UvA/TU Delft topology exchange and path finding in NSI environments

Ralph Koning, Miroslav Živković,
Stavros Konstantaras and Paola Grosso (UvA)
Farabi Iqbal and Fernando Kuipers (TU Delft)

February 13, 2015

1 Introduction

The NSI - Network Service Interface - protocol enables different network domains to negotiate, reserve and provision end-to-end paths on demand. To function properly, NSI requires the knowledge of the network topology of the participating domains; this knowledge in turn relies on mechanisms to exchange such information between the involved parties.

In this document, we propose to address the following requirements:

1. optimal path finding on the provided topologies
2. ability to find disjoint paths (for link protection)
3. security (message exchange, non-repudiation of origin/delivery)
4. possibility of providing representations of the same topology based on:
   (a) requesting party
   (b) peering agreements
   (c) other policy information

2 Topology exchange considerations

When designing a topology exchange mechanisms we need to decide whether we want to rely on a centralized or decentralized system and to which level we intend to disclose the internal topology of each domain. Here we present the general features, pros and cons of these issues. In the next section we will present our chosen solution.

2.1 Centralized topology exchange

A centralized topology exchange implies that the topology information is maintained at a single location, possibly by a dedicated trusted entity. All domains would supply the entity with their topology, and in return are given the topology of other domains when they need them for routing purposes, provided that proper authentication exists. The topology database can also have a dedicated path finder, such that it could compute paths on behalf of a domain without disclosing any topology information of other domains. The advantages of using a centralized approach are the ease of information access, while simplifying the communication between domains. The downside of it is that the topology database entity needs to always be available and reliable, and frequent

updates of all domains are needed to maintain an accurate view of the network. Since a centralized database is a single point of failure, a backup database may be needed to function in case the centralized database fails.

2.2 Decentralized topology exchange

A decentralized topology exchange approach implies that the topology information of each domain is maintained in a database available only to its domain. Disclosure of a domain topology is only provided to other requesting domains for path finding purposes. Hence, frequent topology exchanges could be expected between domain controllers for path finding purposes. The advantages of using a decentralized approach is that the information supplied to other domains is always accurate up to that point of time. The benefits of this, apart from the reduced load, are the confidentiality and the resultant resilience by having no single point of failure. The downside of a decentralized topology exchange is the increased signaling complexity between domains.

2.3 Disclosure level

When exchanging topology information, the level of disclosed information could vary depending on each domain and to whom the information is to be disclosed. In a centralized topology exchange approach, the same topology of each domain could be expected for all requesting domains, unless several topology versions can be maintained by the centralized database. In the decentralized approach, each domain can provide any level of detail to the requesting domains. For example, in the lowest degree of cooperation, the domains may share no internal domain information and only communicate to find a suboptimal loose path across their domains. The loose path only provide information on the order of domains to be traversed, and the corresponding ingress and egress points, without any details on the intra-domain path segments. Each domain that is a part of the loose path must further refine the path segment across their domain, to get an exact multi-domain path. In the highest degree of cooperation, each of the domains will trade full internal domain information, resulting in an optimal end-to-end path computation. However, full disclosure of internal domain information may pose a threat to the domain due to possibilities of attack, sabotage, or competitive intelligence between domains. The scalability of topology information exchanged could also pose a problem. The advantage of providing a minimized internal domain view is that less information needs to be processed during path finding, and for safety concerns. However, insufficient topology information may lead to suboptimal path or even failure to compute paths.

2.4 Announcing reachability

Another way of exchanging topology information is by announcing reachability. This discloses even less topology information. In this case, the topologies are only shared between neighbouring domains, and a neighbouring domain announces that it can provide a connection to that domain. The routing is done locally, per domain, based on the information of itself and its direct neighbours leading to a locally optimal route. Because the domains never have a full view of the network there is a chance of getting suboptimal paths. It is not possible to provide a different view to the requester domain and malicious domains can easily provide false reachability information and break global path finding.

