

# Efficient Policy Conflict Analysis for Autonomic Network Management

Steven Davy, Brendan Jennings  
Waterford Institute of Technology, Ireland  
{sdavy,bjennings}@tssg.org

John Strassner  
Motorola Labs, Schaumburg, IL, USA  
john.strassner@motorola.com

## Abstract

*Autonomic network management strives to reduce the complexity associated to managing large scale communications networks. Policy based management is a critical facilitator for this vision and more importantly policy conflict analysis processes must be efficient and scalable to cope with the dynamicity and size of such communications networks. We present an efficient policy selection process for policy conflict analysis that maintains a history of previous policy comparisons in a tree based data structure to reduce the number comparisons required in subsequent iterations. The ability to incorporate historical information into the selection process stems from the two phase approach we take in our conflict analysis algorithm. The first phase of the algorithm initialises a relationship pattern matrix between a candidate policy and a deployed policy, the second phase matches this pattern against a conflict signature. Previous solutions compare candidate policies against all deployed policies sequentially, however our approach can reuse the patterns already discovered from previous iterations of the algorithm to reduce the number of comparisons. Experimental results presented here show that significant performance improvements can be made using this approach, however the degree of this improvement is dependent on the nature of the relationships between deployed policies.*

## 1. Introduction

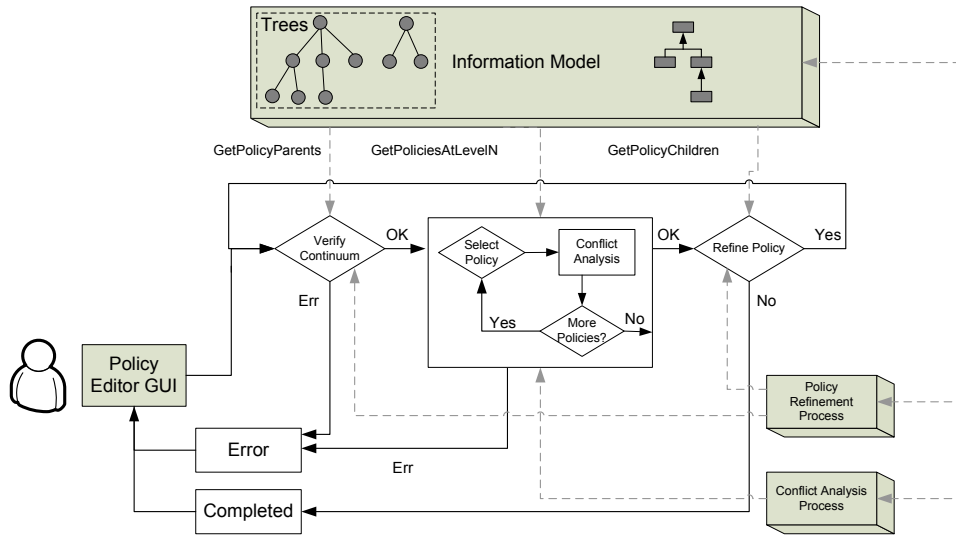
As communications networks become larger in size, less reliable, more heterogeneous and more dynamic, new management paradigms are required. Autonomic network management [8] seeks to automate and distribute the decision making processes involved in optimising network operations. This goal is to allow expensive human attention focus more on business logic and less on low-level device configuration processes. Policy-based network management systems are widely seen as an appropriate facilitator to fulfill the requirement of obeying business logic. Incorporating policy into an autonomic management system requires the

development of effective and extensible algorithms and processes for policy translation, code generation, conflict analysis and policy enforcement. In this paper we focus on enhancing the performance of processes that require the automated detection of potential conflicts between policies.

A policy conflict is defined to occur when a set of policies are simultaneously satisfied; that is, an event has triggered the evaluation of a set of policies whose conditions have subsequently been satisfied, but the combined actions of the policies will result in the system reaching different final states depending on the order of execution of these actions. Policy conflicts can easily occur, especially in circumstances when there are multiple authors defining policies for a given system.

