**Multi-Layer processing in NSI**

**What is “multi-layer”?**

“Multi-layer” path finding refers to routing a connection through regions of different transport technologies where new framing or control headers are added to the flow along the path which must be removed from the flow in the reverse order to deliver the original payload at the destination. A “layer” defines the region where a particular framing/header is pushed upon ingress and popped upon egress. We can discuss this issue by defining a layer as set of contiguous service regions that all express like “client” interfaces. In the layering model, each layer acts as a “service layer” to the “client layer” above, and may also act as a client to other service layers below.

A “layer” provides service between its client interfaces. Since our end to end service is a “connection” which by definition delivers a payload from ingress STP to egress STP unmodified, then the ingress and egress points across the layer must all be similar (use the same framing/header). Within a layer, there may be many interconnected service regions that all use the same framing/header info and thus create a single large service layer domain.

In the conventional layering model, the layers represent specific functional services and are fixed in the layer ordering. However, in emerging inter-domain network environments, the order and function of layers are no longer fixed. Indeed, some layers may be stacked upon themselves creating a single service layer domain that may be transited multiple times by a single connection, each time adding a new header – or popping - a header. Thus, end-to-end path finding must be able to account for these push/pop adaptations.

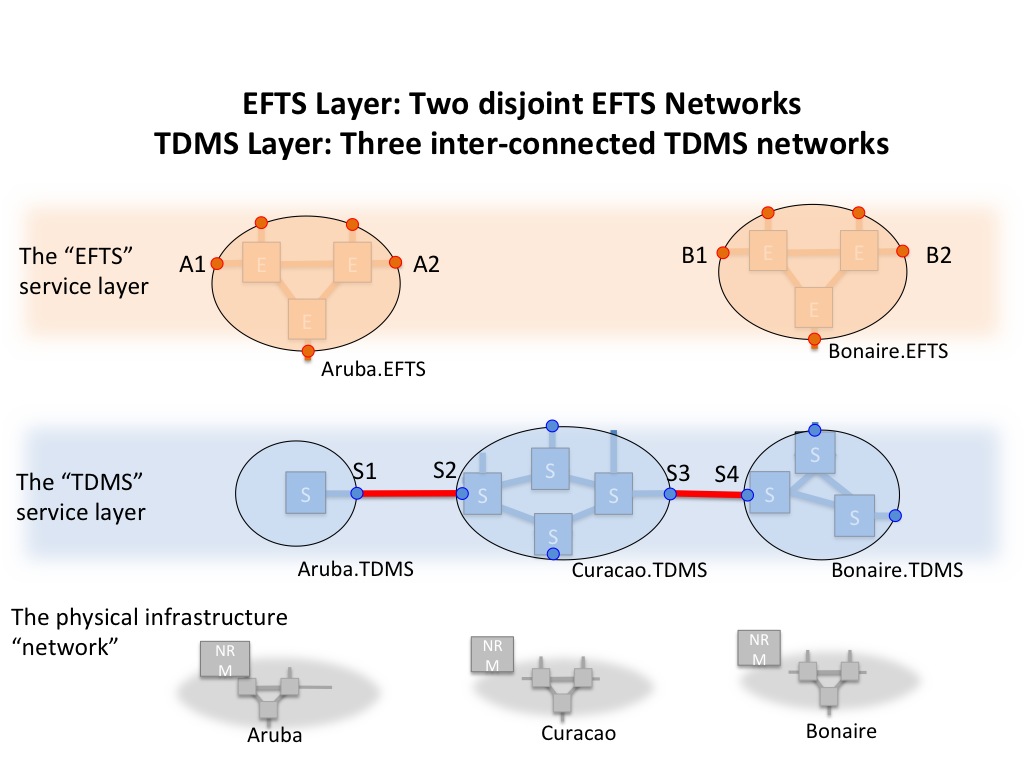
In NSI, the service boundaries do not distinguish when inter-layer framing is pushed or popped from simple intra-layer transit. How does the path finder know if/when a header is pushed or popped in order to insure the original PDU is delivered end to end?

The simplest way to approach this is not to try to figure out what each network does, but rather to assert what each service should be doing. For instance: If we assert that all services must present the original data at the egress STP as was presented at the ingress STP, then we do not care what particular framing changes occur internal to that network service since we are assured that the original PDU will always emerge. This rule is meant to insure that no matter how a core is engineered, an NSI Network Service boundary always guaranties that the same PDU is spit out that was ingested. This will cover most if not all real connection requirements and simplifies path finding immensely.

The implication of this rule (in=out) means that all interconnected network services would have the same client interfaces, i.e. all contiguous services are in the same layer. It also means that seemingly trivial reframing cannot be done internal to a service unless it is undone before egressing that service – for example untagged Ethernet frames presented at an ingress STP cannot be spit out at an egress STP as tagged frames.

(Note: We must be very careful to recognize where even seemingly innocuous adaptations as such pushing an 802.1Q header onto an 802.3 flow changes the definition of the “payload” that is delivered across a service. For instance, if our service was to deliver “Ethernet frames” end to end, then it would accept any conformant 802.3 frame as ingress payload – untagged, tagged, dual tagged, etc. and would be bound to deliver that same frame, tagged or not, at the egress STP.)

The NSI model does not preclude adaptations that take advantage of multi-layer services, but we must be able to clearly express where such pushing and popping will be transparent (if it is done implicitly), or we need to explicitly express these adaptations as part of the topology so that external agents can discover where push or pop of a new framing or header takes place – and which frame/header will be pushed or will be popped.



