A semantic model for complex computer networks: the network description language

van der Ham, J.J.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 6

Emulations of Aggregated Network Topologies

6.1 Introduction

In the previous chapter we have introduced pathfinding using aggregated topologies. We have discussed several studies that have examined the impact of aggregation on pathfinding. Unfortunately, the combined results of these studies are not conclusive about the effects. Two of the studies conflict on the performance of the Star aggregation method. These studies have also been performed in the context of ATM networks.

In this chapter we examine how topology aggregation can be applied to optical networks. We define how different aggregated views can be created in NDL from full topology descriptions. The Full Mesh, Star and Single Node aggregations can automatically be generated from a full NDL topology description.

To accurately determine the effect of the different aggregation strategies we have implemented an emulation experiment. In this experiment we generate random inter-domain topologies and do pathfinding on them. We compare the results of the same pair-sets for the different aggregation strategies and the control case, using the full topology. We also compare our results with the earlier results of topology aggregation in ATM networks.

This chapter continues by examining different aggregation methods in the context of NDL, and defines how these can be created from full NDL descriptions.
in section 6.2. The experimental setup is discussed in section 6.3 and section 6.4 shows the results of the experiments. In section 6.5 we finish this chapter with a discussion of the results.

6.2 Aggregation Methods

As we have shown before, topologies of network domains can be aggregated to a different degree. In our experiments we examine three different aggregation strategies, for which we provide the definitions here:

**Full View** This is the control case, where domains use no aggregation at all and publish their full topologies,

**Full Mesh** Domains only publish their edge nodes, the connections between the edge nodes are described using a full mesh,

**Star** Domains only publish their edge nodes, the connections between the edge nodes are described using a virtual central node,

**Simple Node** The domain is presented as a single node, leaving out all information about the internal network.

We examine the performance of pathfinding in the different aggregation methods with the following assumptions:

**Single Layer** We assume a single layered network. A good understanding of the impact of aggregation in single-layer topologies might later be of use also in multi-layer networks, where path finding is an even more complex problem [70].

**Homogeneous Capacity** We assume a single unit of bandwidth for all links in the network. We do allow for multiple connections between nodes, so it is possible to create a mapping for networks with heterogeneous bandwidths.

**Infinite Reservations** Another assumption is that a reservation is for an infinite amount of time. We generate random topologies, and do multiple passes of the same topology. This means that the total averaged results show the trend of how the network performs under increasing load.
Aggregated Metric The metric of a link is often used to represent the distance of that link. On our topologies the metric for all links is 1, even for links in the aggregated graph which can be based on much longer paths. This is used to minimize the information about the intra-domain connectivity, but also to maximize the distortion caused by the aggregation.

Inter-Domain Pairs In our experiments we only use source-destination pairs that are in different domains. Since nodes always have full knowledge of their own domain, aggregation will not have an effect on intra-domain pathfinding.

Unlabelled Links Previously we described the study by Liu et al. who have researched the effect of aggregation on WDM networks with different numbers of available wavelengths. We are interested in the general effect of aggregation on inter-domain pathfinding. In this study we use unlabelled links.

In the descriptions below we do use a mapping of network topologies onto graphs. In this case this is possible because we only use a single layer topology, and for the aggregation methods it is not necessary to identify different interfaces. The actual aggregated descriptions are created in NDL, complete with interface objects.

6.2.1 Formal Definitions of Topology Aggregation

In the previous chapter we have described what different aggregated topologies look like. However, we have not defined explicitly how to map a full topology to an aggregated version. Below we formally define how we have implemented the aggregation strategies in our experiments.

