Journal of Network and Computer Applications 84 (2017) 14–24

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

BDI2DoS: An application using collaborating BDI agents to combat DDoS attacks

Ingrid Nunes[a,b], Frederico Schardong[a,*], Alberto Schaeffer-Filho[a]

[a] Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
[b] Department of Computer Science, Technische Universität Dortmund, Dortmund, Germany

ARTICLE INFO

Keywords:
Network resilience
Multi-agent systems
BDI architecture

ABSTRACT

Computer networks are critical to many tasks in our daily lives. Therefore, mechanisms that guarantee the resilience of the network must be provided. Different approaches have been proposed to address potential challenges to network operation, but they rely on rigid solutions, which work only in anticipated scenarios. To address this issue, this paper presents BDI2DoS, which is a multi-agent innovative application that ensures the resilience of networks against the widely known Distributed Denial-of-Service (DDoS) attack. We take an existing network resilience strategy based on event–condition–action (ECA) policies to combat DDoS attacks, and use it as a requirement to specify the behaviour that must emerge from the interaction among agents, which together are capable of detecting and remediating anomalies that are considered a DDoS threat, in a flexible way. Agents in our multi-agent system follow the widely used BDI architecture, and were implemented with the BDI4JADE agent platform. In order to evaluate the effectiveness of BDI2DoS, we used the PReSET resilience simulator and BDI4JADE to build an integrated testbed. Our experiments compare the effectiveness of the ECA and the BDI-based resilience strategies and show that the latter is more flexible and able to cope with problems that occur during the execution of the anticipated behaviour.

1. Introduction

Network resilience is the key to provide reliable, robust and efficient network operation. Resilience is the ability of the network to maintain an acceptable level of operation when confronted with *challenges*, *e.g.*, equipment failures, device misconfiguration, or malicious attacks (Smith et al., 2011; Sterbenz et al., 2010; Najjar and Gaudiot, 1990). Ensuring network resilience has become critical nowadays given that many tasks in our daily lives rely on networked infrastructures.

Research on network resilience has combined contributions from different research areas, mainly machine learning (Nguyen and Armitage, 2008) and autonomic computing (Lupu et al., 2008; Tcholtchev et al., 2010). Some approaches provided agent-based solutions (Jiang and Jiang, 2005; Nguengang et al., 2006; Yang and Chang, 2011), in which agents collaborate in a sequential pre-defined way. An agent (Ferber, 1999; Wooldridge, 2009) is a software component with autonomous and proactive behaviour, situated in an environment and with social ability. However, these approaches fall short of *emergent behaviour*, which is a key property of multi-agent systems. Moreover, there are many unexploited domain-neutral agent-based approaches, which can arguably provide flexible solutions to handle situations unpredicted at design time (Jennings, 2001).

In this paper we exploit one particular agent-based approach to achieve network resilience: the belief-desire-intention (BDI) architecture (Rao and Georgeff, 1995), which separates the motivational state of a system from its deliberative state. Additionally, given that goals are explicitly represented, alternative ways of achieving them may be executed, when there is a failure in the execution. Using the BDI architecture, we developed BDI2DoS, an innovative application composed of a set of agents that combined form multi-agent collaborations comprising mechanisms capable of detecting and remediating *Distributed Denial-of-Service* (*DDoS*) attacks. In particular, we take an existing network resilience strategy based on event–condition–action (ECA) policies to combat DDoS attacks (Schaeffer-Filho et al., 2012), and use it as a basis to specify the behaviour that must emerge from the interaction among agents. Traditionally, ECA policies have been used in network management as a means of specifying system behaviour (Lupu et al., 2008; Charalambides et al., 2009; Damianou et al., 2001; Sloman and Lupu, 2002). The decision-making process is purely reactive, and policy actions are performed when specific events occur and if certain conditions are met. Differently from the pre-

* Corresponding author.
E-mail addresses: ingridnunes@inf.ufrgs.br (I. Nunes), fschardong@inf.ufrgs.br (F. Schardong), alberto@inf.ufrgs.br (A. Schaeffer-Filho).

http://dx.doi.org/10.1016/j.jnca.2017.01.035
Received 9 August 2016; Received in revised form 13 December 2016; Accepted 30 January 2017
Available online 02 February 2017

specified interactions defined by ECA policies, a key issue associated with the BDI2DoS design is how to identify the possible local interactions that must occur between agents in order to make the application detect and contain the attack.

Our design consists of a set of capabilities that can be added to agents. Each capability has a set of rules used as part of the agent reasoning process together with a set of plans to achieve goals. Although plans are simple, they effectively provide agent coordination, and it is the interplay among decoupled agent parts (rules and plans) that provides the emergent behaviour. Also, multiple plans can achieve the same goal by different means, challenging agents to select the best plan considering their current knowledge about themselves and their surroundings. The key advantages of BDI2DoS in comparison with the ECA-based resilience strategy are: (i) when an agent is unable to achieve a goal or fails while trying to achieve it, the agent may seamlessly request other agents to achieve this goal, providing robustness to deal with failures; (ii) if something is not done properly while combating an attack, *e.g.*, a malicious flow is misclassified as benign, agents will keep combating the attack because new goals are generated due to the proactive agent nature; and (iii) new agents with specific capabilities can be added to the network (even at runtime) without requiring additional configuration. We not only designed BDI2DoS, but also implemented and evaluated it using a testbed integration between the BDI4JADE (Nunes et al., 2011) agent platform, and the PReSET (Schaeffer-Filho et al., 2013) resilience simulator. This allowed us to run realistic simulations with large-scale networks, which indicate the effectiveness of our approach. As a result of the work presented in this paper, we provide two main contributions: (i) a flexible agent-based solution to combat DDoS attacks and (ii) an experimental comparison of the effectiveness of the ECA and the BDI-based resilience strategies. To the best of our knowledge, this is the first time that ECA and BDI are compared in a simulation environment in order to assess their ability to successfully mitigate malicious network traffic.

The remainder of this paper is organised as follows. Section 2 describes the BDI architecture and the ECA-based resilience strategy to prevent DDoS attacks used as baseline. We then detail the design of BDI2DoS in Section 3, and its implementation and evaluation in Section 4. We discuss related work in Section 5, and finally conclude the paper in Section 6.

2. Background

Several models and techniques were proposed to help design and implement software agents. In this section, we introduce one of these techniques, namely the BDI architecture, which is adopted in this work. We also describe a typical ECA-based resilience strategy against DDoS attacks, which is used to inspire our approach.

2.1. BDI architecture

The BDI architecture is an abstract implementation of the theory of practical reasoning proposed by Bratman et al. (1988), which specifies a reasoning cycle that determines agent behaviour. This architecture structures agents in terms of beliefs, desires and intentions that are used within this reasoning cycle. *Beliefs* consist of the information an agent has about the world. They do not necessarily represent the state of the environment, but rather the view an agent has about its surroundings. Desires, also referred to as *goals*, represent what an agent wants to achieve. Goals can be either explicitly created by system designers or generated at runtime. Finally, *intentions* are goals that an agent is committed to achieve. Intentions are associated with *plans*, which are structured sequences of actions to achieve goals, given that an agent can be committed to achieve a goal only if there is a means of achieving it.

These BDI components allow separation of what an agent wants to achieve (goals) from how a goal is achieved (plans). Multiple plans may be available to achieve the same goal. Consequently, if a plan fails to achieve a goal, another plan may be selected and executed, by the reasoning cycle, to do so. When there are multiple plans to achieve a goal, a *plan selection function* may be specified to choose the most adequate plan, instead of randomly choosing plans. The BDI architecture is abstract in the sense of letting its instances to fulfil gaps, such as the plan selection function. There are approaches (Singh et al., 2010; Nunes and Luck, 2014; Guerra-Hernández et al., 2004) that proposed alternative implementations of this function.

In this work, we leverage one of these approaches, which is suitable to our problem. The approach proposed by Nunes and Luck (2014) provides plans with metadata in order to calculate the plan with the maximum utility to an agent. It relies on the multi-attribute utility theory (MAUT) (Keeney and Raiffa, 1993) to select plans. In this approach, agents are provided with *softgoals*, which are goals that can be more or less satisfied, typically associated with plan side effects, such as those related to non-functional requirements (*e.g.* maximise performance). Plan actions have different outcomes, which lead to possible contributions to each softgoal, each with its probability. Therefore, plans may have a set of contributions c_i associated with a probability pr_i, with respect to a given softgoal, and c_i, $pr_i \in [0, 1]$ and $\sum_i pr_i = 1$. Based on this, each plan may have expected contributions according to different softgoals, which is $\sum_i pr_i \times c_i$. An agent has preferences over softgoals, represented by values $p_i \in [0, 1]$ and $\sum_i p_i = 1$. The plan utility is a weighted sum of the expected plan contribution associated with each softgoal, weighted by the agent preferences over it. The plan with the maximum utility is selected. For further details, we refer the interested reader to elsewhere (Nunes and Luck, 2014).