3 Topology Exchange Architecture

Since we believe sharing topology information provides more flexibility over sharing reachability information, we propose a hybrid approach to exchange topologies: the topology information is stored in a decentralized manner augmented by a centralized index to quickly point to where the topology information is stored. Because topologies are provided by the domains themselves we leave it up to them what topology information they disclose.

Our approach distinguishes three components that work together to form a topology exchange; a domain can choose which components it will run itself or share them with befriended domains:

Topology Index (TI) is a database that holds pointers to the topology providers and summary information. We describe this in detail in section 4;

Topology Provider (TP) hosts the topology descriptions. TPs are described in detail in section 5;

Topology Consumers (TCs) are the components that use topologies:

- Path finding component
- Monitoring component
- Topology validation component
- Lookup components

TCs are described in section 6

All components take security into account; by default all communication is signed and encrypted. It also provides a basic infrastructure to share public keys, yet we recommend using a PKI infrastructure for this.

In the following sections we will elaborate on the different components. For an overview of the interaction between them refer to subsection 7.1

4 Topology index

For practical purposes the Topology Index (TI) is considered a trusted third party. In this way it can also be used to share public keys amongst domains. These keys can be used by the topology consumer to verify the signed information it retrieves from topology providers or to encrypt the data that is being sent from the provider to the consumer.

All domains will maintain and publish their own topology files, *i.e.* all domains participating act as TPs; a central index holds the time stamped pointers to the topology description files in the various domains. Domains are the only holders of policy information: this is not propagated to the central index.

4.1 Data format

Table 1 shows the data format of the topology index. The various fields have the following meaning:

domain is the domain name of the topology provider;

version of the topology file or a time stamp;

topology location is a pointer to the location of the topology file; the URL of the file needs to be reachable by the topology index before it can be included in the list; as we outline later it is still possible that some domains cannot directly retrieve each others information but this validation step guarantees that topologies are always indexed;

public key of the domain originating the information; the public key plays an essential role in our infrastructure as it allows

Domain	Version	Public key	Topology location	Neighbours	Foreign domains	Signature

Table 1: Topology index entry

to verify digital signatures and encrypt messages so they can be only decrypted by the intended recipient;

neighbours is a list of domains that are directly connected; the domain has peering relationships with them;

foreign domains is used to publish external domains with data plane connections to a domain listed in the index. This may allow basic path finding through domains that are not known to the repository. This concept will be explained next.

4.2 Foreign Domains

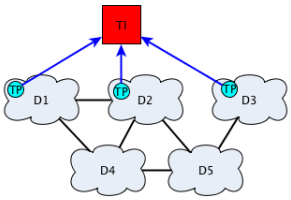
Foreign domains are domains that have direct data plane connections to domains listed in the index but do not report to this topology index, in other words it is a list of domains external to this system. The list may be augmented with information describing the connectivity in more detail.

We can use foreign domains to indicate connectivity to domains that use different systems for exchanging topologies. In addition to this, a topology provider can share a link to (partial) topology information of the foreign domain.

4.3 Populating the Topology Index

Figure 1 shows how the topology index looks like for an example network. In this example we excluded the key and signature fields. Domains `D1`, `D2` and `D3` are topology providers registered in the TI and therefore they can only show in the neighbour list.

The other domains `D4`, and `D5` do not run a TP that registers to this TI but are connected



Domain	Topology	Neighb	Foreign
D1	https://d1.org/topo/	D2	D4
D2	https://d2.org/topo/	D1	D4, D5
D3	https://d3.org/topo/		D5

Figure 1: Retrieving keys using DNS

to `D1`, `D2` and `D3`, therefore they are listed in the foreign domains list of the respective domains.

The foreign domain list is composed as follows: The Topology provider sends a notification to the TI that the topology is updated, and augments it with all domains it connects to on the data plane. The TI receives this full list, and compares it with the domains known to the system. The domains that are known will be listed in the neighbour's field and all the others end up in the foreign domains field. If a new domain registers at the TI, the TI checks all foreign domains fields and will move them to the neighbours field where necessary.