In previous work [6] we defined a generic and extensible policy conflict analysis algorithm that is independent of the application domain to which the policies relate. The algorithm is split into two phases. In the first phase the algorithm uses the information model to identify relationships between a candidate policy (either a newly created policy or a modified policy) and a deployed policy. In the second phase these relationships are examined in the context of an application specific conflict pattern extracted from the information model. This algorithm is defined as an integral part of an encompassing policy authoring process. We assumed the presence of a system information model (typically specified using the UML) which models the structure and relationships between the system managed entities and the policies that effect their management. Given the presence of such an information model, and making minimal assumptions regarding the nature of the information model (for example, that policies are specified as event-condition-action tuples) we defined a generic policy conflict analysis algorithm.

The policy authoring process currently passes policies in an exhaustive pairwise manner into the conflict analysis algorithm. It is this pairwise dependency that we intend to eliminate by making intelligent selections as to which policies must be analysed at each iteration of the algorithm by using a history of comparisons. We use some basic assumption about what we can infer from previous iterations of the



**Figure 1. Policy Authoring Process**

algorithm to reduce the number of comparisons and therefore reduce runtime complexity. For example, if we discover that the subject component of a candidate policy is identical to that of a deployed policy, then we can infer that the candidate policy is also related via subject to all policies that the deployed policy is related to via subject, therefore eliminating the need for the associated comparisons. In this paper we present an enhancement to the policy analysis process by improving the way we select policies to be input into the analysis algorithm by using multiple tree data structures to maintain a history of comparisons.

The paper is structured as follows. Section 2 describes the policy authoring process of which policy conflict analysis is an integral part. It also outlines the two-phase conflict analysis algorithm, discussing how it integrates with a system information model. Section 3 discusses the strategy we take to provide an intelligent policy selection approach and thus improve the performance of the algorithm, we also describing the tree data structure and modification process. Section 4 describes the experiments devised to test the sensitivity of the algorithm to distribution of node sizes in the tree and number of nodes versus number of policies and how these parameters affect the performance of the algorithm. Section 5 discusses related work on policy conflict analysis. Finally, section 6 offers conclusions to the paper and outlines topics for future work.

## 2. Policy Authoring Process

The policy conflict analysis algorithm is an integral part of an encompassing policy authoring process. The authoring process, as outlined in previous work [5], is an interactive process that guides a policy author through a set of steps that revolve around policy analysis, transformation and refinement. The end result of the authoring process is a successfully authored policy that is consistent with existing deployed policies in respect to the constraints imposed by the target management system and the combined effect of existing deployed policies. The authoring process as depicted in figure 1 manages the selection process of policies that are input into the conflict analysis algorithm. Currently this process of selecting policies is not intelligent and iterates over all deployed policies to ensure that the new or modified policy does not cause a potential conflict.

From the perspective of this work, the model we assume policy is based on is rich enough to be applicable to the majority of policy applications. It follows an Event-Condition-Action rule format, that is related to specific Subjects and Targets. This model coupled with application specific information included in the information model can be readily adapted to suit many popular policy application domains, including firewall filtering policies and access control policies as described in [6].

As stated earlier the algorithm we have devised for policy conflict analysis is split into two phases [6]. The first phase of the algorithm is concerned with compiling a relationship pattern matrix describing the different relationships that can exist between the components of two policies, the

---

**Algorithm 1** PolicyRelationship of Phase 1

---

**PolicyRelationship** :  $(Policy \times Policy \times PolicyMap)$

$\rightarrow Matrix$

**PolicyRelationship**  $(p_1, p_2, pm) \hat{=}$

$m = \mathbf{ZeroMatrix}$

$m = \mathbf{associateBySubject} (p_1, p_2, pm) \circ$

$\mathbf{associateByTarget} (p_1, p_2, pm) \circ$

$\mathbf{associateByEvent} (p_1, p_2, pm) \circ$

$\mathbf{associateByCondition} (p_1, p_2, pm) \circ$

$\mathbf{associateByAction} (p_1, p_2, pm) m$

---

candidate policy and the deployed policy. The second phase of the algorithm examines the relationships established in the first phase in tandem with a set of conflict signatures that indicate a potential occurrence of conflict. The conflict signatures are stored within the information model of the target managed system and are therefore application specific. The conflict signatures being used change depending on which specific application domain the policies currently analysed policies are defined. We now expand on the description of each phase of the algorithm and discuss the need for improvements in runtime complexity.

