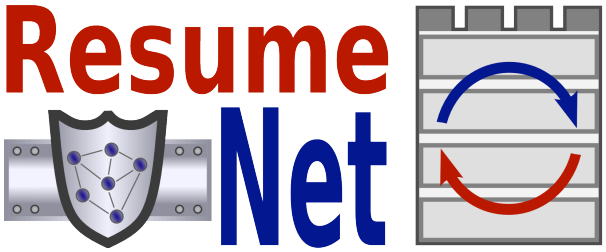
Resilience and Survivability for future networking: framework, mechanisms, and experimental evaluation



Deliverable number	1.3a
Deliverable name	Policies for resilience (Interim)
WP number	1
Delivery date	28/02/2010
Date of Preparation	18/3/2010
Editor	M. Schöller (NEC)
Contributor(s)	M. Schöller (NEC), P. Smith (ULanc), A. Schaeffer Filho (ULanc), N. Kheir (FT)
Internal reviewer	N Kammenhuber (TUM)

Summary

The ResumeNet project proposes a policy-based approach to network and service resilience. This deliverable presents an overview of the policy frameworks that we are considering for using as the basis for building a resilience architecture. Currently, three policy frameworks are investigated: XACML, Ponder2, and Or-BAC. For each framework we present the features that are usable for a resilience architecture. The assessments presented in this deliverable are used by various activities on network and service resilience, which are expected to provide further insights into their applicability to design and build a resilience framework on the presented policy frameworks. Therefore, this deliverable presents only preliminary results and will be updated at the end of the project.

Contents

1 Introduction

Network and service resilience of a communication system adds new features to its management and control functionality. The defensive measures of the system have to be adapted to the current system context. In case of a detected challenge, a different configuration of these measures might be required to fulfil the self-protection requirements. In addition, the challenge detection by highly specialized challenge detectors that allow a root cause analysis of ongoing challenges might be too costly to be operational at all times. They are only enabled after the detection of basic symptoms hinting at the challenge that causes the system to behave erroneously or to fail. This dynamic reconfiguration of the system is called *remediation* in ResumeNet's $D^2R^2 + DR$ strategy. The objective of the strategy is to allow the system to maintain an acceptable level of service during challenging conditions. After the detection of the challenge's cessation, the system should be brought back to its normal operational state. Therefore, the changes of the configuration have to be rolled back for both the defensive measures and the challenge detection mechanisms. This second reconfiguration is called *recovery* in the ResumeNet strategy. Both reconfigurations have to adapt the system to the current system context. Moreover, as we envision to use machine learning techniques to refine the system, the selection of a remediation strategy has to be changeable in a way that strategies that have been successful are more readily selected in the future than those which have not. ResumeNet proposes a policy-based approach to the selection of remediation and recovery mechanisms to assist with these issues.

Besides the steering of the remediation and recovery strategy selection, ResumeNet proposes to use policies to configure the resources and mechanisms of user systems. In a communication system run by a network operator, defensive measures and challenge detection mechanisms are provided during system deployment. The nodes are dimensioned in a way that the hardware satisfies the resource requirements of these mechanisms. In non-operator network scenarios, e.g., ad-hoc networks or community mesh networks, legacy systems are used to build the network. This implies that no dedicated resources for resilience are installed in the system, nor can a unified set of adaptation mechanisms be expected on the various network nodes. In order to support network and service resilience, each of these nodes has to advertise which resilience mechanisms it supports and what resources it allocates for their execution. Especially if adaptation mechanisms include the dynamic addition of new functionality to the communication protocols or the flexible introduction of completely new protocols into the protocol stack. How many resources are reserved for resilience on each device is configured by the owner of the device. Moreover, we assume that mobile devices will use different configurations for different environments they operate in. For example, a user is expected to configure his device to contribute more resources in his home network than in an ad-hoc network in a public setting, such as a festival. These different settings can be expressed via policies, creating such profiles.

In this deliverable, we document our assessment of the suitability of three policy frameworks to build a resilience architecture: XACML, Ponder2 and Or-BAC. Ponder2 [D$^+$01] is an obligation policy framework which has been designed to direct entities in a system. In addition, Ponder2 can support access control to the managed object it controls. In contrast, XACML [OAS09] and Or-BAC [KBB$^+$03] are access control policy frameworks to which obligation policy support is being added to. The objective of this deliverable is to inform resilience framework architects about the benefits of these three frameworks and to exemplify their use. We do this by providing a publication we currently prepare for submission on how to use Ponder2 to build resilience into an operator network. For XACML and OrBAC we provide a set

of different features which might prove useful for network resilience, where a combination of access control and obligation enforcement is required.

2 Policy Frameworks

In this section, we describe some of the features of three prominent policy-based management frameworks: XACML, Ponder2 and Or-BAC. We aim to show the features of the frameworks that could prove useful for resilience.

2.1 XACML

OASIS XACML [OAS09] (eXtensible Access Control Markup Language) has become a well recognized standard for the specification of access control policies as well as a generic framework for access control. The Policy Enforcement Point (PEP) intercepts any access request to the controlled resources and sends access requests to the Policy Decision Point (PDP). There, the request is evaluated and a decision derived whether the access to the resource should be granted or denied. Upon receiving the response from the PDP the PEP is responsible for enforcing the PDP's decision.

XACML uses XML [W3C06] to express the well-established ideas of access control policies. Thereby, it profits from XML's flexibility and extensibility. XACML defines three top-level elements: $< Rule >$, $< Policy >$, and $< PolicySet >$. The $< Rule >$ element contains a Boolean expression that can be evaluated in isolation. It is intended to form the basic unit of management, and be re-used in multiple policies. The $< Policy >$ element contains a set of $< Rule >$ elements and a specified procedure for combining the results of their evaluation. The $< PolicySet >$ element contains a set of $< Policy >$ or other $< PolicySet >$ elements and a specified procedure for combining the results of their evaluation. It is the standard means for combining separate policies into a single combined policy.

In addition to the specification and evaluation of access right to resources, extensions to XACML to support obligation policies are being investigated. In [Lis10], a proposal for dynamic obligation specification and negotiation is presented. There, XOML (eXtensible Obligation Markup Language) is introduced as an extension to XACML. It extends the XACML response message with a list of obligations, which may be sent back to the PEP. The PEP is responsible for enforcing these obligations. XOML, as a metalanguage, allows the description of the obligation method and their parameters. In addition, XOML allows for modelling potential conflicts of obligation methods either in a general way, or depending on the assignment of specific values to the parameters. This is achieved by providing relationships between obligations in the obligation specification (see Section 2.1.1 for details).

Extending XACML to support obligation policies raises the issue of which order access and obligation policies should be enforced. In [CSL08] a proposal for timing access control and obligation policies is proposed. This has been extended and incorporated in the XACML standard in [OAS07] where four modes are specified: *'obligation before access'* mode assures that if the access control decision is successfully enforced, then the obligation is also successfully enforced; *'obligation after access'* mode assures that if the obligation is successfully enforced the access control decision is also successfully enforced; *'obligation with access'* mode assures that the obligation is successfully enforced if, and only if, the access control decision is successfully enforced; *'any timing'* mode defines a best effort attempt to enforce both obligation

and access control decision.

A policy framework with obligation support in heterogeneous environments requires a negotiation method (as proposed in [Lis10]) to ensure compatibility of obligations during PDP and PEP deployment. The negotiation phase allows the PDP and PEP to ensure the compatibility of their required and supported obligations, respectively. In the case of a mismatch, resolution methods can be applied. More details on the negotiation of obligation definitions are presented in Section 2.1.1.

2.1.1 Relationships between obligations

Detecting and resolving conflicts has been well investigated in the access control policy domain. Adding obligation policies requires methods to detect potential conflicts in a set of obligations contained in the XACML response. These relationships between obligations have to be specified beforehand, either by an administrator or by deriving them from meta information on the obligations, which in the end also have to be provided by an administrator or system engineer. Two types of conflicts are currently distinguished: conflicts depending solely on the obligation method and conflicts on the actual parameter values in the XACML response. Regarding these types of conflicts lead to the definition of the following relationships:

independent: there is no relationship between two given obligations, thus their execution has no dependency.

conflict: two obligations are in opposition to each other (e.g., adapting the current network layer protocol configuration and substituting the network layer protocol).

contradiction: one obligation would annul the effect of the other, when executed in a specific order (e.g., increasing link layer information redundancy and re-routing flows over different links)

inclusion: the objective of one obligation is included in the other

subordinated, superordinated: if executed in a specific order, the two obligations could be executed in a meaningful way (e.g., substituting the network layer protocol and then adapting its configuration).