First some notation and definitions. We define a multi-domain topology as $T = (N, D, L)$:

$N$ is a set of nodes: $N = \{n_1, n_2, \ldots\}$

$D$ is a set of domains: $D = \{d_1, d_2, \ldots, d_x\}$ such that:

- Domains are sets of nodes: $\forall d_i \in D : d_i \subset N$
- All nodes must be part of a domain: $\bigcup_{i=1}^{x} d_i = N$
- Nodes can only be in one domain: $\forall d_i, d_j \in D : d_i \neq d_j \rightarrow d_i \cap d_j = \emptyset$
\( L \) is a set of links: \( L \subseteq N \times N \), we assume bi-directional connections:
\[
\forall n_i, n_j : (n_i, n_j) \in L \Rightarrow (n_j, n_i) \in L.
\]

A **Full Mesh aggregation** \( T_f = (N_f, D_f, L_f) \) of a topology \( T \) is defined as follows:

- \( N_f \) contains only domain-edge nodes:
  \[
  \forall n_i, n_j \in N_f \exists d_x, d_y \in D : (n_i, n_j) \in L \land n_i \in d_x \land n_j \in d_y \land d_x \neq d_y \Rightarrow n_i, n_j \in N_f,
  \]

- \( D_f \) is a set of domains \( D_f = \{d'_1, d'_2, \ldots, d'_x\} \) such that:
  \[
  \forall d'_i \in D_f \forall n_x \in d'_i : n_x \in N_f \land \exists d_i \in D \land d'_i \subseteq d_i
  \]

- \( L_f \) is constructed in two steps, first we take all the original inter-domain links:
  \[
  \forall n_i, n_j \in N_f \exists d_x, d_y \in D : (n_i, n_j) \in L \land n_i \in d_x \land n_j \in d_y \land d_x \neq d_y \Rightarrow (n_i, n_j) \in L_f,
  \]
  and secondly we add intra-domain connections based on the original connectivity. Even if multiple paths are available between different edge nodes, we only add a single link:
  \[
  (n_i, n_j) \in L_f \text{ if there is a path } n_i n_x \ldots n_y n_j \text{ with } n_x, \ldots, n_y \in d \text{ and } (n_i, n_x), \ldots, (n_y, n_j) \in L.
  \]

A **Star aggregation** \( T_s = (N_s, D_s, L_s) \) of a topology \( T \) is defined as:

- \( N_s \) contains domain-edge nodes, plus a centre node for each domain:
  \[
  \forall n_i, n_j \in N_s \exists d_x, d_y \in D : (n_i, n_j) \in L \land n_i \in d_x \land n_j \in d_y \land d_x \neq d_y \Rightarrow n_i, n_j \in N_s
  \]
  For each of the domains, we add exactly one centre node:
  \[
  \forall d \in D \exists ! n_d : n_d \in N_s \land n_d \notin N,
  \]

- \( D_s \) is a set of domains \( D_s = \{d'_1, d'_2, \ldots, d'_x\} \) such that:
  \[
  \forall n_i \in N_s \exists d_x \in D \exists d'_x \in D_s : n_i \in d'_x \Rightarrow n_i \in d'_x \land d'_x \in D_s
  \]

- \( L_s \) is constructed in two steps, first we take all the original inter-domain links:
  \[
  \forall n_i, n_j \in N_s \exists d_x, d_y \in D : (n_i, n_j) \in L \land n_i \in d_x \land n_j \in d_y \land d_x \neq d_y \Rightarrow (n_i, n_j) \in L_s,
  \]
  secondly, all the edge nodes are connected to the centre node, if they have available intra-domain connections:
  \[
  \forall n_i \in N_s \exists d_x \in D : n_i \in d_x \land \exists n_k \in d_x \exists n_d_x \in N_s : (n_i, n_k) \in L \Rightarrow (n_i, n_d_x) \in L_s.
  \]

A **Simple Node aggregation** \( T_n = (N_n, D_n, L_n) \) of a topology \( T \) is defined as:
\( N_n \) contains only one node per domain: \[ \forall_{i=1}^x n_i \in N_n \text{ with } x = |D| \]

\( D_n \) contains the same amount of domains, but each only has a single element:
\[ \forall d_i \in D \exists d'_i \in D_n \exists n_i \in N_n : d'_i = n_i \text{ and } \bigcup_{i=1}^x d'_i = N_n \text{ with } x = |D_n| \]

\( L_n \) contains only the inter-domain links:
\[ \forall (n_i, n_j) \in L \exists d_x, d_y \in D \exists n_x, n_y \in N_n : n_i \in d_x \land n_j \in d_y \land d_x \neq d_y \rightarrow (n_x, n_y) \in L_n \]

### 6.2.2 Topology Aggregation from NDL Descriptions

We can use these logical definitions to create different aggregated views from a full NDL domain topology description. Below we describe how we select and transform the NDL objects to create the aggregated views.