## 2.1. Phase 1 : Relationship Analysis

The algorithm initially creates a matrix that encodes the various types of relationships between a pair of input policies, namely the candidate policy and the deployed policy. This matrix represents a *policy relationship pattern*. As described later, different relationship patterns may constitute different forms of potential policy conflict, depending on the particular semantics of policy enforcement in the application domain in question. Construction of the relationship matrix is done by the *PolicyRelationship* function, outlined in algorithm 1, which examines the two policies in order to discover whether specific relationships exist between them.

An initially zeroed matrix is modified by a set of operations that populate the matrix with ones depending on their result. For example, the *associateBySubject* operation takes as input the two policies (denoted  $p1$  and  $p2$ ) and tests for a set of relationships that exist among their subjects. There are initially four tests to compare each policy via subject, where the tests examine the subject of the policies for subset, superset, equality or correlation (intersection). For example, if the test for subset relation returns true, 1s are placed in the input matrix at the positions that signify that the subject of  $p1$  is a subset of the subject of  $p2$ , and inversely that the subject of  $p2$  is a superset of the subject of  $p1$ . Similarly, the same operation can be performed for testing for subject superset membership, subject equality and subject correlation. After the relationship matrix has been

populated with information associated to subjects, the targets of policies can also be related in the same manner.

The computation required to determine whether the subjects of one policy are a subset of the subjects of another policy requires specific information about the application domain over which the policy is defined. We can make use of some well defined interfaces to a system information model to simplify the computation of these functions, such as an operation that can ascertain the type hierarchy of an object. Events, conditions and actions can be related in a similar manner. For example, when the events of one policy are computed to be a superset of the events of another policy, then we are sure that when the first policy is triggered, the second policy is also triggered. Similarly, logical inference among conditions is similar to the subset, superset, and equality relationships. Therefore, when we relate policies via the condition component, we can begin to draw conclusions as to which policies infer the condition components of other policies.

Relating action components can reveal information about what actions may be repeated by other policies and therefore may be redundant. There are additional ways of relating these components. The *associatedByAction* operation may check if there are relationships between actions that signify that the simultaneous execution of the two policies may yield contradicting states for the managed system, this type of relationships between policies is a good indication of some forms of policy conflict. However, even if the actions of two policies contradict, we may still not assume that the two policies conflict because the application domain defines what is and is not deemed to be a conflict among policies. The reader is referred to [6] for more in depth description of action contradiction detection. The next phase examines the established relationship matrix against application specific conflict signatures to establish if a potential conflict exists for the a specific application domain.

## 2.2. Phase 2 : Conflict Signature Matching

The manner in which policies are analyzed for potential conflict is highly dependent on the application domain of those policies. For example, when determining if two access control policies conflict, there must be an overlap among the subjects, targets and actions. Therefore, we can associate policies by subject, target and action as a first step and those policies that can be seen to overlap after this step can be flagged to signify potential conflicts. On the other hand, for filtering policies (such as firewall rules) we may not be interested in subject, target, action overlap, but instead in event, condition, action overlap, in particular only when the target entities are identical. Clearly the pattern of relationships between policies give a clear indication of the potential for conflict; however different relationships are

$$\begin{bmatrix} ssb & ssp & seq & scor & 0 \\ tsb & tsp & teq & tcor & 0 \\ esb & esp & eeq & ecor & emux \\ csb & csp & ceq & ccor & cmux \\ asb & asp & aeq & acor & actd \end{bmatrix}$$

**Figure 2. Conflict Signature Matrix**

relevant in different application contexts.

In phase 2 of the algorithm a matrix containing the application domain specific conflict signature (extracted from the information model) is used to make the decision whether to flag the policies as potentially conflicting. We use an operation that matches the relationship matrix produced in phase 1 with a conflict signature that represents the set of relationships that may or must hold for conflict. Figure 2 depicts a sample conflict signature that is examined in this phase. Each row represents the relationships of a component of policy. To describe a condition for conflict, a 1 is placed in the associated position in the conflict signature matrix. For example, if a specific type of conflict requires subject subset membership (ssb), then a 1 would be placed in the upper left position of the matrix.