2.2. DDoS resilience strategy

Resilience is the ability of the network to maintain acceptable levels of operation in the face of challenges, such as malicious attacks, operational overload, or equipment failures (Sterbenz et al., 2010). Central to a *resilience strategy* is the management and reconfiguration of detection and remediation mechanisms, which operate as autonomous components in the network infrastructure. Detection mechanisms such as link monitors, anomaly detection systems and traffic classifiers permit the identification and categorisation of anomalies and attacks to the network. Remediation mechanisms, in turn, can be used in the subsequent mitigation of these challenges, *e.g.*, rate limiters, load balancers and firewalls. The various mechanisms can be deployed and activated on different parts of the network topology.

Previous work (Schaeffer-Filho et al., 2012) investigated how to organise network mechanisms providing detection and remediation functionality to combat specific types of network attacks, such as DDoS and worm propagations. Mechanisms can be configured and federated into resilience strategies to address a specific type of challenge or attack, *e.g.*, a strategy for combating a DDoS as depicted in Fig. 1. This strategy consists of a number of stages, detailed next, which are triggered by declarative *event–condition–action* (ECA) policies according to the conditions monitored in the network infrastructure.

In the ECA-based strategy presented in Schaeffer-Filho et al. (2012), a *Link Monitor* monitors the incoming traffic rate on the ingress link and raises an alarm (*notify_load*) in case the traffic volume observed exceeds a given threshold. On receiving this event, an initial rate limiting of the link can be performed by a *Rate Limiter*. Policies are deployed to filter a percentage of all incoming traffic in order to protect downstream servers. In addition to the preventive rate limiting started at this point, an *Anomaly Detection* component can further start processing packets to gather more evidence about the traffic anomaly and identify the destination IP address of the potential victim of the attack. As a result, it can raise an event (*notify_detection*) when one destination IP address accounts for a large portion of all packets, *e.g.*, 60%. This additional evidence permits the reconfiguration of the

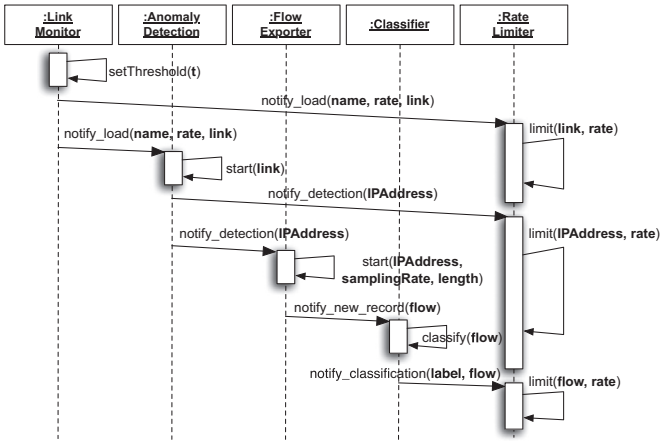


Fig. 1. DDoS resilience strategy.

Rate Limiter to drop a larger portion of the network traffic destined to the victim only. Legitimate traffic that is not destined to the victim is now allowed to go through. If the attack is sustained for a longer period, the *Classifier* and *Flow Exporter* mechanisms are started. In particular, the *Classifier* attempts to identify the specific attack flows, among all traffic flows gathered by the *Flow Exporter*. Classification events are generated (*notify_classification*), and rate limiting is confined to the specific attack flows only, whereas legitimate traffic is now allowed to continue unthrottled.

3. BDI2DoS design

The DDoS resilience strategy presented in the previous section requires a pre-specified arrangement of components and the anticipation of their interactions via reactive ECA policies. In this section, we present a new design, in which devices that run resilience mechanisms are associated with agents that follow the BDI architecture. This design, implemented in the BDI2DoS application, not only allows dynamic instantiation of components without requiring human intervention, but also captures motivational state while preventing the attack. Thus, actions are proactively performed until the attack is prevented.

3.1. BDI2DoS Ontology

We first detail in this section the ontology that specifies the concepts and predicates that can be part of agent knowledge bases. Agents in BDI2DoS have a belief base that captures the informational state of each agent. Beliefs in the base have the semantics specified in the BDI2DoS Ontology. Table 1 details the concepts in this ontology.

3.2. BDI agent capabilities

BDI agents are often structured in terms of three main components, namely beliefs, goals and plans. The concept of *capability* (Busetta et al., 2000) was proposed to support modularisation and reuse in BDI agents, by putting a set of beliefs, goals and plans in a module (i.e., capability) that can be added to agents or combined with other capabilities. BDI2DoS largely relies on the capability concept: different agent parts are specified as capabilities, and agents are built as an aggregation of capabilities. In this section, we detail the different BDI2DoS capabilities, which are split into two groups: reactive capabilities and proactive capabilities.

3.2.1. Reactive capabilities

BDI agents may have goals that they are committed to achieve, and execute actions to achieve them. Two of the BDI2DoS capabilities do not generate goals but have the ability to achieve goals that other capabilities cannot. These are the *Rate Limiter* and *Flow Exporter* reactive capabilities, which serve as services needed by other capabilities.

Table 1
BDI2DoS Ontology.

Concept	Meaning
link(*id*)	Link in the network topology, identified by an id, which connects two devices
ip(*address*)	Address of a network device, according to the Internet Protocol
flow(*sIp*, *dIp*, *pro*)	An information flow that is transmitted from a device with a source IP address *sIp* to a device with a destination IP address *dIp*, following the protocol *prot*
anomalous(*ip*)	The information flow being sent to the device with IP address *ip* has an anomalous behaviour (but not necessarily malicious)
anomalousUsage(*link*)	Network traffic in the link is anomalous
attackPrevented(*link*)	A possible attack in a link is prevented
benign(*ip*)	The information flow being sent to the device with IP address *ip* is benign
flowRateLimited(*flow*)	The traffic rate of a flow is limited
ipRateLimited(*ip*, *rate*)	The traffic rate destined to a device with IP address *ip* is limited to a rate *rate*
linkRateLimited(*link*)	The traffic rate in an entire link is limited
overUsage(*link*)	The traffic of a link is above a given threshold
overUsageCause(*link*, *ip*)	The traffic to the device with IP address *ip* is a cause for the link to be overused
restricted(*ip*)	The access to a device with IP address *ip* is restricted
threat(*flow*)	A traffic flow is a threat to the network

Rate limiter: Preventing a flooding attack such as a DDoS requires limiting network traffic in different granularities and under specific circumstances. The rate limiter capability has a set of plans to perform this, which are detailed in Table 2. When such plans are executed, information about the current state of the network is updated by the addition or removal of beliefs. Each plan has a name and is described in terms of: (i) the *goal* it can achieve; (ii) the required *context* for the plan to be executed; and (iii) the set of *actions* that comprise the plan body. Belief actions, *belief*(*b*), indicate that a belief *b* is being added (*belief*(*b*) or *belief*(¬ *b*)) or removed (*belief*(*not b*)) from the belief base. Primitive actions, `action`(param), are agent native actions, which can be seen as agent actuators.

The rate limiter capability can limit and unlimit a link (LLR and ULR plans), limit and unlimit the traffic for a device with a specified IP address (LIR and UIR plans), and limit a specific flow (LFR plan). Limiting actions receive as parameter a *RATE*, which indicates the percentage of the traffic that is allowed to be transmitted. For links and flows, we assume that this rate is fixed and specified at design time,

Table 2
Plans of the rate limiter capability.

Plan: LimitLink (LL)
Goal: *linkRateLimited*(*link*)
Context: –
Actions:
`limitLink`(*link*, *RATE*)
belief(*linkRateLimited*(*link*))

Plan: UnlimitLink (UL)
Goal: ¬*linkRateLimited*(*link*)
Context: *linkRateLimited*(*link*)
Actions:
`unlimitLink`(*link*)
belief(¬ *linkRateLimited*(*link*))

Plan: LimitIP (LIP)
Goal: *ipRateLimited*(*ip*, *rate*)
Context: –
Actions:
`limitIP`(*ip*, *rate*)
belief(*ipRateLimited*(*ip*))

Plan: UnlimitIP (UIP)
Goal: ¬*ipRateLimited*(*ip*)
Context: *ipRateLimited*(*ip*)
Actions:
`unlimitIP`(*ip*)
belief(¬ *ipRateLimited*(*ip*, *rate*))

Plan: LimitFlow (LF)
Goal: *flowRateLimited*(*flow*)
Context: –
Actions:
`limitFlow`(*flow*, *RATE*)
belief(*flowRateLimited*(*flow*))

Table 3
Plans of the flow exporter capability.

