Reengineering Network Resilience Strategies using a BDI Architecture

Ingrid Nunes and Alberto Schaeffer-Filho

[1]Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil

{ingridnunes,alberto}@inf.ufrgs.br

Abstract. *Network resilience is a challenging task. It requires constantly monitoring network devices to guarantee quality of service and to address potential challenges to network operation, such as malicious attacks, equipment failures or device misconfiguration. Although many approaches that provide automated support to this task rely on autonomous and pro-active software components, they do not exploit domain-neutral agent-based techniques, which can arguably provide flexible solutions to handle situations unpredicted at design time. In this paper we present an investigation in which we show how a pattern-based approach that addresses a distributed denial-of-service (DDoS) attack using policy-driven Self-Managed Cells (SMCs) can be reengineered using a BDI architecture and a logic-based language. Based on the new proposed design, we derive lessons learned, which discuss similarities, strengths and weaknesses of each approach. One of the main contributions of this study is to explore the synergy between network resilience and multi-agent system research areas.*

1. Introduction

Computer networks are becoming increasingly critical in supporting software applications related to business, leisure and daily life in general. Thus it is fundamental to support reliable network services. This is the main concern of *network resilience* [Sterbenz et al. 2010, Smith et al. 2011], which involves constantly monitoring network devices to guarantee quality of service and to address potential *challenges* to network operation, such as malicious attacks, equipment failures or device misconfiguration.

Due to its complex nature, network resilience combines ideas from several research areas, such as autonomic computing [Lupu et al. 2008] and machine learning [Nguyen and Armitage 2008]. Approaches that provide automated support to this task typically rely on autonomous and pro-active software components [Nguengang et al. 2006], which are related to the definition of a *(software) agent* [Jennings 2001]. Although some of these approaches refer to these components as agents, they do not exploit domain-neutral agent-based techniques, which can arguably provide flexible solutions to handle situations unpredicted at design time.

In this paper, we discuss how network resilience strategies [Schaeffer-Filho et al. 2012] can be reengineered using a widely used agent architecture, namely the BDI architecture [Rao and Georgeff 1995]. Initially, resilience strategies are described in terms of *management patterns* to address specific network challenges that are implemented using policy-driven *Self-Managed Cells*

(SMCs) [Lupu et al. 2008]. In particular, we focus on a pattern for a *Distributed Denial of Service (DDoS)* attack. As result of this investigation, we propose a new agent-based solution to address this challenge, structured with BDI agents — composed of beliefs, goals and plans — using a logic-based language. We also detail domain-specific steps of the BDI reasoning cycle, such as belief revision and goal generation. Based on the new proposed design, we derive lessons learned, which discuss similarities, strengths and weaknesses of each approach.

Our main contribution is thus the investigation of a new design solution for providing network resilience, using agents and a BDI architecture. Our aims are: (i) to better understand the synergy between the network resilience and multi-agent system research areas; (ii) to show the applicability of an existing agent-based approach to a real-world problem; and (iii) to provide a flexible alternative solution to network resilience.

This paper is organised as follows. Section 2 provides background on network resilience and the pattern-based approach in which this work is based on. Section 3 presents how the current approach can be reengineered using a BDI architecture, detailing individual agent components and how they are used to form a multi-agent system. Section 4 discusses lessons learned derived from our investigation. Finally, related work is outlined in Section 5 and the concluding remarks are presented in Section 6.

2. Background

This section describes the background work on network management and network resilience, and discusses how previous work advocated means of composing and federating resilience mechanisms into management patterns.

2.1. Network Management and Resilience Strategies

Central to a resilience strategy is the management and reconfiguration of detection and remediation mechanisms, which operate as autonomous components in the network infrastructure. On the one hand, detection mechanisms such as link monitors, anomaly detection systems and traffic classifiers permit the identification and categorisation of anomalies and attacks to the network. On the other hand, remediation mechanisms such as rate limiters, load balancers and firewalls can be used in the subsequent mitigation of these challenges. In previous work [Schaeffer-Filho et al. 2012], we have investigated how to flexibly organise network mechanisms providing detection and remediation functionality in order to combat specific types of network attacks, such as Distributed Denial of Service (DDoS) and worm propagations.

The various mechanisms can be deployed and activated on different parts of the network topology. Therefore, the functions of monitoring, anomaly detection, rate limiting and so on could be carried, for example, at the edge of the subnetwork under attack. We assume that mechanism configuration and adaptation can be expressed in terms of *event-triggered condition-action (ECA) policies* [Sloman and Lupu 2002]. Policies allow the specification of how a component must react in response to specific types of events, in a *declarative manner*. These mechanisms can be configured and federated into resilience strategies to address a specific type of challenge or attack, as the one depicted in Figure 1. This strategy consists of a number of stages, detailed next, which are triggered by declarative policies according to the conditions monitored in the network.