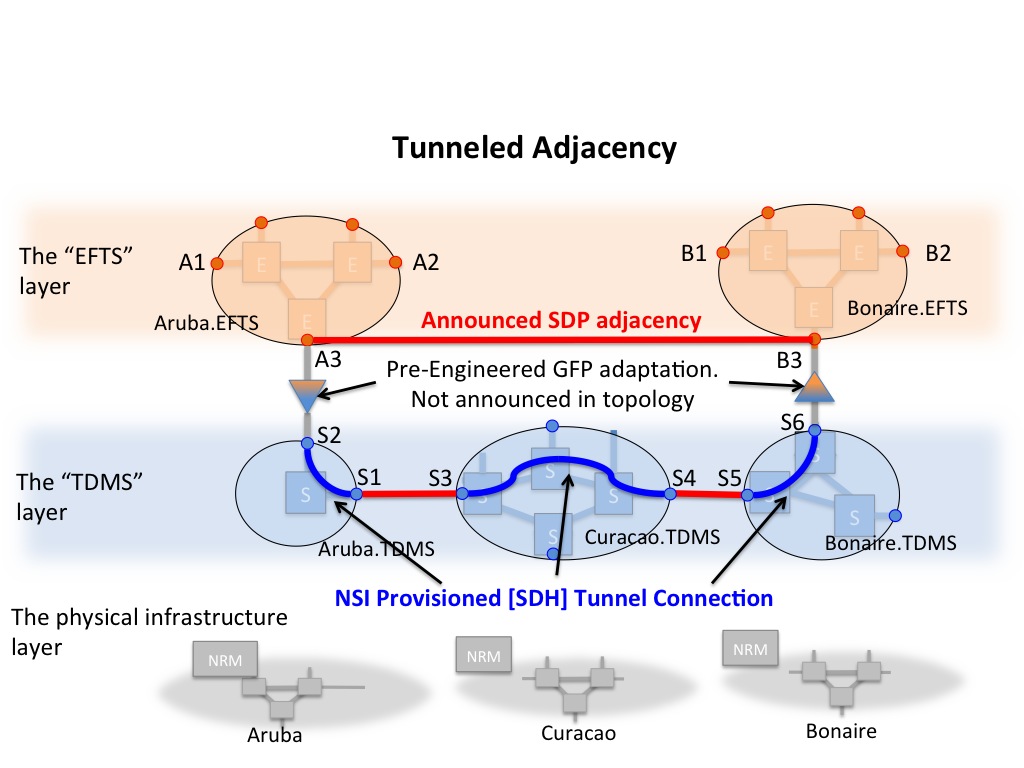
Take the example in fig 1, The Aruba and Bonaire networks offer an identical Ethernet Frame Transport Service (EFTS), Aruba.EFTS and Bonaire.EFTS respectively, to a set of clients. The EFTS service definition stipulates that the client interface is IEEE 802.3 “Ethernet” frames. Since these services offer transparent connectivity among their clients, they can be called the EFTS service “layer”. But these service regions are not connected, and so do not make up a single contiguous service layer domain. With no adjacencies indicated in the topology, there is no path that can be found to link clients in Aruba.EFTS to clients in Bonaire.EFTS.

Now, we insert a network Curacao that offers a Time Division Multiplexed Service (TDMS) that stipulates SDH based G709 STP framing. Aruba and Bonaire networks become clients of the Curacao.TDMS service – which means implicitly that they each constitute a TDMS service themselves, albeit perhaps with only a single STP. So the Aruba.TDMS service, and the Curacao.TDMS service, and the Bonaire.TDMS service all offer the same transparent connectivity between like client interfaces, thus they constitute a TDMS service layer.

*But there has been no adaptation engineering done to link these service layers.* A pathfinder trying to discover a path from Aruba.EFTS to Bonaire.EFTS will give up in disgust – there is no path in the topology even though the Aruba network participates in both the EFTS service and the TDMS service.

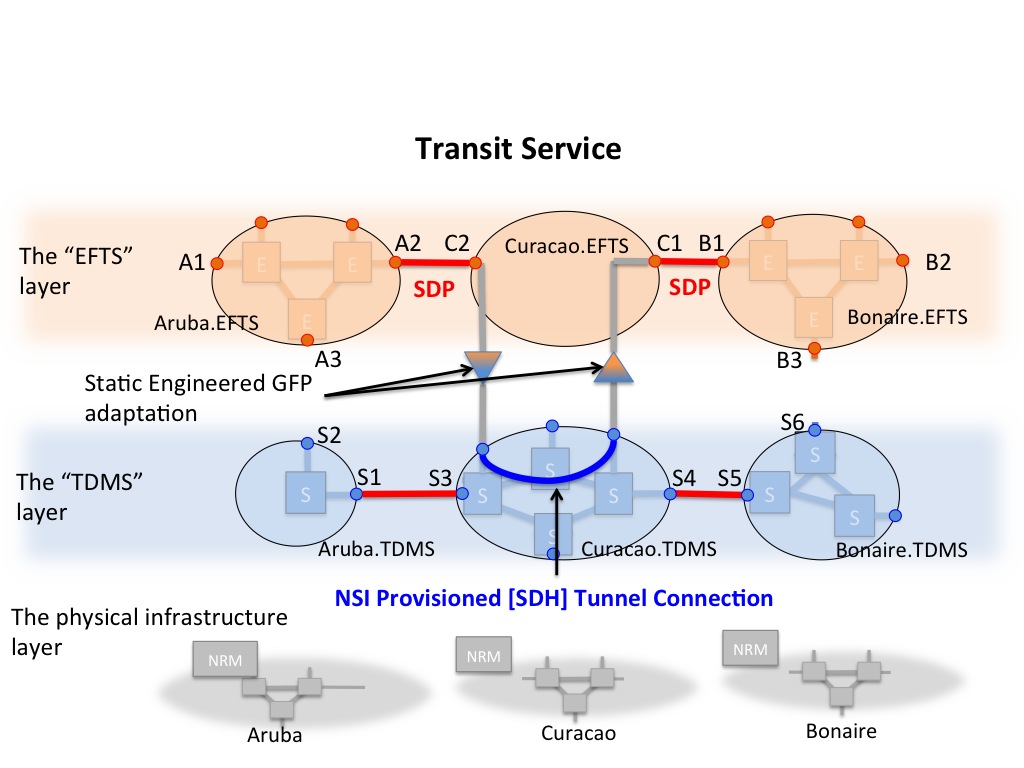
We can resolve this with any one of three basic approaches:

**First method: Tunneled Adjacency**. Aruba and Bonaire both install Ethernet-GFP/SDH adaptation equipment (line cards) and put the proper physical cross connects in place and configure the internal hardware to connect Ethernet hardware in the EFTS service to the GFP capable line cards. Then Aruba sets up a NSI Reservation across the TDMS layer from Aruba.TDMS:S2, to the Bonaire.TDMS:S6. Aruba and Bonaire then each do the following: They Define a set of new STPs in their EFTS domains, say Aruba.EFTS:A3 and Bonaire.EFTS:B3 respectively. Aruba internally maps Aruba.EFTS:A3 to the Ethernet interface of the Ethernet-to-GFP line card it just installed. Bonaire does likewise. At this point, the two EFTS services are linked across the Ethernet/GFP/SDH tunnel connection. Aruba then creates an SDP (Aruba.EFTS:A3, Bonaire.EFTS:B3) to show adjacency with Bonaire.EFTS, and Bonaire defines a similar SDP showing adjacency with Aruba.EFTS. Both EFTS service now announce the SDPs in the topology.



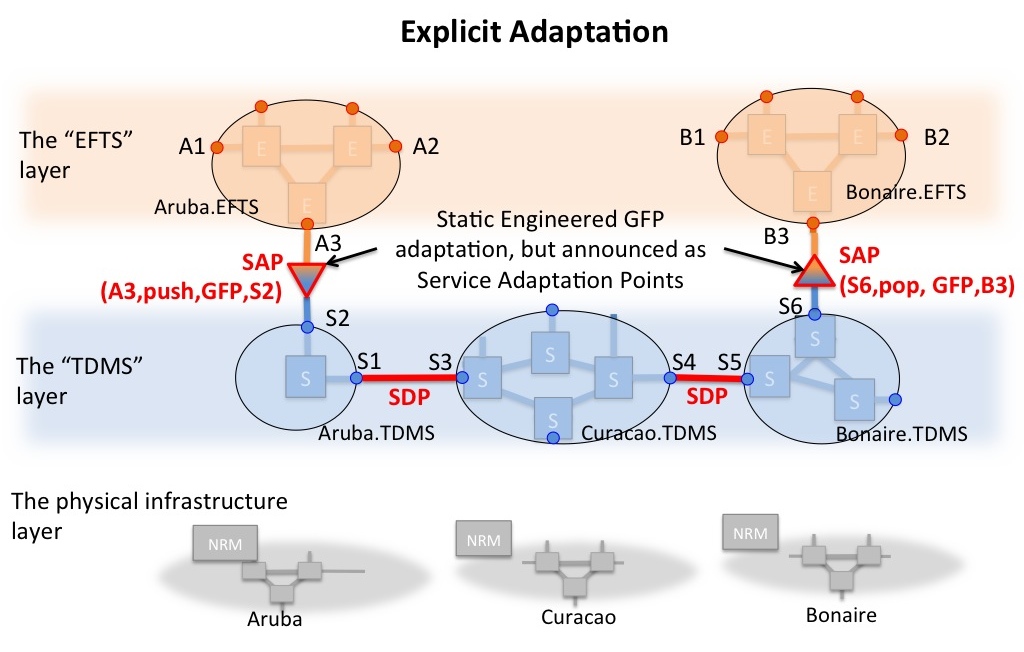
The two EFTS services now show adjacency (connectivity) at the EFTS layer between them. The TDMS layer services know only that a TDMS connection was requested from S2 to S6 – they know nothing about its content or purpose. And pathfinders do not know there is a SDH connection carrying frames between Aruba and Bonaire.

**Second method. Transit Service**. Aruba and Bonaire go to Curacao and ask Curacao to create an EFTS transit service. To do this, Curacao installs Ethernet-GFP/SDH line cards and defines a set of 802.3 framed STPs to create a Curacao.EFTS domain. Aruba and Bonaire arrange to connect their EFTS services to the Curacao.EFTS service. Now Aruba and Curacao define SDPs between their respective EFTS services; Bonaire and Curacao do similarly. Aruba.EFTS is now announcing adjacency with Curacao.EFTS, Bonaire.EFTS is also announcing an adjacency with Curacao.EFTS, and Curacao.EFTS is announcing adjacencies with both Aruba and Bonaire EFTS services.



