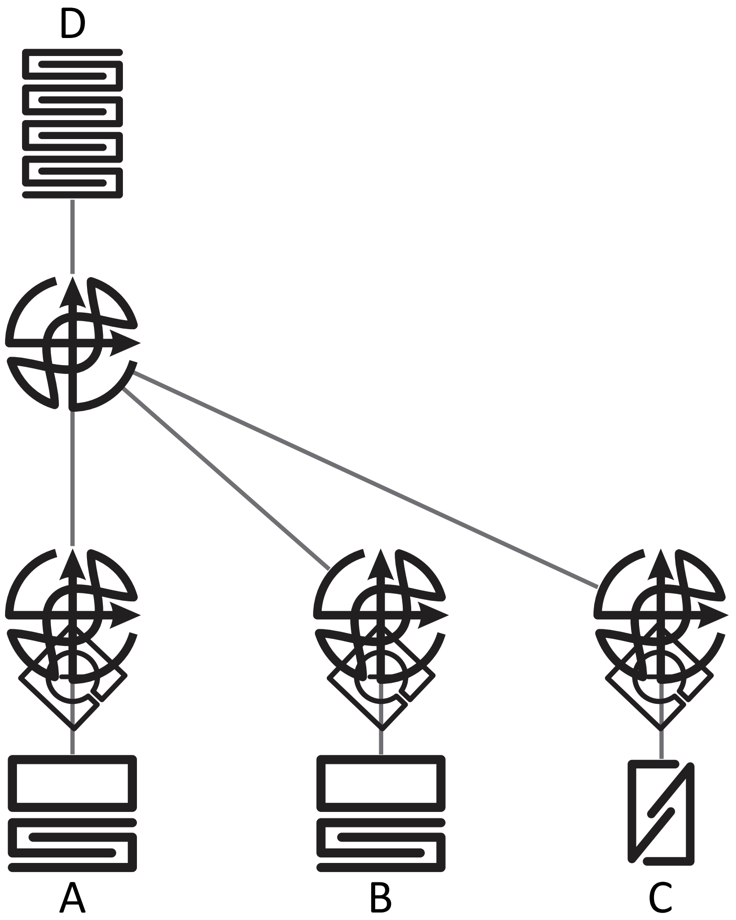
Optimal Forwarding

- Traffic originating at A, B, and C must pass through deep packet inspection before reaching D
- Where should we put this policy?



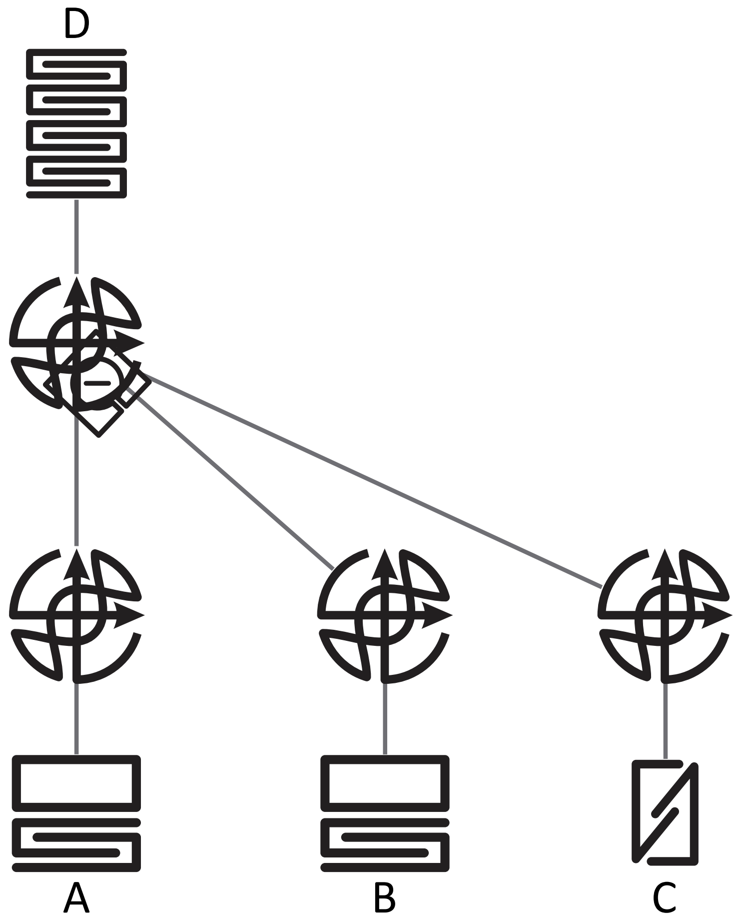
Optimal Forwarding

- At the first hop router?
- We have to manage per edge node
- I can automate these configurations, but…
  - Now I have to manage a new set of tools and processes
- No matter how I slice this, dispersing policy closer to the edge adds complexity