Plan: RecordFlow (RF)
Goal: *flowRecord*(*ip*) **Context**: – **Actions**: `recordFlow`(*ip*) *belief*(*flowRecord*(*ip*))

while for IP addresses this is specified as part of the goal—this feature is used later.

Flow exporter: The second reactive capability is the flow exporter. In order to identify whether an anomalous traffic is malicious or benign, it is necessary to record the network traffic flows. Therefore, the flow exporter capability has a plan, detailed in Table 3, that enables it to accomplish this task.

3.2.2. Proactive capabilities

BDI2DoS has three proactive capabilities, which we refer to as *controllers*. These capabilities must monitor and control the network at different granularity levels: link, IP address and flow. Besides the presented plans, proactive capabilities may also have specific implementations of customisable steps of the BDI reasoning cycle. These steps are: (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 ones they are committed to achieve; and (iv) *plan selection*: agents select the most suitable plan to accomplish the goal it has committed to achieve.

Note that plans presented here have a new kind of action: goal actions. These actions add (sub)goals to the agent. *goal*(*b*) adds the goal of having the predicate *b* as part of the agent belief base (*b* may also be ¬*b*). *goal*(? *b*) adds the goal of having *b* or ¬*b* as part of the agent belief base, i.e., if *b* is unknown, the agent must find out whether *b* or ¬*b* is true.

Link controller: Network links transport traffic flows between devices. Under normal conditions, only part of the link bandwidth is used, and the link usage should always be below a given threshold. A link controller capability has two main goals: (i) monitor links to detect an anomalous usage and (ii) keep links fully operational under normal conditions.

The link controller perceives events associated with links through monitoring probes. The event *event*(*overUsage*(*link*)) indicates that a link whose traffic was below a given threshold now is above the threshold, while *event*(¬ *overUsage*(*link*)) indicates the opposite. When such events occur, beliefs are updated accordingly during the belief revision step. These updates are shown below, in the form of rules, in which the expression on the right-hand side of the symbol → takes place when the expression on the left-hand side holds:

$$\begin{aligned} event(overUsage(link)) \rightarrow\ & belief(overUsage(link)) \\ & \wedge\ belief(not\ \ anomalousUsage(link)) \\ & event(\neg\ overUsage(link)) \\ & \rightarrow belief(not\ \ overUsage(link)) \end{aligned}$$

When the traffic volume for a given link is above the established threshold, there are two possibilities: this is due to either a peak in network usage, deviating its behaviour from the usual, or due to a malicious attack. Consequently, when agents perceive the event *event*(*overUsage*(*link*)), their belief base has an additional update (*not anomalousUsage*(*link*)).

As said above, 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 generate goals. A goal is associated with a state of affairs that an agent wants to achieve. Two goals are generated because an agent believes that *overUsage*(*link*). Firstly, though the agent is unsure whether an attack is indeed occurring, it needs to prevent a possible attack, thus generating a goal *attackPrevented*(*link*). Secondly, because the agent does not know whether *overUsage*(*link*) is a malicious attack, it needs to find this out, by generating a goal ?*anomalousUsage*(*link*). This goal generation is shown below:

$$\begin{aligned} & overUsage(link) \wedge not\ \ anomalousUsage(link) \\ & \quad \rightarrow goal(?\ anomalousUsage(link))overUsage(link) \\ & \quad \wedge \neg\ attackPrevented(link) \quad \wedge (anomalousUsage(link) \\ & \quad \vee not\ \ anomalousUsage(link)) \quad \rightarrow goal(attackPrevented(link)) \end{aligned}$$

Finally, ideally, links should always be fully operational. As a consequence, if an agent believes that a link is not fully operational, i.e., *attackPrevented*(*link*) holds, it must have a goal to make them fully operational. However, this goal is generated only when it is safe to do so, as detailed next:

$$\begin{aligned} & attackPrevented(link) \wedge \neg\ anomalousUsage(link) \\ & \quad \rightarrow goal(\neg\ attackPrevented(link)) \end{aligned}$$

In general, the goals *goal*(? *anomalousUsage*(*link*)) and *goal*(*attackPrevented*(*link*)) are generated together. Given that the ultimate goal is to guarantee that the network remains operational when confronted with a challenge, it is important to protect the network before finding out if an anomalous behaviour is due to a malicious attack. This prioritisation between the two goals is performed in the goal deliberation step. The prevention of an attack is a (palliative) action that must be performed before learning whether there is an attack or finding its root cause.

To achieve the goals presented above, the link controller has two plans, detailed in Table 4, which put and remove an agent in a state of preventing an attack. Currently, there is only one implemented form of preventing an attack, which is *throttling* (*i.e.*, rate limiting) the network traffic. Note that both the goals of limiting the link rate in the plans and *goal*(? *anomalousUsage*(*link*)) have no plans that achieve them within the link controller capability. The first can be achieved by the rate limiter capability, while the second by the IP controller capability, which is detailed next.

IP controller: An IP controller is similar to the link controller, but it throttles traffic at a finer granularity, *i.e.*, according to the destination IP address rather than entire network links. It has the goal of preventing attacks directed to particular IP addresses but, under safe conditions, all IP addresses must ideally be accessible with no restrictions. Hence, if an IP controller learns that an IP address is being targeted by anomalous traffic, *i.e.*, *anomalous*(*ip*), the IP controller should prevent an attack by restricting the traffic towards this IP address, and find out whether this anomalous behaviour is benign or malicious. These two goals, *restricted*(*ip*) and ?*benign*(*ip*), are generated as shown below:

$$\begin{aligned} anomalous(ip) \wedge (\ \neg\ benign(ip) \vee notbenign(ip)) \quad & \rightarrow goal(restricted(ip)) \\ anomalous(ip) \wedge notbenign(ip) & \\ \rightarrow goal(?\ benign(ip)) & \end{aligned}$$

Table 4
Plans of the link controller capability.

Plan: LimitLinkRate (LLR)	**Plan: RestoreLinkRate (RLR)**
Goal: *attackPrevented*(*link*) **Context**: **Actions**: *goal*(*linkRateLimited*(*link*)) *belief*(*attackPrevented*(*link*))	**Goal**: ¬*attackPrevented*(*link*) **Context**: *linkRateLimited*(*link*) **Actions**: *goal*(¬ *linkRateLimited*(*link*)) *belief*(¬ *attackPrevented*(*link*)) *belief*(*notanomalousBehaviour*(*link*))

Table 5
Plans of the IP controller capability.

Plan: AnalyseLinkStatistics (ALS)
Goal: ?*anomalousUsage*(*link*)
Context: –
Actions:
`detectIntrusion(link)`
$\forall\ ip.\ (outlier(ip) \rightarrow belief(anomalous(ip))$
$\wedge belief(not\ \ benign(ip))) \wedge belief(overUsageCause(link, ip))$
$\exists\ ip.\ (overUsageCause(link, ip)) \rightarrow belief(anomalousUsage(link))$
$\nexists ip.\ (overUsageCause(link, ip)) \rightarrow belief(\neg\ anomalousUsage(link))$

Plan: LimitIPRate_*Parameter* (**LIPR**_*Parameter*)
Goal: *restricted*(*ip*)
Context: –
Actions:
$goal(ipRateLimited(ip, rate))$
`detectIntrusion(link)`
$\nexists ip.\ (outlier(ip) \rightarrow belief(restricted(ip)))$
$\nexists ip.\ (outlier(ip) \wedge \exists\ link.\ (overUsageCause(link, ip) \wedge \nexists ip'.\ (ip \neq ip' \wedge$
$overUsageCause(link, ip')) \rightarrow belief(\neg\ anomalousUsage(link))$
$\nexists ip.\ (outlier(ip) \wedge \exists\ link.\ (overUsageCause(link, ip)$
$\rightarrow belief(not\ \ overUsageCause(link, ip)))$

Plan: RestoreIPRate (RIPR)
Goal: $\neg restricted(ip)$
Context: *ipRateLimited*(*ip*)
Actions:
$goal(\neg\ ipRateLimited(ip))$
$belief(not\ \ restricted(ip))$
$belief(not\ \ anomalous(ip))$

In addition, when the access to an IP address is restricted, a goal to enable full access to it is generated, but only when the traffic destined to that IP address is benign:

$$restricted(ip) \wedge benign(ip) \rightarrow goal(\neg\ restricted(ip))$$

Table 5 details the plans of IP controllers, which achieve these generated goals and the ?*anomalousUsage*(*link*) goal discussed in the previous section. The ALS plan makes statistical analysis of the traffic that flows in a link and, if there is a flow destined to an IP address that is an outlier, it is considered anomalous, and consequently the link usage is considered anomalous. The LIPR and RIPR plans are responsible for controlling the access to the IP address, similar to the plans that control the access to links of the link controller. The LIRP plan has, however, a particularity that is explained below.