The codes used in Figure 2 represent the associated relationships considered so far. The first letter of the codes indicates which component is being analysed, s for subject, t for target, and similarly for event, condition and action. The tail of the codes indicate the type of relationship being established, sb for subset, sp for super set, eq for equal, cor for correlation, mux for mutually exclusive and ctd for contradict. It is important to note that the list of relationship types given in the matrix can be expanded upon. Until now, we considered a set of popular relationship types among policies, but as there are certainly more ways to relate two policies we have decided to leave the algorithm extensible. More importantly, both the conflict signature and the policy relationship pattern can be increased in the number of columns and rows so that new ways of relating policies can be developed in the future.

### 3. Enhanced Policy Selection

The policy authoring process as discussed so far implements a pairwise comparison of the candidate policy against each deployed policy to test for potential conflict when using the conflict analysis algorithm. For very large communications networks with a high volume of policies this approach may cause potentially significant efficiency issues as it is  $\mathcal{O}(n)$ ,  $n$  being the number of currently deployed policies. This section discusses how we improve the efficiency of selecting policies to input into the algorithm by reusing the results of previous iterations of the algorithm to

reduce the number of comparisons required in future iterations. The enhancement is therefore part of the authoring process that uses the algorithm to detect for potential conflict. The approach we take is to associate deployed policies to multiple tree data structures that maintain the relationships among the components of policies. We now describe the tree data structure that is used to enhance the selection of policies, this is followed by experimental analysis to illustrate the potential reduction in runtime complexity.

We leverage previous iteration by maintaining a set of tree data structures for each relation type, to which policies can be added. The emerging behaviour is that policies that are similar in respect to a particular relationship and thus a particular tree, are grouped into nodes of that tree that are close together, as they related to other policies in a similar fashion. Therefore, if a candidate policy is deemed similar to a group of existing policies, it simply copies their existing relationships with other deployed policies and eliminates the need for the associated number of comparisons.

$$\begin{aligned} t \in Tree = Node &\rightarrow \mathbb{P}Node \\ n \in NodeDetails = Node &\rightarrow \mathbb{P}Policy \end{aligned} \quad (1)$$

We now define the tree data structure that is used to store a specific relationship type, however the structure is identical for each type of relationship. The tree data structure as defined in (1) illustrates that a tree is made up of a node map, where each node is mapped to a set of nodes. We can also see that a node can be mapped to a set of policies. The tree data structure aids in establishing relationship patterns among policies for phase 1 of the algorithm. The relationships ascertained in phase 1 of the algorithm are equality, superset, subset and correlation for each policy component type at a minimum. Each tree can be defined to efficiently store equality, subset and superset and to a degree correlation. Equality, subset and super set relationships are transitive, therefore if we compare a candidate policy to a deployed policy and establish that a relationship holds for one of their components, then we can infer that the candidate policy is also related to those policies that the deployed policy is related to. As more policies are added to the policy repository they begin to form groups around common managed entities in regards to subject and target trees, and common event, conditions and actions for the other trees respectively. Each tree is kept unbalanced, with the nodes sorted in order of size. The type of tree we have implemented is more commonly known as a lopsided or unbalanced tree, where the branches of the tree are always ordered by size. This way policies are compared to the most popular policies first to maximise the elimination of comparisons required.

We define an algorithm for policy insertion in algorithm 2 that adds a policy to one of the trees and maintains the tree structure appropriately. The algorithm takes as input an existing tree and a candidate policy and outputs the new tree.