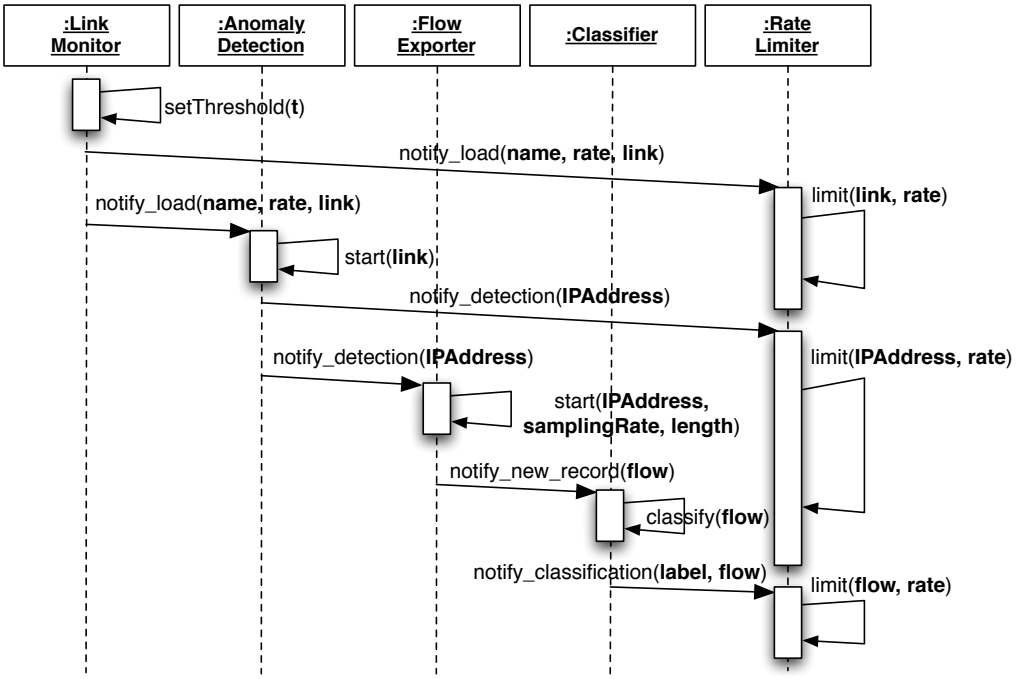

Figure 1. DDoS resilience strategy using a number of mechanisms that must operate autonomously in the network [Schaeffer-Filho et al. 2012].

1. Policies configured in the *Link Monitor* should monitor the incoming traffic rate on the ingress link and raise an alarm (*notify_load*) in case the traffic volume observed exceeds a given threshold.
2. On receiving this event, an initial rate limiting of the link can be performed by the *Rate Limiter*. Policies are used to filter a percentage of all incoming traffic in order to protect downstream servers.
3. In addition to the preventive rate limiting started at this point, the *Anomaly Detection* component can further start processing packets to identify the destination IP address of the victim.
4. As a result, it raises an event (*notify_detection*) when one destination IP address accounts for 60% of all packets. This additional evidence permits the reconfiguration of the *Rate Limiter* to drop 70% of the traffic destined for the victim only. Legitimate traffic that is not destined for the victim is now allowed to go through.
5. If the attack is sustained for a longer period, the *Classifier* and *Flow Exporter* mechanisms are started. *Classifier* attempts to identify the specific attack flows. Classification events are generated (*notify_classification*), and rate limiting is confined just to the attack flow, and legitimate traffic is allowed to continue.

2.2. A Pattern-based Approach for Network Resilience

Previous work has introduced the *Self-Managed Cell* (SMC) [Lupu et al. 2008] as a paradigm for structuring the management aspects of ubiquitous and network applications. An SMC typically consists of a set of autonomous hardware and software components with management and adaptation strategies specified as easily modifiable policies. For example, SMCs may represent autonomous components supporting resilience functionality, e.g., link monitoring, anomaly detection and traffic shaping. These components

must autonomously react to conditions monitored in the network, such as increased traffic load or suspicious network packets. In essence, an SMC provides an implementation of a MAPE-K loop [IBM 2005]. In the SMC model, ECA policies are used to specify adaptation rules that determine the reconfiguration actions to be performed in response to changes in the context, component failures, new components joining the system, or application-specific alarms.