For limiting IP addresses, we exploited the approach proposed by Nunes and Luck (2014), introduced in Section 2. The LIPR is an abstract plan, which has three instances, each throttling the traffic at different rates (High, Medium and Low). These plan instances have different outcomes, which impact in two softgoals: (i) *maximise performance* and (ii) *maximise protection*. With respect to the first softgoal, there is only one possible outcome: how much the traffic is throttled. The contribution to the softgoal is inversely proportional to the rate used in the plan, and as the traffic is always throttled, the probability that this occurs is 1.0. With respect to the second softgoal, there are two possible outcomes: the action performed can cause the server to keep being operational or it can cause the server to be unable to process requests. The contribution of the former outcome is 1.0 (highest utility) and of the latter outcome is 0.0 (worst utility). The probabilities are based on the history of the effectiveness of rates used. According to these contributions and probabilities, we specify in Table 6 their values for each plan, as well as their expected utility with respect to each softgoal.

Based on the values presented in Table 6, an agent with preferences over the softgoals *maximise performance* and *maximise protection* selects the plan with the maximum utility to execute first. That is, if its preference is stronger for maximising performance, it may select the plan with the lowest limiting rate first. This causes the traffic to the IP address to be throttled, but not necessarily *restricted*(*ip*) holds. In this plan, the action `detectIntrusion` is executed, in order to verify whether the IP address is not an outlier anymore. If this is the case, the plan ends in a successful state. If this is not the case, the plan fails, and the next candidate plan is executed in order to try to achieve the *restricted*(*ip*) goal (possibly with a higher limiting rate). This mechanism allows dynamic selection of rates to be used to throttle the traffic to an IP address, balancing performance and protection, according to an agent preferences.

Flow controller: Finally, a flow controller is responsible for monitoring and controlling network traffic flows, which are described in terms of a source IP address, a destination IP address, and a protocol. When a flow is considered a threat, the flow controller generates a goal to make this flow not a threat.

$$threat(flow) \rightarrow goal(\neg\ threat(flow))$$

The plans of a flow controller are presented in Table 7. The AIPF plan is able to determine whether the traffic destined to an IP address is benign with its primitive action `classifyFlows`, which uses a machine learning algorithm to classify flows (*e.g.*, K-means, Naïve Bayes, SVM). Flows classified as malicious are considered a threat, and if there is at least one flow that is a threat, the traffic destined to the IP address is considered malicious. These flows are limited by the LFR

Table 7
Plans of the flow controller capability.

Plan: AnalyseIPFlows (AIPF)
Goal: ?*benign*(*ip*)
Context: –
Actions:
$goal(?\ flowRecord(ip))$
`classifyFlows(ip)`
$\forall\ flow.\ (malicious(flow) \rightarrow belief(threat(flow)))$
$\exists\ flow.\ (threat(flow) \wedge dstIP(flow) = ip) \rightarrow belief(\neg\ benign(ip))$
$\nexists flow.\ (threat(flow) \wedge dstIP(flow) = ip) \rightarrow belief(benign(ip))$

Plan: LimitFlowRate (LFR)
Goal: $not\ \ threat(flow)$
Context: *threat*(*flow*)
Actions:
$goal(flowRateLimited(flow))$
$belief(not\ \ threat(flow))$
$\nexists flow'.\ (flow \neq flow' \wedge threat(flow')$
$\wedge dstIP(flow) = dstIP(flow')) \rightarrow benign(ip)$

Table 6
LIPR plan metadata: contributions, probabilities, expected utilities for each softgoal.

Softgoals		Maximise performance			Maximise protection				
Outcomes		Available bandwidth		Expected utility	Server is operational		Server is down		Expected utility
Plan	Rate	Probability	Contribution		Probability	Contribution	Probability	Contribution	
LIPR_High	70	1.0	0.3	0.3	0.9	1.0	0.1	0.0	0.9
LIPR_Medium	40	1.0	0.6	0.6	0.7	1.0	0.3	0.0	0.7
LIPR_Low	20	1.0	0.8	0.8	0.4	1.0	0.6	0.0	0.4

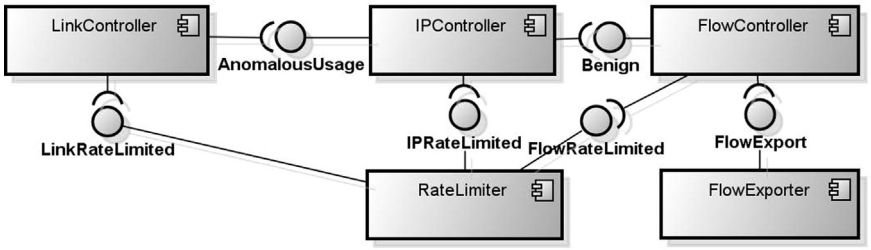


Fig. 2. BDI2DoS capability dependencies.

plan and, when all threats destined to an IP address are mitigated, the IP address is considered benign.

3.3. BDI2DoS multi-agent system

In the previous section, we showed the capabilities that are used as agent building blocks in BDI2DoS. In our application, each device that runs resilience mechanisms in the network has an associated BDI agent, which together are used to combat DDoS attacks. An agent is built as an aggregation of capabilities, and such capabilities are selected according to the primitive actions that a mechanism provides. For example, *link monitors* are able to monitor links, and therefore can perceive *overUsage*(*link*) events.

In order to permit capabilities to be flexibly assigned to devices, *i.e.*, without having to perform additional configurations besides adding capabilities to agents, there are two challenges that must be faced. Firstly, when an agent is unable to achieve a certain goal, it must collaborate with other agents to achieve it. This collaboration must, however, be performed in a seamless way; this means that there is no difference between adding a goal to an agent that will achieve it or to an agent that will request another agent to achieve this goal. Secondly, capabilities cannot share knowledge, *i.e.*, beliefs, because otherwise it would be necessary to add to the same agent all capabilities that share knowledge. Next we detail how we address these two challenges.

3.3.1. Seamless agent collaboration

Some described capabilities depend on others to achieve goals. For example, controllers must limit link, IP or flow rates, and only the rate limiter capability has plans to perform this. The dependencies among these capabilities are shown in Fig. 2, where UML *components* represent capabilities. Also, goals that a capability cannot achieve (and thus in order to be achieved require the use of additional capabilities) are represented as *required interfaces*, and goals that a capability can achieve to others are represented as *provided interfaces*.

To enable agent collaboration, we add to every agent two additional plans: (i) `GoalRequest` plan, which can be used to request other agents to achieve a goal, but has the lowest priority, that is, it is the last plan executed to achieve a goal and (ii) `GoalResponse` plan, which handles messages with goal requests. This agent communication follows the FIPA Contract Net Interaction Protocol.[1] When an agent requests other agents to achieve a goal, they reply if they can achieve it and, if so, they specify a cost. This cost can be associated with the current agent availability in terms of memory or processor, or physical location, when a significant amount of data will be exchanged between agents. Such costs allow the agent that issued a request to decide to which agent it will delegate the goal. Cost is currently given as a fixed value, and it is future work to make the cost a function of the parameters that may influence agent effectiveness to achieve delegated goals as described above. Alternatively, existing task allocation approaches, *e.g.*, Jiang and Jiang (2005), can be adopted.

To illustrate how agent collaboration occurs, we show in Fig. 3 messages being exchanged among agents to combat a DDoS attack. Rectangles located at the top of this figure represent agents, while arrows represent messages. Agent names are given as a composition of the initials of the capabilities that agents have. For example, agent *FC+RL* has the capabilities flow controller and rate limiter. When an agent is not able achieve a goal, *e.g.*, agent *FC+RL* is not able to achieve ?*anomalousUsage*(*link*), it sends a call for proposals (CFP) message to all agents that have the IP controller capability. Such agents are discovered by consulting an agent that provides an yellow pages service (shown as *Other* agents). Agents may respond that they cannot achieve the goal or inform a cost for achieving it. Based on such costs, as said above, the requester agent delegates the goal to a selected agent; the `GoalRequest` plan selects the agent associated with the lowest cost. The `GoalRequest` plan not only allows agents to delegate goals when they cannot achieve them, but also to delegate goals after plan failures, thus providing robustness to the system.