---

**Algorithm 2** Add a Policy to the Tree

---

 $\text{AddPolicyToTree} : (\text{Policy} \rightarrow \text{Tree}) \rightarrow \text{Tree}$  $\text{AddPolicyToTree}(p) t \triangleq$  $\forall n \in (I \rightarrow \pi^1) t :$ 

```
if ( $p = \text{getFirst}(n)$ )
  AddPolicyToNode( $p$ )  $n$ 
elseif ( $p \subset \text{getFirst}(n)$ )
  AddPolicyToNodeSubset( $p$ )  $n$ 
elseif ( $p \supset \text{getFirst}(n)$ )
   $n_e = \text{newNode}(p) t$ 
  AddNodeSubset( $n_e, n$ )
  DeleteNodeFromTree( $n$ )  $t$ 
```

```
if ( $\text{noPlaceFound}$ )
```

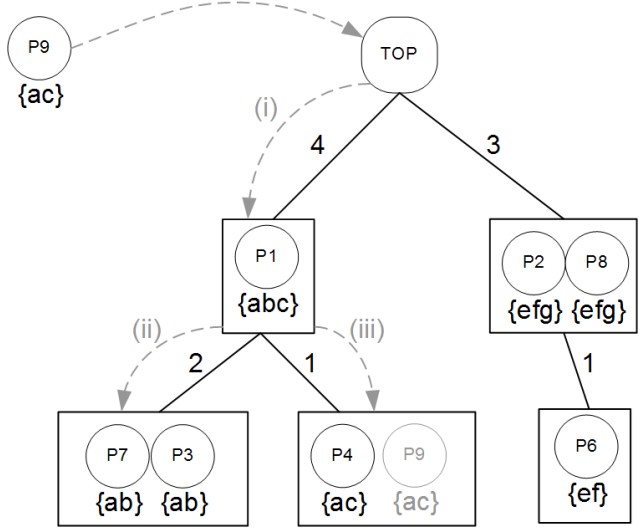
```
   $n_e = \text{newNode}(p) t$ 
```

---

For each tree, the candidate policy is compared against the top level nodes of the tree, where each node contains a list of policies related via, for example, equality. If the candidate policy matches the node via equality, it is merged into that tree at that node and the algorithm finishes. If the candidate policy is a sub-set of the current node, then the policy is recursively added to the child nodes. If the candidate policy is a super set of the current node, then a new node is created for the candidate policy, and the sub-set node is added to the new node as a child node. If the candidate policy is not related to any existing node, then a new node is created and the candidate policy is added to it.

Each tree grows as policies are added, subsequently the least specific policies are arranged at the top of the trees, and the most specific are arranged at the leaf edges of the trees. A count is maintained for each node representing the total number of policies associated to that node including the count of its child nodes. The nodes at the same level are sorted so that the node with the highest count is compared against first as it is associated to the biggest number of policies.

To illustrate the concept we have depicted a simple insertion operation (see figure 3) for a single tree, we represent the contents of a policy as a set of letters. Therefore policies that reference the same set of letters can be grouped together. Also we can represent subset and superset relationships. By organising the policies in to a tree structure where the top node is a superset of all policies, we can group the policies into nodes of the tree. From this tree we can derive relationships among policies such as, policies located in nodes at a higher degree of the same branch (closest to the root) are a superset of the policies contained in lower nodes. Policies located at the same node reference equivalent entities. Establishing the order of policies can be carried out per component, therefore based on different policy compo-



**Figure 3. Example Policy Insertion**

nents, they may be related differently to other policies.

At step (i) the candidate policy is compared against the node with the highest count at the top level, the node containing P1. A subset relationship is established and the policy is recursively added to that nodes child nodes. We can now assume that the policy is not related to any other branch of the tree at the top level and thus eliminate the need to compare against policies P2, P8 and P6. At step (ii) the candidate policy is compared against the largest node which contains P7 and P3, however as we have previously established that P7 and P3 are equivalent, we need only to compare against a single policy in the group, thus eliminating another comparison. We establish that the candidate policy is not related to that node, so we continue to search. The next node we compare against, step (iii), is of size one and contains P4. Subsequently we establish that the candidate policy is equivalent to this policy and it is inserted into this node. Policies that exist in the tree within other branches are thus pruned from the search.

