Please refer to the NSI-Automatedd GOLE topology diagram as presented at SC2011 for this discussion.

In the AutoGOLE network, we have four STPs in Uva: ps80, ps81, ps82, ps83. Four similar endpoints exist in KrLight. While not indicated in the NSI topology, and not known to NSI protocol, these endpoints map to one of four ethernet vlans. There are 4x4=16 possible end-to-end connections that could be requested from Uva to KrLight.

A “user” ReservationRequest() is issued for a connection from UvA:psxx to KRlight:psxx. We assume this request was issued from some unknown user agent to an Aggregator NSA not on the path under analysis: UvA > NL > SL > KrL. It is also assumed that the NSAs associated with each network only have detailed visibility to their own resources and the information present in the global topology file.

The analysis is presented in terms of the number of ReservationRequests() that must be issued to reserve a confirmed path end to end. The pathfinding completes when a) it successfully finds and confirms a path meeting all user constraints, or b) it exhausts all possible known paths without being able to confirm one that meets the constraints.

**Scenario #1: None of the networks can swap VLANs.**

In scenario#1 there is no way within the physical layer to cross vlan planes. Thus, we know that of the 16 possible end to end connection permutations, only 4 end-to-end connection pairs (those that map to like vlans) are physically viable.

Starting in UvA, there are four paths from the source STP to the next hop network NL. For non-swapping networks, only one of the four paths is physically viable. E2E, this results in 4(uva) x 4(nl) x 4(sl) x 1(krl) = 64 possible paths expressed in the topology between any two specific endpoints. Given a request from uva:ps80 to krl:ps80 there is only one actual viable path that exists due to poor intra-domain engineering issues in each network (i.e. use of non-swapping Ethernet switches.) However, the topology nevertheless indicates that there 64 possible paths.

**Using the chain method:**

The **best** case is 4 Reservation Requests. The external PF/NSA issues a **[chain]** ReservationRequest directly to the UvA NSA for UvA:ps80 > Krlight:ps80. UvA recognizes NL as the next hop and finds four SDPs from UvA to NL. UvA performs an availability check on these local resources and sees that ams80 is the only viable exit SDP and it is available, so it reserves the local UvA segment ps80>ams80, and then issues a ReservationRequest to NL for the downstream segment from NL:Uva80 to krlight:ps80. And so on thru NL and SL. At krlight the Reservation describes a local segment krlight:chi80 > krlight:ps80. So krLight reserves it locally and confirms back to SL. SL now has both its local segment confirmed and the downstream segment confirmed, so it confirms back to NL. And so on back to the external NSA/PF. A total of 4 Reservations were issued.

The **worst** case chain model only occurs when a downstream segment fails. Had any of the downstream segments been rejected, the RA NSA receiving the rejection would attempt to find an alternate path requiring additional ReservationRequests. In scenario #1, there are no viable alternates, so all NSAs will simply reject the request back upstream, and the chain unravels upstream culminating at the external PF with a reject. So the worst case chain scenario #1 is still 4 Reservation requests.

**Using Tree method:** A remote PF using a tree segmentation for this example must select transit points at each network boundary in order to issue the Reservation Request to that transit network. The PF does not have direct access to intra-domain information regarding availability of those transit STPs, nor does it have the means or authority to actually reserve intra-domain resources. And worse, the PF believes all paths it found in the topology description to be truthful – i.e. viable. So, the external PF will be obligated to try all of those announced alternate paths.

Since the external PF has no state for the intra-domain SDP resources in each network, its only recourse is an exhaustive search of all SDP at a transit boundary to find one that works. In the **best** tree case, the external PF *randomly* selects the single viable path across each network on the first try, generating 4 ReservationRequests. The odds of a first guess being the correct path are inversely proportional to “r” – the size of the transit SDP/label space. i.e. the larger the SDP space, the less likely a guess will strike the correct path. So, the probability of a hit first try is 1/r(uva) x 1/r(nl) x 1/r(sl) x 1/r(krl) = 1/r^(n)=1:256 for the AutoGOLE fabric. However, as the SDP/label space (“r”) increases, the odds of being lucky and finding a path on a first try becomes increasingly unlikely. For r=100, the odds of getting best case are 1/100^4= 1/100,000,000==10^-8. The odds of finding the correct path in the first 100 Reservations? 10^-6 (one in a million). This is not going to work.

In the **worst** case tree, the external PF performs 4 Reservations across the UvA network only to find the last one works. It uses this good SDP to try 4 Reservations in the NL network only to find one that works, and so on. And it only needs to try one segment in KrLight. In this case, there are 4+4+4+1= **13 reservations issued**. (note that the tree method is performing a breadth first search and by performing the reservations in sequence it can prune rejected downstream branches.)

**Analysis:**

For chain model, both the best case and the worst case is 4 Reservations. (!)

For tree model, the best case is 4 Reservations, while the worst case is 13 Reservations.

Worst case performance is O(n) for Chain, and O(r\*(n-1)+1) for tree, where r=number of SDPs (label space), and n is the number of networks in the path. So if we increase r to 100, Chain still exhibits 4 Reservations worst case, where Tree climbs to 301 worst case. If we further increase r to 4000 (e.g. Vlan ids), our example#1 generates 4 reservations for chain, while tree needs 4000 x 3 =12,000 Reservations worst case. While tree best case is still 4, the odds of that best case occurring are 1:12000.

In this scenario, Tree is hampered by the fact that it must select a specific destination SDP for its transit segments but it doesn’t know which are used/unused. The only way to determine this is to try Reserving them – one by one – until a segment succeeds. This is slow. A more insidious problem is that the PF does not know that the overwhelming majority of the advertised paths do not even exist. It believes all paths advertised in the topology are real viable possibilities. So the PF will try to perform an exhaustive search over all the announced paths to just find the one viable path to the next hop network – issuing Reservations one at a time and most of these reservations requests are utterly futile. Wasted effort.