The higher the number of agents, the higher the number of exchanged messages. However, the overhead caused by such increase in the number of exchanged messages is not highly significant, because: (i) each agent receives a CFP and must reply the request, thus the number of messages increases linearly according to the number of agents and (ii) only a reduced number of agents receive the request, which are those agents that have the capability associated with the goal to be achieved.

3.3.2. Decoupling knowledge

In addition to support goal delegation, we need an additional mechanism to allow capabilities to be freely assigned to agents. There are capabilities that share knowledge and must be kept consistent. For example, when a ?*benign*(*ip*) goal is added to an agent by the IP controller capability, it is achieved by an agent that has a flow controller capability. However, if as a result of achieving this goal the agent that has the flow controller capability learns that ¬*benign*(*ip*), goals to respond threats that are the reason why the IP address is not benign are generated. When all threats are responded, the traffic of the IP address becomes benign. But, this update must be also sent to the IP controller capability, so that it can generate a goal to unlimit the IP address.

To address this issue it is possible to establish a set of beliefs that is shared knowledge and, whenever such beliefs are updated, this is broadcasted to agents. Yet, this can generate unnecessary network traffic. Therefore, by analysing beliefs that are updated by more than one capability, we identified two beliefs—*anomalousUsage*(*link*) and *benign*(*ip*)—that must be kept partially consistent among capabilities.

The IP controller capability must be aware of the belief *anomalousUsage*(*link*) only when it is true, so that it can take actions to restrict anomalous IP addresses. After this, the *anomalousUsage*(*link*) becomes false, and this belief can be removed. The link controller capability, after learning the *anomalousUsage*(*link*) belief, must receive an update when it becomes false. The `GoalRequest` plan has thus an additional feature: when a goal is delegated to another agent, the requester agent can make a subscription to receive belief updates. The `GoalResponse` plan in turn is able to handle such subscription and send updates until the belief is removed. This solution is also adopted with the *benign*(*ip*) belief,

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

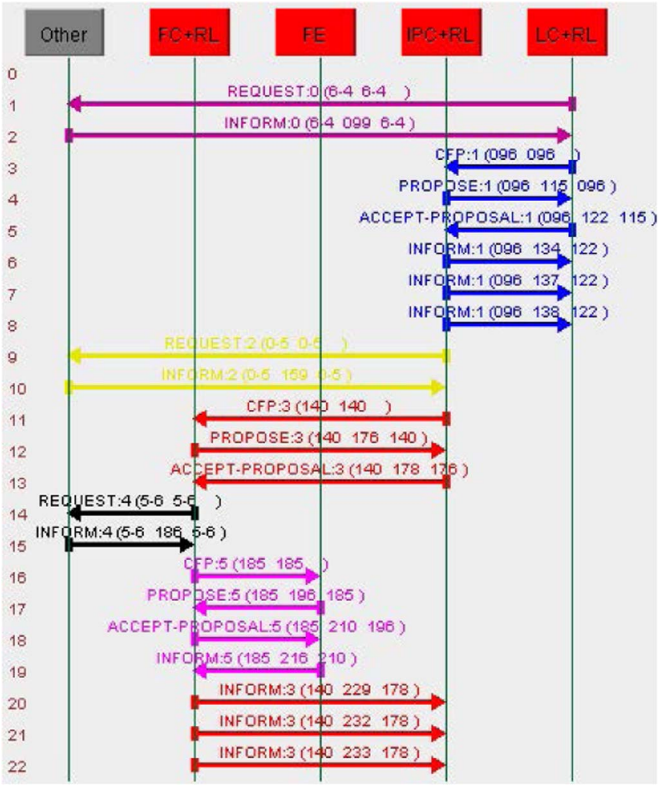


Fig. 3. Example of BDI2DoS agent message exchange.

and the IP and flow controllers. In Fig. 3, this mechanism can be seen in agent conversations which, after the `accept-proposal` message, exchange more than one `inform` message. The first `inform` message is sent when the delegated goal was achieved, and the following ones correspond to updates, when the belief associated with the goal is updated, until it is removed from the agent belief base.

4. Implementation and evaluation

BDI2DoS was implemented using the BDI4JADE platform, which was integrated with the PReSET resilience simulator to build an evaluation testbed. This integration as well as the experimental evaluation of BDI2DoS are presented next.

4.1. BDI agent implementation and testbed

From the many BDI agent platforms available, we selected BDI4JADE (Nunes et al., 2011) to implement BDI agents. BDI4JADE was chosen mainly because agents can be implemented in pure Java thus facilitating the implementation of mechanisms to decouple capability knowledge and integration with a testbed to run simulations. It also implements the concept of capabilities, which have their own reasoning strategies (*e.g.*, goal generation function). Hence, capabilities specified in our design can be directly implemented in BDI4JADE. Each agent has a capability that has the `GoalRequest` and `GoalResponse` plans; and other capabilities, such as rate limiter, are added to it as *part* capabilities (Nunes, 2014). An implemented plan selection strategy uses the `GoalRequest` plan as the last alternative to achieve goals. When agents cannot achieve a goal, they request other agents to achieve it, and then the mechanism described previously is used to keep particular beliefs consistent. Otherwise, a belief that is shared among capabilities is stored in a single Java object and is added to the belief bases of the capabilities that have this belief. Agents register themselves in a directory facilitator (yellow pages service) and inform the goals they can achieve to others. When an agent needs a goal to be achieved, it searches this directory and sends requests only to agents that offer what the agent needs to be accomplished.

To evaluate BDI2DoS we built a testbed integrating the BDI4JADE agent platform and the PReSET resilience simulator (Schaeffer-Filho et al., 2013). PReSET is based on OMNeT++ (Varga and Hornig, 2008), which is a network simulator implemented in C++, and supports the use of a policy specification language to orchestrate the behaviour of instrumented mechanisms providing resilience functionality in the simulated network, *e.g.*, anomaly detection and traffic shaping systems. These instrumented mechanisms export a management interface that allows them to be dynamically reconfigured, *e.g.*, setting flags, dropping connections, triggering or stopping monitoring sessions, etc. Further, a socket interface is used to communicate and translate observed events from the simulation environment to the policy framework. Events are used to indicate conditions observed in the simulated network that may require management actions, such as the detection of an attack. Thus, policy-driven resilience strategies running within the simulation environment can be readily evaluated using PReSET.

Despite being a valuable tool for evaluating resilience strategies, PReSET still requires all the orchestration between resilience mechanisms to be anticipated and described in the form of *event–condition–action* (ECA) policies. For this reason, we extended this environment with BDI4JADE adapters to enable agent communication with PReSET components, using XMLRPC for remote communication. Fig. 4 shows an architectural overview of how the various components of our solution are integrated. Components of PReSET, located on the left-

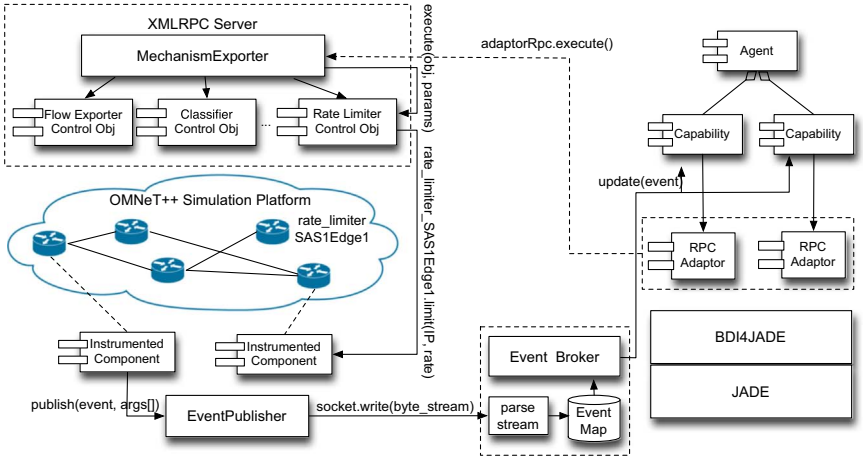
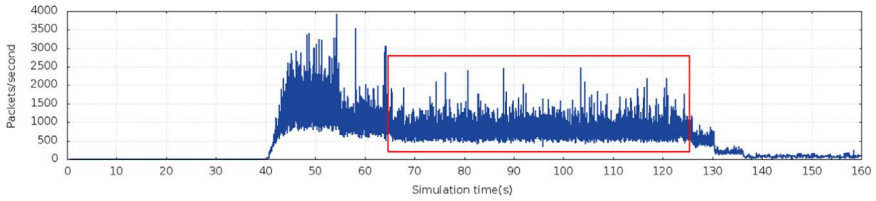
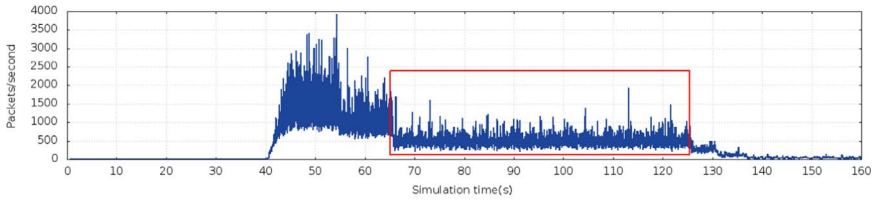


Fig. 4. Integrated testbed for the BDI2DoS evaluation.



