PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Gerenciamento Autônomo de Redes na Internet do Futuro

Alexandre Passito de Queiroz

Manaus
dezembro de 2012

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Alexandre Passito de Queiroz

Gerenciamento Autônomo de Redes na Internet do Futuro

Tese apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas, como requisito parcial para a obtenção do título de *Doutor em Informática*, área de concentração em Inteligência Artificial.

orientador:
Prof. Edjard de Souza Mota, Ph.D

Manaus
dezembro de 2012

Dedico este trabalho aos
meus avós Gercina, Deolinda e Raimundo,
pela criação dentro de princípios honrosos e
pelo incansável estímulo à educação.

Agradecimentos

Não tenho dúvida alguma que o doutorado é um longo processo de formação. Se olharmos para trás é possível ver como amadurecemos em todas as dimensões, principalmente intelectualmente e como pessoa. Um processo que só quem chega ao final é capaz de entender e valer-se dessa experiência para construir o futuro. Também não tenho dúvida alguma, assim como em outros processos na vida, que só podemos terminar um doutorado com a ajuda de outras pessoas, que desempenham diversos papéis, e que, com o passar do tempo mudam de papel ao longo do processo. Gostaria de agradecer imensamente:

- Ao meu orientador Prof. Edjard Mota, um pesquisador visionário que sempre me ajudou do início ao fim. Agradeço aos momentos de discussão científica. Agradeço aos estímulos aos estudos e diversos votos de confiança durante o processo. Agradeço às inúmeras ajudas tanto relacionadas à essa pesquisa quanto ao lado pessoal. Reconheço agora que o papel do orientador é um dos mais difíceis que existem. Bom ainda é quando ao final do doutorado podemos dizer que temos um novo amigo, como é no meu caso.

- Agradeço à diversas pessoas no IComp/UFAM. Em ordem cronológica, ao Prof. Edjair Mota, quem apostou em 2003 que eu tinha um potencial para a pesquisa em computação e que me ofereceu uma bolsa de iniciação científica. Os anos no grupo de redes foram engrandecedores. Ao Prof. Leandro Carvalho, pelas inúmeras discussões na área de redes e computação autonômica no início do doutorado. Aos meus colegas do LabCIA/UFAM, em especial ao Ricardo Bennesby, Paulo Fonseca e Rodrigo Braga, que ajudaram em inúmeras discussões e em parte do código em SDN.

- Agradeço também à CAPES e SUFRAMA pela bolsa de doutorado e auxílio financeiro oferecidos durante 4 anos. Ao projeto de extensão *iOS at UFAM*, na figura do Prof. Edjard Mota, pela ajuda na minha participação no primeiro Open Networking Summit na universidade de Stanford/EUA. Sem dúvida alguma esse foi um momento de grande crescimento profissional.

- À pesquisadora Monica Nogueira da Universidade da Carolina do Norte/EUA, pelas suas inúmeras ajudas com o texto da tese e nossos artigos científicos, bem como conselhos técnicos em momentos importantes. Suas mensagens de motivação fizeram grande diferença.

- Agradeço à toda minha família. É difícil nomear, pois todos de certa forma, em algum momento, contribuíram com algo, desde uma palavra de conforto até a paciência em aceitar minhas diversas ausências. Acreditem, foram muitas.

- Um agradecimento especial aos meus tesouros mais valiosos: minha esposa Diene Passito e aos meus filhos João Guilherme Passito e Ada Sofia Passito. A família é a maior de todas as motivações. Agradeço pelos momentos de paciência e pela renovação das energias quando chegava em casa. Aos meus filhos que lerão essa tese com consciência daqui a 12∼15 anos: no final, a educação vale a pena, não importa o preço que você tenha que pagar por ela.

- Agradeço aos meus amigos, em especial ao Jacall®. Vocês proveram a descontração sempre necessária e um porto seguro.

- Agradeço aos professores do IComp/UFAM, pela excelente formação profissional ao longo desses anos de estudos desde a graduação em Ciência da Computação, e aos professores da coordenação do curso de Ciência da Computação do Uninorte, que deram uma força extra no último semestre desse trabalho.

- Finalmente, agradeço à banca examinadora dessa tese: Prof. Edjard Mota, Prof. Edjair Mota, Prof. Eduardo Nakamura, Prof. José Neuman e o Prof. Dorgival Neto. Após a defesa, entendi que o papel de vocês é apenas contribuir para que o trabalho tenha a excelência obrigatória para um doutorado. Agradeço às inúmeras contribuições com relação ao texto, a organização das ideias e conceitos e, principalmente, ao *viva*, que foi sem dúvida alguma um momento de grande crescimento profissional/acadêmico.

Resumo

A pesquisa em redes autônomas aplica a teoria de agentes inteligentes e sistemas multiagente em mecanismos de controle de redes. Implantar esse mecanismos autônomos e racionais na rede pode melhorar seu comportamento na presença de cenários de controle muito complexos e dinâmicos. Infelizmente, a construção de mecanismos baseados em agentes para redes não é uma tarefa fácil. A principal dificuldade é criar representações concisas de conhecimento sobre os domínios de redes e mecanismos de raciocínio para lidar com elas. Além disso, a Internet faz com que o projeto de sistemas multiagente para o controle da rede seja uma atividade intrincada envolvendo a modelagem de diferentes participantes com diversas crenças e intenções. Esses tipos de sistemas geralmente apresentam problemas de escalabilidade devido à falta de incentivos para cooperação entre domínios administrativos. Finalmente, como a estrutura corrente da Internet geralmente impede inovações, mecanismos de redes autônomas construídos não são totalmente implantados em cenários de larga escala.

O paradigma das redes definidas por software (SDN) está na esfera dos esforços da Internet do Futuro. No paradigma SDN, o hardware de repasse de pacotes é controlado por software sendo executado como um plano de controle separado. Softwares de gerenciamento utilizam um protocolo aberto que programa as tabelas de fluxo em diferentes *switches* e roteadores.