A foreign domain is represented as a tuple containing the domain name and the URL for additional topology information, e.g., `["d4.net","http://d4.net/topology"]`. The additional topology information is optional.

4.4 Security concerns

We assume that the topology index is trusted and that the domain participating will have provided the repository with their public key

at the start of the process and that the public key of the topology index is known to the participating domains.

We assume that if a domain hosts a TC and a TP it will use the same private/public key for both components; this allows a TP to look up the key of a TC and we do not have to maintain a second registry.

Updates or insertion of data in the table need to be signed by the originating domain with its own private key. The topology index will update the table content only when the digital signature is verified by the public key. It will also list the signature over the data so consumers can verify this. This will prevent malicious domains to tamper with information of other domains.

To prevent false index information updates the topology index must verify who pushes the index updates. This can be done in two ways:

1. The TI maintainer first verifies and obtains the domain name and public key via an out of band channel and it adds them to the system. Any updates can now be verified by checking a signature.

2. The public key can be distributed using for example the Domain Name System. Figure.2 illustrates this. The `A` or `AAAA` record of the server can be augmented by a `CERT` record containing the public key. This trust is not ultimate unless the domain uses DNSSec to sign their zone files. The advantage of this method is there's no manual intervention needed to add a domain to the system and you can verify the domain keys without using the topology index as a trusted party. Also the domain name used here must match the domain name used in the DNS system.

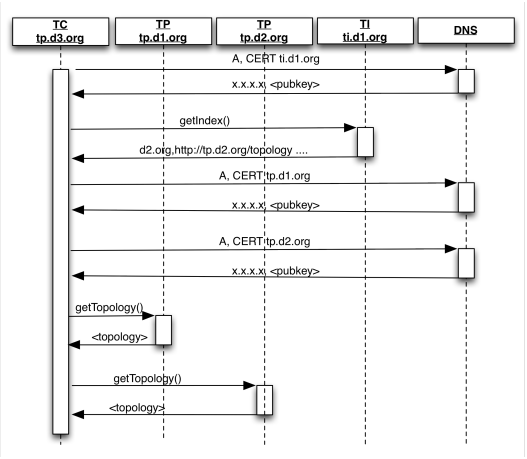


Figure 2: Retrieving keys using DNS

4.5 Proxy requests

If a domain cannot reach the topology URL of the domain it can ask the topology index to retrieve it on its behalf and act as a proxy. This assumes that all URL/pointers to topologies are reachable by the repository.

For foreign domains we cannot guarantee that the topology index can reach that topology information since they might not adhere to the prerequisites defined in this document.

4.6 Synchronization and fail over

If the topology index becomes unreachable there will be a severe degradation of performance. Although the topology information remains available and the topology consumer can still retrieve them based on cached information this is an unwanted situation.

Replicating the index would solve this problem. A topology index can be a topology consumer of another topology index. This would allow index A to receive updates when index B changes and merge this information when the version is newer. When a conflict arises index A might have to ask the conflicting topology providers to re-send their summary informa-

tion. Then index A sends out update notifications and topology consumers including index B can update their information. Topology consumers of course need to be aware of this backup server and know its public keys.

5 Topology Provider

The topology provider is the service that provides a topology for a specific network or domain. This service can be relatively simple, it needs to be able to share a topology file to topology consumers and send information to the topology index in case of an update. If necessary more advanced functionality can be implemented, like different topologies based on the consumer or decisions on via what domain the consumer comes from.

5.0.1 Topology files

A topology file contains the topology information for that domain. The format can essentially be anything as long as it can be interpreted by the consumer. It's up to the domain what information will be shared. The most common use will be sharing inter-domain topologies, this is a summarized version of the full topology which only contains the service endpoints and connections between domains though there is no restriction on sharing the full topology.