(a) Simulation of ECA policies with 30% reduction of traffic to the victim.



(b) Simulation of ECA policies with 70% reduction of traffic to the victim.

Fig. 5. Simulation results using ECA policies: traffic volume measured in a specific link leading to the victim IP address (packet/sec vs. time (s)). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.).

hand side, run on top of the OMNeT++ simulation platform. These components communicate with agents that run on top of the BDI4JADE and JADE platforms (right-hand side). This communication adopts a publish-subscriber architecture, and events are delivered to agents through an event broker. Typically, agents have primitive actions, which can be implemented *locally* or *remotely*. In our implementation, we chose to run primitive agent actions *remotely*, *i.e.*, deploying and executing agents in the BDI4JADE platform, and associated network devices running in the simulation platform are accessed remotely. Consequently, agents can be seen as the "brain" and PReSET components as the "body". Agents and the PReSET components only exchange messages when either events are detected by these components or decisions are taken by the agents. Moreover, the agent infrastructure is separated from PReSET components (*i.e.*, sensors). Consequently, agent-to-agent communication is performed in a dedicated environment. Therefore, agent-to-agent and agent-to-sensor communications do not add significant network overhead to the infrastructure. This approach is similar to that adopted in recent work on simulation (Padgham et al., 2014). When agents are instantiated in BDI4JADE, they receive an *abstract factory* (Gamma et al., 1995) as parameter, which allows them to instantiate the appropriate implementation of the interface to invoke primitive actions.

By integrating BDI4JADE and PReSET we were able to replace the reactive ECA policies by a more sophisticated BDI reasoning strategy. Thus, the effects of actions performed by collaborating agents can be readily observed in the simulation. As an additional contribution, the evaluation testbed presented in this section can be used to assess the behaviour emerged from collaborating agents when different types of attacks and network conditions are simulated.

4.2. Experimental evaluation

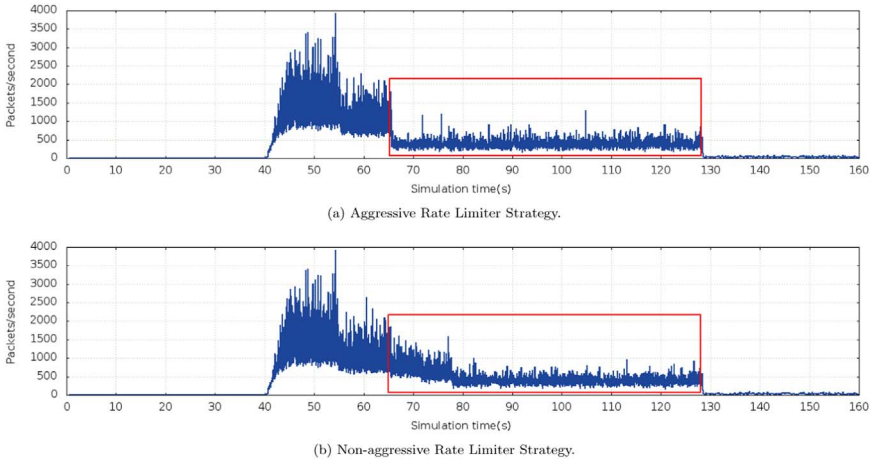
To simulate large-scale IP networks and attacks PReSET uses *ReaSE* (Gamer and Scharf, 2008) for creating realistic topologies and for generating background and attack traffic, including DDoS attack traffic. In particular, ReaSE can generate DDoS attack traffic based on the Tribe Flood Network (Dittrich, 1999). In our experiments, we simulate a network topology in which a specific victim web server is being attacked by a number of *DDoSZombie* hosts across the network. In Section 4.2.1, we explore the effectiveness of ECA policies in containing a DDoS attack, and in Section 4.2.2 we simulated the same attack on the same network but instead of policies, BDI2DoS was used to orchestrate the behaviour of the resilience mechanisms on the ingress link of the subnetwork under attack. Functions such as link monitoring, intrusion detection, and so on were carried out by agents that autonomously coordinate resilience mechanisms in order to contain the attack.

4.2.1. Policy-based resilience strategy

To explore the behaviour of ECA policies we have reproduced the resilience strategy presented in Section 2.2, which used Ponder2 (Twidle et al., 2009) as the policy engine to control the resilience mechanisms in a simulated DDoS attack. In our simulated network, we monitored the traffic passing through a specific switch that provides access to the victim web server. As can be seen in Figs. 5a and b, the attack started around 40 s from the beginning of the simulation. This can be noticed by the sudden increase in the number of packets/second observed in the monitored network link. After a few seconds the criteria for limiting the amount of traffic in the link that reaches the victim is met and a policy activates the rate limiter component, which throttles the traffic in 50% as can be seen at around 55 s in Figs. 5a and b. This ensures that the network remains operational while more evidence about the ongoing anomaly is gathered. Shortly after, the anomaly detection mechanism starts monitoring the traffic in order to find out the IP address of the victim of the ongoing flooding attack. This component is capable of examining the destination address of each incoming packet in the monitored link, and determining if a given destination (in this case, the victim web server) accounts for a large percentage of all incoming traffic.

We illustrate the use of ECA policies in two different simulations that explore alternative resilience strategies. First, we explored a policy that drops only 30% of the traffic to the victim after the anomaly detection mechanism found out which IP address was being targeted (red rectangle), as can be noted at around 66 s in Fig. 5a. Due to the nature of ECA policies, if it is decided that 30% is too low, and that the sustained attack traffic is compromising the operation of the network, then an operator would have to manually change the value of such condition or create an alternative policy. To explore a more aggressive interference, we performed a second experiment where, instead of 30%, the traffic was reduced by 70% (red rectangle). This change can be seen at around 66 s in Fig. 5b.

In both Figs. 5a and b, immediately after the above policies are activated, the flow monitor and classifiers are started in order to record the traffic targeting the victim IP address and analyse it. This is performed in order to identify the specific attack flows and allow the incoming traffic from legitimate hosts to reach the web server without any restriction. Consequently, the rate limiter completely blocks the traffic from the identified attackers at around 125 s. This process is repeated until classifier stops identifying attackers, which happens at around 136 s.



(a) Aggressive Rate Limiter Strategy.

(b) Non-aggressive Rate Limiter Strategy.

Fig. 6. Simulation results using agents to control the resilience strategy: traffic volume measured in a specific link leading to the victim IP address (packet/sec vs. time (s)). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Table 8
Agent preferences.

Agent	Maximise performance	Maximise protection
Aggressive	0.25	0.75
Non-agressive	0.7	0.3

Table 9
Plan utilities according to agent preferences.

Agent	LIPR_High	LIPR_Medium	LIPR_Low
Aggressive	0.75	0.675	0.50
Non-agressive	0.48	0.63	0.68

4.2.2. BDI-based resilience strategy

Differently from policy rules, agents are flexible and can adapt to different strategies by applying plans with distinct restriction rates. For example, one might prefer a more aggressive resilience strategy that promptly reduces suspicious traffic to a minimum. Considering that a harsh restriction might also have an impact on legitimate traffic, a distinct approach for agents is to have plans with different restriction rates and preferences over softgoals that slowly contain traffic until it reaches the desired level.

Consider a scenario in which there is a single network switch (*i.e.*, an agent) that provides access to the victim web server. We also have five agents, each with one capability type. Figs. 6a and b show the onset of the DDoS attack shortly after 40 s from the start of the simulation. After perceiving an event, the link controller believes *overUsage*(*link*) and *not anomalousUsage*(*link*), and consequently generates goals to prevent an attack and find out whether there is an anomalous usage, *attackPrevented*(*link*) and ?*anomalousUsage*(*link*) respectively. With the aim of protecting downstream servers and the infrastructure, the link controller starts a preventive measure of partially limiting the link usage (LLR plan). This is performed with the help of the rate limiter agent, and its effects can be noticed around 55 s into the simulation (Figs. 6a and b). Next, it requests the IP controller to achieve ?*anomalousUsage*(*link*), which concludes that there is an IP address receiving anomalous traffic. This is achieved by the ALS plan, whose action `detectIntrusion` enables a PReSET component that performs intrusion detection in the simulated network. The IP controller then generates goals to restrict the access to the victim IP address and to identify whether the traffic destined to it is benign (?*benign*(*ip*)). The former is achieved by requesting the rate limiter to partially limit only the traffic directed to that specific victim IP address, rather than all the traffic passing through the link (LIPR plan).