Este trabalho apresenta uma discussão geral sobre a integração de redes autônomas e redes definidas por software. Baseado no conhecimento oferecido por essa discussão, é apresentado um arcabouço que provê autonomia para domínios SDN, permitindo que eles atuem cooperativamente quando implantados em cenários com gerenciamento distribuído.

Dois estudos de caso são apresentados para importantes questões em aberto na Internet: (1) o problema da mitigação de ataques DDoS quando milhares de atacantes realizam inundação por pacotes e os domínios SDN precisam cooperar para lidar com o filtro de pacotes na origem; (2) o problema do gerenciamento de tráfego da rede quando múltiplos domínios devem cooperar e realizar modificações nas primitivas de roteamento de redes.

Palavras-chave: Redes Autônomas; Sistemas Multiagente; Internet do Futuro; Redes Definidas por Software .

Abstract

Autonomous networking research to applies intelligent agent and multiagent systems theory to network controlling mechanisms. Deploying such autonomous and rational entities in the network can improve its behavior in the presence of very dynamic and complex control scenarios. Unfortunately, building agent-based mechanisms for networks is not an easy task. The main difficulty is to create concise knowledge representations about network domains and reasoning mechanisms to deal with them. Furthermore, the Internet makes the design of multiagent systems for network controlling a challenging activity involving the modeling of different participants with diverse beliefs and intentions. Such type of system often poses scalability problems due to the lack of incentives for cooperation between administrative domains. Finally, as the current structure of the Internet often prevents innovation, constructed autonomous networking mechanisms are not fully deployed in large scale scenarios.

The Software-Defined Networking (SDN) paradigm is in the realm of Future Internet efforts. In the SDN paradigm, packet forwarding hardware is controlled by software running as a separated control plane. Management software uses an open protocol to program the flow-tables in different switches and routers.

This work presents a general discussion about the integration of autonomous networks and software-defined networks. Based on the knowledge offered by this discussion, it presents a framework that provides autonomy to SDN domains allowing them to act cooperatively when deployed in scenarios with distributed management.

Two case studies are presented for important open issues in the Internet: (1) the problem of mitigating DDoS attacks when thousands of attackers perform malicious packet flooding and SDN domains must cooperate to cope with packet filtering at the source; (2) the problem of network traffic management when multiple domains must cooperate and modify routing primitives.

Key Words: Autonomous Networks; Multiagent Systems; Future Internet; Software-Defined Networks.

Contents

List of Figures

List of Tables

List of Acronyms and Variables

ACL	Access Control Lists
ACL	Agent Communication Language
API	Application Programing Interface
AI	Artificial Intelligence
AS	Autonomous System
BGP	Border Gateway Protocol
DDoS	Distributed Denial-of-Service
EBNF	Extended Backus-Naur Form
FIPA	Foundation for Intelligent Physical Agents
FML	Flow-based Management Language
HMM	Hidden Markov Models
IP	Internet Protocol
IDS	Intrusion Detection System
JVM	Java Virtual Machine
KB	Knowledge Base
MIB	Management Information Base
OS	Operating System
QoS	Quality of Service
RPC	Remote Procedure Call
SSL	Secure Socket Layer
SNMP	Simple Network Management Protocol
SDN	Software-Defined Networks
TCP	Transport Control Protocol
XML	Extensible Markup Language

Chapter 1

Introdução

I hear and I forget; I see and I remember; I do and I understand..
Confucius (551–479 BCE)

Contents

1.1 Motivação

A comunidade de pesquisa na área de redes tem se engajado em um esforço contínuo para expandir o campo de pesquisa, e também a própria Internet [Rexford and Dovrolis, 2010]. Essa ideia é chamada de Internet do Futuro e seu objetivo é desenvolver arquiteturas de redes que resolvam questões enfrentadas pela Internet atual, como segurança, privacidade, confiabilidade, mobilidade, roteamento e gerenciamento de redes.

Há duas abordagens utilizadas por pesquisadores dessa área: compreender e melhorar a Internet atual [Dovrolis and Streelman, 2010] ou projetar novas arquiteturas de redes que não são limitas pelo sistema corrente. A última, chamada de abordagem *clean-slate*, tem como objetivo o reprojeto da Internet a partir do princípio a fim de oferecer melhores abstrações e desempenho, enquanto provendo funcionalidades similares baseadas em novos princípios fundamentais [Feldmann, 2007].

O paradigma das redes definidas por software (SDN) [McKeown et al., 2008] está na esfera do esforço da Internet do Futuro e tem suas raízes no projeto *clean-slate* [Stanford University, 2012]. No paradigma SDN, o hardware de repasse de pacotes é controlado por software sendo executado em um plano de controle separado. Softwares de gerenciamento utilizam um protocolo aberto que programa as tabelas de fluxo em diferentes *switches* e roteadores.

O paradigma SDN tem sido reconhecido pelos pesquisadores de redes e pela indústria como a arquitetura mais promissora da Internet do Futuro. Soluções inovadoras têm sido desenvolvidas para questões como o controle da mobilidade [Yap et al., 2010], gerenciamento de redes de *datacenters* [Tavakoli et al., 2009] e particionamento de redes de produção [Sherwood et al., 2010a]. Além disso, SDNs têm sido implantadas em várias redes de empresas, *datacenters* e *backbones* [Rexford and Dovrolis, 2010].

Apesar da abordagem inovadora onde softwares de gerenciamento acessam recursos de redes e controlam seu comportamento através de uma interface programática centralizada, as SDNs ainda sofrem com problemas encontrados no presente projeto da Internet: o gerenciamento distribuído entre domínios de redes ainda é difícil de ser realizado. Não há nenhum mecanismo para cooperação entre domínios com o objetivo de lidar com otimizações de redes, falhas e ameaças.