For the **Full Mesh** we take the device objects that have an inter-domain connection, including their switching matrix. We then describe only the interfaces of those devices that have inter-domain connections. For each device we check whether it is possible to reach the other inter-domain devices through the domain. If so, we create a new interface object for each, with a `connectedTo` statement between them, and we also connect them through the switching matrix. The result is a topology description containing only the edge devices, with their edge interfaces, and an aggregated description of the intra-domain topology through the virtual interfaces.

Initially in an unused network, all edge nodes will be able to reach each other, since domains are usually connected graphs. When more and more links are reserved it is possible that domains become disconnected graphs. The aggregation method Full Mesh, and Star can describe this, Full Mesh without updating will not.

In the **Star** aggregation we also start with the edge devices. We describe only the interfaces of those devices that have inter-domain connections. We add a new virtual device to the aggregated graph for the centre of the domain, the nucleus. For each edge device we check whether it has any intra-domain interfaces with available bandwidth. If it does, we add a connection between the device, and the centre node. This results in a topology description containing only the edge nodes and a (virtual) intra-domain star network.

Also in the Star network, edge nodes will initially be able to reach the intra-domain network. Once the network usage grows, it is possible that an edge node will no longer have available internal connectivity. This is reflected in the Star aggregation, the node then becomes disconnected from the rest of the domain.
This node can still be of service by switching from one of its inter-domain connections to the other.

The Simple Node topology is simpler to create: We take the network domain object with the name of the domain, \( n_d \) in the mathematical description, and we add a switching matrix object to the network domain object. Then from the nodes with inter-domain connections we select the interface objects of those inter-domain connections. We add these interface objects directly to the domain object. We also copy the inter-domain \( connectedTo \) properties of the original view into the aggregated view, and connect all the interfaces to the switching matrix. The result is a network domain object with interfaces, which are connected to other domains, and internally connected with a full mesh through the switching matrix.

This Simple Node topology does not publish any details about the intra-domain connectivity. Other domains reading this topology will assume that the intra-domain network is fully connected. Once the network usage increases, it is possible that the intra-domain network is no longer connected. However, there is no way to publish this using the Single Node strategy.

### 6.3 Experimental Setup

We have implemented an experimental setup for testing the different aggregation methods. The steps in the experiment are as follows:

- Take a number \( d \) for the number domains, and \( n \) for the number of nodes in each of the domains,

- Generate a random graph \( G \) with \( d \) domains, and \( n \) nodes,

- Create a randomly ordered list \( l \) of inter-domain endpoint pairs,

- For each of the aggregation methods:
  - Create a simulated network of the graph \( G \),
  - For each pair \((x, y)\) in \( l \), find a path between \( x \) and \( y \) in the aggregated view:
    * If there is no path, continue.
    * If there is a path:
      - Translate the path in the aggregated view to a full path,
If the full path is available, reserve the path and record the result.
If the full path is not available, record a false positive.

In the rest of this section we describe the details of our experimental setup. First we describe how we generate the network graphs and the source and destination pairs, and in section 6.3.2 we describe the pathfinding in aggregated views, and how we translate these paths back to the underlying graph.

6.3.1 Generating the Graphs and Pairs

At the start of each experiment run we generate a random graph and pair-lists. In the experiment we can use different graph sizes to test the impact of the number of domains, and the number of nodes per domain.

There are a large number of ways to generate random graphs. Barabási and Albert have shown in 1999[95] that many complex networks exhibit a scale-free property. That is, the probability $P(k)$ that a vertex in the network interacts with $k$ other vertices decays as a power law, following $P(k) \sim k^{-\gamma}$. The value of $\gamma$ varies with different types of graphs, but is usually $2 \leq \gamma \leq 3$. In case of the BGP router network, the value of $\gamma \sim 2.3$.