Based on our requirements analysis for network resilience, we propose to extend the provided list of relationships with two new obligation relationships:

alternative: the obligations feature the same purpose and a single one should be selected for enforcement (e.g., reduction of packet loss due to bit errors on a wireless link can be achieved by using different forward error correction codes (FEC) algorithms, e.g., Red-Salomon-Codes or Raptor codes).

require: the enforcement of an obligation is only meaningful if the required obligation is also successfully enforced (e.g., a challenge detection module requires sensor information to be available).

The conflict or contradiction relationships could be resolved by assigning priorities to the obligations. These priorities reflect the outcome of a risk assessment procedure, such as the one described in [SS09]. The three latter relationships can be resolved in a relatively straightforward

manner. In the case of an inclusion relationship, the included obligation becomes obsolete and is not executed. In the case of sub- and superordinated relationship, an order between the obligation is given, thus a valid and meaningful execution is possible. Finally, in the case of an alternative relationship, the PDP only selects one of these obligations and therefore alternative obligations must not appear in an XACML response message.

In addition, some conflicts can only be detected on the PEP, such as obligation conflicts due to the actual value of the parameters. It is important to note that there might be more than one relation between two obligations depending on the matching of the parameters. Three matching categories are introduced which help the detection of such conflicts:

direct: the obligations are directly related, independent of current parameter values.

partially: depending on some parameter values of the obligation matching with the values of the same parameter in another obligation.

full: depending on all parameters values of the obligation have to match with their corresponding parameter of the other obligation (e.g., two configuration parameter settings for the forward error correction code).

2.1.2 Negotiation of obligation definitions

Using obligations policies to drive remediation in large deployments has to deal with heterogeneous devices and their installed software. A result of this is that different devices support different sets of obligations. Therefore, a negotiation of supported obligations has to take place during PDP and PEP deployment. This negotiation process is also executed if policies are updated during system operation. [Lis10] proposes a three-way handshake for the negotiation process. Either the PDP or the PEP initiate the negotiation by sending a request message containing a list of all obligations. The peer analyses each obligation and indicates either that it supports the obligation, that it does not support the obligation, or that the obligation should be casted using a resolution function. The sender checks the reply message and finishes the negotiation by sending the third message. This message informs the peer that the supported obligations are accepted, that the unsupported obligations are ignored, that the cast is either accepted or rejected, or that the negotiation of a policy has failed.

2.1.3 Obligation subject

Many resilience mechanisms require the activation of functionality on multiple system components to successfully deploy an adaptation strategy. One way to achieve this is to integrate signalling functionality into the initialization procedure of the mechanisms composing the adaptation strategy. This, of course, is a bad design choice, as this signalling functionality is required by many different adaptation strategies. An alternative and better approach to this is to integrate the enforcement of obligations on many entities within the policy framework. Therefore, an *obligation subject* has to be introduced into the obligation extension of XACML. Currently, XOML [Lis10] is designed to execute obligations only on the PEP which has requested the policy decision from the PDP. In contrast, [CF09] proposes an independent obligation service which is responsible to enforce obligations at multiple locations in the system.

2.1.4 Example of obligation policy usage in XACML

The specification of XACML with its proposed support for obligation policies is an interesting option to build network resilience systems on. We present a detailed example of how to use XACML in a wireless mesh backhaul network in ResumeNet deliverable D2.3a [DSSK10]. Here, we illustrate the different features with examples based on the ResumeNet resilience architecture design.

```
1  <xoml:ObligationDefinition
       xmlns:xoml="http://www.example.org/xacml-obligation"
       internalID="packet_loss_sensor">
2    <xoml:obligationId>
3      urn:example:obligation:packet_loss_sensor
4    </xoml:obligationId>
5    <xoml:parameter name="monitoring_interval"
       datatype="urn:example:int" />
6  </xoml:ObligationDefinition>
```

Figure 1: XOML definition of a Packet Loss Sensor

The first example (see Figure 1) provides the XOML definition for the deployment of a sensor that can report the percentage of lost packets within a monitoring interval. The monitoring interval is the only parameter of this command. It defines in which interval the sensor provides its information to an information consumer. The actual length of the monitoring interval is deployment specific and will be set at run-time.

```
1  <xoml:ObligationDefinition
       xmlns:xoml="http://www.example.org/xacml-obligation"
       internalID="packet_loss_detector">
2    <xoml:obligationId>
3      urn:example:obligation:packet_loss_detector
4    </xoml:obligationId>
5    <xoml:parameter name="threshold "
       datatype="urn:example:float" />
6    <xoml:full>
7      <xoml:obligation>
8        urn:example:obligation:packet_loss_sensor
9      </xoml:obligation>
10     <xoml:relation>require</xoml:obligation>
11   </xoml:full>
12 </xoml:ObligationDefinition>
```

Figure 2: XOML definition of Packet Loss Detector

The second example (see Figure 2) shows the XOML specification of a packet loss detector. Again, the detector has a single configuration parameter. This parameter defines the threshold when the detector has to report an ongoing challenge. As the packet loss sensor is the only source for detector to get data from, we defined the relationship *requires* in order to make sure that the detector does not get deployed without the appropriate sensor.

The third example (see Figure 3) illustrates how potential conflicts on the parameter level

```
1  <xoml:ObligationDefinition
       xmlns:xoml="http://www.example.org/xacml-obligation"
       internalID="new_firewall_rule">
2   <xoml:obligationId>
3      urn:example:obligation:new_firewall_rule
4   </xoml:obligationId>
5   <xoml:parameter name="IPv4_quintuple"
        datatype="urn:example:ipv4_tuple" />
6   <xoml:parameter name="Firewall_verdict"
        datatype="urn:example:string" />
7   <xoml:partial>
8    <xoml:obligation>
9      urn:example:obligation:new_firewall_rule
10   </xoml:obligation>
11   <xoml:relation>conflict</xoml:relation>
12   <xoml:conflictingParameter>
13     Firewall_verdict
14   </xoml:conflictingParameter>
15  </xoml:partial>
16 </xoml:ObligationDefinition>
```

Figure 3: XOML definition of a remediation mechanism

can be modelled with XOML. The presented obligation is used to add new rules to a firewall. For one IPv4 flow – characterized by its IPv4 quintuple (IPv4 source and destination address, Layer 4 protocol, source and destination port number) – only a single verdict (allow, drop, by-pass, etc.) can be set. This modelling allows the PDP to detect potential conflicts in an obligation policy set and prevent the enforcement of these conflicting configurations.

2.1.5 XACML summary

XACML, together with the ongoing activities of introducing obligation support, provide a promising basis to build a policy-based remediation framework upon. The proposed extension, XOML, provides many features that are required for resilience. Its main features include an open and extensible obligation policy definition language for obligation policies. Based on the policy definition, the conflict resolution scheme can detect and prevent the enforcement of conflicting obligations. Complementary to this, PDP and PEP dynamically negotiate the supported obligation to prevent enforcement of obligations not supported by the PEP. A shortcoming of XOML is the lack of support of an obligation subject which would simplify integration significantly. Proposals in the literature exist, and their integration into XOML is currently being investigated.

2.2 Ponder2

Ponder2 [PON] implements a policy execution framework that supports the enforcement of both obligation (event–condition–action) and authorisation (access control) policies. Policies can be dynamically loaded, enabled, disabled and unloaded to change the behaviour of the Ponder2 instance without interrupting its functioning. Ponder2 incorporates a Domain Service,

which provides an hierarchical structure for managing objects, and a command interpreter that interprets PonderTalk (a high-level language based on SmallTalk) commands via a range of communication interfaces, such as RMI, and invokes actions on Managed Objects that exist in the Domain Service. An overview of these components is presented in the sections below.

One of the design goals of Ponder2 is that it should scale to the deployment context. For example, it should be deployable on small devices, such as GumstiX, as well as traditional operator network devices. As enabling resilience in the future Internet will require co-operation across potentially multiple network architectures, e.g., a WMN that connects to a wired back-haul, the ability of a policy framework to scale to different network architectures is undoubtedly advantageous.