## 4. Experimental Analysis

To test the performance of the enhancement to policy selection for the conflict analysis algorithm we devised a set of experiments that tests the process's sensitivity to distribution of node sizes, and its sensitivity to the ratio of policies to nodes. To reduce the number of variables measured, the experiments were carried out to only compare the equality of policy subjects between the candidate policy and deployed policies, and therefore only a single tree was generated for this relationship. However, if more relationships were to be included this would have the effect of further reducing the number of comparisons, as the node sizes would

**Table 1. Distributions of Policies per Subject**

Trail	Distribution
1	1, 1, 1, 1, 2, 5, 30, 215
2	2, 3, 5, 8, 16, 31, 63, 128
3	4, 6, 9, 14, 23, 38, 61, 101
4	12, 15, 19, 24, 31, 39, 50, 66
5	32, 32, 32, 32, 32, 32, 32, 32

increase as more policies become associated and more trees are created. The efficiency of the process depends on the number of nodes in the resulting policy tree, therefore it is independent of how a policy tree creates a node. These assumptions do not affect the accuracy of the results that are outlined throughout the rest of this section.

#### 4.1. Node Size Distribution Sensitivity

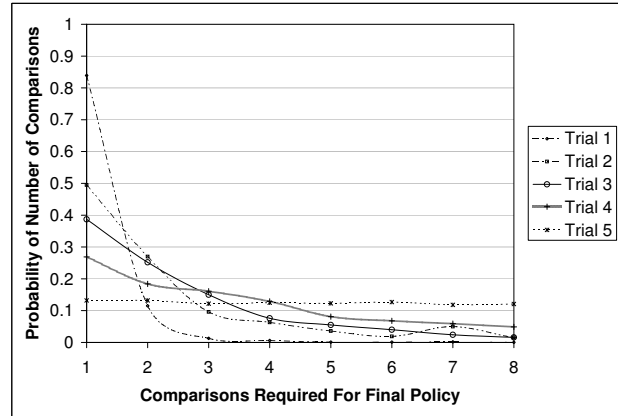
Policy repositories can have very different characteristics when it comes to how much we can leverage the existing patterns. We devised an experiment to demonstrate the applicability of the process to repositories of different types. For this experiment we assume that there are a fixed number of policies, and a fixed number of nodes within the tree data structure. The distribution of policies among the nodes are varied to simulate the popularity of some managed entities over others. A node is established if one or more policies reference a subject, we say that the policies are grouped via subject equality. There are in total 256 policies and 8 subjects, therefore there is a maximum of 8 nodes in the tree. The distribution of policies per node was generated using an exponential distribution with varying weights to give distributions of different gradients. We generated five distributions to represent the varying nature of policy repositories. The distribution of policies per subject is outlined in table 1.

The experiment consisted of generating a random sequence of policies, based on a predetermined seed, to be added to the tree. When the final policy (the candidate policy) is added to the tree, i.e. policy number 256, we counted the total number of comparisons required until that policy had established relationships with all existing deployed policies. A pair wise comparison would take 255 comparisons. We performed the test for each set of policies, and it was performed 1000 times where each iteration had a different random seed. The average number of comparisons for our approach along with the reduction in computational complexity is outline in table 2, the distribution of the number of comparisons is depicted in figure 4.

From figure 4, we observed that the distribution of policies per tree node affected the performance of the policy selection process. The behaviour observed is directly related to the nature of each repository type. This demonstrates

**Table 2. Average number of comparisons**

Trail	Average	Percent Reduction
1.1	1.295	99.49%
1.2	2.032	99.23%
1.3	2.456	99.04%
1.4	3.254	98.72%
1.5	4.666	98.17%

**Figure 4. Distribution of Comparisons**

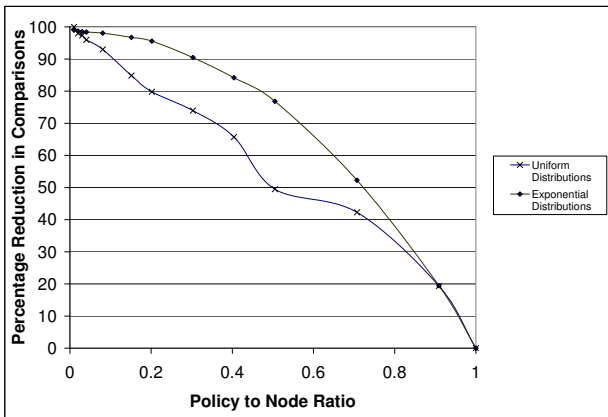
why we decided to keep the tree sorted in order of node count (or node size) after each policy insertion, thereby comparing against the node that covers the most amount of policies. The probability of the candidate policy being compared against a group that is most related to it is increased and subsequently increases the probability of it being compared against fewer nodes. We also observe that the distribution with the least amount of variability (trial 1.5) has roughly an equal probability of matching against any node as the final policy is added. Notice that the number of comparisons is not bounded by the number of established nodes as opposed to the number of policies. We also observe that the number of comparisons is of the order of the number of nodes in the tree, never raising above eight. The experiment assumes that the addition of new candidate policies follow the same probability distribution as defined by the policy repository they are being added too. The next experiment tests the sensitivity of the algorithm to node to policy ratio.