A topology provider has to share a topology file by default this is what is supplied to any topology consumer making a request this is the default topology file.

Based on the requesting topology consumer the topology provider can choose to supply other topology files. This gives the ability to create multiple views of the network, it can also be used to reflect policy information and to remove links that cannot be used by the consumer.

If the topology consumer knows roughly what path it wants to take or knows the neighbouring domain it wants to traverse through the consumer can supply this information in the request. Based on this more filtering of topology information can be done.

This filtering is not intended to enforce policy or restriction, yet reflecting the policies in the topology file will aid the path finder skipping links that cannot be used later on, making the overall process more efficient.

6 Topology Consumer

The topology consumer, e.g. a pathfinder, is the component that performs an operation on the topology information.

Depending on whether the topology index is open, first the consumer needs to exchange public keys with the topology index. Only then the consumer can consult the topology index. The consumer looks for the domains it is interested in, processes the summary information when necessary and then requests the topology files from the topology providers.

6.1 Update notifications

A consumer can also subscribe itself to the index. When a topology provider publishes a new or updated topology to the index, a broadcast message is sent to all subscribed consumers with the information that the new topology has been added or updated. This message is signed by the topology index.

The consumers could then determine whether they want to update potentially cached information by retrieving new information from the topology index. For example adding a new domain to their list of domains, or updating existing topology information for known a domain.

In order to receive update notifications the consumer should provide a callback address to

which those notifications should be sent.

These services can be different implementations ran by different domains. The only prerequisite is that there is a trust relation between the consumer and the topology index.

6.2 Authentication

It is not necessary for the consumer to have a private and public key pair to retrieve topology information. It can verify the topology information with the public key of the provider and use the public key to securely send a symmetric session key to the provider so secure communication can take place.

However, if some topology providers want to customize topologies based on the requesting domain the consumer should still be able to authenticate this with the private key of the requesting domain.

7 Architecture operations

7.1 Interaction between components

The assumption is that each domain has a single topology provider, though some domains might agree to use a path finder from another domain or share one.

7.2 Bootstrapping

In order to exchange topology information in a secure way some prerequisites have to be met:

- Topology index needs to be known and accessible by all consumers and providers
- Topology providers need to be accessible by at least the index
- Topology index and providers both need a private/public key pair
- Topology providers always exchange public keys with index
- Topology consumers optionally have a private/public key pair

This information can be communicated by email or communicated automatically. Since there's no secret information shared (only public keys) this can be done via a unsecured connection, of course with the risk of someone else impersonating the index or provider when no PKI infrastructure is in place.

7.3 Topology update

Figure 3 shows that when an update is made to the topology the provider notifies the index and provides some summary information (1). The Topology index will in turn notify all registered consumers that there is an updated index (2).

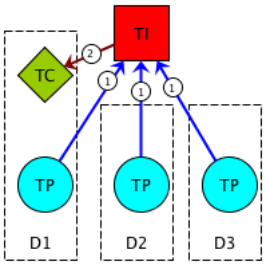


Figure 3: Topology update

7.4 Topology retrieval

In Figure 4 the topology consumer contacts the central index and looks for updates from the topology providers it is interested in and gets their public key (1). The topology consumer contacts all the providers it is interested in and retrieves the topology files (2).

Figure 5 shows the key distribution during the bootstrap phase:

1. The topology providers in D2 and D3 send their domain's public key to the index and the index verifies the providers.

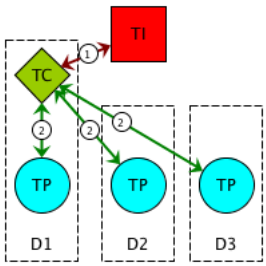


Figure 4: Topology retrieval

2. The topology index sends its public key to consumers in D1 and D2, the consumers may verify the index.

3. The topology consumer optionally sends its domain key to the index. In case of D2 this is already done since it also runs a provider. In this way a consumer doesn't have to be known by the index to request information.