In addition to the aspects of the framework described below, Ponder2 implements the notion of a Self-Managed Cell (SMC) [SFLD+07] – an autonomous policy-driven administrative domain. We discuss some of the benefits of using SMCs to enable inter-domain interactions for resilience, e.g., between autonomous systems, in D2.3a; here we provide a summary. A number of requirements are defined for peer and composition (hierarchical) SMC interactions that can be used as a basis for developing inter-domain protocols for resilience. Dynamic policy loading is supported by SMCs using a "mission" abstraction. A mission relates to an activity, such as mitigate a DDoS attack, that permits one SMC to remotely install policies in another SMC. Furthermore, SMCs provide interfaces that can be queried by other components to determine the SMC's capabilities. SMC features such as these provide mechanisms that enable entities in distinct administrative domains that collectively provide resilience, such as the resilience framework presented in D2.3a, to co-operate via well-defined interfaces.

2.2.1 Managed Objects and Domains

Ponder2 comprises a general-purpose object management system. The policy service maintains managed objects for each of the components on which management actions can be performed. This includes sensors and local resources, services within those devices, and adapters for remote instances of Ponder2. Managed objects representing remote devices adapt invocations received to platform-specific actions, thus providing a uniform interface for accessing their services. Managed objects are kept in a domain structure that implements a hierarchical namespace, similar to a file system. A domain provides a way of grouping objects for the purposes of policy specification, as the specification of policies in terms of a large number of individual subjects and targets is impractical [ST94]. Grouping of objects can be based on their type, management functionality, or simply convenience.

Policies are written in terms of managed objects. Domains and policies are managed objects in their own right on which actions can be performed; for example, adding or removing an object from a domain, or enabling or disabling other policies. Thus, events can trigger obligation policies that enable or disable other policies, allowing an instance of Ponder2 to modify its own adaptation strategy during run-time. Policies specified in terms of a specific domain will apply to all objects inside that domain. In short, events generated by managed objects within the Ponder2 instance trigger obligation policies, which specify what management actions must be invoked in other objects, provided these actions are allowed by corresponding authorisation policies.

Managed objects are typically programmed in Java. Managed objects may also be held transparently in a remote Ponder2 system, and management actions are invoked in remote devices via the adapters stored in the local domain. Different underlying transport protocols

are natively supported to facilitate remote communication, e.g., RMI, HTTP, etc.

2.2.2 Factory Objects

The policy service has been designed with particular focus on flexibility, in that all the code needed can be loaded on demand. This is done via factory objects, which are managed objects themselves and permit the creation of new objects in the domain. Factories can be loaded dynamically and used for creating new policies, for creating domain objects to form a hierarchy, and for creating adapters to communicate with external devices. Ponder2's factories provide built-in support for the creation of a set of core managed objects, e.g., events, policies, etc. However, the infrastructure is extensible and allows the creation of user-defined custom factory objects, e.g., adapters for interfacing with a temperature sensor.

2.2.3 PonderTalk and the Command Interpreter

A command interpreter provided by Ponder2 supports a high-level configuration and control language called *PonderTalk*, which allows the invocation of actions on these managed objects. PonderTalk's syntax is based on Smalltalk, in which messages can be sent to objects. A PonderTalk statement is defined as a reference to a managed object (possibly stored in the domain hierarchy), followed by zero or more messages to be sent to the object. A message may be a simple command or it may be parameterised. The example below illustrates an example PonderTalk statement:

```
root/myObject print: Hello World.
```

In this example, the message "print:" is sent to the object "myObject" which is stored in the "root" domain of the local Ponder2 instance. The message receives as an argument the string "Hello World". This example assumes that "myObject" accepts the "print:" message, otherwise an exception will be raised.

PonderTalk commands are linked to Java methods defined in the corresponding managed object by using Java annotations (i.e., `@Ponder2op()`). Thus, a PonderTalk command such as:

```
root/myObject addPolicyBehavior: ‘‘policy’’ to: ‘‘role’’.
```

is linked to a Java implementation in the corresponding managed object, by using a Java annotation of the form:

```
@Ponder2op(addPolicyBehavior:to:)
public void p2_addPolicyBehaviorTo(P2Block policy, String roleName)
{
  // Implementation of the method comes here
  // ...
}
```

This decouples the Java implementation of the methods from their invocation for the purposes of configuring and controlling a Ponder2 system. New factory objects can be loaded in a Ponder2 interpreter and objects can be created from these factories on demand, permitting commands to be sent to these objects dynamically via PonderTalk messages. This ability of Ponder2 to de-couple the invocation of actions on managed objects from their implementation allows new resilience policies to be created at run-time, which may be necessary in light of the changing nature of challenges, in the short-term, and learned behaviour as part of the

diagnosis and refinement phase, in the longer-term.

2.2.4 Ponder2 Summary

In summary, the Ponder2 policy framework at its core is intended to be lightweight, such that it can execute on relatively resource scarce devices. Additional features, such as those described here for XACML and OrBAC, can be built on top of Ponder2. For our purposes, this means that Ponder2 can execute on a range of devices, which means that interoperability across network architectures is more straightforward. Within Ponder2, the concept of a Self-Managed Cell is implemented, which facilitates interactions between autonomous administrative domains. This could prove useful when entities need to perform some collective actions to improve network resilience. By grouping policies into convenient Domains, realising policies for resilience can be more tractable; we appreciate that developing policies for resilience is a complex problem. In the submitted publication, which is attached to this deliverable as an appendix, we discuss some of the complexities of defining configurations for resilience, and show how policies can be used to manage this complexity.

2.3 The Organization-Based Access Control (Or-BAC) Model

The Organization-Based Access Control (Or-BAC) model [KBB+03] connects security monitoring to security policies, and thus provides automated and high-level remediation management. Or-BAC uses an expressive language, which relies on first order logic. Or-BAC handles policies at two abstraction levels, identically abstract and concrete policies. Abstract policies include global specifications whose enforcement in the real-world environment is constrained by contexts [CM03]. Concrete policies are instantiations of abstract policies that are inferred with knowledge about the current system contexts. Or-BAC thus integrates dynamic remediation, especially the possibility to adapt policy instantiation to current challenges, through the use of appropriate context definitions.

An Or-BAC remediation framework adds a Policy Instantiation Engine (PIE) to the traditional Policy Decision Point (PDP) and Policy Enforcement Points (PEPs) [DTCCB07]. The PIE considers a generic policy, defined at the abstract level, which is able to activate concrete policy instances depending on some input alerts. The PIE characterizes challenges and triggers adequate remediation measures, as new policy instances. The PDP in an Or-BAC framework is relegated to the role of a local decisional entity [KDC+09]. It mainly deals with policy instance translation according to the capabilities of the PEPs. However, it may also be in charge of local strategy functionalities, which are not in the scope of the PIE.

Or-BAC is used to derive many different policy instantiations according to current contexts. By carefully modelling contexts and specifying the way they are activated and deactivated, the adaptation of policies to existing events would be a straightforward approach. The Or-BAC model defines policies at a totally abstract level, that is not to use subjects, actions and objects; but abstractions of these components using roles, activities and views respectively. An Or-BAC rule thus includes abstract components, and is associated to a given context, which is either an elementary context or a combination of elementary contexts. Activating this context for some concrete elements automatically infers a concrete security rule from the abstract one, and which applies to these elements (see Section 2.3.3 for details). Or-BAC handles context combination (e.g., conjunction, disjunction and negation) and context prioritization. Priorities define a partial order on the set of contexts so that when a conflict occurs between two rules,

preference is given to the rule with the higher context priority. Priority assigned to security rules must be compatible with hierarchies defined on entities such as role, activity and view [CCBG07]. Thus, in case of conflict, if a given security rule is inherited by a given entity, this rule will have lower priority than another security rule explicitly assigned to this entity. Conflict resolution is addressed through dynamic priority assessment based on the priorities of activated contexts.

An Or-BAC security policy includes access control rules (i.e., permissions and prohibitions) and obligation rules. A first form of remediation would be to update the access control policy by activating and deploying new permissions or prohibitions. In a second case, a remediation may be specified using obligations. According to the subject specified in an Or-BAC obligation, this obligation either applies on the system side or the user side. A system-side obligation is typically enforced by some security device (PEP), and usually corresponds to an *immediate* obligation. A user-side obligation specifies some action to be enforced by some user. User-side obligation might be enforced within a certain delay (e.g., obligation to backup files and disconnect from an overloaded server within a given delay). The failure to enforce a user-side obligation in Or-BAC could result in some provisional contexts to be activated, and subsequently to activate other (usually strict) security rules.