However, it should be noted that it is unclear whether current optical networks, such as the GLIF network, are scale-free. The current networks are too small and nodes have too small degrees to come to a definitive conclusion[96]. We expect that as optical networks grow larger, they will also follow the power-law distribution.

The graphs are generated using the Barabási-Albert algorithm as implemented in the NetworkX Python module[97]. This algorithm will generate graphs with a value of $\gamma$ that tend to $2.9 \pm 0.1$. Using this algorithm we generate $d$ graphs with their $n$ nodes. We then generate another graph with the same algorithm with $d$ nodes, and use that as the inter-domain graph, randomly picking nodes from each of the domain graphs to provide the inter-domain links. Analysis of the results of this generation method shows that $\gamma$ averages to $\sim 2.3$, which is comparable to the BGP network.

In our experiments we then enumerate all the pairs $(x, y)$ where $x$ and $y$ are in different domains. So in total we have $(d \cdot n) \cdot ((d - 1) \cdot n) \cdot \frac{1}{2}$ number of pairs. And before each run we shuffle this pair-list.

The graph is then converted to an emulated network using our pynt package. The result of the conversion is the same if the descriptions of the domains would have been imported from NDL descriptions or from OSPF data.
6.3.2 Pathfinding Using Aggregations

We perform four different pathfinding operations using the same shuffled set of endpoints pairs on different views of the same graph. We use a full view, a Full Mesh aggregation with and without updates, a Star aggregation, and a Simple Node aggregation.

As a baseline we perform pathfinding on the graph using complete information of the graph, that is, we work with the full graph. This shows the ideal situation which the aggregation methods should approximate. As with all the other aggregation methods, we perform pathfinding using a standard Dijkstra’s shortest path algorithm.

In the Full Mesh method the topology shows only the boundary nodes. The intra-domain connections are created by taking each boundary node and performing a path find call on it to all other boundary node of that domain. If there is a path between that node and another boundary node, then we add a connection between those two nodes. The inter-domain connections are not aggregated, and used as is. The Full Mesh view is recreated before each pathfinding attempt to reflect the current status of the network.

For each pathfinding attempt on pair \((x, y)\), we add \(x\) and \(y\) to the aggregated graph. To connect them we iterate over the boundary nodes of their respective domains, and if a path can be found to that boundary node, then we add a connection in the graph.

With the nodes \(x\) and \(y\) added to the Full Mesh graph, we try to find an inter-domain path between them. If a path is found in the Full Mesh aggregated graph we convert each path through a domain by mapping it back to the underlying path in the full topology. Before moving on the next pair, the source and destination nodes \(x\) and \(y\), are removed from the graph, if they are not edge-nodes.

We also perform the experiment with the same initial Full Mesh view of the topology, but without any intermediate updates. The updates in this case refer to the updating of intra-domain topology after a path has been found. Without the updates it is possible that a path is found in the aggregated graph where the underlying path is no longer available. In this case we record the result as a false-positive.

In the Star aggregation method we also keep the boundary nodes, and link them to a virtual device, the nucleus. The links from a boundary node to the nucleus is created if the boundary node has any intra-domain connectivity available. After each successful reservation we check the intra-domain connectivity again and perform updates where needed.
The last aggregation method is to collapse domains into a *Simple Node* in the aggregated graph. We have implemented this by creating an aggregated view of the graph with only domains. The inter-domain interfaces of the boundary nodes are added to aggregated graph, along with their connections. As noted before, the Simple Node view does not provide details about the intra-domain network. Once the network usage increases, it is possible that the domain becomes a disconnected graph. If a path request in the aggregated view traverses such a domain and cannot be translated to an underlying path, we record the result as a false-positive.

For each pathfinding attempt on pair \((x, y)\) in the Star and the Simple Node aggregation we add \(x\) and \(y\) to the aggregated graph with a connection to their centre and domain nodes respectively. If a path is found between \(x\) and \(y\) we convert that path into a full path by replacing each domain hop in the path with the result of a path find operation between the boundary interfaces of that domain. If the complete path can be converted it is reserved. If no path can be found within one of the domains, we record a false-positive. Before moving on to the next pair, the nodes \(x\) and \(y\) are removed from the graph.