Qualquer solução proposta para essa questão deverá lidar com redes extremamente dinâmicas e que, por exemplo, podem apresentar novos padrões de tráfego ou serviços nunca antes solicitados. A cooperação entre domínios de redes deve funcionar bem em larga escala e os administradores devem concordar em ter *scripts* e regras para promovê-la.

Uma solução natural para superar a natureza bastante distribuída da Internet e sua crescente complexidade em suas interações é usar redes autônomas. Redes autônomas aplicam teoria de sistemas multiagente nos mecanismos de controle da rede. De acordo com Wooldridge [2009], um *sistema multiagente* consiste em um número de agentes que interagem entre si em favor dos seus donos, com diferentes metas e motivações. Para alcançar uma interação com sucesso entre eles, os agentes devem ter a habilidade de cooperar, coordenar e negociar um com o outro. A implantação dessas entidades autônomas e racionais na rede pode melhorar seu comportamento na presença de cenários apresentados acima.

Infelizmente, construir mecanismos de controle de redes baseados em agentes não é uma tarefa fácil, apesar de alguns importantes avanços descritos na literatura, como o trabalho de Tesauro et al. [2004], Meskaoui et al. [2003] e Bullot et al. [2008]. A principal dificuldade é criar representações de conhecimento e mecanismos de raciocínio concisos com o objetivo de lidar com relações complexas entre os diversos componentes de redes, como serviços, protocolos e etc. O agente também deve lidar com informações de baixo nível sobre a rede, como endereços IP e endereços da camada de enlace, ou precisa executar processamentos e análises por pacote a fim de gerar conhecimento. Essa última abordagem impacta diretamente nas suas eficiência e aplicabilidade.

O projeto de sistemas multiagente para a Internet é muito intrincado devido a sua natureza altamente distribuída. O processo envolve a modelagem de diferentes participantes, cada um com diferentes crenças e intenções. Protocolos para cooperação e coordenação devem ser cuidadosamente otimizados para cada arquitetura de redes, que geralmente consistem em milhares de nós. Finalmente, como a estrutura atual da Internet torna difícil a inovação, os mecanismos de redes autônomas não são aplicados em cenários de larga escala atualmente.

1.2 Hipótese da Tese

O projeto corrente das redes definidas por software pode ser aperfeiçoado pelo uso de agentes inteligentes no gerenciamento do plano de dados. Este plano de controle autônomo provê meios eficientes para a cooperação, coordenação e negociação entre domínios. A interface programática centralizada implantada pelas SDNs provê uma abstração que reduz a complexidade de representação e raciocínio sobre conhecimento da rede pelos agentes.

1.3 Objetivos

Este trabalho tem como objetivo construir um arcabouço que provê autonomia para domínios SDN, permitindo a eles agir cooperativamente quando implantados em cenários com gerenciamento distribuído.

Adicionalmente, nós projetamos mecanismos de cooperação para dois importantes problemas na Internet: (1) a detecção/mitigação de ataques de negação de serviço distribuídos (DDoS) e (2) o gerenciamento do tráfego da rede por meio de roteamento entre domínios.

1.4 Contribuições da Tese

Na sequência, nós descrevemos as contribuições da tese na ordem em que elas aparecem no documento.

Um *survey* sobre redes autônomas. O propósito desse *survey* é duplo. Primeiramente, nós discutimos os princípios de SDN, arquitetura e a aplicação corrente da tecnologia. Adicionalmente, nós discutimos as questões em aberto relacionadas com a cooperação entre domínios SDN. Como afirmado nessa tese mais à frente, redes autônomas são uma opção natural para resolver problemas de cooperação em SDN. Nós discutimos o estado da arte das pesquisas em redes autônomas, arquiteturas e aplicações. Além disso, nós apresentamos suas fraquezas e limitações, assim como superá-las para aplicá-las em domínios SDN.

AgNOS: um arcabouço para controle autônomo de SDNs. Nós apresentados um arcabouço baseado em sistemas multiagentes, chamado *AgNOS*, para a construção de SDNs cooperativas na Internet do Futuro. Esse arcabouço é construído sobre as abstrações providas pelas SDNs e estende seus domínios para além de redes corporativas. Nós também provemos uma descrição da arquitetura e sua implementação.

Aplicação do AgNOS na Internet do Futuro. Nós apresentamos dois estudos de caso onde aplicamos o arcabouço AgNOS em importantes questões abertas na Internet: (1) o problema da mitigação de ataques DDoS quando milhares de atacantes executam inundações por pacotes e os domínios SDN devem cooperar para filtrar esses pacotes na origem; (2) o problema do gerenciamento do tráfego da rede quando múltiplos domínios devem cooperar e efetuar modificações nas primitivas de roteamento. Nós desenvolvemos provas de conceito para esses dois estudos de caso e avaliamos empiricamente o desempenho do arcabouço AgNOS.

1.5 Organização do Documento

O Apêndice B define SDN, sua arquitetura e aplicações. Esse apêndice provê uma revisão da literatura da teoria de agentes e sistemas multiagente aplicada à redes, assim como descreve as arquiteturas de sistemas autônomos. Os motivos para a falha das soluções propostas em resolver adequadamente o problema — prover redes com autonomia por meio de agentes inteligentes — são discutidos. Finalmente, nós propomos a implementação de SDN usando os princípios de redes autônomas para lidar com cooperação entre domínios. Essa proposta é alcançada com a pesquisa conduzida nessa tese.

O Apêndice C apresenta o arcabouço AgNOS para a cooperação entre domínios SDN. Os aspectos formais do arcabouço são descritos e o processo de projeto e implementação desses aspectos é retratado. A arquitetura é discutida, juntamente com sua implementação.