Other EFTS services may connect to Curacao.EFTS and use Curacao as not just a transit service, but as a convenient “meet me” service, or an exchange, to reach many other EFTS services. This is not unlike the emerging Open Lightpath Exchange architecture.

**Third Method: Inter-domain Explicit Adaptation.** In this scenario, an adaptation function is created and announced in the inter-domain topology in which an EFTS client STP is “adapted” to the TDMS client STP. We cannot refer to these functions as a “layer” since they do not provide transparent connectivity between like clients, and since they adapt a single STP it is hard to think of these as a separate NSI service. Indeed, since these adaptations occur between certain service STPs, they look and feel more like connection segments or SDPs than network services.



Existing SDPs express topological equivalence of two STPs in different services of the same layer. These are essentially dimensionless transit “segments” that bind to endpoints of the connection segments chosen by a pathfinder. We can define a similar new relation that expresses a similar transit segment between STPs in different layers. Where as SDPs express horizontal adjacencies, “Service Adaptation Points” (“SAP”s?) would express vertical adjacencies.

**Discussion:**

In both the Tunneled Adjacency and the Transit Service approaches the adaptation is performed symmetrically internal to the EFTS services. Strictly speaking, this would be fine for the vast majority of service requests and simplifies path finding as multi-domain layer adaptation is not required. While we don’t need to worry about the frame stacking in these methods, the NSI Query function would not reveal the underlying tunnel(s) because these are not segments generated by the NSI service request. So in neither of these two approaches do we have visibility into the internal topology that the path takes.

In the Tunneled scenario the push/pop adaptation is coordinated and pre-engineered within the two EFTS services and then just the EFTS SDP adjacency is announced. The tunnel is treated as infrastructure internal to the Aruba network. Since the tunnel is administratively part of Aruba’s internal infrastructure, it could be set up manually in advance or it could itself use NSI to set it up on-demand or on a scheduled basis.

In the Transit Service scenario, the push/pop adaptation is again pre-engineered within, and internal to, the transit service. The adaptations are not externally visible to external agents – they only see the transit network service as an EFTS domain. But the transit service insures the push and pop of framing is symmetric across it’s domain.

Both of these implicit layering methods (tunnel and transit service) can be made somewhat more topologically explicit without breaking the implicit layering. For instance, the tunnel can be promoted to an explicit inter-domain EFTS service itself with only two endpoints – essentially resembling a simple circuit between Aruba and Bonaire. Indeed, it may map directly to an underlying resource such as a trans-Pacific SONET circuit that is framed as WANPHY. This strategy would allow that service to enforce unique authorization constraints, or to express other attributes such as latency or geo-location. But the path still appears to external PFs as an Ethernet Frame Transport Service – not an SDH resource.

The third approach, Explicit Adaptation:

A Service Adaptation Point must indicate the type of adaptation is being performed. Consider an Ethernet to SONET adaptation- there are at least three methods to encapsulate Ethernet over SONET: WANPHY. GFP-F, and GFP-T. Knowing which adaptation is pushed onto the flow is critical to finding the proper inverse SAP downstream in order to recover the original Ethernet framing. The adaptation type cannot be inferred simply from the source and target layers.

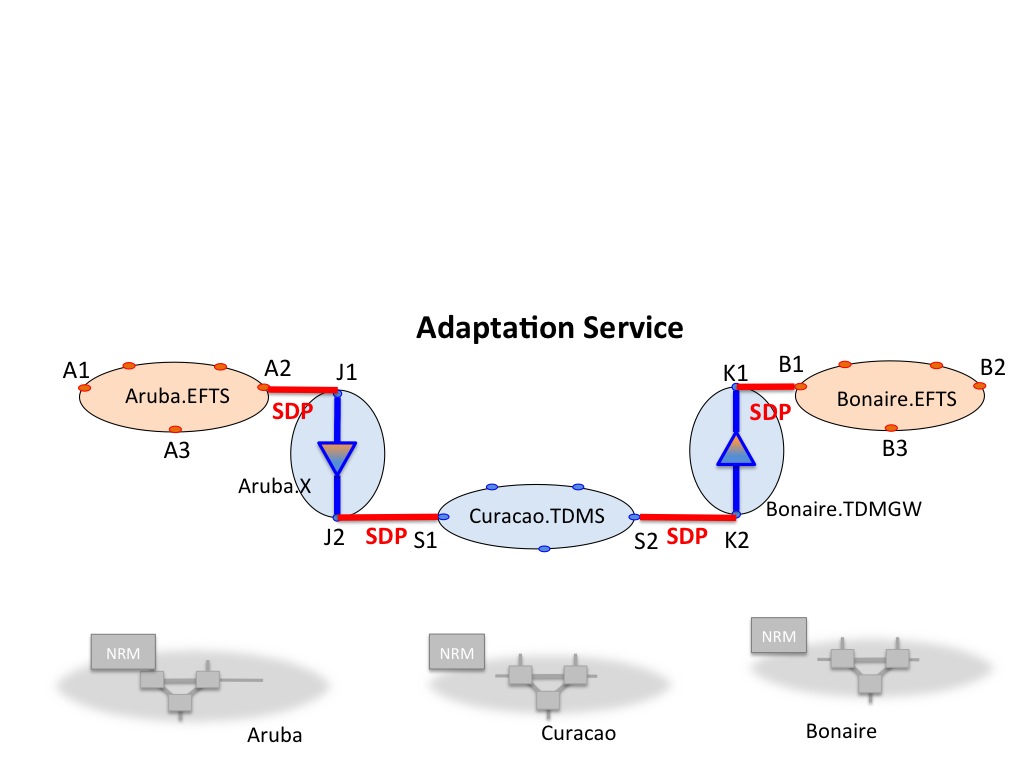
Further, as a path finder transits service boundaries hunting for a path, it must be able to determine when it crosses layer boundaries. By definition, a layer transition incurs an adaptation or an inverse adaptation, i.e. a new framing/header is “pushed” onto the stream or being “popped” off of the stream. Since the push and pop are simply reciprocal functions, we must know which side of the adaptation has the additional framing/header. This cannot be inferred simply by knowing the layers and/or the adaptation type. For example: In the GFP examples, we *assume* a push is being done as we transit a SAP from an EFTS service to the TDMS service because of our personal knowledge of Generic Framing Procedures. But PFs don’t know this. What if the adaptation type was PBB encapsulation? With PBB both source and target layers are conventional Ethernet. Is the adaptation function “popping” the PBB framing as we go from EFTS service A to EFTS service B? Or is it “pushing” a PBB frame onto the flow? A similar situation occurs when stacking MPLS LSP headers. Thus, we need to assert an “orientation” to the adaptation. We must be able to say “From *STP A*, a *GFP* framing is *pushed* onto the flow at *STP B.*