Note that the Full Mesh without update is very similar to the Simple Node method. The Full Mesh without update shows the edge nodes, with a full mesh between them. The Simple Node graph only shows a single point, implicitly assuming full connectivity between all its inter-domain interfaces.

Note also that regardless of what kind of intra-domain updating occurs, all of the aggregation methods do update the inter-domain connectivity.

### 6.4 Results of the Emulations

In each experiment run, when a path is found, we record the result (success or false-positive), length of that path, and the new resource usage of the network. From each run we have a large set of results (\(10^4\) data points per aggregation method).

Since we are using results from experiments with different graph sizes, we normalize the results by using a relative index on the pair number. So the first pair has a relative index of 1 divided by the total number of pairs. This puts all results on a scale between 0 and 1. The resource usage numbers have also been normalized to their total graph sizes.

We analyse our results using the *R* statistical analysis tool[98]. We use the coefficient of determination (\(R^2\)) to determine the goodness of fit of our fit functions. This is calculated using the formula given in equation 6.1 where
$SS_{\text{errors}}$ and $SS_{\text{total}}$ are the sum of squares of the errors and the total sum of squares. $y_i$ and $\hat{y}_i$ are respectively the observed and predicted value, and $\bar{y}$ is the mean of the observed values. $R^2$ indicates the explained variance of a model, and measures how well future outcomes are likely predicted by the model. The result is a number between 0 and 1, where a value of 1 means that the model explains all the variability of the data.

$$R^2 = 1 - \frac{SS_{\text{errors}}}{SS_{\text{total}}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$  \hspace{1cm} (6.1)

### 6.4.1 Fit Functions

Before we examine the performance of the different aggregation strategies, we first inspect the behaviour of our control case, pathfinding using the complete view. The length of the paths will behave differently as the network is gradually filled up. Figure 6.1 shows a scatterplot of the path length development. We can see that initially it is fairly stable, the network is still empty, almost any path will succeed, and paths will start at the average path length in that network, and gradually increase. As the network starts to fill up, most requests will still succeed, but the path length peaks as longer and longer detours are taken. This grows until the network becomes nearly saturated, meaning that large parts of the network are in use. Then only small disconnected parts of the network remain available, and the chance of success depends on the distance in the network. The path lengths will gradually decrease to the minimum, i.e. paths between neighbours.

To show the combined effects of the path length and the success rate we plot the development of inter-domain resource usage over the successive requests, see figure 6.2. There are two different behaviours, the initial increase in path lengths, and the slow decrease once the network reaches a saturation point. Therefore we use two different functions for fitting to the results. We determine the split by the path length peak, the peak and everything before is the first part, and afterwards is the second part.

Initially there is a constant success rate, and the path length increases linearly, and the inter-domain usage also increases linearly. We use a linear function starting at zero to fit with, as shown in equation 6.2. This fit explains roughly 90% of the variance, and is shown in figure 6.3.

$$\text{InterDomainUsage} = A \cdot \text{RelativeIndex}$$  \hspace{1cm} (6.2)
RESULTS OF THE EMULATIONS

Figure 6.1: A scatterplot showing the path length distribution in full view pathfinding

Figure 6.2: A scatterplot showing the two different phases in inter-domain resource usage distribution in full view pathfinding
In the second part of the plot the behaviour is dictated by a decreasing success rate, and path length. Initially we presumed that this behaviour could be predicted using a standard growth function towards an asymptote as shown in equation 6.3. In this case the asymptote is a completely filled network where the inter-domain usage is 1.

\[
\text{Inter Domain Usage} = 1 - A \cdot e^{(-B \cdot \text{Relative Index})}
\]  

(6.3)

The fitted function shows a reasonable result, explaining about 87% of the variance. However, if we overlay this function to the scatterplot of the results, it clearly shows a different trend than the actual results. The trend shows more of a logarithmic form, so we have also fit the function given in equation 6.4. This fit shows a much better result, explaining over 94% of the variance.

\[
\text{Inter Domain Usage} = A \cdot \log(\text{Relative Index}) + B
\]  

(6.4)

Both fits are shown on the scatterplot in figure 6.4, with the exponential fit in red, and the logarithmic fit in green. The only caveat with the second fit is that it over-predicts when the network has been nearly filled. However, the paths in that final section will all be very short ones, regardless of the aggregation strategy. We expect to see most significant differences in the first part of this second section.