In order to assemble and federate SMCs into large-scale applications we rely on the notion of *management patterns* [Schaeffer-Filho et al. 2014]. A management pattern is a configuration, including policies to be deployed and properties to be satisfied, between a set of SMC types. For example, a management pattern for addressing a DDoS attack will contain the policies and configurations necessary for a set of mechanism types such as the ones listed above. SMC types are specified in the pattern using the notion of *roles* [Lupu et al. 2008], which are placeholders for actual instances that will be manually assigned during runtime by a human expert. The use of patterns supports the construction of SMC interactions in a methodical manner, by reusing simpler abstractions as design elements of a more complex interaction. The exchange of policies, events and interfaces define the task-allocation, communication and structural aspects, respectively, of a cross-SMC interaction. Ultimately, management patterns can capture best practices and the experience of network operators into reusable policy configurations, which can be automatically deployed in the network when a challenge manifests.

3. Providing Network Resilience with BDI agents

Previous work [Schaeffer-Filho et al. 2012] has shown using simulations that the proposed pattern to handle DDoS attacks can be effective. Nevertheless, such an approach together with ECA policies focuses on establishing actions that should be performed given the occurrence of an event, without capturing the reasoning *why* a certain set of actions are the more adequate to be performed. The BDI agent architecture [Rao and Georgeff 1995] — adopted in this work — captures it in goals: the BDI architecture decouples *what* should be done from *how* it should be done. Consequently, it is possible to perform a higher-level reasoning that first chooses goals to be achieved, and then selects which actions are more adequate to achieve them according to the current context.

In this section, we show how the previous solution proposed to provide network resilience can be modelled using agent components following a BDI architecture and using a logic-based language, similar to the AgentSpeak(L) language [Rao 1996]. We first present, in Section 3.1, individual BDI agent components, such as goals and plans, and then show, in Section 3.2, how such components are used to form a multi-agent system able to respond to DDoS attacks.

3.1. Individual BDI Agent Components

BDI agents are software components structured with three main finer-grained components, namely beliefs, goals and plans. Such agents are provided with a reasoning cycle, including the following customisable steps: (i) *belief revision*: based on perceived events, agents update their current beliefs; (ii) *goal generation*: agents generate goals that they want to achieve, considering their current state; (iii) *goal deliberation*: agents choose, among their current goals, which they are committed to achieve; and (iv) *plan selection*: agents select a plan to achieve each goal they are committed to achieve. Given this

internal structure of each individual agent, next we present details on the finer-grained components that are part of our multi-agent solution to prevent DDoS attacks.

3.1.1. Belief Revision

Different network devices (which can be regarded as agents) are connected by links, where network traffic flows. Under normal conditions, only part of the link bandwidth is used, and the link usage should always be below a given threshold. Such links are part of the environment in which agents are located, and agents perceive events that occur in it. Therefore, agents perceive how much of the link capacity is being used, i.e. $usage(link)$, and have a belief — $overUsage(link)$ — that is updated according to the perceived link usage, as shown in Equations 1 and 2. Such equations are rules, in which the expression in the right-hand side of the symbol $\longrightarrow$ takes place when the expression in the left-hand side is true.

$$greater(usage(link), threshold) \longrightarrow belief(overUsage(link)) \quad (1)$$

$$lesseq(usage(link), threshold) \longrightarrow belief(\neg overUsage(link)) \quad (2)$$

When the link bandwidth is being used above the established threshold, there are two possibilities: this is due to either regular users that, for a certain reason, are deviating from the usual behaviour, or a malicious attack. Therefore, when agents perceive that the link bandwidth is being used above the threshold, they do not know if this is associated with a regular usage or not, i.e. they believe that $\sim regularUsage(link)$. This absence of knowledge is shown in Equation 3, where $\sim$ means that $regularUsage(link)$ is *unknown*.

$$overUsage(link) \longrightarrow belief(\sim regularUsage(link)) \quad (3)$$

3.1.2. Goal Generation and Deliberation

The belief revision step updates beliefs according to perceived external events. Given the current agent state, which is composed of its current beliefs, agents may have different goals. A goal is associated with a state of affairs that an agent would like to achieve. We will now analyse goals generated by agents according to certain beliefs.

If an agent believes that $overUsage(link)$, there are two goals to be achieved. Firstly, although the agent is not sure whether an attack is indeed occurring, it needs to prevent a possible attack, i.e., it generates a goal $attackPrevented(link)$. Secondly, because the agent does not know if $overUsage(link)$ is due to regular users, it needs to find this out, by generating a goal $?regularUsage(link)$. "?" indicates that the target state of affairs should either contain $regularUsage(link)$ or $\neg regularUsage(link)$. These two generated goals are shown below.

$$\begin{aligned} &belief(overUsage(link)) \longrightarrow \\ &\quad goal(attackPrevented(link)) \wedge goal(?regularUsage(link)) \end{aligned} \quad (4)$$

Another situation that may occur is that an agent, somehow, may have detected that one of the IP addresses that use a link has an anomalous behaviour, i.e.