Optimal Forwarding

- At the second hop router?
- Reduces the number of devices to manage
- But…
  - Potentially wastes bandwidth between the first and second hop router
  - Leaves the first hop routers without the packet inspection protection offered at the edge

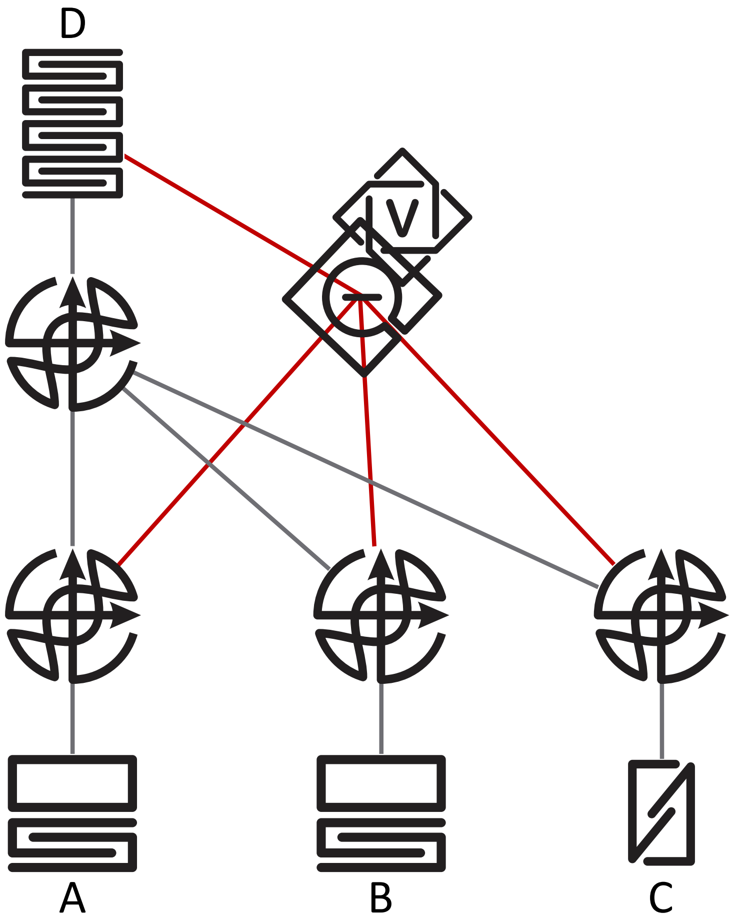


Service Chaining

- *I know! I'll just virtualize my services*
  - *Then I can tunnel the traffic service to service starting from where it enters the network!*
- Good try…
  - But you can't fool the demons of complexity that easily…