#### 4.2. Number of Nodes Sensitivity

For some policy repositories it may not be common to make equality relationships among most policies, therefore this experiment demonstrates the sensitivity of the policy selection process to the number of nodes produced by the tree to reflect the equality of policy subjects. For this experi-

**Table 3. Number of Nodes and Distribution of Policies**

Trial	Nodes	Average Comparisons (Exp)	Average Comparisons (Uni)	Exp % Reduction	Uni % Reduction
2.1	1	1	1	99.0%	99.0%
2.2	2	1.296	2	98.7%	98.0%
2.3	3	1.466	2.623	98.5%	97.4%
2.4	4	1.586	4	98.4	96.0%
2.5	8	1.914	8	98.0	93.0%
2.6	15	3.216	15	96.8	85.0%
2.7	20	4.407	20	95.6	80%
2.8	30	9.464	25.796	90.5	74.2%
2.9	40	15.688	33.93	84.3	66.0%
2.10	50	22.927	50	77.1	50%
2.11	70	47.241	57.128	52.8	42.9%
2.12	90	79.756	79.839	20.2	20.1%
2.13	99	99	99	0.0%	0.0%

**Figure 5. Node to Policy Ration**

ment we assume that there are 100 policies and 100 subjects that policies can reference, however for each trial not all subjects are referenced. The number of subjects referenced determines the number of nodes in the tree because we are only considering equality for the moment. We performed the experiment with exponentially distributed policies per subject and uniformly distributed policies per subject because from the previous experiment we observed that the node distribution impacts the performance of the process. The results for the experiment are outlined in table 3.

We observed that the number of nodes produced per 100 policies directly affects the performance of the policy selection process. The results are depicted graphically in figure 5. We can see the difference the distribution of node size has on the process as it is designed to leverage any commonalities among policies. Therefore the performance of the exponential distributions sustains better performance than the

uniform distributions.

Most notably, for trial 2.1 one subject was referenced in common to all policies. When the last policy was added, it too referenced the the same subject as all currently deployed policies and so it required only a single comparison to establish its association to all deployed policies. This case represents the best performance case for the algorithm, specifically when there is only a single node generated to cover all policies.

In contrast, for trial 2.13 each deployed policy referenced a distinct subject. When the last policy was added, it too referenced a distinct subject, therefore it would have to be compared against each deployed policy to establish a relationship pattern. This case represents the worst case performance of the algorithm, specifically when the number of nodes generated approaches the total number of policies in the repository.

## 5. Related Work

Policy conflict analysis has been a highly active research domain for the past 10-15 years due the potential advantages it offers policy based management systems. Most popular policy languages address some form of policy conflict in one way or another where the work is primarily based in the original research of Sloman and Lupu documented in [10]. Later research on detecting policy conflict for the Ponder policy language by Charalambides et. al [3][4] is based on defining conflict rules, that are separated from the algorithm for detecting conflict, unfortunately the policy repository is exhaustively search to determine is there exists policies that breach the conflict rule. In contrast the approach we present performs an efficient preprocessing of policy to eliminate the requirement of exhaustively searching for conflicts. Policy analysis for the ponder

policy language has been more recently documented in [11] where they describe new forms of policy conflict, assuming that policies can be both subject based and target based. Their work focuses on automating policy conflict resolution, however their detection process is based on pair wise analysis of policy and is confined to only detect a restricted set of conflicts. Similarly the KAoS [12] and Rei [9] policy languages treat a limited set of pre-defined policy conflict types, and search for conflict on a pair wise policy basis.