(This “orientation” of the adaptation in the path is confused further by the current NSI bi-directional service model. In a bi-directional circuit model both a push and a pop are always performed at each SAP as the bi-directional data-flow transits layers, but it must be known which STP represents the “upper layer” STP or the “up stream” (closer to the origination) STP in order to determine whether a push or pop is to be accounted for in the path. The orientation varies depending upon which direction you think of the “flow” as going relative to one or the other domains. If we had unidirectional STPs and unidirectional service model, this orientation would be self evident.)

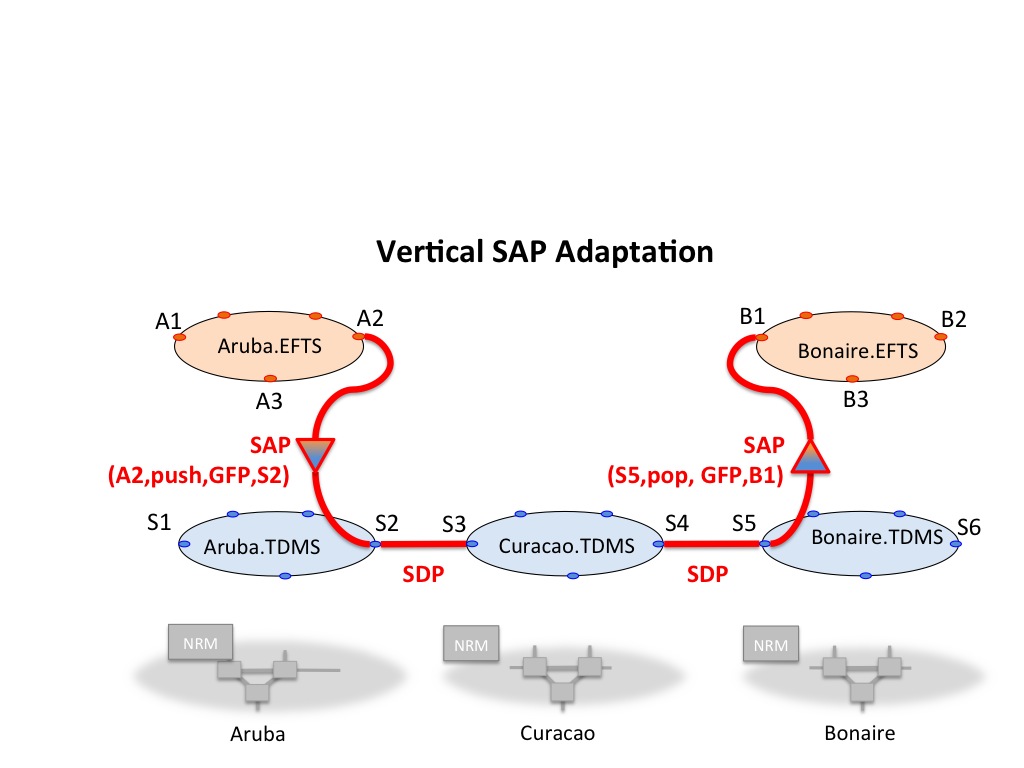
Finally, an adaptation relation is, in essence, a static pre-engineered connection segment waiting to be incorporated into a connection. Since STPs reside at the edge of a network service, they bind an internal connection segment to an external segment. For instance: an SDP peering (i.e. “connectedTo” relation) constitutes an external binding; and an intra-domain reservation segment would constitute an internal binding at an STP. Thus, the *adaptation segment* must likewise be bound to either the internal side of the STP or the external side of the STP. To date, the NSI topology model always stipulated SDPs (ConnectedTo relations) as the only means of external linkage. So all other bindings were implicitly internal. Now, we need to be more explicit.