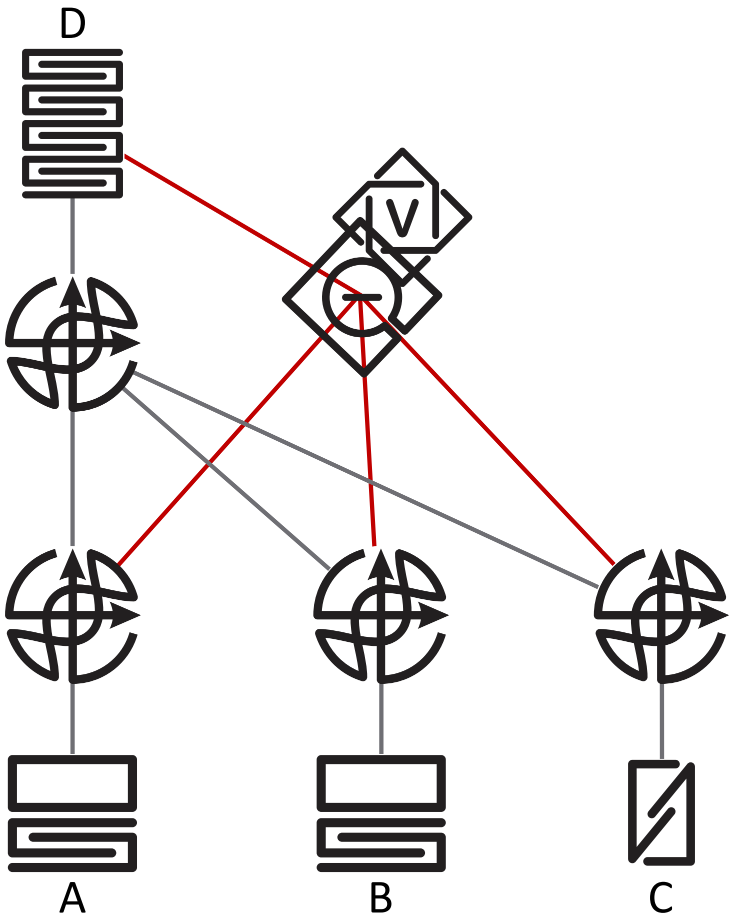
Service Chaining

- Create a new virtual service containing the packet inspection process someplace close to D
- At the network entrance…
  - Look up the destination D
  - Determine the class of service, based on the source A
  - Tunnel the traffic to the virtual packet inspection service



Service Chaining

- We've kept the service logically close to the network edge, while physically centralizing it
- You can bring the policy to your packets, or you can bring the packets to your policy
  - To paraphrase Yaakov's rule…

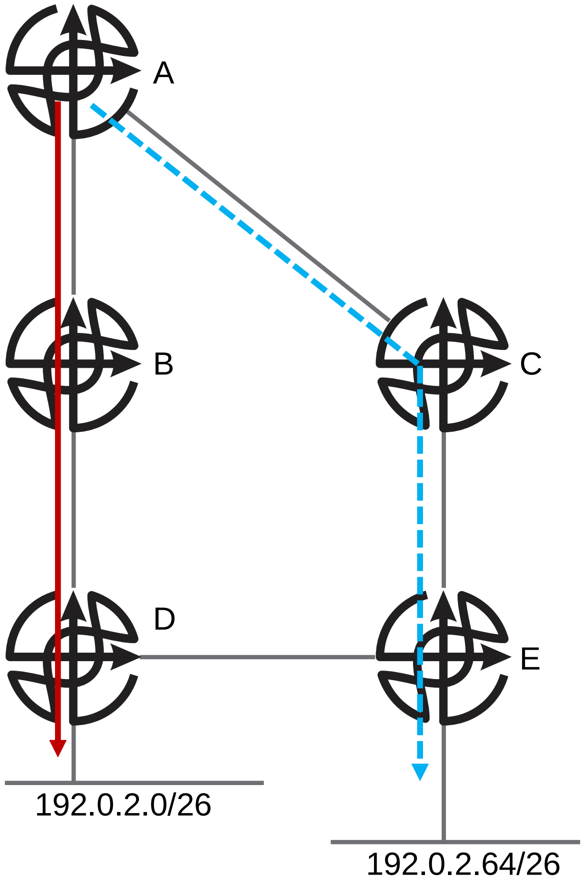


Service Chaining

- We've still added complexity
  - The policy about which packets to put in which tunnels to chain to which services must be programmed in at the edge devices
- And we've still reduced optimality
  - Traffic must be tunneled through the network
    - Potentially wasting bandwidth for packets that will be dropped at some future policy point
    - Tunnels must be configured and maintained
  - Managing quality of service becomes more complex
  - The length of the real path of the packets has increased