### 6.4.2 Domain Sizes

In our experiments we have also examined the effect of domain sizes on the results. In figure 6.5 we show the initial fits for \((d = 50, n = 5)\) and \((d = 50, n = 50)\), in figure 6.6 the fits for the second section. Note that the horizontal scale is different for each of the plots. The fitted values \((A, B)\), the error \((\sigma)\), and the explained variance \((R^2)\) are shown in tables 6.1 and 6.2.

Both graphs show a very similar performance of the aggregation methods. However, the performance difference is larger in the configurations with less nodes per domain. Part of this difference can be explained by the different ratio of the number of pairs to inter-domain resources. However this would only cause the graphs to shift on the scale, keeping the relative difference. The smaller difference between the fitted lines with \((d = 50, n = 50)\) can only be explained by the increase in intra-domain resources. Since we are interested in the performance difference of the different aggregation strategies, we will use small domain sizes.
Figure 6.3: A scatterplot showing only the initial linear growth section, along with the fitted function in full view pathfinding.

Figure 6.4: A scatterplot of the inter-domain resource usage distribution in full view pathfinding, along with the exponential fit in red, and the logarithmic fit in green.
Figure 6.5: Plotted fits for the initial growth phase for \((d = 50, n = 5)\) and \((d = 50, n = 50)\)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>(A)</th>
<th>(\sigma)</th>
<th>(R^2)</th>
<th>Strategy</th>
<th>(A)</th>
<th>(\sigma)</th>
<th>(R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>930</td>
<td>3.6</td>
<td>0.97</td>
<td>A</td>
<td>88026</td>
<td>308</td>
<td>0.98</td>
</tr>
<tr>
<td>Full Mesh</td>
<td>896</td>
<td>3.3</td>
<td>0.97</td>
<td>Full Mesh</td>
<td>89184</td>
<td>332</td>
<td>0.98</td>
</tr>
<tr>
<td>FM no updates</td>
<td>497</td>
<td>3.0</td>
<td>0.92</td>
<td>FM no updates</td>
<td>84374</td>
<td>324</td>
<td>0.97</td>
</tr>
<tr>
<td>Star</td>
<td>544</td>
<td>3.2</td>
<td>0.93</td>
<td>Star</td>
<td>86646</td>
<td>314</td>
<td>0.98</td>
</tr>
<tr>
<td>Single Node</td>
<td>518</td>
<td>3.3</td>
<td>0.92</td>
<td>Single Node</td>
<td>84715</td>
<td>315</td>
<td>0.98</td>
</tr>
</tbody>
</table>

(a) \((d = 50, n = 5)\)  
(b) \((d = 50, n = 50)\)

Table 6.1: Fitted values in the initial growth phase
### RESULTS OF THE EMULATIONS

<table>
<thead>
<tr>
<th>Strategy</th>
<th>variable</th>
<th>value</th>
<th>σ</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>A</td>
<td>0.1015</td>
<td>0.0004</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.1727</td>
<td>0.0027</td>
<td></td>
</tr>
<tr>
<td>Full Mesh</td>
<td>A</td>
<td>0.1016</td>
<td>0.0004</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.1594</td>
<td>0.0024</td>
<td></td>
</tr>
<tr>
<td>FM no updates</td>
<td>A</td>
<td>0.1058</td>
<td>0.0003</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.1535</td>
<td>0.0019</td>
<td></td>
</tr>
<tr>
<td>Star</td>
<td>A</td>
<td>0.1059</td>
<td>0.0004</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.1561</td>
<td>0.0026</td>
<td></td>
</tr>
<tr>
<td>Single Node</td>
<td>A</td>
<td>0.1053</td>
<td>0.0003</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.1386</td>
<td>0.0017</td>
<td></td>
</tr>
</tbody>
</table>