There are three ways we could define the explicit adaptations:

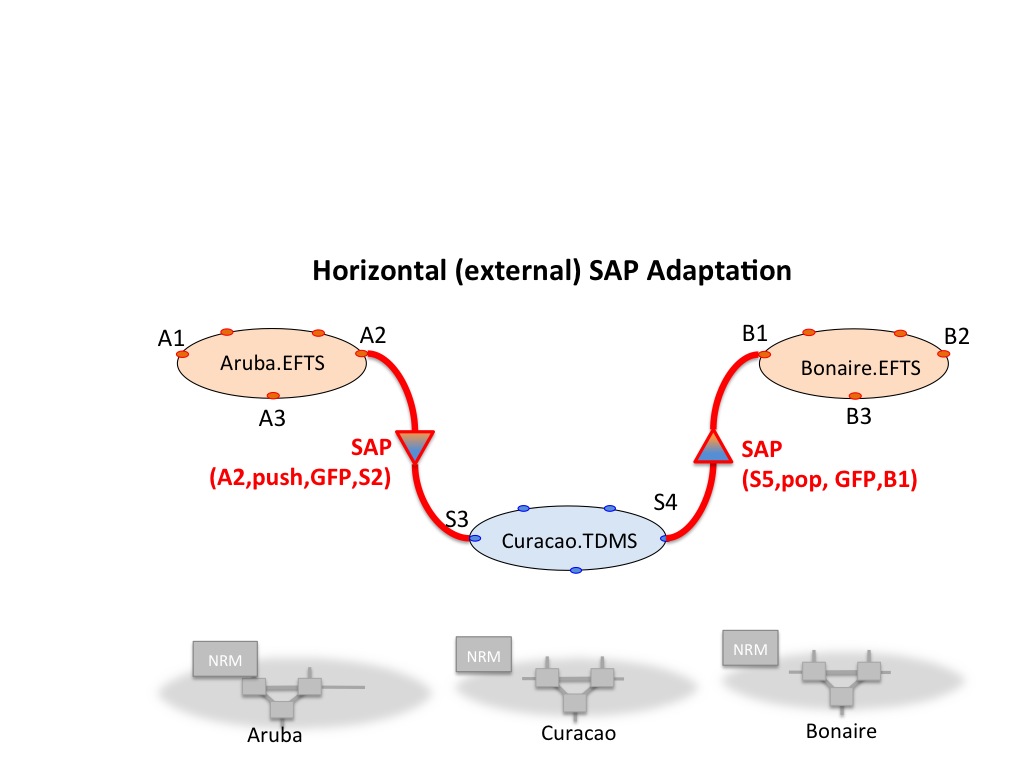
1. The “adaptation service” approach: This approach preserves existing topology. We define the adaptation segment as its own network, with two STPs. In this approach, the adaptation segment is a conventional intra-domain segment bound to the internal side of the source and target STPs. And a conventional SDP is defined externally between the adaptation service STPs and the layer services. The NSI CS protocol handles this just like any other network in the path. However, for this approach to be viable, we still need to announce the adaptation type and stacking orientation of the adaptation. This approach differs from the “Tunneled” examples above where both a push and pop are statically pre-engineered in the tunnel. In this approach, only one of the push and pop adaptations are present in a segment provisioned across this adaptation service domain – so external agents need to know what that adaptation is, and in which orientation it is applied. The drawback to this approach is that a NSI Network must be defined in the topology for each pair of STPs that constitute an adaptation. While this is inconvenient, it is not a scaling issue as it could be stated in a compact fashion.



1. The “vertical SAP adaptation” approach: In this approach, each network domain implements a set of vertically overlapping “services” corresponding to each layer they anticipate interacting with. The adaptation segment (SAP) transits vertical layers between corresponding STPs in each layer, but it does not transit horizontally service domains. I.e. the adaptation segment is bound to opposite sides of the source and target STPs respectively. For example: the GFP adaptation between Aruba.EFTS:A2 STP and the Aruba.TDMS:S2 STP would have the adaptation segment bound externally to A2, and internally to S2. Or vice versa. In this approach, the adaptation segment is treated in a similar fashion to an SDP – If a connection segment is bound to one of the STPs in the SAP, then it is connected to the other by definition. However, since we now have at least two ways to indicate linkage between service domains, and two possible binding sites, we need to generalize the existing model to indicate explicitly the internal/external bindings for connectedTo or adaptedTo relations.



1. The “horizontal SAP adaptation” approach. In this approach, the SAP is bound to the external side of both STPs. In essence, the SAP becomes more like an SDP, flattening the inter-domain network to be just a set of service domains interconnected by SDPs/SAPs. Or put another way, an SDP becomes an SAP whose adaptation function is “null.” The drawback in this model is that, unlike the SDP, the SAP STPs do not reference the same topological location. There is tangible topological functionality/resource existing between the STPs in the SAP. We need to allocate this adaptation segment to one service domain or the other. If we allocate the SAP to Aruba.EFTS, then that means the SAP is internal to Aruba.EFTS. Which means A is announcing internal topology – an internal resource that connects the two STPs…i.e. a internal “service” and we are now back to the “adaptation service” approach mentioned above.



So we really have only two approaches: The vertical internal-to-external SAP mapping where SAPs and SDPs are very similar, and the adaptation service approach where the adaptation is treated as a service. There is really no reason these both can’t co-exist in the same topology ontology, though some enhancements are needed.