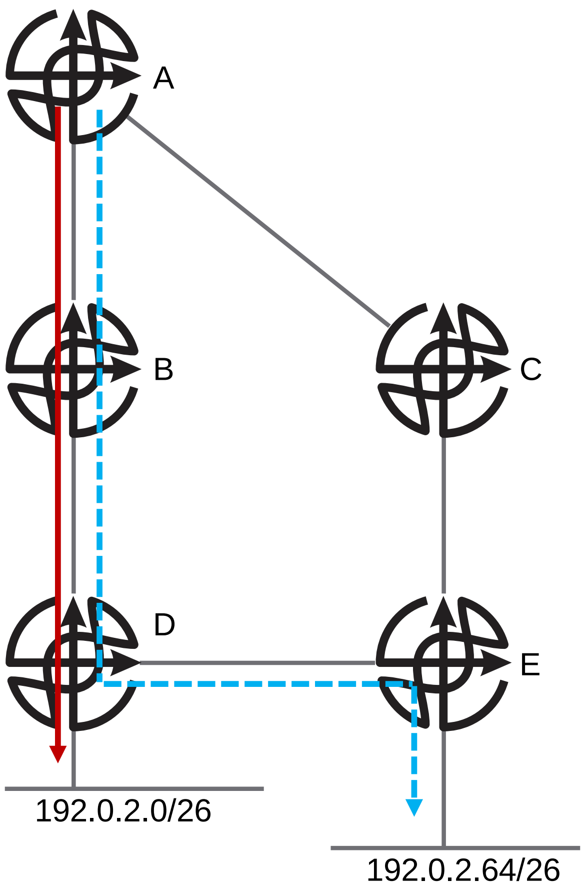
Aggregation/Stretch

- If B and C do not aggregate
  - A will have the optimal route to reach both 192.0.2.0/26 and 192.0.2.64/26
- But…
  - A will have more routes in its local routing table
  - A will receive topology state changes for all the links and nodes behind B and C
  - So more routes, more state change visibility, more complexity



Aggregation/Stretch

- Assume A aggregates to 192.0.2.0/24
  - A will choose either A or B for everything within this subnet (ignoring ECMP)
  - Hence A will choose a suboptimal route to either 192.0.2.0/26 or 192.0.2.64/26
- Reduces complexity
  - A has fewer routes in its local table
  - A deals with less state change over time



Aggregation/Stretch

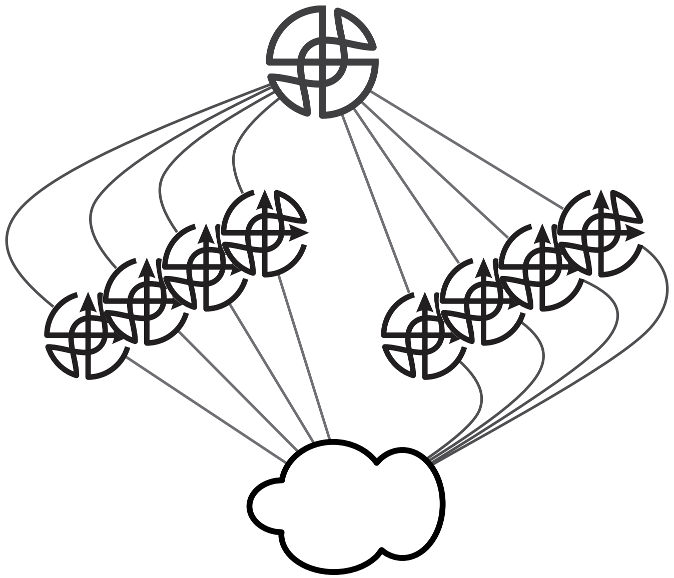
- Aggregation almost always confronts us with the state verses stretch tradeoff
- More state == more optimal paths
- Less state == less optimal paths
  - Or more stretch – the difference between the optimal path through the network and the path the traffic actually takes

Control Plane Centralization

- Let’s centralize the entire control plane!
- Won’t this be simpler?
  - Policy will be in one place
  - Easier design
    - No thinking through aggregation, etc.
    - Just install the routes where they need to be in real time
  - Can dynamically interact with the control plane in real time
    - Applications can tell the control plane when new paths/etc. are needed
- Sounds neat
  - But… has anyone read RFC1925 recently?
  - *It is always possible to agglutinate multiple separate problems into a single complex interdependent solution. In most cases this is a bad idea.*

Control Plane Centralization

- Complexity Points
  - North/South interface
    - This isn’t as simple as it sounds
    - Particularly as there is a “kitchen sink” tendency in these things
  - Resilience
    - The controller is a single point of failure
    - This has to be mitigated somehow…
  - Fast convergence
    - We can always precompute and install alternate paths
    - But double failures and rapidly changing local conditions can stress the system, possibly causing a control plane failure