When the management of abstract policies is centralized by a single PIE which is aware of all contexts within a single domain, multiple local strategies for enforcing those rules could be implemented by multiple PDPs. Each PDP therefore recuperates a sub-part of the global concrete policy and locally enforces this sub-part on the local PEPs according to their capabilities. In an Or-BAC framework, PDPs do not negotiate policies between them as these policies are already divided for these PDPs by the PIE.

2.3.1 Abstract Policies in Or-BAC

The Or-BAC model implements abstractions of the traditional triplets $\langle subject, action, object\rangle$ into $\langle role, activity, view\rangle$. The entities $subject$, $action$ and $object$ are known as concrete entities whereas the entities $role$, $activity$ and $view$ are called organizational entities. A $view$ is a set objects that process the same security-related properties within an organization, thus these objects are accessed in the same way. Abstracting them into views avoids the need to write one rule for each of them. Another abstraction is that of action into activity. An activity is considered as an operation that is implemented by some actions defined in the organization. This is why they can be grouped within the same activity for which we may define a single security rule. One of the main contributions of the Or-BAC model is that it can model context that reduces the applicability of the rules to some specific circumstances. Thus, context is another organizational entity of the Or-BAC model. The Or-BAC model defines four predicates:

- *empower (subj, role)* means that the subject $subj$ is empowered in the role $role$.
- *consider (act, activity)* means that the action act implements the activity $activity$.
- *use (obj, view)* means that the object is used in the view $view$.
- *hold (subj, act, obj, ctxt)* means that the context $ctxt$ is true between the subject $subj$, the action act and the object obj.

A security rule is specified in Or-BAC by the following: *SR (decision, role, activity, view, context)*. It specifies that the decision (i.e., permission, prohibition and obligation) is applied

to a given role when requesting to perform a given activity on a given view in a given context. We call these organizational security rules. An example of such a security rule is *SR (permission, private_host, HTTP, Internet, nominal)* which specifies that hosts assigned to the role $private\_host$ are permitted to use the HTTP protocol for Internet access in the nominal context (the nominal context is always true).

2.3.2 Remediation policies in Or-BAC

The remediation policy corresponds to security requirements that are activated when challenges occur. In Or-BAC, this is modelled using appropriate contexts called *threat context*. For this purpose, challenge scenarios are associated with threat contexts. These threat contexts are activated when challenges are detected, and are used to specify the remediation policies. The activation of these contexts leads to the instantiation of the policy rules in response to the considered challenge. For instance, a network congestion challenge is reported by a certain drop in the throughput of some devices. The network device which is concerned by this challenge corresponds to some gateway Gtw which is used as access path to some service $serv$. Then the $Net\_congestion$ is specified as in listing 1. Notice that the subject field is not specified

Listing 1: Threat context definition

```
hold(any, serv, Gtw, Net_congestion):−
    alert (Time, Source, Target, Classification),
    reference (Classification, 'bad throughput'),
    service (Target, serv), hostname(Target, Gtw).
```

since network congestion is an accidental challenge which is not produced by some specific subject. Therefore, and when new alerts are launched by some detection device, new *hold* facts are derived for threat context $Ctxt$. Therefore, all security rules associated with the context $Ctxt$ are triggered to remediate to the challenge.

2.3.3 Concrete policy inference

We model concrete security rules using the predicate $Sr(decision, subject, action, object)$. A concrete security rule is derived from an organizational (i.e. abstract) security rule by the general inference rule in listing 2. The inference of security rules which specify both permissions,

Listing 2: Threat context definition

```
Sr (decision, subject, action, object):−
    empower (subject, Role),
    consider (action, activity), use (object, View),
    hold (subject, action, object, context),
    SR (Decision, Role, Activity, View, Context)
```

prohibitions and obligations may lead to conflicts between the security requirements of these rules. These conflicts are classified in Or-BAC into three different classes:

Access conflicts occur when we may derive, for the same concrete components, both a permission and a prohibition. In Or-BAC, it is expressed by the following rules: $Sr(permission, subj, act, obj)$ and $Sr(prohibition, subj, act, obj)$.

Prohibited obligations occur when a given subject is obligated to make some action on some object while being prohibited to do so in the same time. In Or-BAC, it is expressed by the following rules: $Sr(obligation, subj, act, obj)$ and $Sr(prohibition, subj, act, obj)$.

Obligation conflicts occur when a given subject is obligated to make two conflicting actions (act_1 and act_2), that is to be impossible to simultaneously make these actions. In Or-BAC, it is expressed by the following rules $Sr(obligation, subj, act_1, obj)$ and $Sr(prohibition, subj, act_2, obj)$.

The management of these conflicts in Or-BAC is made at the abstract level by managing potential conflicts between abstract rules [CCBG07]. In case of access conflicts and prohibited obligations, a potential conflict exists if two conflicting rules (one prohibition and one permission or one prohibition and one obligation) may apply to the same subject, action and object. The approach used to manage such conflicts is based on the definition of *separation constraints* assigned to organizational entities. A separation constraint assigned to two roles specifies that a given subject may not be empowered in these two roles. The same concept applies to activities and views. In case of obligation conflicts, Or-BAC introduces the so-called $antinomic$ [CCBB+08] constraint which applies to activities. Two activities are antinomic if it is not possible to execute these two activities simultaneously. The combined use of separation and antinomic constraints enables the detection of potential conflicts. We may therefore use priorities to specify the rules which take precedence when such potential conflicts are inferred at the concrete level.

2.3.4 Or-BAC summary

It appears from this overview that Or-BAC is a well-suited model to specify security policies that dynamically change when a challenge is detected. In the absence of a challenge, the policy to be applied is said to be an operational policy and corresponds to nominal contexts. Other contexts must be defined to specify additional security rules to be triggered when challenges are detected. We have explored in this section some methods for defining such contexts. Or-BAC also provides means to manage conflicts between security rules. The management of conflicts at an abstract level guarantees that no conflicts would occur at the concrete level.

Or-BAC provides an abstract and global view of the security policy. It is the purpose of the PIE to manage this global security policy. The PIE must clearly separate the global policy from its implementation in the PEPs. In particular, the conflicts are to be solved at the abstract level before generating PEPs configurations.

3 Conclusion and Outlook

In this deliverable, we have introduced three prominent policy frameworks that we are currently considering the suitability of to build a resilience architecture. XACML, Ponder2 and Or-BAC provide similar basic feature sets with different specialities. In various activities within ResumeNet, mainly in *WP2 network resilience*, we use these policy frameworks to get further insights on their suitability to steer remediation and recovery strategy selection. This deliverable will be updated at the end of the project to reflect the lessons learned from these activities, and to provide a guidelines to system architects that wish to build resilient systems about the benefits and short-comings we discovered in our experiments.

Appendix: **List of publications**

- "Strategies for Network Resilience: Capitalising on Policies", Paul Smith, Alberto Schaeffer-Filho, Azman Ali, Marcus Schöller, Nizar Kheir, Andreas Mauthe and David Hutchison, publication in preparation

References

[CCBB+08] Frédéric Cuppens, Nora Cuppens-Boulahia, Yacine Bouzida, Wael Kanoun, and Aurélien Croissant. Expression and deployment of reaction policies. In *SITIS '08: Proceedings of the 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, pages 118–127, Washington, DC, USA, 2008. IEEE Computer Society.

[CCBG07] Frederic Cuppens, Nora Cuppens-Boulahia, and Meriam Ben Ghorbel. High level conflict management stategies in advanced access control models. *Electronic notes in theoretical computer science*, 186:3–26, 2007.

[CF09] David W. Chadwick and Kaniz Fatema. An advanced policy based authorisation infrastructure. In *DIM '09: Proceedings of the 5th ACM workshop on Digital identity management*, pages 81–84, New York, NY, USA, 2009. ACM.

[CM03] Frederic Cuppens and Alexandre Miege. Modelling contexts in the or-bac model. In *19th Annual Computer Security Applications Conference (ACSAC)*, 2003.

[CSL08] David W. Chadwick, Linying Su, and Romain Laborde. Coordinating access control in grid services. *Concurr. Comput. : Pract. Exper.*, 20(9):1071–1094, 2008.

[D+01] Nicodemos Damianou et al. The Ponder policy specification language. In *Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks (Policy)*, pages 18–39, Bristol, U.K., January 2001. IEEE Computer Society.

[DSSK10] C. Doerr, M. Schöller, P. Smith, and N. Kheir. D2.3a: First draft on the remediation, recovery, and measurement framework. ResumeNet Deliverable, February 2010.