(a) \(d = 50, n = 5\)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>variable</th>
<th>value</th>
<th>σ</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>A</td>
<td>0.1236</td>
<td>0.0008</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.9416</td>
<td>0.0095</td>
<td></td>
</tr>
<tr>
<td>Full Mesh</td>
<td>A</td>
<td>0.1223</td>
<td>0.0008</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.9196</td>
<td>0.0094</td>
<td></td>
</tr>
<tr>
<td>FM no updates</td>
<td>A</td>
<td>0.1220</td>
<td>0.0008</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.913</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>Star</td>
<td>A</td>
<td>0.1280</td>
<td>0.0008</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.9853</td>
<td>0.0092</td>
<td></td>
</tr>
<tr>
<td>Single Node</td>
<td>A</td>
<td>0.1211</td>
<td>0.0008</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1.9005</td>
<td>0.0094</td>
<td></td>
</tr>
</tbody>
</table>

(b) \(d = 50, n = 50\)

**Table 6.2:** Fitted values in the logarithmic growth phase
6.4.3 Results on Inter-Domain Pathfinding

With the fit functions selected, we can examine the performance of inter-domain pathfinding in the different aggregation strategies. In figure 6.7 we show the fits for the results on the graphs with \((d = 150, n = 5)\). Table 6.3 shows the fitted values, along with their errors, and the coefficient of determination.
Figure 6.7: Fits of initial and logarithmic growth phases for \((d = 150, n = 5)\)
Table 6.3: Fitted values in the initial and logarithmic growth phases for \((d = 150, n = 5)\)
RESULTS OF THE EMULATIONS

Initial Linear Growth Phase  The graph of the first section in figure (a) shows the initial growth. Recall that the boundary for the first section is determined by the peak in the path lengths. The fitted graphs all end around 60% resource usage, however the more aggregated the longer it takes to get there. The results in table (a) also show that the Full Mesh aggregation strategy performs very close to the Full View, and that there is a large gap in performance with the other aggregation strategies. Of these the Star aggregation performs better than the last two, the Full Mesh without updates, and Single node, which both show very similar performance.

Logarithmic Growth Phase  The same difference in performance continues in the second section of the fit. The Full Mesh aggregation performs almost perfectly compared to the control. However, here the difference with the other strategies is far less pronounced. The Star aggregation method shows a significantly lower performance, with the Full Mesh without updates just below it. The Single Node aggregation method clearly performs worst of all the aggregation methods.

Figure 6.8: The fraction of false positives in the linear and logarithmic phases for \((d = 150, n = 5)\)

False Positives  In figure 6.8 we show the fraction of false positives in a bar-plot. It should be noted that these false positives only occur in the logarithmic growth phase. None of the aggregation methods show false positives in their linear growth phase.

It is not possible to have false positives with the full view, nor with the
Full Mesh view with updates. In the latter case, the update mechanism always makes the graph reflect the current availability in the network. The number of false positives is extremely high in the aggregation strategies without detailed intra-domain connectivity in the logarithmic growth phase. When the Single Node is used, only 1 in 10 attempts will result in an actual path. Even with the Star aggregation, which shows some intra-domain details, over 75% of the attempts is a false positive result. This clearly shows that once the network resources becomes less and less available, detailed knowledge of intra-domain connectivity is required in order to provide accurate results for inter-domain pathfinding.

**False Negatives** We have also examined the false negatives in the aggregation strategies. To examine this, we have slightly adjusted our experiments. At first we perform a normal run on the full graph, but now we record which pairs form a successful path. Then we use only these pairs in the aggregated views of the graph. Figure 6.9 shows the results, the diagonal describes perfect performance. The difference between the diagonal and each of the aggregation methods is the number of false negatives. In this case the plots were created by using non-linear least squares fitting with the following function:

\[
\#\text{SuccessfulPaths} = A \cdot \text{index} + B \cdot \text{index}^2
\]  

(6.5)

The graph in figure 6.9 shows that initially the Full Mesh strategy shows very little false negatives. This number increases as more and more paths are reserved, even up to the point where it is finally overtaken by the other aggregation strategies.