O Apêndice D apresenta o desenvolvimento de dois estudos de caso relacionados com a mitigação de ataques DDoS e o gerencimento de tráfego na rede. As implementações das provas de conceito são retratadas nesse apêndice. Além disso, esse apêndice descreve o projeto e metodologia dos experimentos. Nós discutimos o cenário virtualizado usado, assim como a topologia e as configurações de rede. Métricas de avaliação também são retratadas. Finalmente, apresentamos a discussão dos resultados baseado nos experimentos realizados e analisamos o desempenho do arcabouço do ponto de vista de sistemas multiagente e do paradigma SDN.

O Apêndice E apresenta as conclusões e trabalhos futuros possíveis a partir desta tese.

Appendix A

Introduction

A man provided with paper, pencil, and rubber,
and subject to strict discipline,
is in effect a universal machine.
A. M. Turing (1912–1954)

Contents

A.1 Motivation

The networking research community has been engaged in an ongoing effort to move the field — and the Internet itself — forward [Rexford and Dovrolis, 2010]. The idea is referred to as *Future Internet* and aims to develop network architectures that address the issues faced by the current Internet in areas such as security, privacy, reliability, mobility, routing and network management.

There are two approaches taken by researchers in the field: to understand and improve today's Internet [Dovrolis and Streelman, 2010], or to design new network architectures, unconstrained by the current system. The latter, referred to as the *clean-slate* approach, aims to redesigning the Internet from scratch to offer improved abstractions and/or performance, while providing similar functionality based on new core principles [Feldmann, 2007].

The Software-Defined Networking (SDN) [McKeown et al., 2008] paradigm is in the realm of the Future Internet effort and has its roots in the clean-slate design Stanford University [2012]. In the SDN paradigm, packet forwarding hardware is controlled by software running in a separated control plane. Management software uses an open protocol that programs the flow-tables in different switches and routers.

SDN has been recognized by network researchers and the industry as the most promising architecture for the Future Internet. Innovative solutions have been developed for issues such as mobility control [Yap et al., 2010], datacenter network management [Tavakoli et al., 2009], and production network slicing [Sherwood et al., 2010a]. Furthermore, SDN has been deployed in several enterprise, datacenter, and backbone networks [Rexford and Dovrolis, 2010].

Despite the innovative approach, where network management software accesses network resources and controls their behavior through a centralized programmatic interface, SDN suffers from a problem also encountered in the current Internet design: the distributed management between network domains is still difficult to accomplish. There are no mechanisms for cooperation between domains in order to enable network optimizations, cope with failures, and handle threats.

Any proposed solution to handle this issue must deal with highly dynamic networks that can, for instance, suddenly create new patterns of traffic, or services previously not demanded. Cooperation among network domains must work well at large scales, and administrators should agree to have scripts or rules to promote it.

A natural solution to overcome the highly distributed nature of the Internet and its increasingly complex interactions is to use autonomous networks. Autonomous networks apply multiagent theory to network controlling mechanisms. According to Wooldridge [2009], a *multiagent system* consists of a number of agents which interact with another on behalf of their owners, with different goals and motivations. To achieve a successful interaction among them, they must cooperate, coordinate, and negotiate with each other. The deployment of such autonomous and rational entities in the network could improve its behavior in the presence of the aforementioned scenarios.

Unfortunately, building agent-based controlling mechanisms for networks is not an easy task, despite some important advances described in the literature, such as the work of Tesauro et al. [2004], Meskaoui et al. [2003], and Bullot et al. [2008]. The main difficulty is to create concise knowledge representation and reasoning mechanisms to handle a network's complex relationships among its diverse components, services, protocols, and so forth. Agents also have to deal with low-level information about the network, like IP and link-layer addresses, and they may need to perform per-packet processing and analysis for knowledge generation. This latter approach directly impacts their efficiency and applicability.

The design of multiagent systems for the Internet is very challenging due to its highly distributed nature. The process involves the modeling of different participants, each with diverse beliefs and intentions. Protocols for cooperation and coordination must be carefully optimized for each network architecture, which usually consists of thousands of nodes. Finally, since the actual structure of the Internet makes it difficult to innovate, actual autonomous networking mechanisms are not fully deployed in large scale scenarios.

A.2 Statement of the Thesis

The current design of software-defined networks can be improved by the use of intelligent agents to manage the data plane. This autonomous control plane provides efficient means for inter-domain cooperation, coordination and negotiation. The centralized programmatic interface deployed by SDNs provides an abstraction that reduces the complexity to represent and reason about network knowledge by the agents.

A.3 Objectives

This work aims to build a framework that provides autonomy to SDN domains allowing them to act cooperatively when deployed in scenarios with distributed management.

In addiction, we design cooperation mechanisms for two important problems in the Internet: (1) the detection/mitigation of distributed denial-of-service attacks (DDoS), and (2) the management of network traffic by means of inter-domain routing.

A.4 Thesis Contributions

In what follows we describe the thesis contributions in the order they appear in the document.

A survey about autonomous networks. The purpose of this survey is twofold. First, we discuss the SDN principles, architecture and current applications of the technology. In addition, we discuss open issues related to the cooperation between SDN domains. As stated in this thesis, autonomous networks are a natural option to address problems of cooperation in SDN. We discuss the state-of-the-art research in autonomous networks, architectures and applications. Furthermore, we present their weakness and limitations, and how we can address those limitations to enable the applicability of autonomous networking features to SDN domains.

AgNOS: a framework for SDN autonomous control. We present a multiagent-based framework, called*AgNOS*, for the building of cooperative SDNs in the Future Internet. This framework is built on top of the abstractions provided by SDN and extends its domains beyond the enterprise networks. We also provide a description of its architecture and implementation.

Application of AgNOS in the Future Internet. We present two case studies applying the AgNOS framework to important open issues in the Internet: (1) the problem of mitigating DDoS attacks when thousands of attackers perform malicious packet flooding and SDN domains must cooperate to cope with packet filtering in the source; (2) the problem of network traffic management when multiple domains must cooperate and modify routing primitives. As a proof-of-concept, we implement these two case studies and empirically evaluate AgNOS performance.