Two scenarios were considered in order to evaluate different limiting strategies. In each scenario, we used different preferences for the agent that included the IP controller capability, as shown in Table 8. In the first scenario, this agent was aggressive, *i.e.* it had a stronger preference for the *maximise protection* softgoal. As a consequence, the plan with the highest utility is LIPR_High (see Table 9), which throttles 70% of the traffic addressed to the victim IP address, as can be seen at around 66 s (red rectangle) in Fig. 6a. The execution of this plan was enough to achieve the goal of restricting the access to the victim IP address. In the second scenario, we set a stronger preference for the *maximise performance* softgoal, that is, the agent was non-aggressive. The plan selection function chose to execute the plan LIPR_Low first, which throttles traffic only in 20%, as can be noted at around 66 s in Fig. 6b. The action `detectIntrusion` in the LIPR_Low was executed within this plan, and detected that the victim IP address remains an outlier. Therefore, *restricted*(*ip*) is not added to the agent belief base and the plan fails. As a result, another plan is selected to achieve *restricted*(*ip*) and between the two candidates—LIPR_Medium and LIPR_High—the former has the highest utility and is thus selected. This results in a drop of 50% of the traffic to the victim IP address, as can be seen at around 73 s in Fig. 6b. Given the current attack, LIPR_Medium also fails. Finally, the more intrusive plan, LIPR_High, is selected as the other plans failed to achieve the goal of restricting the traffic to acceptable levels. This plan reduced the traffic in 70% at around 78 s as can be noted in Fig. 6b, limiting the traffic to the victim IP address to approximately the same level as in the first simulation (red rectangle). Thus, successfully reducing the traffic to the victim to acceptable levels.

Partially restricting the traffic directed only to the victim IP address causes the link controller to believe that ¬*anomalousUsage*(*link*) and, as a consequence, it generates and achieves the goal to become fully operational again. This is achieved by the RLR plan. Next, the IP controller delegates the goal ?*benign*(*ip*) to the flow controller. To achieve this goal it is necessary to identify the specific attack flows among all the traffic flows directed towards the victim web server. To do so, the IP controller executes the AIPF plan, which first requests the flow exporter to record flows, which are classified by the flow controller. The latter finds out that there are flows that are threats, and thus coming from a malicious IP address (¬*benign*(*ip*)). The flow controller must thus respond to these threats, and so performs the LFR plan for each flow. Consequently, malicious flows are blocked (after 130 s of simulation), and the victim IP address becomes unrestricted because the IP controller generates a goal to do it and executes the

Table 10
Belief Base Updates after Plan Executions.

Belief		LLR /LL	ALS	LIPR /LIP	RLR /RL	AIPF /RF	LFR	RIPR
Link Controller								
$overUsage(l)$	⊤	⊤	¬[1]	¬	¬	¬	¬	¬
$anomalousUsage(l)$	*not*	*not*	⊤	¬	*not*	*not*	*not*	*not*
$linkRateLimited(l)$		⊤	⊤	⊤	¬	¬	¬	¬
$attackPrevented(l)$		⊤	⊤	⊤	¬	¬	¬	¬
IP Controller								
$anomalousUsage(l)$			⊤	¬	*not*[1]	*not*	*not*	*not*
$overUsageCause(l,i)$			⊤	¬	¬	¬	¬	¬
$anomalous(i)$			⊤	⊤	⊤	⊤	⊤	*not*
$restricted(i)$				⊤	⊤	⊤	⊤	¬
$ipRateLimited(i)$				⊤	⊤	⊤	⊤	¬
$benign(i)$			*not*	*not*	*not*	¬	⊤	⊤
Flow Controller								
$benign(i)$			*not*	*not*	*not*	¬	⊤	*not*[1]
$flowRecord(i)$						⊤	⊤	⊤
$threat(f)$						⊤	¬	¬
$flowRateLimited(f)$							⊤	⊤
Rate Limiter								
$linkRateLimited(l)$		⊤	⊤	⊤	¬	¬	¬	¬
$ipRateLimited(i)$				⊤	⊤	⊤	⊤	¬
$flowRateLimited(f)$							⊤	⊤
Flow Exporter								
$flowRecord(i)$						⊤	⊤	⊤

[1] Updated due to belief revision.
[2] l: link; i: ip address; and f: flow.

RIPR plan. Ultimately, the link is fully operational, IP addresses unrestricted, and malicious flows are limited.

By ensuring that plans are automatically selected until the threat is finally contained is a clear indication that the BDI-based resilience strategy is more robust than the one based on ECA policies. In particular, the BDI strategy is able to cope with problems that occur during the execution of the anticipated behaviour, *e.g.*, in case the thresholds specified by policy parameters were ill-defined. The evolution of agents' beliefs over time due to the execution of plans is summarised in Table 10 (for the scenario in which an aggressive agent is used). Each column represents the current beliefs after the execution of the plan that is indicated in the column title: (i) ⊤: belief is true; (ii) ¬: belief is false; and (iii) *not*: it is unknown whether the belief is true. Grey cells indicate belief changes. As said, we assume initial beliefs indicate that the link is being overused, and it is unknown whether it is anomalous or not. In this table, we show only one flow that is a threat, but in our experiments we simulated multiple attacking flows.

5. Related work

Many approaches use autonomous software components to ensure network resilience. However, these approaches, *e.g.*, Nguengang et al. (2006); Yang and Chang (2011); Preetha et al. (2014), often only decompose a proposed solution into software components, each with a specified responsibility, executed reactively like in object-oriented software, and typically do not employ any particular agent-based technique. Likewise, focusing on cloud computing, a software component referred to as security agent was used to monitor and detect DoS attacks (Kim et al., 2016). This agent may migrate to different nodes if there are not enough computing resources to support its operations. However, this work does not employ any particular technique proposed in the context of multi-agent systems, in which multiple agents interact to collectively resolve a problem.

For supporting new architectures for the Future Internet, Netlets (Martin et al., 2011) and autonomic functional blocks (AFBs) (Sifalakis et al., 2011) have been proposed. Both are inspired by component-based development. They promote the composition of network protocols by using building blocks during design-time. 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.

Self-Managed Cells (SMCs) (Lupu et al., 2008; Schaeffer-Filho et al., 2015)s were introduced as a paradigm for structuring the management aspects of ubiquitous and networked applications. An SMC typically consists of a set of autonomous hardware and software components with management and adaptation strategies specified as easily modifiable policies. In SMCs, ECA policies are used to specify adaptation rules that determine the reconfiguration actions to be performed in response to conditions observed. Further, a *sentient object* (Biegel and Cahill, 2004) 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.

By comparing ECA policies, used in SMCs, with our approach, we can observe that their key difference is that BDI agents capture motivational state, while ECA policies are purely reactive. This improves flexibility by, for example, having multiple plans to achieve a goal. Moreover, because agents delegate goals to other agents that are identified and selected at runtime, agent interaction is also flexible, and is dynamically adapted when new devices join the network. Furthermore, if a plan makes a mistake, as exemplified above, agents proactively generate new goals causing them to continue to take actions to identify and remediate the attack.

6. Conclusion

In this paper, we presented BDI2DoS, a multi-agent application composed of BDI agents to combat DDoS attacks and ensure network resilience. The BDI2DoS design is based on capabilities that are building blocks to instantiate agents. Agents collaborate in our application by delegating goals to other agents, when an agent cannot itself achieve a goal. Differently from existing approaches, if problems occur during the execution of the anticipated behaviour, agent interactions lead to the prevention of the attack as well. We detailed how we implemented BDI2DoS with the BDI4JADE platform, and how agents communicate with components within the PReSET resilience simulator. Also, we compared the BDI-based strategy with an experiment using ECA policies to achieve the same resilience strategy and showed that the proposed approach is more flexible and able to cope with problems that occur during the execution of the anticipated behaviour.

Nevertheless, there were issues left unaddressed. Guaranteeing robustness is not trivial, because testing all possible scenarios is unfeasible. Moreover, there should be further investigation on how different resilience strategies would interact using a BDI agent approach. Finally, organisational frameworks can possibly improve the coordination among agents, and will be investigated as part of our future work.