7.5 Key distribution

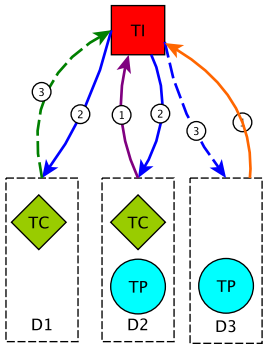


Figure 5: Key distribution bootstrap phase

Now that the bootstrapping is done and there is trust with the index, this can be used for key exchange in the operational phase. This can be seen in Figure 6.

1. Topology consumer requests the public keys of the provider domains from the index. After this step the consumer contacts the providers directly.

2. Topology providers may request the consumer's key from the index if they want to authenticate.

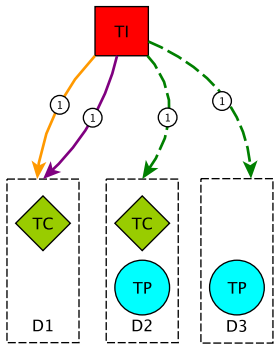


Figure 6: Key distribution operational phase

7.6 Path finder interaction example

First the path finder fetches all the information from the central index (i.e. the table entries). This information includes pointers to topology files and summary information. The path finder creates a network graph topology, and it can cross out the domains it is not going to use. The path finder requests the full topology for the relevant domains. The path finder uses the retrieved topologies to create a more detailed graph. The path finder prunes out mismatches and unusable links such that they will not be considered during the routing process. The path finder tries to find the most suitable paths and returns the best.

7.7 Validation of topologies

Authenticity of information The topology document of a domain is stored by the domain itself and it can be validated by other domains in order to identify tampering with the document.

1. All topologies are digitally signed by private keys of the providing domain.

2. The domain sending the topology will encrypt the topology using a shared secret that the requester shares with the sending domain. The shared secret is communicated to the sending domain by encrypting this with its public key which was retrieved from the topology index.

3. Optionally the sending domain can sign the request. In this way the sending domain can be authenticated by the provider.

Correctness of format We expect that both the provider and consumer can validate the correctness of the topology information. If the topology information of a domain is unreadable, the path finder should treat this domain as non-existent and cannot use the domain as part of a path.

Correctness of information Only the provider knows whether this information is correct, and should check this before publishing. Mistakes can be made and this can become a problem when there is a mismatch of information regarding a connection between domains. In the majority of cases this needs to be solved by human intervention of the domains network engineers.

Path finders should be resilient to these mismatches and not provide them as an option to the user. This means mismatches in topology information go unnoticed to the user unless you specifically make a request to use such a link.

A way to mitigate this is to provide a separate service that contacts the topology index to request the topologies of a domain and its neighbours and to perform cross checks to aid in identifying topology mismatches. The domains network engineers can then act upon this and update their topologies.

8 Path Finding

8.1 Path Requirements

Certain clients or providers may prefer a more detailed inter-domain path. Though conventional shortest path algorithms such as the Dijkstra's algorithm [**?**] may be used by the path finder to find a simple, shortest inter-domain path between two distinct domains, a more intelligent algorithm is needed to accept more path requirement details, and provide an inter-domain path that satisfies the given requirements in return.

Examples of path requirements that could be requested by the clients are:

- *notviaN*: a set of domains must not be part of the multi-domain path.
- *notviaE*: a set of inter-domain links must not be part of the multi-domain path.
- *viaN*: a set of domains must be part of the multi-domain path.
- *viaE*: a set of inter-domain links must be part of the multi-domain path.
- *orderN*: a set of domains must be in a predefined sequence in the multi-domain path.