A.5 Document Outline

Appendix B defines SDN architecture and applications. That appendix also provides a literature review of the agent and multiagent theory applied to networking and describes autonomous networking architectures. The reasons why many of the current proposed solutions did not properly address the problem — providing networks with autonomy by means of intelligent agents — are discussed. Finally, we propose the implementation of SDN using autonomous network principles to cope with cooperation between domains. This proposal is achieved with the research conducted in this thesis.

Appendix C presents the AgNOS framework for the cooperation between SDN domains. The formal multiagent and communication aspects of AgNOS are described and the process of designing and implementing these aspects are depicted. The architecture is discussed, along with its implementation.

Appendix D presents the development of two case studies related to mitigation of distributed denial of service attacks and network traffic management. The proofs-of-concept are depicted in that appendix. Furthermore, it describes the evaluation methodology. We discuss the virtualized scenario used and the topologies and network configurations. Metrics of evaluation are also specified. Finally, we discuss the results based on the experiments conducted and analyzes the framework performance from the perspective of multiagent systems and the SDN paradigm.

Appendix E presents the conclusions and future work.

Appendix B

Background and Related Work

What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle.

Marvin Minsky (1927–)

Contents

THIS APPENDIX presents the software-defined network paradigm, discussing the architecture and applications, as well as open issues related to the cooperation between

ASes built with this paradigm. The aim of this cooperation is to address large scale problems in the Internet. Then, we examine mechanisms which could address the problem of cooperation between these networks by means of autonomy. We review the various architectures for autonomous networks and highlight the reasons why current approaches fail to provide scalable and efficient performance. Then, we propose a new framework for building autonomous networks in the Future Internet.

B.1 Software-Defined Networks

In the software-defined network paradigm, packet forwarding hardware is controlled by software running in an separated control plane. This control plane is composed of: (1) a network operating system, also referred as network controller, (2) management applications which reside on top of this OS, and (3) a protocol which provides the interface between the control and data planes [Das et al., 2011].

A network operating system (also known as network controller) [Gude et al., 2008] provides a centralized and uniform programmatic interface to networks, allowing management applications residing on top of them to run monitoring and controlling tasks. The largest contribution of a network OS is to allow applications to be developed using a centralized programming model and written in terms of high-level abstractions, instead of using low-level parameters from the current Internet architecture. Additionally, directives are deployed independently of the network topology, because the network operating system maintains the mapping between the created abstractions and the low-level configurations. Tootoonchian et al. [2012] provides information about network controller performance.

Figure B.1 depicts a typical SDN deployment. The network is controlled by a network OS and a set of controllable switches or wireless switches (data plane). The data plane allows the network OS to specify flow table entries into switches through the OpenFlow protocol [McKeown et al., 2008]. Entries in the flow table are in the form $< header, action >$, where the first parameter specifies the flow identifying pattern and the second specifies the actions which must be applied to it, such as forwarding, multicasting, dropping, modifying and queuing. If a packet flow matches a pattern, then the action specified in the flow table is deployed. If a packet flow does not match any pattern, then the switch forwards one or more packets to the network controller that processes and eventually updates the flow table with directives for the next packets in that flow. These directives depends on the management applications deployed on top of the network controller and their network policies.

Management applications maintain maps for names and addresses, monitor the network for topology changes or traffic variation. They create a slowly changing network view, which en-

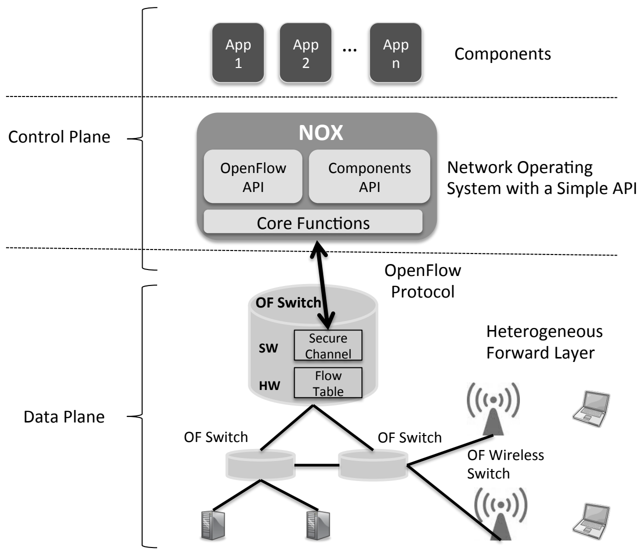

Figure B.1: Software-defined network architecture.

ables the centralized control of the network. The network OS core functions handle concurrency, provide an OpenFlow protocol's API, and keep the communication between network OS instances and switches secure.

Management Applications

A management application (also known as component) is a software module which implements a specific task in the network. Components access the network through an API provided by the network operating system. For each relevant event in the network, the network OS generates different type of notifications which are passed to each component registered for that particular notification type. Each component has a handler module which is in charge of processing this notification. Components can also generate event notifications and send them to other components (e.g. a switch component generates a packet-in event message that is processed by an authenticator component in charge of registering users).

Components can perform basic and internal functions, manage aspects related to networking, and provide web services. Components can work together to maximize network resources or can be instantiated independently. A particular component can provide a task which can be useful for many other components or it can provide a task useful only to its own operation. It is also possible to enforce dependences between components when a task is necessary for the execution of other component.

Network Policy

SDN provides new ways to enforce network policies in the network. Many policy languages have been proposed and the intelligent agents designed in this thesis explore one of these very efficient high-level declarative policy languages, the Flow-based Management Language (FML) [Hinrichs et al., 2009].