[DTCCB07] Herve Debar, Yohann Thomas, Frederic Cuppens, and Nora Cuppens-Boulahia. Enabling automated threat response through the use of a dynamic security policy. *Journal in Computer Virology (JCV)*, Volume 3:195–210, 2007.

[KBB+03] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Mige, C. Saurel, and G. Trouessin. Organization based access control. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks (Policy)*, 2003.

[KDC+09] Nizar Kheir, Herve Debar, Frederic Cuppens, Nora Cuppens-Boulahia, and Jouni Viinikka. A service dependency modeling framework for policy based response enforcement. In *6th International Conference on Detection of Intrusion, Malware and Vulnerability Assessment (DIMVA)*, 2009.

[Lis10] M. Lischka. Dynamic obligation specification and negotiation. In *to appear in Network Operations and Management Symposium Workshops (NOMS)*, 2010.

[OAS07] OASIS. *XACML v3.0 Obligation Families Version 1.0 Working Draft 1*, December 2007.

[OAS09] OASIS. *eXtensible Access Control Markup Language (XACML) Version 3.0 Committee Draft 1*, April 2009.

[PON] Ponder2. http://www.ponder2.net.

[SFLD+07] Alberto Schaeffer-Filho, Emil Lupu, Naranker Dulay, Sye Loong Keoh, Kevin Twidle, Morris Sloman, Steven Heeps, Stephen Strowes, and Joe Sventek. Towards supporting interactions between self-managed cells. *Self-Adaptive and Self-Organizing Systems, International Conference on*, 0:224–236, 2007.

[SS09] M. Schöller and P. Smith. D1.1: Understanding challenges and their impact on network resilience. ResumeNet Deliverable, August 2009.

[ST94] Morris Sloman and Kevin Twidle. Domains: a framework for structuring management policy. In *Network and distributed systems management*, pages 433–453. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[W3C06] W3C. *Extensible Markup Language (XML) 1.1 (Second Edition)*, August 2006.

Strategies for Network Resilience: Capitalising on Policies

Paul Smith[1], Alberto Schaeffer-Filho[1],Azman Ali[1], Marcus Schöller[2], Nizar Kheir[3], Andreas Mauthe[1] and David Hutchison[1]

[1] Computing Department, Lancaster University, UK
{p.smith, asf, a.ali, andreas, dh}@comp.lancs.ac.uk
[2] NEC Laboratories Europe, Heidelberg, Germany
marcus.schoeller@neclab.eu
[3] France Télécom R&D Caen, 14066 CAEN, France
nizar.kheir@orange-ftgroup.com

Abstract. Networked systems are subject to a wide range of challenges whose nature changes over time, including malicious attacks and operational overload. Numerous mechanisms can be used to ensure the resilience of networked systems in light of these challenges. However, it can be difficult to define how these mechanisms should be configured in networks that support many services that have differing and shifting requirements, such as the Internet. In this paper, we present an exploration of the potential benefits of using of policy-based management frameworks for defining the configuration of mechanisms that can be used for resilience. Using an Internet Service Provider scenario, we show how policies can be used to realise a strategy for resilience that incorporates a set of potential conflicting mechanisms. The use of policies allows a configuration to be specified independently of its implementation; this approach provides necessary flexibility, given the changing nature of challenges and requirements. We discuss some of the complexities of defining configurations, such as identifying conflicts, and highlight how existing policy-based management frameworks could be used or extended to manage this complexity.

1 Introduction

The cost of failure of communication systems, and the Internet in particular, can be extremely high, in some cases. We depend on them to support many aspects of our daily life – e-commerce, governance, and the control of infrastructure, such as electricity grids. Outages of the networks that support these activities can have significant monetary, societal, and humanitarian impact. Developing strategies to ensure the resilience of networked systems is of primary importance.

The challenges that can negatively impact networked systems are wide-ranging. For example, they are susceptible to random component faults, large-scale natural disasters, human mistakes (e.g., mis-configurations), properties of the communication environment – particularly for mobile wireless networks –

unusual but legitimate request for service (e.g., flash crowd events), and malicious behaviour from intelligent adversaries. Furthermore, the nature and set of challenges a networked system will have to endure can change over time.

To combat the challenges to networked systems, a wide range of mechanisms can be used, such as firewalls and intrusion detection systems. What is often not clear is how they should be configured to realise high-order resilience requirements (managing trade-offs with other forms of requirement), such as the availability of critical services, and in response to the networked system's changing context and the existence of challenges. Continuing evidence suggests that high complexity when configuring mechanisms leads to mistakes being made [1].

In this paper, we explore the use of policies to define configurations of mechanisms that can ensure the resilience of networked systems. We believe that using policies to define configurations is beneficial for a number of reasons: by using them we de-couple the implementation of the resilience mechanisms from the strategy used to enable resilience, which is a desirable property considering the changing nature of challenges a network may face over time. Furthermore, specific features of policy-based management frameworks may assist in tackling a number of challenging problems that stem from the complexities of deriving resilience strategies for multi-service networks.

This exploration of configuring resilience mechanisms via policies is considered in the context of a general high-level strategy for resilience, called $\mathsf{D}^2\mathsf{R}^2$ + DR – *Defend*, *Detect*, *Remediate*, *Recover*, and *Diagnose* and *Refine* [2]. Using this high-level strategy one can consider the forms of configurations that are possible (or necessary) for resilience. Specifically, one must define configurations for *defensive* measures, such as firewalls or redundant network paths, to ward foreseen challenges off. Inevitably, challenges will occur that have not been accounted for, or are too great for defensive measures, these events should be *detected* in real-time, and *remediated*. The configuration of detection mechanisms, e.g., monitoring mechanisms and intrusion detection systems, needs to be considered, and their coupling to remediation mechanisms, such as packet filtering systems. During remediation, the configuration of the network is likely to be sub-optimal for normal operation; when challenges have abated they should be disengaged, via specific *recovery* configurations. It is assumed the real-time aspect of the strategy will need to evolve over time; the *diagnose* and *refinement* stages are used to identify shortcomings, e.g., in configurations, and improve them. The core of this paper is concerned with configuring mechanisms to realise aspects of the $\mathsf{D}^2\mathsf{R}^2$ stages of the strategy, we discuss how policies could be used to aid longer-term learning in Sect. 3.3.

As a proof-of-concept, we show, in Sect. 2, the use of policies for a resilience scenario – resource starvation of an Internet Service Provider's (ISP's) infrastructure caused by high volumes of traffic from a Distributed Denial of Service (DDoS) attack or flash crowd event. Based on this scenario, we then present, in Sect. 3, a discussion on the difficulties of deriving the configuration of mechanisms for resilience, and how features available in policy-based management frameworks could be used to address these. Specifically, we describe

problems relating to deriving concrete configurations of resilience mechanisms from high-level requirements, such as SLAs, identifying conflicting configurations, and evolving configurations over time in response to the changing nature of challenges and requirements.

2 Network Resilience Case Study

One of the key problems related to resilience in networks is to discriminate operational overload due to legitimate service requests from malicious attacks, and then apply adequate countermeasures to remediate the problem [3]. *Distributed Denial of Service (DDoS)* attacks are often launched from a large number of distributed sources, making the attack traffic characteristics appear similar to a *flash crowd*. However, these distinct incidents require different treatments by reliably distinguishing between attack and legitimate flash traffic and then, for example, dropping the former and rate-limiting the latter.

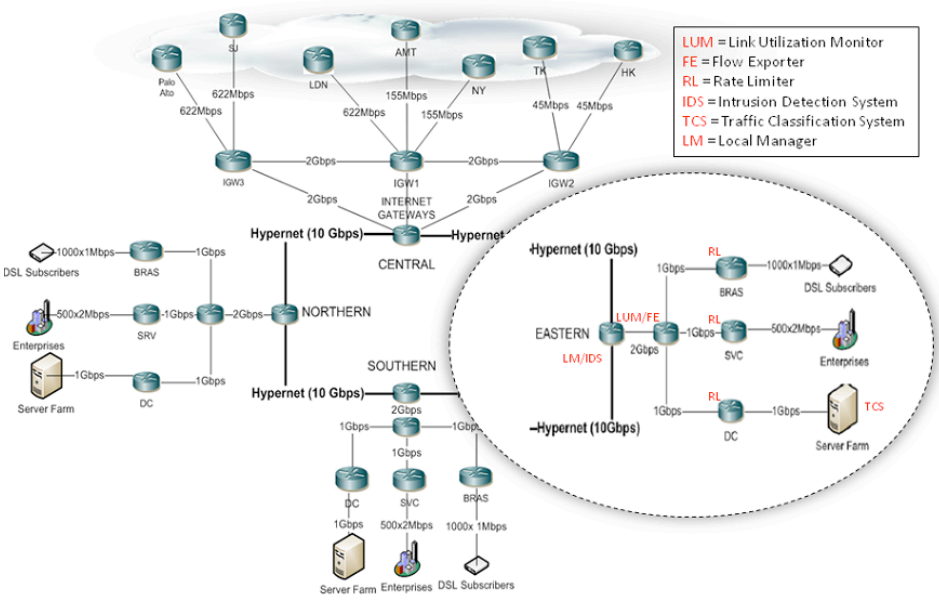


Fig. 1. An abstraction of Telekom Malaysia ISP's topology, physical resources and logical components for the resilience strategy.