Acknowledgements

This work receives financial support of CNPq, project ref. 442582/2014-5. Ingrid Nunes thanks for research grants CNPq ref. 303232/2015-3, CAPES ref. 7619-15-4, and Alexander von Humboldt, ref. BRA 1184533 HFSTCAPES-P. Alberto Schaeffer-Filho would like to thank CNPq for research grant ref. 311088/2015-5.

References

Biegel, G., Cahill, V., 2004. A framework for developing mobile, context-aware applications. In: Pervasive Computing and Communications, 2004. PerCom 2004. In: Proceedings of the Second IEEE Annual Conference on, pp. 361–365. http://dx.doi.org/10.1109/PERCOM.2004.1276875.

Bratman, M.E., Israel, D.J., Pollack, M.E., 1988. Plans and resource-bounded practical reasoning. Comput. Intell. 4 (3), 349–355.

Busetta, P., Howden, N., Rönnquist, R., Hodgson, A., 2000. Structuring bdi agents in functional clusters. In: Proceedings of the 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL), ATAL '99, Springer-Verlag, London, UK, pp. 277–289.

Charalambides, M., Flegkas, P., Pavlou, G., Rubio-Loyola, J., Bandara, A.K., Lupu, E.C., Russo, A., Dulay, N., Sloman, M., 2009. Policy conflict analysis for diffserv quality of service management. IEEE Trans. Netw. Serv. Manag. 6 (1), 15–30. http://dx.doi.org/10.1109/TNSM.2009.090302.

Damianou, N., Dulay, N., Lupu, E., Sloman, M., 2001. The ponder policy specification language. In: Policies for Distributed Systems and Networks, Springer, pp. 18–38.

Dittrich, D., The tribe flood network distributed denial of service attack tool. Tech. rep., University of Washington (October 1999).

Ferber, J., 1999. Multi-agent systems: an introduction to distributed artificial intelligence, Vol. 1, Addison-Wesley Reading.

Gamer, T., Scharf, M., 2008. Realistic simulation environments for ip-based networks. In: Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, pp. 1–7.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Guerra-Hernández, A., El Fallah-Seghrouchni, A., Soldano, H., 2004. Learning in bdi multi-agent systems. In: International Workshop on Computational Logic in Multi-Agent Systems, Springer, pp. 218–233.

Jennings, N.R., 2001. An agent-based approach for building complex software systems. Commun. ACM 44 (4), 35–41. http://dx.doi.org/10.1145/367211.367250.

Jiang, Y.-C., Jiang, J., 2005. A multi-agent coordination model for the variation of underlying network topology. Expert Syst. Appl. 29 (2), 372–382. http://dx.doi.org/10.1016/j.eswa.2005.04.015.

Keeney, R.L., Raiffa, H., 1993. Decisions with multiple objectives: preferences and value trade-offs. Cambridge university press.

Kim, D., Jung, S., Hwang, D.-J., Kim, S., 2016. Mobile-based dos attack security agent in sensor networking. Wirel. Pers. Commun. 86 (1), 91–107.

Lupu, E., Dulay, N., Sloman, M., Sventek, J., Heeps, S., Strowes, S., Twidle, K., Keoh, S.-L., 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., Zitterbart, M., 2011. A flexible framework for future internet design, assessment, and operation. Comput. Netw. 55 (4), 910–918. http://dx.doi.org/10.1016/j.comnet.2010.12.015.

Najjar, W., Gaudiot, J.-L., 1990. Network resilience: a measure of network fault tolerance. IEEE Trans. Comput. 39 (2), 174–181.

Nguengang, G., Hugues, L., 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, Springer-Verlag, pp. 64–75. http://dx.doi.org/10.1007/11880905_6.

Nguyen, T., Armitage, G., 2008. A Survey of Techniques for Internet Traffic Classification using Machine Learning. IEEE Commun. Surv. Tutor. 10 (4), 56–76. http://dx.doi.org/10.1109/SURV.2008.080406.

Nunes, I., Luck, M., 2014. Softgoal-based plan selection in model-driven bdi agents. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems, pp. 749–756.

Nunes, I., Lucena, C., Luck, M., 2011. Bdi4jade: a bdi layer on top of jade. In: ProMAS 2011, Taiwan, pp. 88–103. ⟨http://www.inf.ufrgs.br/prosoft/bdi4jade/⟩.

Nunes, I., 2014. Capability relationships in BDI agents. In: Proceedings of the 2nd International Workshop on Engineering Multi-Agent Systems (EMAS 2014), Paris, pp. 56–72.

Padgham, L., Nagel, K., Singh, D., Chen, Q., 2014. Integrating BDI agents into a matsim simulation. In: ECAI 2014 - Proceedings of the 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), pp. 681–686. http://dx.doi.org/10.3233/978-1-61499-419-0-681.

Preetha, G., Devi, B.K., Shalinie, S.M., 2014. Autonomous agent for ddos attack detection and defense in an experimental testbed. Int. J. Fuzzy Syst. 16 (4), 520–528.

Rao, A.S., Georgeff, M.P., 1995. Bdi agents: From theory to practice. In: Proceedings of the First International Conference on Multiagent Systems, San Francisco, pp. 312–319.

Schaeffer-Filho, A., Smith, P., Mauthe, A., Hutchison, D., Yu, Y., 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), IEEE Computer Society, Maui, Hawaii, USA, pp. 401–408.

Schaeffer-Filho, A., Mauthe, A., Hutchison, D., Smith, P., Yu, Y., Fry, M., 2013. PReSET: A toolset for the evaluation of network resilience strategies. In: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, 2013, pp. 202–209. ⟨http://www.scc.lancs.ac.uk/PReSET⟩.

Schaeffer-Filho, A., Lupu, E., Sloman, M., 2015. Federating policy-driven autonomous systems: Interaction specification and management patterns. J. Netw. Syst. Manag. 23 (3), 753–793. http://dx.doi.org/10.1007/s10922-014-9317-5.

Sifalakis, M., Louca, A., Bouabene, G., Fry, M., Mauthe, A., Hutchison, D., 2011. Functional composition in future networks. Comput. Netw. 55 (4), 987–998. http://dx.doi.org/10.1016/j.comnet.2010.12.006.

Singh, D., Sardina, S., Padgham, L., Airiau, S., 2010. Learning context conditions for bdi plan selection. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, pp. 325–332.

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

Smith, P., Hutchison, D., Sterbenz, J., Schöller, M., Fessi, A., Karaliopoulos, M., Lac, C., Plattner, B., 2011. Network resilience: a systematic approach. IEEE Commun. Mag. 49 (7), 88–97. http://dx.doi.org/10.1109/MCOM.2011.5936160.

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

Tcholtchev, N., Grajzer, M., Vidalenc, B., 2010. Towards a unified architecture for resilience, survivability and autonomic fault-management for self-managing networks. In: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, Springer, pp. 335–344.

Twidle, K., Dulay, N., Lupu, E., Sloman, M., 2009. Ponder2: A policy system for autonomous pervasive environments. In: 2009 Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems, pp. 330–335. http://dx.doi.org/10.1109/ICAS.2009.42.

Varga, A., Hornig, R., 2008. An overview of the OMNeT++ simulation environment. In: SIMUTools '08: Proceedings of the 1st International Conference on Simulation Tools and Techniques, ICST, Marseille, France, pp. 1–10.

Wooldridge, M., 2009. An introduction to multiagent systems. John Wiley & Sons.

Yang, S.-Y., Chang, Y.-Y., 2011. An active and intelligent network management system with ontology-based and multi-agent techniques. Expert Syst. Appl. 38 (8), 10320–10342. http://dx.doi.org/10.1016/j.eswa.2011.02.115.

Ingrid Nunes is an Associate Professor at the Institute of Informatics, Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, currently in a sabbatical year at TU Dortmund in Germany. She obtained her PhD in Informatics at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil. Her PhD was in cooperation with King's College London (UK) and University of Waterloo (Canada). She is the head of the Prosoft research group, and her main research areas are agent-oriented software engineering and software maintenance and evolution.

Frederico Schardong is an M.Sc. student in computer science at the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He achieved his Technologist Degree in Information Security in Universidade do Vale do Rio dos Sinos (UNISINOS), in 2015. His research interests include network security and resilience, Network Function Virtualization (NFV), machine learning and multi-agent systems. See http://www.inf.ufrgs.br/~fschardong for more information.

Alberto Schaeffer-Filho is an Associate Professor at the Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS). He obtained his PhD in Computing from Imperial College London, UK, in 2009. Between 2009 and 2012, he worked as a Research Associate at the School of Computing and Communications (SCC), Lancaster University, UK. His research interests include network management, security and resilience, Network Functions Virtualization (NFV), Software-Defined Networking (SDN) and Policy-Based Network Management (PBNM). See http://www.inf.ufrgs.br/~alberto for selected papers.