FML allows the expression of very common network configurations, such as access control, quality of service, NAT administration, and admission control. FML is based on nonrecursive DATALOG, a subset of PROLOG with negation. It has been used as a query language for deductive databases. Policies in FML are sets of statements in *Horn Clause* notation, $B \Leftarrow A_1 \wedge \ldots \wedge A_n$, where the left hand side is called the head and the right hand side is called the body. It can also be seen as a logical representation of *if-then* relationship. The following FML example, taken from Hinrichs et al. [2009], exemplifies a simple policy statement:

$allow(U_s, H_s, A_s, U_t, H_t, A_t, Prot, Req) \Leftarrow superuser(U_s)$

$superuser(todd)$

$superuser(michelle)$

Variables are denoted by symbols starting with capital letters and denote the fields of a flow. Keywords define constraints over the variables. In the example, the keyword *allow* is used to restrict access based on the defined variables. From the declarative statements above, users *todd* and *michelle* do not have communication restrictions.

A network controller using FML can perform per-flow policy checks and authenticate network entities such as switches, users and hosts. Network policies are defined using the FML language and checked against an evaluation machine for each flow initiation in the network, performing efficient lookup and evaluation of a set of rules. This machine is built around the controller's services such as the name-address biddings.

It was our intention to experiment with FML within the agent architecture for SDN we propose in this thesis. However, since FML implementation is not open source and, as far as we know, there is no comercial version of it available we ported a full linear resolution-based inference engine from Oberon2 [Mota, 1993] into C++.

The intelligent agents designed in this thesis also makes use of a high-level declarative logic-based language to specify network policies, properties and other relevant information. Instead of using a logic programming paradigm, which merges declarative with procedural semantics, we experimented an approach that separates logical reasoning from the control of actions as in [Menezes, 1989](but here, for proof-of-concept purposes, the procedural control is made by component implementation of actions). This was proved to be very efficient to build knowledge processing systems where the declarative specification is written using clausal rules and the engine implements a very efficient complete linear resolution based procedure without the restriction of "ordered clauses" like PROLOG or DATALOG: *structured clauses* [Vieira, 1987], which can be seen as a "linear form" of Proof Trees [Bruynooghe, 1983]. More details are presented in Section C.3.1.

B.1.1 Distributed Management

The top level goal of the Internet architecture was to develop an effective technique for multiplexed utilization of existing interconnected networks [Clark, 1988]. It was early 70s and one of the second-level goals was that the Internet architecture should permit the distributed resource management.

The distributed management of the Internet allows gateways and other resources to be implemented and deployed by different management centers. Routing provides an important example of this cooperation between administrative domains in the Internet since they can exchange routing information, even thought they do not completely trust each other. As remarked by Clark [1988], the lack of sufficient tools for the distributed management of the Internet was, at the time, the most significant source of problems.

Distributed management still remains a challenge for the Internet architecture. Policy-based routing protocols, such as BGP, suffer performance fluctuations due to the complexities involved in the cooperation between multiple ASes [Cittadini et al., 2012]. Security concerns, such as DDoS attacks [Peng et al., 2007b], still compromise millions of hosts in the Internet because establishing cooperative DDoS attack defenses across multiple subnetworks is extremely difficult. Another important concern relates to the deployment of advanced QoS services, because performance guarantees need to be kept consistent along the entire packet's path, across several domains, with different administration and technical characteristics [Kamienski and Sadok, 2004].

The software-defined networking paradigm was initially crafted for enterprise networks [McKeown et al., 2008]. From the basic applications of packet forwarding [Sherwood et al., 2010b] to virtual machine mobility mechanisms [Erickson et al., 2008], SDN deployments found their way into large datacenter implementations [Tavakoli et al., 2009] and virtualized networks [Drutskoy et al., 2013]. To the best of our knowledge, the establishment of multiple SDN domains connected and cooperating to achieve large scale Internet services is still an open issue. The benefits of this is the extension of the boundary of SDN deployments from the datacenters to multiple ASes in the Internet.

Unfortunately, even the redesign of the control plane imposed by SDN will not prevent networks to incur in the same problems related to the distributed management existing in the Internet today. Each network will have its controller and the controller will provide features such as routing, forwarding, security mechanisms, and QoS. The controller stills needs to follow the administrative domain's network policies and its behavior must adhere to network policies when dealing with low-level configurations of the network. Then, the cooperation with other domains remains constrained by the initial Internet's design principle of independent management of ASes.

In early related works we extended the SDN concept from the enterprise to multiple ASes setups. The first of these studies was related to the detection of DDoS attacks [Braga et al., 2010]. This research showed the use of intelligent mechanisms in the detection of attacks directed to and from SDN networks. One of the conclusions of the work was that the attack detection can be greatly improved with the new features provided by SDN, such as the flow-based sensing of the network. Despite the powerful abstractions provided by SDN, the mitigation process (e.g the blocking of packets in the origin of the attack) is hard to achieve because of the lack of cooperation features between controllers close to attack sources and destinations.

The other study was related to interdomain routing [Bennesby et al., 2012]. This work built a framework for BGP-based routing between muiltiple ASes using SDN. Despite the fact that controllers in this implementation could communicate and exchange routing information, the behavior of the network application was entirely dependent on the BGP specification. It lacks mechanisms for routing negotiation and cooperation in cases of failures.

These two case studies led us to think about different approaches to the problem of cooperation between multiple SDN domains. Implementing the current practices applied to the Internet were not successful to achieve the type of behavior suitable to network domains perform scalable and efficient forms of cooperation.

As seen in Section B.2, the approach adopted in this thesis was to provide cooperation by means of autonomous entities called agents. The following section describes the concepts of autonomous networks. Unfortunately, the current architectures of agents are not sufficient to deal with the SDN abstractions, then we discuss their limitations in Section B.3.5 and we propose a new framework in Section B.4.