$anomalous(ip)$. Similarly to the generated goals above, the agent must restrict the access of anomalous IPs to the link and find out whether this anomalous behaviour is benign or malicious, and therefore two goals are generated: $restricted(ip)$ and $?benign(ip)$.

$$belief(anomalous(ip)) \longrightarrow goal(restricted(ip)) \wedge goal(?benign(ip)) \quad (5)$$

An agent may also become aware that a certain IP flow is likely to be associated with an attack, that is, it is a threat to the network — $threat(flow)$. When this happens, the agent should respond to the threat, by generating a goal $threatResponded(flow)$.

$$belief(threat(flow)) \longrightarrow goal(threatResponded(flow)) \quad (6)$$

Finally, ideally, links should always be fully operational and accessible from anywhere (i.e. by any IP address). Therefore, if a link is not fully operational, agents have a goal to make them fully operational, or if an IP address is restricted, agents have a goal to make them unrestricted. This is shown in Equations 7 and 8, which generate these goals. Note these goals are generated solely when it is safe to do so.

$$belief(\neg fullyOperational(link)) \wedge belief(regularUsage(link)) \longrightarrow goal(fullyOperational(link)) \quad (7)$$

$$belief(restricted(ip)) \wedge belief(\neg anomalous(ip)) \longrightarrow goal(\neg restricted(ip)) \quad (8)$$

Given these goal generation rules, agents may have many goals to achieve. However, according to the current context, agents may be committed to achieve only some of the goals, for instance those with higher priority, to later achieve others. This is performed in the goal deliberation step. In our scenario, agents should prevent an attack, before learning about a certain information. Therefore goals that will lead to (palliative) actions to prevent and respond to attacks have the highest priority, which are: $attackPrevented(link)$, $restricted(ip)$, $threatResponded(flow)$. Goals that restore full operation, $fullyOperational(link)$ and $\neg restricted(ip)$, also have higher priority than the informational goals $?regularUsage(link)$ and $?benign(ip)$.

3.1.3. Plan Selection

We showed how agents reason about what should be done, without concerning with how it should be done. Now, we will focus on the latter, by detailing the plans to achieve goals. All plans, which are shown in Table 1 and will be described next, are used to achieve the presented goals and consequently to respond to a DDoS attack. Each plan has a name (with an acronym that will be used later to refer to it), a goal that it can achieve, the context that state the conditions in which the plan is applicable, and a set of actions that are executed to achieve the goal associated with the plan.

In order to describe plans, we will consider an initial scenario in which a specific flow is attacking a server in a network, and there is a single network switch (i.e., an agent) that provides access to this server through a single link. Consequently, after perceiving events, the agent believes that $overUsage(link)$ and $\sim regularUsage(link)$, and consequently has the goals $attackPrevented(link)$ and $?regularUsage(link)$. So, the agent

Table 1. Plans for responding to a DDoS attack.

Plan: LimitLinkRate (LLR) **Goal**: $attackPrevented(link)$ **Context**: $overUsage(link)$ **Actions**: $\texttt{limit}(link, rate)$ $belief(\neg fullyOperational(link))$ $belief(attackPrevented(link))$	**Plan**: AnalyseLinkStatistics (ALS) **Goal**: $?regularUsage(link)$ **Context**: - **Actions**: *// Analyse link and detect outliers* $\forall ip.(outlier(ip)) \longrightarrow belief(anomalous(ip)) \wedge belief(\sim benign(ip))$ $\exists ip.(anomalous(ip)) \longrightarrow belief(\neg regularUsage(link))$ $\nexists ip.(anomalous(ip)) \longrightarrow belief(regularUsage(link))$
Plan: RestoreLinkRate (RLR) **Goal**: $fullyOperational(link)$ **Context**: $regularUsage(link)$ **Actions**: $\texttt{unlimit}(link)$ $belief(fullyOperational(link))$ $belief(\sim attackPrevented(link))$	**Plan**: LimitIPRate (LIPR) **Goal**: $restricted(ip)$ **Context**: $anomalous(ip)$ **Actions**: $\texttt{limit}(ip, rate)$ $belief(ipRateLimited(ip))$ $belief(restricted(ip))$ $\nexists ip'.(anomalous(ip') \wedge \neg restricted(ip')) \longrightarrow regularUsage(link)$
Plan: RecordFlow (RF) **Goal**: $flowRecord(ip)$ **Context**: - **Actions**: $\texttt{recordFlow}(ip)$ $belief(flowRecord(ip))$	**Plan**: AnalyseIPFlows (AIPF) **Goal**: $?benign(ip)$ **Context**: $anomalous(ip)$ **Actions**: $goal(?flowRecord(ip))$ *// Classify flow record using machine learning* $\forall flow.(malicious(flow)) \longrightarrow belief(threat(flow))$ $\exists flow.(threat(flow) \wedge srcIP(flow) = ip) \longrightarrow belief(\neg benign(ip))$ $\nexists flow.(threat(flow) \wedge srcIP(flow) = ip) \longrightarrow belief(benign(ip))$
Plan: RestoreIPRate (RIPR) **Goal**: $\neg restricted(ip)$ **Context**: $benign(ip) \wedge$ $ipRateLimited(ip)$ **Actions**: $\texttt{permit}(ip)$ $belief(\neg ipRateLimited(ip))$ $belief(\neg restricted(ip))$ $belief(\sim anomalous(ip))$	**Plan**: LimitFlowRate (LFR) **Goal**: $threatResponded(flow)$ **Context**: $threat(flow)$ **Actions**: $\texttt{limit}(flow, rate)$ $belief(flowRateLimited(flow))$ $belief(threatResponded(flow))$ $belief(\sim threat(flow))$ $\nexists f.(threat(f) \wedge srcIP(flow) = srcIP(f)) \longrightarrow belief(benign(ip))$