To confront these types of challenges, our case study aims to show how, in the context of the $\mathsf{D}^2\mathsf{R}^2$ strategy, policies can be used to configure the *detection* and *remediation* components within the network. Initially, we describe an algorithm that can be used to detect and remediate high-volume network traffic challenges. We then show how this algorithm can be realised using policies. The algorithm and policies are presented within the context of typical ISP network topology,

specifically that of Telekom Malaysia's ISP topology[4], shown in Fig. 1. We limit the scope of our case study to the management of an access and corresponding distribution layer, highlighted in Fig. 1. This involves protecting the routers interconnecting with the core of the network, as well as the server farm present in that network, relying on local management and information collected in those routers and links alone.

2.1 Algorithm for Network Traffic Volume Resilience

Our algorithm relies on a number of *managed objects* (mechanisms) that must co-operatively enforce the resilience of the network. A physical device, e.g., a router, will typically implement several logical managed objects, e.g., a link monitor and an IP flow exporter. Table 1 summarises these components. The algorithm configures and facilitates the interaction between these components, and is outlined in Fig. 2. The algorithm is roughly divided in the following steps:

Table 1. Detection and remediation managed objects

Managed Object	Description
LinkMonitorMO	Measures the current utilisation of a given link
FlowExporterMO	Generates flow records at a given time out and with a given sampling rate
IntrusionDetectionMO	Anomaly detection based algorithms that calculates how much a flow deviates from the expected profile
RateLimiterMO	Applies rate limiting to an entire link or to selected flows within a link
ClassifierMO	Used for diagnosing the root cause of an anomaly, thereby allowing a more tailored remediation
LocalManagerMO	Performs local decisions and management actions in a subset of components
VisualisationMO	Used to notify the network operators of alarms or important conditions

1. *LinkMonitorMO* evaluates link utilisation at given periodicity, and notifies *LocalManagerMO* of significant changes (e.g., traffic rate above certain threshold).
2. *LocalManagerMO* configures components in the subnetwork accordingly: on high link utilisation, *FlowExporterMO* is enabled and starts recording *IP flows* at a default sampling rate. On low link utilisation, and if anomalies are not present in the link, flow recording is disabled, thereby keeping monitoring resources to a minimum, during periods of low demand.
3. When enabled, *FlowExporterMO* will truncate flow records after a specific time out period, e.g., 60s or 180s, and send records to the intrusion detection component with a given sampling rate.

[4] We base our example on Telekom Malaysia's topology because we have direct experience with it amongst the authors.

4. When receiving a new flow record, *IntrusionDetectionSystemMO* will use an anomaly-based algorithm (e.g., *entropy* [4]) to estimate how much a flow deviates from the expected link behaviour and, if an anomaly is detected, it will be flagged. *LocalManagerMO* will be notified, and *ClassifierMO* will be enabled for identification of the route cause of the anomaly. Subsequent records from that flow will be sent to *ClassifierMO* automatically.
5. When *LocalManagerMO* receives high risk event from *IntrusionDetectionSystemMO*, *FlowExporterMO*'s time out and sampling rate are adjusted. *LocalManagerMO* also obtains link utilisation from *LinkMonitorMO* and if above threshold, configures *RateLimiterMO* to perform preventive rate limiting on the whole link capacity in order to keep it operational.
6. Meanwhile, flow records are classified using more expensive algorithms (e.g., *Naïve Bayes* [5]) by *ClassifierMO*. Choice of the algorithm is a policy decision in itself.
7. Based on the cause of the anomaly, *LocalManagerMO* applies a tailored remediation strategy, such as blocking bad traffic coming from a DDoS attack or rate limiting legitimate traffic coming from a flash crowd, via configuration of the *RateLimiterMO*. A *VisualisationMO* may also be used to notify the network operator, if needed.
8. Eventually, *IntrusionDetectionSystemMO* will detect that an anomaly no longer exists for a specific flow. It will stop forwarding flow records to *ClassifierMO* and notify *LocalManagerMO* (not shown in Fig. 2).
9. When *LocalManagerMO* receives the low risk event, if there are no other anomalies for the same link, *RateLimiterMO* is configured to stop limiting and *FlowExporterMO* is notified. Finally, *FlowExporterMO* adjusts time out and sampling rate accordingly (not shown in Fig. 2).

This is a simplified strategy for network resilience. A more sophisticated solution may include additional detection and remediation components, which

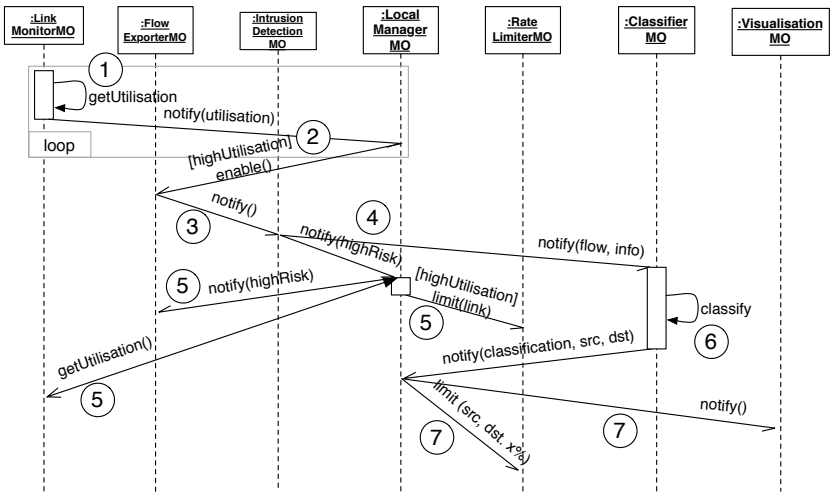


Fig. 2. Algorithm for network traffic volume resilience.

are configured in a similar manner, such as *Pushback* [6] for filtering bad traffic from upstream routers and *REPLEX* [7] for configuring dynamic multi-path routing in order to reduce stress on default paths. Thus, this is not the only resilience strategy but rather one of several possible strategies.

```
on classification(fl, value, conf)
    if ((value == ''DDoS'') and (conf < 0.4))
        do
        {
            VisualisationMO notify(alert(high));
            RateLimiterMO limit(fl.src, fl.dest, x%);
        }
    if ((value == ''DDoS'') and (conf >= 0.4) and (conf <= 0.8))
        do
        {
            VisualisationMO notify(alert(high));
            RateLimiterMO limit(fl.src, fl.dest, y%);
        }
    if ((value == ''DDoS'') and (conf > 0.8))
        do
        {
            VisualisationMO notify(alert(high));
            RateLimiterMO block(fl.src, fl.dest);
        }
```

Fig. 3. Policy handling classifications based on root cause (configuration of remediation mechanism).

Two important insights were obtained when defining this case study: *(1)* the use of different sets of mechanisms to achieve resilience, e.g., ensuring link availability via a *Rate limiter* or *REPLEX*, may lead to similar resilience properties; and *(2)* by having a resilience strategy implemented with the aid of policies, as opposite to having it hardcoded, one can easily change it by editing, adding or removing the corresponding policies.

2.2 Policy-based Resilience Strategy

Policies separate the management strategy from the implementation of the management actions. This permits the modification of the run-time adaptation strategy without interrupting system operation [8]. This flexibility is of particular importance for resilience, as strategies for detection and remediation of network challenges are subject to frequent modifications, due to changes in application requirements, context changes or new types of challenge manifestation. As an example, Fig. 3 shows the policy that defines how *LocalManagerMO* reacts to the outputs of *ClassifierMO*. According to *step 7* of the algorithm, different root causes will require distinct remediation strategies, for example, when a flow is classified as a possible *DDoS attack*, a preventive rate limiting action may be applied with actual parameters defined by the confidence level, ranging from the slight limitation to the complete blocking of a given flow.