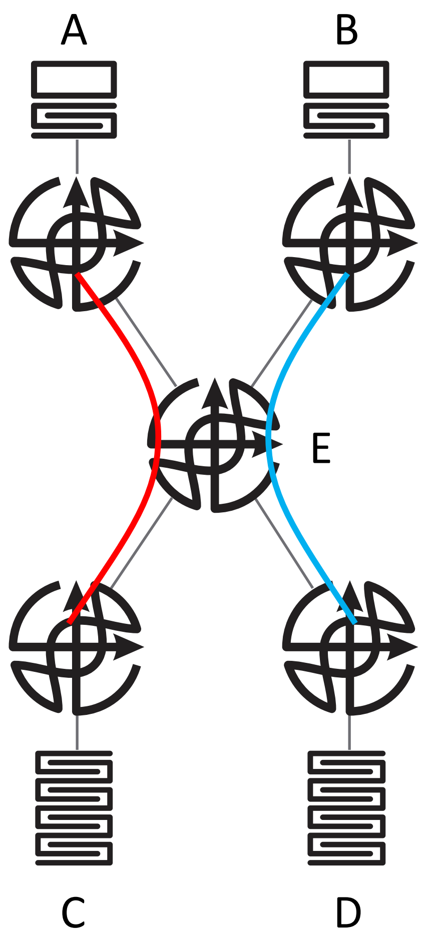


- Maybe we need a new rule of thumb…
  - Distribute where you can, centralize where you must…

Control Plane State Example

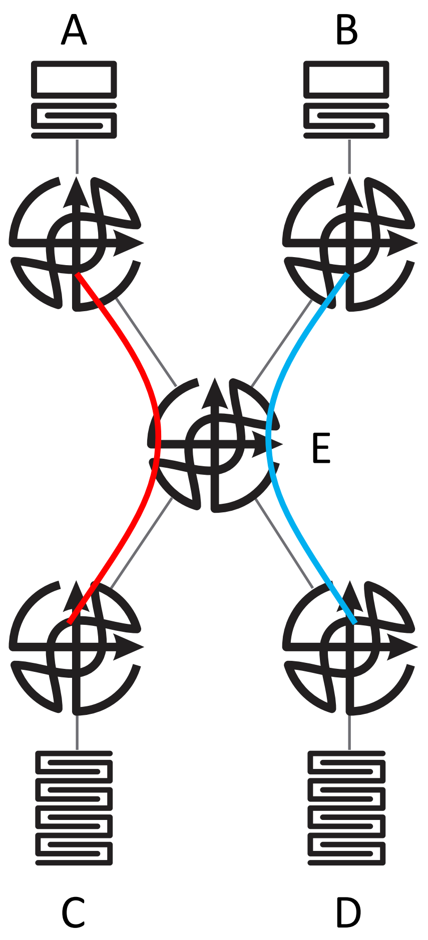
Control Plane Virtualization

- Virtualizing the control plane can control the amount of state and the rate of change
- The setup
  - A only needs to talk to C
  - B only needs to talk to D
  - So let's create two tunnels that allow this traffic flow
- E only needs to know how to reach the tunnel end points, not the individual destinations



Control Plane Virtualization

- But how do we know where to set the tunnels up?
- We've essentially moved control plane forwarding state from the control plane…
  - …into policy
  - …into an overlay protocol
- Does increasing policy or adding another protocol really reduce overall complexity?
  - Or are we just moving complexity around in the network?



Control Plane Virtualization

- We could autoconfigure the tunnels…
  - Blast the traffic for destinations we don't know how to reach everywhere
  - Someone, someplace, answers us with the right mapping
  - Build a tunnel based on what we've just discovered
  - Hold the tunnel until no more traffic is passing through the path…
- Many schemes have used this type of mechanism
  - LISP
  - TRILL (following bridging in general!)
  - ATM/LANE

Control Plane Virtualization

- But is it really simpler?
  - Throws complexity onto the edge device
    - A sends a packet to C and never receives a reply…
    - Has the path has failed, or the destination?
  - Introduces entropy based on traffic flow
    - What should the forwarding table look like at any given moment?
    - What size will the forwarding table be under any given network conditions?
- Not really…
  - The complexity has moved to the transport protocol and caching algorithms