executes first the LLR plan to achieve $attackPrevented(link)$, causing the link to be partially operational ($\neg fullyOperational(link)$). Then, the ALS plan is executed to achieve $?regularUsage(link)$ and performs statistical analysis of the link usage, concluding that there is an IP address with an anomalous behaviour ($anomalous(ip)$), which may be benign or malicious ($\sim benign(ip)$). Because now the agent believes that $anomalous(ip)$, it also believes $\neg regularUsage(link)$. Given that the agent knows that there is an IP address with anomalous behaviour, it generates goals to restrict it ($restricted(ip)$) and to know whether it is benign ($?benign(ip)$). So the LIPR plan is executed to achieve the former, and the AIPF to achieve the latter. Restricting the only IP address that is anomalous causes the agent to believe that $regularUsage(link)$ and, as a consequence, it generates the goal of becoming fully operational again. As this goal has a higher priority than that of the $?benign(ip)$ goal, the RLR plan is executed, restoring the link rate. To achieve $?benign(ip)$, the AIPF is executed. This plan uses a flow record, which is a belief that the agent does not currently have. So the AIPF plan adds a new goal to the agent, which executes the RF plan to obtain this information. Then, the AIPF plan continues its execution, by using a machine learning algorithm to classify flows (e.g., K-means, Naïve Bayes, SVM), and concludes that a particular flow is a threat ($threat(flow)$), and thus the IP address is malicious ($\neg benign(ip)$). Because there is a threat, the agent must achieve the goal $threatResponded(flow)$, and executes the LFR plan to do so. As the malicious flow is now limited, the IP address is considered benign and can be unrestricted. The agent generates the goal $\neg restricted(ip)$, and executes the RIPR plan to achieve it. In the

Table 2. Example of belief base updates given a DDoS attack.

Belief	Start	LLR	ALS	LIPR	RLR	AIPF(1)	RF	AIPF(2)	LFR	RIPR
$overUsage(link)$	p	p	$\neg$[1]	$\neg$	$\neg$	$\neg$	$\neg$	$\neg$	$\neg$	$\neg$
$regularUsage(link)$	$\sim$	$\sim$	$\neg$	p	p	p	p	p	p	p
$fullyOperational(link)$		$\neg$	$\neg$	$\neg$	p	p	p	p	p	p
$attackPrevented(link)$		p	p	p	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	p
$anomalous(ip)$			p	p	p	p	p	p	p	$\sim$
$restricted(ip)$				p	p	p	p	p	p	$\neg$
$ipRateLimited(ip)$				p	p	p	p	p	p	$\neg$
$benign(ip)$			$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\neg$	p	p
$flowRecord(ip)$							p	p	p	p
$threat(flow)$								p	$\sim$	$\sim$
$flowRateLimited(flow)$									p	p
$threatResponded(flow)$									p	p

[1] Updated due to belief revision.

end, the agent is fully operational and the malicious flow is limited.

This evolution of agent beliefs over time due to the execution of plans is summarised in Table 2. As mentioned before, we assume that the agent has as initial beliefs the knowledge that the link is being overused, and does not know whether this is due to regular users. Then, each column represents the current agent beliefs after the execution of the plan that is indicated in the column title: (i) p: belief is true; (ii) $\neg$: belief is false; and (iii) $\sim$: it is unknown whether the belief is true. Grey cells indicate belief changes.