In figure 6.10 we show a box-plot of the path lengths for the aggregation strategies with only the success pairs. There we can see that the paths in Full Mesh are significantly longer than in the other strategies, and they are also slightly longer than in the control. These longer paths take up more resources, and therefore make it less likely that successive paths will succeed. In the other strategies the successful paths are somewhat shorter. The initial longer paths fail, keeping more resources available so that the later shorter paths will succeed.

### 6.5 Discussion and Conclusion

In this chapter we have examined what kind of impact aggregation has on the performance of inter-domain pathfinding. We have described different aggregation methods, and performed experiments to test and quantify this impact.
Figure 6.9: The fitted functions describing the sum of successful path requests

Figure 6.10: A box plot for the lengths of only the paths that succeeded in the full view
Table 6.4: Fitted values and their error for false negatives in \((d = 150, n = 5)\)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>variable</th>
<th>value</th>
<th>(\sigma)</th>
<th>(R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Mesh</td>
<td>A</td>
<td>0.934</td>
<td>0.004</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.00233</td>
<td>0.00005</td>
<td></td>
</tr>
<tr>
<td>FM no updates</td>
<td>A</td>
<td>0.603</td>
<td>0.004</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.00134</td>
<td>0.00004</td>
<td></td>
</tr>
<tr>
<td>Star</td>
<td>A</td>
<td>0.652</td>
<td>0.004</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.00057</td>
<td>0.00005</td>
<td></td>
</tr>
<tr>
<td>Single Node</td>
<td>A</td>
<td>0.517</td>
<td>0.004</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0.00191</td>
<td>0.00004</td>
<td></td>
</tr>
</tbody>
</table>

Our analysis in the previous section clearly show that aggregation does indeed have an impact on the performance of inter-domain pathfinding. In the initial linear growth phase the Full Mesh aggregation strategy performs close to the Full View. The other aggregation strategies perform significantly worse.

This difference in performance becomes much smaller in the logarithmic growth phase, where the growth of inter-domain resource usage in all aggregation strategies is very similar. However in this phase the number of false positives in the aggregation methods without accurate intra-domain connectivity is extremely high. This means that once the network becomes reasonably filled, these aggregation strategies become almost unusable without a way of filtering out these false positives.

A possible way of using the information from false positives is by using crank-backs. This method of locally updating the view of the topology using false-positive information ultimately creates a similar view on the graph as the Full Mesh method does. The difference is that with crank-back the majority of the effort of creating the updated graph lies with the requester. The resulting graph is also not shared with the other domains. The load of pathfinding is then shifted from the domains, performing less updates, to the source, which has to perform the crank-backs. We have seen from the false-positive results that clients will very often have to perform these crank-backs.

An argument that is often used in favour of aggregation is scaling: finding paths in large detailed graphs takes more time than finding a path in an aggregated graph. This argument fails to take the cost of constructing and updating the aggregated graph into account. In the case of the Full Mesh graph with updates, the cost of maintaining the graph is distributed over all the domains. The total distributed processing time is then higher than finding a path in a
full detailed graph. In the Star, Simple Node, or Full Mesh without it is less hard to maintain the aggregated topologies, but these views show a very large number of false-positives. To get a reasonable performance in these strategies, crank-backs will have to be used, which will be very time consuming.

It is somewhat difficult to compare our results to the results of Awerbuch et al. since they use crank-backs. However, the general performance trends in their results are similar to ours. Full Mesh (‘Complete’ in their terminology) performs best, while both their Star aggregation methods show a slightly worse performance.

Comparing our results to the results of Guo and Matta, we see a significant difference in the performance of the Star aggregation. In their study the Star performs almost equally with the Full Mesh aggregation, while both in Awerbuch et al. and our results the Star aggregation performs significantly worse. Unfortunately, we cannot reproduce their results, even when using their topology we see a significant difference between the Full Mesh and Star aggregations. Our results are shown in figure 6.11 and fitted values are in table 6.5. The high inter-domain connectivity in their topology makes the network saturation occur very late in our results, this is why we only use the linear fit.
Figure 6.11: The performance of the different aggregation methods on the topology of Guo and Matta.

Table 6.5: Fitted values and their error for the topology of Guo and Matta