Requirements *notviaN* and *notviaE* can be fulfilled by pruning out all of the forbidden domains and inter-domain links before creating the topology graph that would be used for path computation. The resultant topology graph $G = (\mathcal{N}, \mathcal{L})$ would be the input to our problem, where G is the multi-domain network after the forbidden domains and inter-domain links have been pruned, $\mathcal{N}$ is the set of domains

and $\mathcal{L}$ is the set of inter-domain links. Requirements *viaN*, *viaE* and *orderN* need to be considered during the path computation process. The client can also request any combination of the listed path requirements. The requirements also relate to each other, as shown in the following equations.

$$orderN \subseteq viaN \quad (1)$$

$$viaE \subseteq viaN \quad (2)$$

Eq. 1 implies that all domains that are included in *orderN* must also be part of *viaN*. Eq. 2 implies that all domains with at least one adjacent inter-domain links in *viaE* must also be part of *viaN*. We define the Selective Path Finding (SPF) problem:

Selective Path Finding (SPF) problem - Given a graph $G = (\mathcal{N}, \mathcal{L})$ consisting of a set $\mathcal{N}$ of N domains, a set $\mathcal{L}$ of l inter-domain links, and a set $\mathcal{R} = \{notviaN, notviaE, viaN, viaE, orderN\}$ of R requirements. A specific domain from the set $\mathcal{N}$ is denoted by n, and a specific inter-domain link from the set $\mathcal{L}$ between domains u and v is denoted by (u, v). Find a simple path P from a source domain s to a destination domain d satisfying $\mathcal{R}$, such that the total path hop count is minimized.

We limit to simple multi-domain path since using only simple multi-domain paths, instead of allowing loops, would lower the signaling complexity between domains in providing the lightpaths between them. Requirements *notviaN* and *notviaE* play a minor role in the complexity of the SPF problem, since if only *notviaN* and *notviaE* is given, the forbidden domains and inter-domain links can be pruned from the graph and a run of a conventional shortest path algorithm such as Dijkstra's algorithm [**?**] is sufficient to find an optimum path. The SPF problem is NP-hard, when any of the size of *viaN*, *viaE* or *orderN* is more or equal to two, and can be proven via a reduction to the NP-hard Hamiltonian Path problem [**?**] when $|viaN| = N$.

8.2 Routing Algorithm

We focus on solving the SPF problem via an exact algorithm which we refer to as the SPF algorithm that is based on the k-shortest paths approach [**?**], where k is not predetermined but incremented adaptively. Before the SPF algorithm is run, we conduct a preprocessing phase that eliminates domains in requirement *notviaN* and inter-domain links in *notviaE* from the network graph G. The pseudocode of the algorithm is given in Algorithm 1.

During the path finding process, each domain n will keep at most k paths, with each path denoted by P_{nk}. A set of paths maintained by a domain n is denoted by P_n. Each P_{nk} is paired with its path length (in hops) D_{nk} and last visited domain n whenever it is inserted into (or extracted from) the priority queue Q.

The algorithm first initializes a path consisting of just the source domain, P_{s1} and adds it into P_s in line 1. The path length (in hops) of P_{s1} is set to zero. P_{s1} is then inserted into Q in line 2. While Q contains at least a path, the path with the lowest path length, P_{uk} is extracted from Q in line 4. If domain u is the destination domain d and all of the requirements in $\mathcal{R}$ are satisfied by P_{uk}, P_{uk} is returned as a solution to the problem in line 7. If domain u is not the destination domain d, or the requirements in $\mathcal{R}$ are not satisfied, each adjacent domain v of domain u is checked for possible path extension P_{vk} in lines 8-10. Line 11 ensures that the extended path P_{vk} is simple, i.e. it does not repeat any vertices. Line 12 ensures that the sequence of P_{vk} obeys $\mathcal{R}$. Line 12 implements the *Look-Ahead* routine, which will be explained in subsection 8.2.1. Lines 14-16 implements the *Path Comparison* routine,