Two further observations can be made with respect to these plans. First, plans are independent from each other, and there is no action that invokes any of the plans. Current agent beliefs and goals are the factors that lead to plan execution. Therefore, Table 2 shows just an example of a sequence in which plans are executed. Second, there are beliefs — $ipRateLimited(ip)$ and $flowRateLimited(flow)$ — that are currently not explicitly used in our proposed design. They are used to distinguish the intervention made (such as limiting an IP address) from an ultimate goal (such as restricting an IP address). There may be different possible interventions (i.e. plans) to achieve a certain ultimate goal and, if we provided these alternative plans, agents may choose the best according to a certain scenario, or use them in case a plan fails.

3.2. Multi-agent Architecture

In previous sections, we showed components that are used to compose a BDI agent, and illustrated a scenario in which we have a single agent with all these presented components. Although this single agent provides the whole solution to respond to DDoS attacks, due to hardware restrictions, it is more adequate to distribute responsibilities among different network devices. For example, running traffic classification algorithms requires significant RAM memory and a powerful processor, and it is thus convenient to have a dedicated machine to perform this task.

We will consider a network configuration that is composed of the following devices: (i) *application servers*: responsible for running applications; (ii) *IP load balancers*: responsible for distributing user requests among different replicated application servers; (iii) *routers*: responsible for forwarding network packets to their proper destination; (iv) *firewall*: responsible for blocking or not the flow of network traffic; and (v) *classifiers*: responsible for analysing flows and classifying them as malicious or benign. As can be seen in Table 1, there are some plans that involve some primitive actions, such as limiting

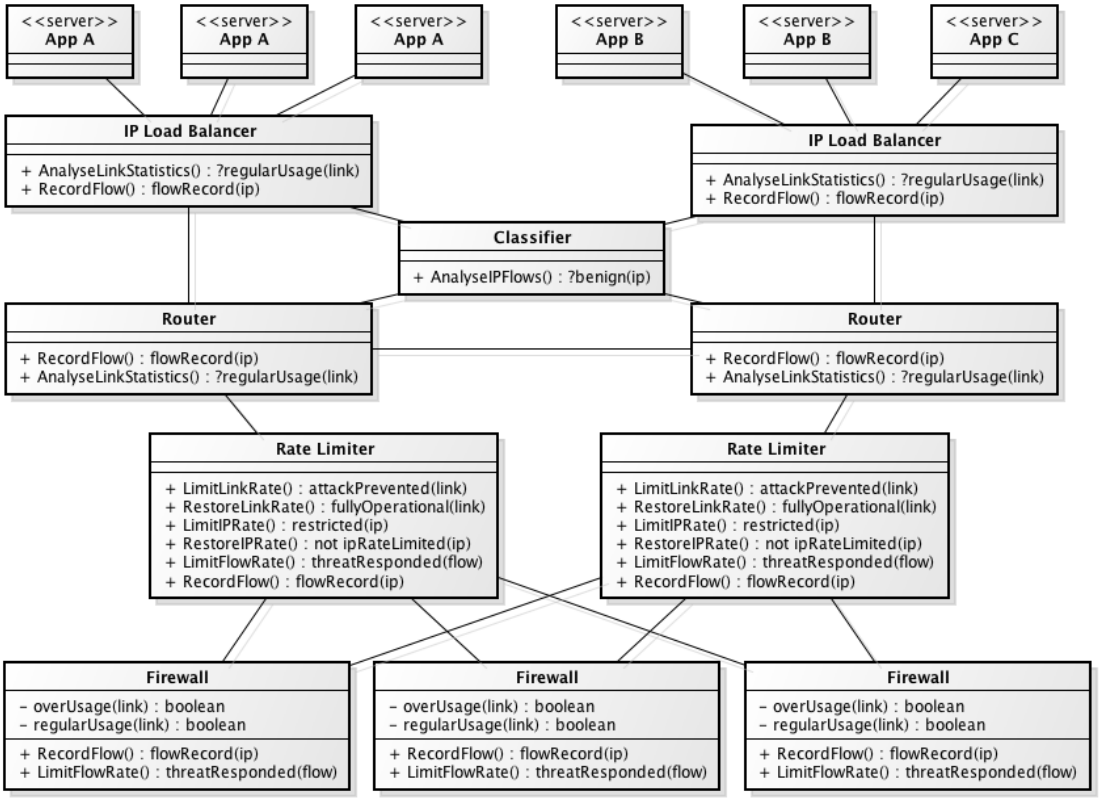

Figure 2. Configuration of network devices with Initial beliefs and plans.

traffic. Therefore, these plans must be added only to agents (i.e., network devices) that are able to perform such actions. We show an example of network topology, together with the distribution of plans of Table 1 among agents in Figure 2.