Similarly, Fig. 4 illustrates the policy which implements *step 9* of our algorithm and enforces the recovery of the network into normal operation. When an event indicates that a given source-destination flow is no longer anomalous, if

```
on lowRisk (link, src, dst)
    if ((anomalyList remove(link, src, dst)) isEmpty(link))
        do
        {
            FlowExporterMO notify(lowRisk(link));
            RateLimiterMO limit(link, 100%);
        }
```

Fig. 4. Policy configuring flow recording and rate limiting upon low risk (recovery into normal operation).

there are no other anomalies for the same link, the operation of *FlowExporterMO* and *RateLimiterMO* is adjusted accordingly, e.g., lightweight flow recording and no rate limiting for the link. Policies implementing each one of the other steps of our algorithm were defined in a similar manner but are not shown here.

3 Complexities of Defining Configurations

Using the case study presented in Sect. 2, here we discuss some of the complexities of determining appropriate configurations of resilience mechanisms, and highlight where support can be found in policy-based management frameworks to address these complexities.

3.1 Deriving Configurations from High-level Requirements

In our case study, we present an algorithm and derive a set of policies that realise it, which can be used to protect a region of an ISP's network. We assume the policies will realise a high-level requirement to ensure the resilience, e.g., defined in terms of availability and reliability, of the server farm and the services, such as Websites, that it provides. It is clear that our algorithm could improve the resilience of the infrastructure mentioned, but it is not clear if it is sufficient to ensure that a specific high-level goal, e.g., defined in a Service Level Agreement, is met. Moreover, our case study is relatively straightforward in nature, more complex scenarios, for instance, that require co-operation across ISPs, would make deriving concrete policies by hand, as we did here, intractable. Clearly, toolsets are required that can be used to aid this process of deriving configurations from high-level requirements.

We seek to (semi-)automatically derive strategies of implementable policy configurations from high-level specifications and requirements, with minimum human involvement. In [9], a technique for refining high-level goals into concrete policy specifications is presented, which consists of *(1)* recursively applying goal elaboration, iteratively transforming high-level goals into more concrete ones, until they can be expressed as implementable operations in the underlying system, and then *(2)* using logical reasoning and abduction to derive a *strategy*, which will determine how low-level operations need to be executed sequentially or in parallel, in order to realise the high-level goal. The invocation of these operations in turn can be co-ordinated via policies. This goal elaboration technique was used in [10] to refine high-level QoS requirements into the concrete configuration of routers. Our task is more difficult, as we must generalise these concepts

to address not only QoS requirements, but to co-ordinate the resilience aspects of the system, both at the network and service levels. Other strategies for transforming high-level goals into concrete policies, for example, using rules [11, 12] are also being considered.

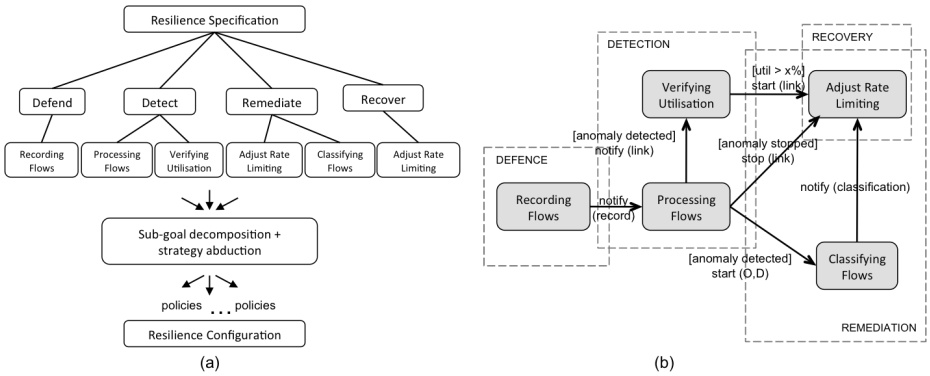


Fig. 5. (a) Goal decomposition of a resilience specification plus abduction of low-level configuration policies, and (b) resilience strategy for the scenario (intermediary step).

By decomposing specific sub-goals that independently realise the different phases of the $\mathsf{D}^2\mathsf{R}^2$ strategy, each phase is thus refined into more concrete sub-goals, until it consists only of concrete implementable operations. Fig. 5 outlines the partial decomposition of high-level resilience goals with respect to the case study described in Sect. 2, e.g., *detection* is achieved by *processing incoming flows* and *verifying link utilisation*. In its most concrete level, the resilience strategy is represented by the algorithm presented in Fig. 2.

3.2 Identifying and Resolving Conflicting Configurations

In complex multi-service networks, determining the existence of conflicting configurations is of critical importance. Such conflicts can, for example, lead to the resilience requirements of a set of services being met, unnecessarily at the expense of another set, or no requirements being met for any service. We envisage a number of different ways that conflicts could manifest, as depicted in Fig. 6[5]:

***Vertically*, across protocol levels:** Consider the ISP scenario described in Sect. 2 in the presence of two concurrent challenges – a DDoS attack targeted at the server farm in East Malaysia and a flash crowd event directed toward the West Malaysia farm. Because of the DDoS attack, our remediation algorithm may lead to rate limiting being invoked on the routers associated with East Malaysia; a *network-level* mechanism. In the presence

[5] To ease presentation, we show two *levels* – network, which approximately maps to L1 to L4 of the OSI model, and service, which approximately maps to L7.

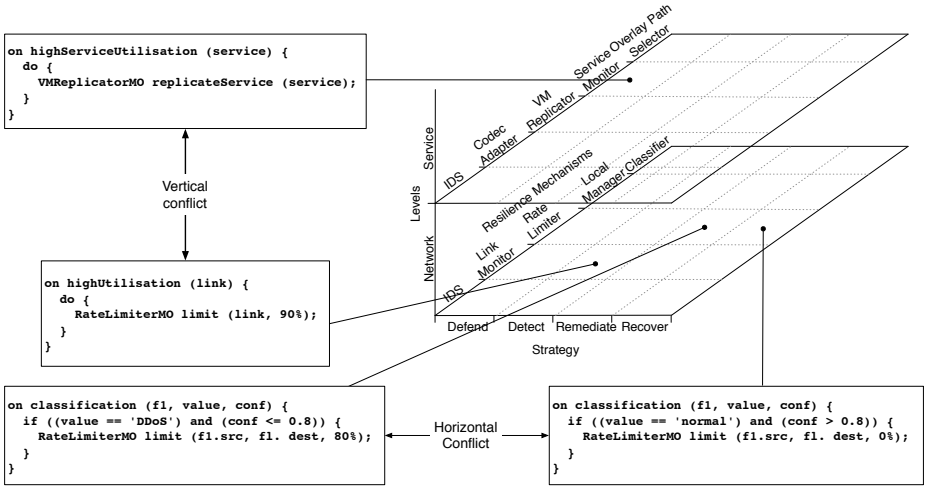


Fig. 6. Defining configurations for resilience is a multi-level problem, with vertical configuration conflicts across levels, and horizontal conflicts along the $\mathsf{D}^2\mathsf{R}^2$ strategy.

of a flash crowd event, we could reasonably decide to replicate a service to another server farm; a *service-level* mechanism. However, because of the naïve rate limiting that is operating as a consequence of the DDoS attack, this replication may not be able take place in East Malaysia, and by trying to replicate a service there, make the resource starvation situation worse. What is required in this context is a way of determining that the two configurations at different levels will conflict. The two example policies that could lead to this situation are shown in Fig. 6.

***Horizontally*, along the $\mathsf{D}^2\mathsf{R}^2$ strategy:** Again, considering our ISP scenario, an attack may be targeted at both a server farm and a corporate customer. Collectively, the attack traffic could saturate the 2Gb links that provide access to the core network; this would make *pushing back* the rate limiting of malicious traffic to the Internet Gateways (shown at the top of Fig. 1) desirable. For various reasons, detection mechanisms would still reside in close proximity to the targets, identifying malicious hosts, as in our algorithm. A detection mechanism at the server farm may, for instance, determine that for a particular target a node has ceased to behave maliciously, and initiate a *recovery* configuration for that node by removing it from a list of nodes to rate limit at the Internet Gateway. However, this same node may still be behaving maliciously in relation to the corporate customer; unless care is taken, the recovery configuration for the server farm could inadvertently disengage the remediation configuration for the corporate network. Example policies that demonstrate this potential problem are shown in Fig. 6.