Algorithm 1 SPF$(G, s, d, \mathcal{R})$

```
1: P_s1 = [s], D_s1 = 0, add P_s1 into P_s
2: INSERT(Q, s, P_s1, D_s1)
3: while Q ≠ φ
4:     (u, P_uk, D_uk) ← EXTRACT-MIN(Q)
5:     if P_uk ∈ P_u
6:         if u = d and R is fulfilled
7:             return P_uk
8:         if u ≠ d
9:             for each v adjacent to u
10:                P_vk = P_uk + v
11:                if P_vk is simple
12:                    if P_vk fulfills orderN
13:                        if all domains in viaN is still reachable
14:                            if P_vk is superior to any P ∈ P_v
15:                                remove all the inferior P
16:                            if no P ∈ P_v is superior to P_vk
17:                                add P_vk into P_v
18:                                D_vk = D_uk + 1
19:                                INSERT(Q, v, P_vk, D_vk)
```

which will be explained in subsection 8.2.2. If the extended path P_{vk} is valid, its corresponding path length D_{vk} is updated in line 18, and P_{vk} is inserted into Q. Though we use hop count as the path finding metric in this paper, the SPF algorithm is also usable with the inter-domain length as the path finding metric, provided that the *Path Comparison* routine is omitted.

Without the *Look-Ahead* and *Path Comparison* routines, the algorithm is essentially a brute-force approach on solving the SPF problem. Since a brute-force approach would take a very long time to compute the path when the network size is large or when the number of requirements is large, we introduce the *Look-Ahead* and *Path Comparison* routines that significantly reduce the computation time of finding the feasible multi-domain path with minimum hop count.

8.2.1 Look-Ahead routine

The *Look-Ahead* routine checks if all the domains in *viaN* that have not been visited by the path extension P_{vk} are still reachable. We only consider *viaN* since all of the domains included in the requirements *viaE* and *orderN* are also part of *viaN*. As an initialization step to the *Look-Ahead* routine, we make a copy of the original graph G as G_2, and a copy of *viaN*, as *viaN2*. We then prune all domains that are part of P_{vk} (except v) from G_2 and *viaN2*. Next, we run a breadth-first-search (BFS) algorithm [**?**] from v to all remaining domains in G_2, to get the shortest path tree T. If *viaN2* is a subset of T, then all domains in *viaN* that have not been visited by P_{vk} are still reachable from domain v. If the *Look-Ahead* routine decides that any further path extension from P_{vk} will never lead to a path that fulfils R, P_{vk} will not be considered.

8.2.2 Path Comparison Routine

The *Path Comparison* routine compares the candidate multi-domain path P_{vk} with all existing candidate paths in P_v, and eliminates paths that are inferior to at least another path. A path P_1 is superior to another path P_2, if P_1 is a subset of P_2, and all the domains in P_2 that are not part of P_1 are not in *viaN*, and without these domains, $P_1 = P_2$. For example, any two domains u and v that is directly connected by an inter-domain link (u, v) in P_1, but instead connected through several intermediate domains that does not consist of any domains from *viaN* in P_2, would imply that P_2 is inferior to P_1.

If P_{vk} is superior to any existing path(s) maintained in the list of P_v, the inferior paths are removed since they will not lead to a more optimal path than P_{vk}. Even though they may still exist in Q, once they are extracted from Q, they will not be processed due to line 5. On the other hand, if P_{vk} is inferior to at least a path maintained in P_v, P_{vk} will not be added into P_v since a more superior path already exist.