So, for explicit adaptation to work, the *adaptation type*, *adaptation stack function*, and the *adaptation orientation* needs to be expressed for every announced SAP. In general, it may be sufficient to simply have unique identifiers for each *type* of adaptation. Since the push/pop and orientation is indicated at the SAP, a PF can simply push/pop that adaptation type identifier onto a stack associated with the candidate path and insure that a reciprocal SAP is incorporated downstream. The PF need not care about the specifics of the adaptation as it has been pre-engineered into the physical network infrastructure that constitutes the adaptation point and will be configured appropriately by the NRM.

We also need to introduce a polarity or directionality to the STPs. We need to differentiate their internal linkage from their external linkage. If future NSI is enhanced to support unidirectional connections, it will need to support unidirectional STPs, and this topology issue is solved intrinsically. (In fact, unidirectional STPs more accurately reflect the physical infrastructure.)

**Recommendation #1:** We maintain the existing NSI topology model and allow or even encourage tunnels or exchange point NSI Services that implicitly hide multi-layer complexities.

**Recommendation #2:** For explicit multi-layer path finding at the inter-domain level, we define a Service Adaptation Point (“SAP”) construct to indicate the type and orientation of the inter-layer adaptation at that point. The SAP is a 4-tuple that contains <sourceSTP> <adaptationFunction><adaptationType><targetSTP>.

**Recommedation #3:** In conjunct with SAPs, we reform the STP/SDP representation in the OWL topology to support enumeration and internal/external bindings.

SAP Examples:

GFP from EFTS to TDMS, evaluated from left to right as a vertical SAP:

(Aruba.EFTS:A3, push, GFP, Aruba.TDMS:A1)

Using the OWL/RDF topology representation from the SC demo, the following fragment could describe an Ethernet to SONET adaptation at StarLight facing Netherlight:

<owl:NamedIndividual rdf:about="urn:ogf:network:stp:**starlight.efts:ams**">  
 <rdf:type rdf:resource="http://www.glif.is/working-groups/tech/dtox#STP"/>  
 <externalBinding>

<adaptedTo rdf:resource="urn:ogf:network:stp:**starlight.tdms:ams**">

<adaptationFunc>**Push**</adaptationFunc>

<adaptationType>**GFP-F**</adaptationType>

</adaptedTo>

</externalBinding>

<internalBinding></internalBinding>

</owl:NamedIndividual>

The corresponding STP in the TDMS layer would then look like this:

<owl:NamedIndividual rdf:about="urn:ogf:network:stp:**starlight.tdms:ams**">  
 <rdf:type rdf:resource="http://www.glif.is/working-groups/tech/dtox#STP"/>  
 <internalBinding>

<adaptedTo rdf:resource="urn:ogf:network:stp:**starlight.efts:ams**">

<adaptationFunc>**Pop**</adaptationFunc>

<adaptationType>**GFP-F**</adaptationType>

</adaptedTo>

</internalBinding>

<externalBinding>

<connectedTo> rdf:resource="urn:ogf:network:stp:**netherlight.tdms:chi**"/>

</externalBinding>

</owl:NamedIndividual>

In the RDF/OWL representation above, we define an “adaptedTo” relation between the local “source” STP in one layer to a “target” STP in a corresponding service in a different layer. The “adaptationType” relation specifies the adaptation being performed in going from the source STP to the target STP. The “adaptationFunc” relation specifies the orientation – i.e. if the adaptation is a “push” then the target STP has the new framing/header applied, if the adaptationFunc is a “pop” then the adaptation has been removed from the target STP.

These adaptations are encapsulated in <externalBinding> or <internalBinding> clauses to indicate the how the adaptation segment meets the STP.

**Related issues to the proposals:**

First, the proposed syntax is just that: a proposal. A broader discussion should be taken to weigh the various approaches and how their representations integrate with other issues such as enumeration and the general issue of how we represent topology for exchange.

It should be also noted that the tunneled adjacency and the transit service methods could be implemented using NSI and explicit adaptations as well. All three of these techniques can coexist.

However, it should be also observed that the adaptations themselves are – in all three cases – still relegated to one network or another. Thus the announcement of explicit adaptations is still the purview of the network that owns that adaptation. There is no obligation on a network’s part to announce explicit adaptations. Where appropriate, the NSI Best Practice may encourage or recommend that NSI providers announce such multi-layer service adaptation capabilities. In any case, an external PF will only be able to incorporate explicit adaptations where networks have announced the capability in the expressed topology.

There is an unresolved issue where the external “horizontal” SAP is announced. There is no obvious ”owner” for these inter-domain adaptation segments. So while they are simple, they break a fundamental rule of NSI: All resources are assigned to a network. Since these adaptation resources are bound to the external side of the STPs, they are essentially outside both networks…in no-man’s land. Thus we need to assert a semantic or attribute that allocates ownership to one of the services. This is an issue for discussion.

The issue of internal topology announcements is also relevant here. As can be gathered by this discussion, “NSI Networks” are really NSI “service domains”. The notion that all NSI domains are conventional networks is probably not a useful model. If we want to expose internal topological detail, we probably need to modify our concept of an “NSI Network” to be more one of a “service” - a construct that is both lighter weight and a more flexible representative of topological functions than simple collections of hardware. Thus, a generalized “service” can represent an aggregated large scale network service (e.g. GEANT) and also recursively represent finer detail such as NRENs or regionals or individual switching nodes (GOLEs? Switch?) and/or other groupings of STPs. The notion of both a Transfer Function (e.g. Any-to-any switching, GFP encaspusltion, [possibly] pairwise switching, etc.) could be useful along with an “Oracle” or agent (e.g. NSA) to handle service requests where generalized transfer functions can’t represent all facets of a network service. The point is that our basic NSI topology is still simple an works elegantly, but we need more sophistication in terms of enumerating resources and describing connectivity.