As no agent has all plans, they are not able to achieve all goals. Agents must, thus, collaborate to respond to attacks. Due to space restrictions, we limit ourselves to a broad description of how agents collaborate to do so. When an agent is unable to achieve a goal (i.e., it has no plan to achieve it), it will request other agents of the network to achieve them. Other agents reply to this request by answering whether they can achieve the requested goal and an associated cost. This cost can be associated with the current agent availability in terms memory or processor, or physical location, when a significant amount of data will be exchanged between agents. Such costs allow the agent that made the request to decide to which agent it will delegate the goal. This process can be performed using the FIPA Contract Net Interaction Protocol[1], for example.

4. Discussion

This paper proposes reengineering an existing solution that addresses DDoS attacks. Therefore, our agent-based design is expected to achieve similar performance than the original solution in standard scenarios. As said in the introduction, our main goal with this new design is to investigate, in the form of an exploratory study, the use of a BDI architecture to provide network resilience, and its impact in the design. We thus in this section provide a qualitative discussion of major issues that were identified.

[1]http://www.fipa.org/specs/fipa00029/

Table 3. Comparison of adopted terms.

BDI Agent-based approach	SMC-based approach
External Event	Event
Goal	
Plan Context	Condition
Plan	Action
Organisation	Management Pattern
Roles	Roles
Interaction Protocols	Interface

Table 4. Strengths (+) and weaknesses (-) of each approach.

BDI Agent-based Approach	SMC-based Approach
(-) It is harder to understand its behaviour, because there are different factors that influence it (current beliefs, generated goals, etc.).	(+) It has a more predictable behaviour, because it is easier to analyse the chain of events generated by each ECA policy, thus facilitating testing.
(+) It has a flexible behaviour, because of the different factors that influence agent behaviour, thus allowing agents to handle situations unpredicted at design time.	(-) It is able to handle only situations predicted at design time.
(+) As it captures motivational state, an agent may try different plans until it achieves a goal.	(-) It does not capture motivational state, therefore trying different alternatives to solve an issue must be implemented manually.
(+) It isolates concerns: (i) informational state and how it is updated based on perceived events; (ii) motivational state; (iii) actions to achieve goals.	(-) All these concerns are tangled in ECA policies.
(+) Plugging a new device in the network does not require human intervention to make it collaborate with other devices.	(-) When a new device is added to the network, an expert should manually re-assign roles to devices.

Terminology. Both the pattern-based approach used as reference in this work and the BDI architecture consider autonomous software components as building blocks. Nevertheless, such software components (SMCs and BDI agents) are structured with different finer-grained components. With respect to this, we identified several similarities among these components, as detailed in Table 3. As can be seen, most of the concepts exist in both approaches. The key difference is the goal concept, which specifies a motivational state. This makes SMCs reactive, while agents are pro-active. Some agent concepts — organisation, roles, and interaction protocols — are not explored in this paper, but there are agent approaches that address agent organisations [Zambonelli et al. 2001].

Strengths and Weaknesses. Although many concepts used in the BDI architecture can be mapped to those in the SMC-based approach, the BDI reasoning cycle and the use of the goal abstraction make the two designs significantly different. We identified strengths (+) and weaknesses (-) of each of these approaches, which are summarised in Table 4.

One of the weaknesses of using the BDI-based approach is the difficulty in testing the system (in fact, this is a challenge of multi-agent system development), because agent behaviour depends on many different factors and it is hard to perform a limited number of tests to guarantee that the network will behave adequately in all scenarios. However, because of the adoption of a logic-based language, it is possible to prove desired properties. This is left out of the scope of this paper.

Open Issues. Based on an existing pattern for responding to a possible DDoS attack, we proposed a design solution based on BDI agents. Although it provided a flexible implementation of the proposed pattern, there are many issues left unaddressed. Firstly, as discussed above, guaranteeing safety is not trivial, because testing all possible scenarios is unfeasible. Secondly, the proposed solution should consider that malicious users may learn how threats are contained, thus exploiting the resilience strategy for their own benefit. The proposed solution must be robust enough to prevent this. Finally, there should be further investigation on how different resilience mechanisms would interact using a BDI agent approach. For example, if an agent concluded that there is a benign peak of network usage (e.g., as in a *flash crowd*), actions to handle this should also be implemented.

5. Related Work

Several approaches use autonomous software components to provide network resilience. However, agent-based designs [Nguengang et al. 2006] only decompose a proposed solution into software components, each with a specified goal. Thus, they typically do not employ any particular technique proposed in the context of autonomous agents and multi-agent systems.