B.2 Autonomous Networks

In their seminal work "A Knowledge Plane for the Internet", Clark et al. [2003] propose a new direction for networking research: creating networks which autonomously control themselves in the presence of changes or failures. This is to be accomplished by a third plane, the *knowledge* *plane*, which uses tools of Artificial Intelligence (AI) and cognitive systems to create models of good network behavior, possibly closing the loop between perception, reasoning, and acting. Thus, autonomous networks, as such, should self-assemble according to high level instructions, reassemble due to requirement changes, detect problems, and recover from failures (due to malicious acts or not).

The idea of introducing a new plane to overcome existing problems of the Internet's data and control planes was unprecedented by 2003, but the use of AI techniques to autonomously control the network was proposed about a decade earlier. Three approaches were the use of intelligent agents systems [Magedanz et al., 1996], mobile agent systems [Bieszczad et al., 1998], and multiagent systems [Boutaba et al., 2003]. Furthermore, correlated research borrowing concepts from AI emerged at the same time as Clark's proposal including cognitive networks [Thomas, 2007] and autonomic networks [Dobson et al., 2006].

All those research efforts attempted to address problems of the current Internet architecture by using frameworks primarily based on AI's *agent* metaphor. Network security, mobility, routing, and quality-of-service, for example, are handled by autonomous mechanisms provided by agents.

Agents are entities with capabilities such as reactivity, pro-activity, and social ability [Wooldridge, 2009]. An agent-based approach enables the programmer to create artifacts that allow networks to self-govern their behavior. Social abilities, like cooperation and negotiation, are used to tackle large-scale Internet problems through interaction between network domains, modeled as *multiagent* environments.

In this section, we list and analyze approaches based on intelligent agents. We classify these approaches using broad characteristics of agent design: (1) agent programs; (2) agent execution environments; and (3) task environments. The attributes of these categories are then used to detail the *status quo* on agent architectures within broad areas of research in autonomous networks (e.g. directly applied AI methods, autonomic networks). The limitations of these approaches are discussed taking into account the main problems affecting the current Internet's architecture and design.

Essentially, the main disadvantage of current autonomous approaches is the lack of a suitable mechanism for handling knowledge without involving complex representations of low-level network information and their reasoning. In a multiple network domain interaction, such as that deployed between the Internet's Autonomous Systems, this information gets even more difficult to deal with. Additionally, it seems that possible solutions for such problems are hard to deploy within the current architectural design of the Internet.

B.2.1 Some Definitions

The research community's interest in applying theories of agents and multiagent systems on networks has increased in the last two decades. As new challenges have emerged due to the increased complexity in managing the Internet, the application of intelligent agents in networks became known under different names, usually related to areas of research or deployment in the networking field.

An early analysis of the application of intelligent agents in the field of networks was presented by Magedanz et al. [1996], where agents are described as autonomous software entities that behave according to some internal mechanism of intelligence, usually based on knowledge and inference. These agents would form what is called *smart networks*, providing the necessary intelligence to the networks to perform tasks autonomously. The agent technology overview presented by Hayzelden and Bigham [1999] complements this view by showing that these agents can select actions when unexpected events happen in the network, increasing its survivability.

Agents can be distributed among the network devices, performing computations, communicating and cooperating with each other. They behave and act in an autonomous way, but seeking to meet common goals of the network [Thottan and Ji, 1998]. These agents can be mobile, moving between different nodes in the network, have different execution environments, and act autonomously while they optimize network resources [Bieszczad et al., 1998].

For Clark et al. [2003], networks should be capable of performing a recognize-act cycle, while learning and reasoning to self-adapt to the Internet's constant evolution in many dimensions. Network tools should embed mechanisms for representing and reasoning with constraints and policies. Furthermore, for Clark et al. [2003], "the ultimate goal of these networks is to allow the expression of goals and policies at high level and use those to generate low-level configurations". In this approach, the *knowledge plane* should be distributed and decentralized, but all mechanisms should keep a global perspective with knowledge integration of the network.

Based on the concept of autonomic systems [Kephart and Chess, 2003], it has been proposed that *autonomic networks* are able to work in a completely unsupervised way, being able to self-configure, self-monitor, self-adapt, and self-heal. They adapt dynamically to the needs of users, reducing complexity and increasing reliability of network services [Dobson et al., 2006]. According to Jennings et al. [2007], these networks are capable of self-governing their behavior within the constraints and pre-established goals, automating and distributing the decision-making process involved during the optimization of network operation. The core of this technology is the closed control loop between the so-called autonomous elements, managed devices, and the environment. This control loop usually performs the tasks of monitoring, analysis, planning, and execution.

A *cognitive network* is defined by Thomas [2007] as “a network with a cognitive process that can perceive current network conditions, and then plan, decide and act on those conditions. The network can learn from these adaptations and use them to make future decisions taking end-to-end goals into account”. Cognitive networks can achieve better end-to-end performance due to the adaptability of the network elements and the flexibility of the process of cognition. Different models of behavior can be used in cognitive networks and multiagent systems capture exactly the distributed aspect of networks.