**Comments on Adaptations and NSI Service Definitions**

We should re-iterate how important it is to clearly understand the payload and framing of each service and each layer in order to clearly define what is being “adapted”. In order to deliver the payload end to end, across varied layers and services, there can be no confusion about which framing format or header is being pushed or popped, and the specific definition of “payload” as it crosses each STP.

For example: An “Ethernet Frame Transport Service” is presumed to transport “Ethernet frames” from the ingress STP to the egress STP. Thus all EFTS STPs should pose an 802.3 “ethernet frame” (the entire PDU) as the ingress payload regardless of the content of the frame. Thus, a true EFTS service would carry the entire Ethernet frame – src mac, dst mac, ethertype/length, any .1Q or .1ad tags, payload, and fcs. And this same frame would be presented at the egress point. Such a service would not allow tagging or VLAN re-write as that would modify the “payload.”

There are two ways to implement such a EFTS service: 1) Use Ethernet as the core switching technology…this has significant drawbacks (see 802.1ah PBB), or 2) encapsulate the entire frame and carry it as payload data in some other core transport framing. The former method would tempt a transit network to tag that frame and place it into a VLAN for transit…*but this would be wrong*. Adding a tag to the frame is adding additional header information that must be removed later – i.e. a layer adaptation. And, due to Ethernet idiosyncrasies, it requires the ingress frame to be untagged to begin with – which is not a valid assumption for an EFTS service, nor would that even be known at the path finding stage. So we must be very explicit about what the “Payload” is across this NSI service, and this includes explicitly specifying whether a) the entire STP framing+payload (i.e. the PDU) is the transport payload, or b) whether the Ethernet framing is just a common client interface framing standard for all STPs in this service but which is not formally part of the transport payload, or c) whether the ingress framing is simply a local framing and the egress framing can be anything as long as the payload section of the ingress framing is presented as the payload section of the egress framing.

If we assert (b) or (c), then we are saying that the Ethernet frame *payload section* is the transport payload and we can route that payload to any STP that is “Ethernet framed” and we can deliver the payload inside an untagged frame, or a tagged frame or a double tagged frame. And we can re-write the VLAN tag where we need to. So, as an alternative to the Ethernet Frame Transport Service, we could instead define a Ethernet Framed *Data* Transport Service (EFDX) that requires all EFDX STPs to be 802.1Q framed STPs, with the transport payload being the user data section of the ingress 802.1Q frame – not the entire PDU frame itself. Thus, we can now push/pop or modify VLAN tags to take advantage of VLAN swapping flexibility (because the frame is not part of the payload) or capabilities like PBB or VPLS. We can implement an Ethernet core to support the service (though there are still other issues to be addressed – see PBB), or we could completely strip the Ethernet framing altogether upon ingress, transport the payload section on hand written messages tied to carrier pigeons, re-adapt the messages into the payload section of Q-tagged Ethernet frame and present that frame at the egress STP. The EFSX service is a much better service definition because it stipulates a standard service client framing *and* clearly defines the service payload.

If we look at the Automated GOLE facility, what we are providing is in fact an “Ethernet Framed Data Transport Service” that stipulates 802.1Q STPs. I.e. we expect an Ethernet 802.1Q PDU to be presented at the ingress STPs, and the service (e.g. NetherLight.ets, or NorthernLight.ets) is only bound to carry the payload across the core and present that payload at the egress STP in the payload section of another 802.1Q PDU. This allows us to re-write VLAN tags along the way (useful if we are doing VLAN translation).

Multi-layer Adaptation relies on knowing *exactly* what constitutes the STP framing and the payload at each service boundary. If all of the services that constitute a single NSI layer agree to a single common STP framing and clearly stipulates whether the <PDU> is the service payload or <the payload section of the PDU> is the service payload, then the adaptation requirements become simple and very clear.

A formal Service Definition should describe the common features of all NSI domains that claim to offer that particular service. Most notably to this discussion, the SD should specify the common client interface attributes such as STP Framing and Service Payload. The Service Definition should be announced with the topology of a network. The name for a Service Definition can be used to match network services to determine which are compatible (same layer) and which are in different layers and require adaptation.

Many of the complexities associated with multi-layer path finding stem from very broadly defined or very poorly defined “services” that try to express every feature of every piece of hardware. NSI is an approach that defines the service definitions first, clearly and deterministically, and then relies on good engineering teams to realize those service definitions. This approach obviates the need in most cases for complex pathfinders that need to understand specifics of a huge variety of hardware and standards in order to construct and useful paths. The theory for path finding and stack encapsulation is well understood and can work practically in real networks if the theoretical abstractions and semantics (such as header and payload specifications, forward and inverse adaptations, service boundaries and stitching points) are clearly defined for each service domain. By doing our homework upfront as we define our service offerings we will resolve many of the challenges that have torpedoed such efforts in the past or relegated them to very limited environments. Taking these issues as a whole will substantially and significantly simply multi-domain multi-layer path finding.