8.2.3 Time Complexity

The initialization phase in lines 1-2 has a time complexity of $O(1)$. The extraction procedure in line 4 takes at most $O(k_{\max}N\log(k_{\max}N))$ time, since Q contains at most $k_{\max}N$ paths. The path validation procedure in lines 6-7 takes at most $O(N + R)$ time. The for loop in line 9 takes at most $O(k_{\max}E)$ time, since it is invoked at most $k_{\max}$ times for each side of each inter-domain link. Checking if a path is simple in line 11 takes at most $O(N)$ time. Checking if *orderN* is satisfied in line 12 takes at most $O(N + R)$ time. The *Look-Ahead* routine in line 13 takes at most $O(N+E+R)$ time. The *Path Comparison* routine in lines 13-16 takes at most $O(k_{\max}NR)$ time. Summing up all the contributions, the worst-case complexity of the algorithm is $O(k_{\max}N\log(k_{\max}N)+ k_{\max}E^2 + k_{\max}^2NER$. Since $k_{\max}$ can grow exponentially with the input, the algorithm has an exponential running time. However, a polynomial-time tuneable heuristic can easily be derived by fixing k (e.g., see [**?**]).

9 Topology exchange as part of NSI

This topology exchange supports distribution of NML documents so this can work with the NSI framework. Instead of domain names we can use the network identifiers used within the NSI community.

Although the topology exchange can be run as a stand alone service, the three components of the topology exchange can be integrated into existing NSAs. In this case support needs to be added to the NSI API. The components can be announced as extra capabilities of an NSA.

9.1 Key distribution

Key distribution is problematic, since many developments within NSI did not focus too much on the security aspects giving no facility to distribute information with end-to-end security.

X.509 certificates are already being exchanged between peering NSAs. So all of the certificates should now also be exchanged with the NSA that acts as an index. This is manual labour. After this every NSA can get all certificates since they can be requested from the index.

An other option would be to drop key distribution or make it optional in the initial implementation.

The discovery service is not the place to exchange keys because documents can be forged easily by intermediate NSAs when the documents are not signed.

9.2 Bootstrapping

The X.509 certificates need to be shared with the index. The most simple way is to let everyone peer with the NSA running this service.

Another way is to distribute information on who acts as index is to use the Discovery Service.

9.3 Resolving STP's

Since we should not derive structure from a STP it's not easy to see what network/topology a STP belongs to. To do this one would have to fetch all topology files and do an exhaustive search on the STP.

Solutions to this problem are to extract the domain name from the STP by doing string manipulation, or to create a lookup service that is keeps a topology database and facilitate the searching on the STP.

9.4 Services

The three components of this topology exchange need to be implemented in the NSI framework. Here we describe the minimum set of functions for our components.

9.4.1 Topology Index (TI)

getIndex() provides the full index to the requesting domain

updateIndex(domain, data) updates the data in the index for the given domain. This will only be done if verified.

subscribeIndex(domain, callback) allows a consumer to receive/subscribe to the index for topology update, when the index is updated the index will send a message to the callback url.

9.4.2 Topology Provider (TP)

getTopology() this will return a default topology.

getTopology(requester) the topology returned will be customized depending on the requesting consumer; in other words this call will account for policies between domains. In most cases we expect that a default topology will be returned.

getTopology(global_route) global_route is a route based on summary information on how it wants to route; based on this a TP can see through which peer the information comes in and apply a peer policy based on the agreement between peers. This means that in this call we do apply potentially two types of policies: domain policies (as in getTopology) and peer policies.

9.4.3 Topology consumer (TC)

Optional:

update() will be triggered by the Topology Index when a new index is available. In order to receive updates the consumer needs to describe with this function as callback.

Pathfinder
In case of the path finder extra function calls are exposed:

The path request could be in the form of

getPath (A **to** B)

getPath (A **to** B **via** $< C, D >$)

getPath (A **to** B **via** $< (E, F) >$)

getPath (A **to** B **notvia** $< C, D >$)

getPath (A **to** B **notvia** $< (E, F) >$)

or any combination of them.

10 Future work

Open issue to be solved in the next stages are:

- index replication methods need to be researched more in depth;
- more in-depth discussion needed within the NSI community to see if we can pass messages on the control plane instead of directly peering and discuss how we can combine topology providers with aggregators;
- more research is needed on how to use the foreign domains information, For now it can be used to hint the requester;
- inclusion of dynamic information, i.e. actual bandwidth used/available on links at times of request.