So a major problem in scene 1 is that networks are making announcements of Any-to-Any connection capability that they are simply unable to honor, and these announcements are enticing PFs to perform long exhaustive searches over large non-viable search spaces. The simplest solution to this problem is to remap the SDPs into “service planes” that properly express the real connection viability. The topology file will increase by a factor of “r” – the size of the SDP/label space. But the result will be worst case Tree in scenario 1 now becomes O(n) – the same as chain! This solves the futile path search problem. But there may still be multiple viable paths to the transit network – and the availability of these segments must still be confirmed. (Scenario #2 and #3).

However, the external PF cannot remotely determine availability of a transit segment. The “availability” of a segment is dependent upon *all* the intra-domain resources necessary to build the transit segment. Thus, the external PF will need visibility to *all* the intra-domain pathfinding topology and state to implement a transit selection algorithm that is not, ultimately, a guess. Assuming a network is willing to part with that much detailed information, and the PF finds an available path, the resources must yet be allocated to this segment before the segment can be confirmed. Only the local NSA is authoritative for intra-domain resource allocation – i.e. the local NSA is the only agent that can check availability and reserve the resource in one atomic operation.

Since the ReservationRequest requires specific unitary STPs as the endpoint constraints, the Tree method must search a large space by trying each egress SDP one by one. The chain method, however, implicitly allows the local NSA to select the next hop SDP. The local NSA has direct access to the local intra-domain information and can quickly determine which SDPs are viable and available and then select one for reservation processing. And all this is done without any NSI CS Reservation calls. What would improve the tree method in this regard would be a means for a Reservation request to specify the endpoint as a bounded set of some sort – e.g. “any STP adjacent to Network X” or “any STP in this list: <…> “ Then, Tree can communicate the real constraint on the endpoint (adjacency to the next hop Network) rather than iteratively trying each individually.

For scenario #1 it is best to simply use Chain mode Reservation and let the local NSAs select the intra-domain paths. A useful new feature would be to modify the Reservation endpoint specification to allow sets of STPs based on lists, topology, or name wildcards.

**Scenario #2: All nodes are non-swapping except for netherlight.**

In this scenario, the external PF still sees only Any-to-Any announcements and so it is still enticed to search the entire 64 path space for a connection. But now, NetherLight has re-engineered their network to provide real any-to-any connectivity. The net result is that we now have at least one path between all 16 end-to-end pairs. This is in itself a giant improvement over just four pairs having connectivity.

With chain, the process is the same until it encounters NL. The Netherlight switch now allows the local NSA to explore alternate paths when downstream reservations fail. Each down stream path tree has only 2 reservations, so 8 downstream reservations in total. Thus 1(Uva)+ 1(nl)+(4\*(1(sl)+ 1(KRL))=10. So this is **10 Reservations worst case using chain**. In the best case, NL chooses wisely on the first shot and we still have **4 Reservations best case chain**.

With Tree, the external PF detects no difference from scenario 1. It will still try all paths as in the topology. But now, as it processes NL, there are now 4 times as many successful cross connects, so the downstream options grow. There are 4 Reservations issued across UvA, 1 succeeds. Four Reservations are then issued across NL– all 4 succeed. For each of those four, the ePF issues four reservations to SL of which only 1 will succeed. For each of those that succeed, ePF will issue one Reservation to KRl. The result is 4(uva)+4(nl)\*(4(sl)+1(KRL))=**24 Reservations worst case using tree.** The NL engineering does not improve our performance over scenario1 – so **best case is still 4 reservations for tree** in scenario 2, but it significantly improves the reachability – all 16 pairs of endpoints are viable now.

Again, scaling up the SDP/label space ruins tree: for r=100, we get 100(uva)+100(nl)\*(100(sl)+1(KRL))=10100+100=10200 Reservations worst case. And as there is still only 1 viable path between endpoint pairs, our odds of finding that path by guessing is inversely propotional to the path complexity. It is truly evident that the Tree method really depends on accurate topology to reduce the effects of this large transit STP/labels sets, or we should be using the chain exclusivelyfor these reservations which performs much better as “r” increases.

**Scenario #3: All nodes provide modern scalable WAN switching technology**

Carrying forward from scenario #2, if UvA inserts a swapping switch, then each UvA endpoint will then have four possible *real* paths to each NL SDP. And if SL then adds a switch capable of actually connecting all SL STPs, there will be 4x more paths = 16 real viable paths from each uva STP to each STP in KrLight. And finally, if KrLight upgrades as well, then all 64 announced paths become real viable paths.

This does several things- First, it inherently removes the futile path issue – as all paths expressed in the topology now accurately reflect the capabilities of the respective hardware.

Second, by reconciling the underlying switching layer to reflect the service announcement, we no longer need to redefine the service layer into a stack of “service planes” just to reflect some limited technology specific capabilities of our switch. Thus our topology file remains compact.

Third, and perhaps most important, every swapping (any-to-any) stage increases path options. And now that all stages have any-to-any switches the STP viability and availability is essentially assured across each network. Only internal resources are yet to be considered.

The bottom line though is the tradeoff between exposing and moving enough current topology detail to remote PathFinders to enable them to make intelligent detailed planning of transit domains, versus just asking the local NSA to build the segment directly. The former will incur substantial overhead and will still require Reservations to guaranty the segments. The latter just needs a better means of specifying endpoint constraints – if a better end point constraint semantic is defined, a single Reservation request would be sufficient to find and reserve a segment across each transit domain. This single modification would make Tree processing much more competitive to Chain mode.