By using policies to define the configuration of our framework we can also benefit from research on analysis and conflict resolution to ensure the correct specification of our resilience strategies. The categorisation of policy conflicts

presented in [13] is particularly relevant. The authors identify operations that can be used for policy ratification, namely: *dominance checks*, for determining when a policy is dominated by another, and therefore does not effect the behaviour of the system, *potential conflict checks*, where two or more policies cannot be applied simultaneously, *coverage checks*, which can be used to identify if policies have been defined for a certain range of possible cases, and *consistent priority assignment*, to prioritise which policies should be executed. In particular, a variation of dominance checks is desirable for a multi-level approach to resilience, requiring appropriate configuration of both network- and service-level mechanisms, to ensure that mechanisms at one level do not render mechanisms at another level redundant (vertical analysis). Similarly, coverage checks can be used for the analysis of configurations of mechanisms at the same level, for example, conditions or range of values where mechanisms activated in response to a DDoS attack are not co-ordinated properly (horizontal analysis).

Whereas the authors in [13] present the theoretical foundations of these types of analysis, they explicitly leave domain-specific information outside the scope of their study. Such domain-specific information is arguably necessary to detect the conflicts in the example horizontal and vertical conflicts described above. It is our intention to build on these general techniques for policy analysis and include domain specific expertise.

3.3 Learning Resilience Behaviour

The mechanisms and configuration that realise a resilience strategy will need to evolve over time. This is because challenges and requirements change – the nature of attacks change and new customer agreements may cause high-level priorities to shift. Furthermore, in light of challenges, a particular strategy may prove to be sub-optimal or incorrect. Ideally, the operation of the mechanisms that realise an instance of the $\mathsf{D}^2\mathsf{R}^2$ strategy must autonomously evolve and learn new resilience behaviour. This is represented by a two-stage background loop in $\mathsf{D}^2\mathsf{R}^2$ the strategy: *Diagnose* and *Refine*, where shortcomings are identified and improved on.

To assist with this learning, we can benefit from existing policy-based research, also. Typically, policy-based learning relies on the use of logical rules for knowledge representation and reasoning [14]. This is for two main reasons: *(1)* policies can be easily translated into a logical program, and *(2)* logical programs provide a way for users to understand (and correct) what has been learned. This is in contrast to mathematical/statistical models, e.g., artificial neural networks, where the system is able to learn "some" behaviour, but it is not possible to explain what or how this was learned.

We intend to use logical reasoning to learn and revise a *resilience theory* R, where R is a logical program formed by logical rules that define the conditions when actions should be executed. R can thus be used as the underlying formal representation of policies governing the system behaviour. A learning approach such as the one in [14] can be used to derive a *resilience theory* R' from R by adding and deleting rules in R or literals in the body of existing rules in R.

These can be, for example, new conditions when a given mitigation mechanism should or should not be activated in response to a challenge.

For example, based on information learned over time, the rule $r \in R$ saying that *"activate rate limiting at time T when high link utilisation is detected"*:

$$do(rate\_limit(link_1, 70\%), T) \longleftarrow holds\_at(utilisation(link_1, high), T)$$

can be transformed into a new $r' \in R'$ saying that *"activate rate limiting at time T when high link utilisation is detected, except if T equals the time of the Football League final"*:

$$do(rate\_limit(link_1, 70\%), T) \longleftarrow holds\_at(utilisation(link_1, high), T) \wedge \\ \neg holds\_at(broadcasting(football\_final), T)$$

The *resilience theory* R can be iteratively refined to better reflect resilience practices and the understanding of the system behaviour. Rules can be amended based on how successful previous attempts to mitigate a challenge were. Similarly, the system must be able to learn entire new rules, for example, that in the conditions described in the latter example, a replication of the server streaming the live match is more appropriate than simply rate limiting the link utilisation.

4 Conclusions

Network resilience is difficult to ensure because the configuration of systems is complex, spans across several levels, and is subject to a wide range of challenges. To help reason about the components necessary to build a resilient networked system, we rely on a six-stage strategy, called $\mathsf{D}^2\mathsf{R}^2 + \mathsf{DR}$. Policies provide flexibility in the configuration of the components that implement this strategy, as forms of detection and remediation are subject to frequent modifications, due to requirements shift, context changes or new types of challenge manifestation.

We examined the applicability of policies in a case study to mitigate high traffic volume challenges in the context of an ISP network. Despite several assumptions, the resilience strategy was still complex, and required numerous configurations and co-ordinated interactions between the mechanisms. We highlighted how policy-based approaches can assist in making the problem more tractable, through the derivation of resilience configurations from high level requirements, the identification and resolution of conflicts in the low-level configuration, and the learning aspects of the system based on past experience.

Future work will investigate how these policy techniques can be extended and incorporated into our framework. For some challenges, resilience can be improved by employing strategies that span multiple administrative domains; we will investigate how policy negotiation could be used to aid this. Furthermore, we will examine how statistical detection and classification mechanisms that yield results with varying degrees of certainty affect policy-based decision making.

Acknowledgment

The research presented in this paper is partially funded by the European Commission in the context of the Research Framework Program Seven (FP7) project ResumeNet (Grant Agreement No. 224619). This work has also been supported by the EPSRC funded India-UK Advance Technology Centre in Next Generation Networking. The authors are grateful to Angelos Marnerides for insights relating to detection approaches that form part of our resilience strategy.

References

1. Wool, A.: Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese. IEEE Internet Computing **99**(PrePrints) (2010)
2. Sterbenz, J.P., Hutchison, D., Çetinkaya, E.K., Jabbar, A., Rohrer, J.P., Schöller, M., Smith, P.: Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines. Computer Networks: Special Issue on Resilient and Survivable Networks (COMNET) (2010) (to appear).
3. Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Comp. Surv. **39**(1) (2007) 3
4. Lakhina, A., Crovella, M., Diot, C.: Mining anomalies using traffic feature distributions. In: SIGCOMM '05, New York, NY, USA, ACM (2005) 217–228
5. Moore, A.W., Zuev, D.: Internet traffic classification using bayesian analysis techniques. In: SIGMETRICS '05, New York, NY, USA, ACM (2005) 50–60
6. Mahajan, R., Bellovin, S.M., Floyd, S., Ioannidis, J., Paxson, V., Shenker, S.: Controlling high bandwidth aggregates in the network. SIGCOMM Comp. Commun. Rev. **32**(3) (2002) 62–73
7. Fischer, S., Kammenhuber, N., Feldmann, A.: REPLEX: dynamic traffic engineering based on wardrop routing policies. In: CoNEXT '06, New York, NY, USA, ACM (2006) 1–12
8. Damianou, N., et al.: The Ponder policy specification language. In: POLICY '01, Bristol, U.K., IEEE Computer Society (2001) 18–39
9. Bandara, A.K., Lupu, E.C., Moffett, J., Russo, A.: A Goal-based Approach to Policy Refinement. In: POLICY '04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, IEEE Computer Society (2004) 229
10. Bandara, A.K., Lupu, E., Russo, A., Dulay, N., Sloman, M., Flegkas, P., Charalambides, M., Pavlou, G.: Policy Refinement for IP Differentiated Services Quality of Service Management. IEEE Transactions on Network and Service Management **3**(2) (2006)
11. Beigi, M.S., Calo, S., Verma, D.: Policy Transformation Techniques in Policy-based Systems Management. In: POLICY '04. (2004) 13–22
12. Brodie, C., George, D., Karat, C.M., Karat, J., Lobo, J., Beigi, M., Wang, X., Calo, S., Verma, D., Schaeffer-Filho, A., Lupu, E., Sloman, M.: The Coalition Policy Management Portal for Policy authoring, Verification, and Deployment. In: POLICY '08, Washington, DC, USA, IEEE Computer Society (2008) 247–249
13. Agrawal, D., Giles, J., Lee, K.W., Lobo, J.: Policy ratification. In: POLICY '05, Washington, DC, USA, IEEE Computer Society (2005) 223–232
14. Corapi, D., Ray, O., Russo, A., Bandara, A., Lupu, E.: Learning rules from user behaviour. In: 2nd Int. Workshop on the Induction of Process Models. (2008)