Detecting conflicts among firewall policies is a special case of conflict analysis as the semantics of the policy languages and the rules that determine conflict have been researched thoroughly. Al-Shaer et. al [2] classifies the types of firewall policy conflict, where all forms of conflict they specify can be represented by different forms of conflict signatures in our algorithm. They differentiate forms of conflict by relating different components from the individual policy rules together, subsequently if a particular set of relationships hold, then a conflict occurs. Our approach is more generic as it can be applied to different policy models. Also we optimise the process further by maintaining an efficient data structure relating policies to each other.

Jajodia et. al [7] developed a formal access control model that enabled the coexistence of multiple access control policies in a single system and introduces features such as rule propagation and conflict resolution policies. The access control model we assume in this paper is considerably more simple than that proposed in [7], however as our approach is independent of policy model used a richer policy model may be incorporated into that information model. Also their conflict detection and resolution strategy is based on an exhaustive search of the policy repository per conflict type, similar to that proposed in [3] and suffers from the same complexity issues.

Policy ratification work by Agrawal et. al [1] investigated the problem of policy conflict from the perspective of the condition component associated to a policy. They implemented a set of algorithms to analyse policies for consistency, coverage and conflict. Their work can be used in coordination with the research presented in this paper as they are primarily focused on analysing the condition components of policy, subsequently it can be used to enhance the population of the relationship matrix we generate between policies.

## 6. Conclusions and Future Work

We presented an efficient policy selection process that uses multiplere tree data structures to maintain the history of policy relationships for each relationship type as the policy repository is amended with new or modified policies. The policy authoring process can leverage the tree data structure to introduce performance enhancements by re-

ducing the required number of comparisons between policies that are input into the conflict analysis algorithm. We demonstrated the potential benefits of using the approach in comparison to traditional pair-wise selection. A sensitivity analysis of node count, and node size distribution yielded a set of results illustrating the reduction in computational complexity the algorithm offers against a range of policy repository types. The experiments presented are designed to demonstrate the applicability of the approach to a range of policy repository types, where specific performance gains for particular policy repositories can be expected to lie between the extremes. Large scale communications networks must maximise performance in regards to management overheads. As autonomic network management paradigms for communications networks are increasingly policy based, policy conflict detection algorithms will continue to play a critical role.

Our future work will encompass implementing richer scenarios such as distributed firewall policies and access control policies and therefore demonstrating the flexibility of the algorithm. The algorithm is still limited to detecting only conflicts that can be represented as relationships among policies, however we intend to investigate the use of ontologies to augment the information model so that we can detect a wider amount of policy conflict types.

## References

- [1] D. Agrawal, J. Giles, K.-W. Lee, and J. Lobo. Policy ratification. *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005.*, pages 223–232, 2005.
- [2] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *Selected Areas in Communications, IEEE Journal on*, 23(10):2069–2084, 2005.
- [3] M. Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, and J. Rubio-Loyola. Policy conflict analysis for quality of service management. *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005.*, pages 99–108, 2005.
- [4] M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, and N. Dulay. Dynamic policy analysis and conflict resolution for diffserv quality of service management. *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 294–304, 2006.
- [5] S. Davy, B. Jennings, and J. Strassner. The policy continuum - a formal model. *2nd IEEE International Workshop on Modelling Autonomic Communications Environments, MACE 2007*, pages 65–79, October 2007.
- [6] S. Davy, B. Jennings, and J. Strassner. Application domain independent policy conflict analysis using information models. *accepted for publication in IEEE/IFIP Network Operations and Management Symposium (NOMS 08)*, 2008.

- [7] S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260, 2001.
- [8] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Bitvich, M. Foghlú, W. Donnelly, and J. Strassner. Towards autonomic management of communications networks. *IEEE Communications Magazine*, October 2007.
- [9] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74, 2003.
- [10] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 1999.
- [11] G. Russello, C. Dong, and N. Dulay. Authorisation and conflict resolution for hierarchical domains. In *Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on*, pages 201–210, 2007.
- [12] A. Uszok, J. M. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lot. Chaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. *Proceedings of Policy 2003. Como, Italy.*, 2003.