Furthermore, a *sentient object* [Gregory et al. 2002] resembles an SMC in that both are intended to model a set of interacting hardware and software components, and provide an infrastructure to support large-scale distributed and networked systems composed of autonomous components. However, sentient objects rely on static rules, so they are less flexible than SMCs. Also, sentient objects follow a structure that cannot be dynamically assembled using more general patterns of interaction [Schaeffer-Filho et al. 2012]. Netlets [Martin et al. 2011] and autonomic functional blocks (AFBs) [Sifalakis et al. 2011] have been proposed with the aim of supporting new architectures for the Future Internet. Both are inspired by component-based software development approaches. They promote the composition of network protocols by using building blocks during design-time. Components are collected within a design repository for further reuse. However, both Netlets and AFBs components are aimed at the specification of network stack protocol functionality and do not cater for the general federation and coordination of autonomous components operating in the network.

6. Conclusion

In this paper, we presented the first step towards an agent-based approach for network resilience. It exploits the synergy between the network resilience and multi-agent system research areas. Our approach involves the reengineering of a previously proposed pattern using Self-Mananged Cells (SMCs) to address distributed denial-of-service attacks using agents following a BDI architecture and a logic-based language. The proposed design not only specifies agent beliefs, goals, and plans, but also how beliefs are updated based on perceived external events and how goals are generated. We showed how these BDI agent components are used to instantiate agents (or network devices) in a network, so that such agents can collaborate to respond to attacks. Based on our design, we discussed lessons learned from it. Although there are many similar concepts between SMCs and BDI agents, BDI agents have a motivational state, which makes them pro-active (while SMCs are reactive). Moreover, BDI agents are able to reason about what should be done, not only about how it should be done. These, and other discussed factors, make agent

behaviour more flexible. This work has left many issues unaddressed, such as proving safety properties of our solution, which will be addressed in future work.

References

Gregory, A. F., Biegel, G., Clarke, S., and Cahill, V. (2002). Towards a sentient object model. In *In Workshop on Engineering Context-Aware Object Oriented Systems and Environments (ECOOSE'2002)*.

IBM (2005). An architectural blueprint for autonomic computing. third edition. Technical report, IBM.

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41.

Lupu, E., Dulay, N., Sloman, M., Sventek, J., Heeps, S., Strowes, S., Twidle, K., Keoh, S.-L., and Schaeffer-Filho, A. (2008). AMUSE: autonomic management of ubiquitous systems for e-health. *Concurrency and Computation: Practice and Experience, John Wiley*, 20(3):277–295.

Martin, D., Volker, L., and Zitterbart, M. (2011). A flexible framework for future internet design, assessment, and operation. *Computer Networks*, 55(4):910–918.

Nguengang, G., Hugues, L., and Gaiti, D. (2006). A multi agent system approach for self resource regulation in ip networks. In *Proceedings of the First IFIP TC6 International Conference on Autonomic Networking*, AN'06, pages 64–75. Springer-Verlag.

Nguyen, T. and Armitage, G. (2008). A Survey of Techniques for Internet Traffic Classification using Machine Learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76.

Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-agent World : Agents Breaking Away: Agents Breaking Away*, MAAMAW '96, pages 42–55. Springer-Verlag.

Rao, A. S. and Georgeff, M. P. (1995). BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco.

Schaeffer-Filho, A., Lupu, E., and Sloman, M. (2014). Federating policy-driven autonomous systems: Interaction specification and management patterns (in press). *Journal of Network and Systems Management*. DOI: 10.1007/s10922-014-9317-5.

Schaeffer-Filho, A., Smith, P., Mauthe, A., Hutchison, D., Yu, Y., and Fry, M. (2012). A framework for the design and evaluation of network resilience management. In *Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS 2012)*, pages 401–408, Maui, Hawaii, USA. IEEE Computer Society.

Sifalakis, M., Louca, A., Bouabene, G., Fry, M., Mauthe, A., and Hutchison, D. (2011). Functional composition in future networks. *Computer Networks*, 55(4):987–998.

Sloman, M. and Lupu, E. (2002). Security and management policy specification. *IEEE Network*, 16(2):10–19.

Smith, P., Hutchison, D., Sterbenz, J., Schöller, M., Fessi, A., Karaliopoulos, M., Lac, C., and Plattner, B. (2011). Network resilience: a systematic approach. *Communications Magazine, IEEE*, 49(7):88–97.

Sterbenz, J. P. G., Hutchison, D., Çetinkaya, E. K., Jabbar, A., Rohrer, J. P., Schöller, M., and Smith, P. (2010). Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks: Special Issue on Resilient and Survivable Networks (COMNET)*, 54(8):1245–1265.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2001). Organizational abstractions for the analysis and design of multi-agent system. In *First International Workshop, AOSE 2000 on Agent-oriented Software Engineering*, pages 235–251. Springer-Verlag.