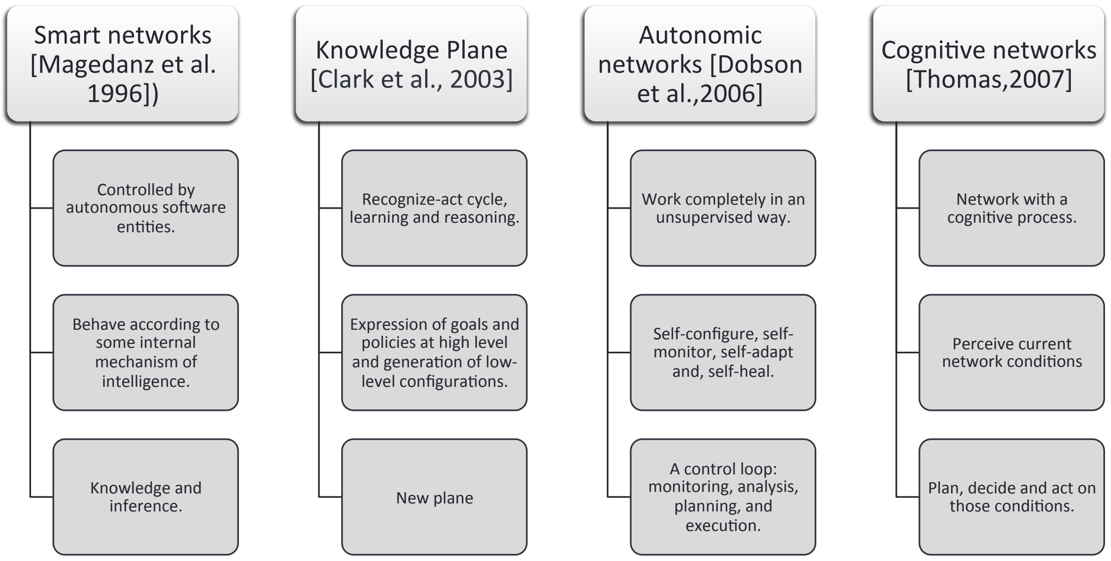
Smart networks [Magedanz et al. 1996])
Controlled by autonomous software entities.
Behave according to some internal mechanism of intelligence.
Knowledge and inference.
Knowledge Plane [Clark et al., 2003]
Recognize-act cycle, learning and reasoning.
Expression of goals and policies at high level and generation of low-level configurations.
New plane
Autonomic networks [Dobson et al.,2006]
Work completely in an unsupervised way.
Self-configure, self-monitor, self-adapt and, self-heal.
A control loop: monitoring, analysis, planning, and execution.
Cognitive networks [Thomas,2007]
Network with a cognitive process.
Perceive current network conditions
Plan, decide and act on those conditions.

Figure B.2: Features of autonomous networking systems from different research domains.

Figure B.2 depicts the many aspects of autonomous networks provided by domains such as autonomic computing, cognitive networks, and smart networks. As it is not the goal of this text to stress all areas which equip networks with some autonomy, we identified the most relevant from the standpoint of scientific literature production and they are used to create a framework of common techniques, in particular those related to intelligent agents.

Herein we adopt the term *autonomous networks* and a definition close to that presented in the networking literature, but oriented by definitions of intelligent agents [Russell and Norvig, 2009] and multiagent systems [Shoham and Leyton-Brown, 2009]. Thus, autonomous networks are those controlled by software entities capable of operating autonomously, perceiving the environment, persisting for a long period of time, adapting to changes, and creating and pursuing goals, always constrained by network policies. Because it is rational, it al also attempts to achieve the best outcome or, when there is uncertainty, the best expected outcome in order to keep network services running. Often, these multiple autonomous entities — when controlling networks with either diverging information or interests — need to interact to achieve a common goal.

B.2.2 Why Autonomous Agents for Networks?

Wooldridge [2009] describes several scenarios where agent and multiagent metaphors are appropriate for problem solving. It is possible to realize how the features of the Internet match the features of these scenarios:

- *The environment is open, or at least highly dynamic, uncertain, and complex.* Undoubtedly, the generality and heterogeneity of the Internet's design led to a great complexity in the treatment of failures and the adoption of changes for networks. Types of problems better handled by more flexible autonomous systems include: packet routing through highly dynamic paths [Akashi et al., 2006], malicious or untrustworthy Internet components [Clark et al., 2003], and the complexity in the management of low-level configuration parameters of large enterprise networks [Casado et al., 2007].
- *Agents are a natural metaphor for modeling.* Some Internet problems require human interpretation in order to handle raw data used in the extraction of useful information for troubleshooting. In such cases, a natural metaphor is the use of agents as intelligent interfaces, assisting and cooperating with humans on some problem [Akashi et al., 2002].
- *Distribution of data, control or expertise.* The Internet design has the assumption that all ASs domains are administratively decentralized [Clark, 1988]. For some problems, these domains need to cooperate or even compete with one another, e.g. during the mitigation of distributed denial-of-service (DDoS) attacks [Chen et al., 2007] or monitoring network traffic [Yalagandula et al., 2006; Jain et al., 2004]. Such domains may often be modeled as multiagent systems and the control of their behavior is directed by cooperative or competitive algorithms and techniques.
- *Legacy systems.* Some obsolete, network managed resources are supported by complex and slow updating information and data models. An *agent layer* functionality can support legacy network components that are technologically obsolete but functionally essential.

In addition to the scenarios noted by Wooldridge [2009], we can cite the problem of dealing with business rules, which should be taken into account when handling the demands of network resources and services; the change of network policies in relation to contextual changes in the network; the adaptation to new users' requirements and environmental conditions [Strassner et al., 2006]; and issues such as the automation of switching/routing and the management of large-scale failures.

All these scenarios require control that is distributed among different organizations, and often their complexity grows rapidly. Therefore, these scenarios can be handled using elements provided by agents, such as the automation and distribution of the decision making process involved in optimizing network operation.

B.2.3 Agents for Autonomous Networks

In this section, agent-based autonomous networks are classified according to properties of (1) agent programs, (2) execution environment, and (3) task environment Russell and Norvig [2009]. The program of the agents can be categorized as reflex agents, goal-based agents, utility-based agents, or learning agents. Execution environment refers to whether they are stationary agents or mobile agents. The task environment property refers to single-agent or multiagent setting. In Section B.3, we use the following classification to characterize current frameworks and architectures used in autonomous networking systems.

Agent Programs

A program embody an agent function, i.e. taking current sensor input into the agent function and applying the actions resulting through actuators. The classification of agent programs is used to review the application of agents to autonomous networks.

Reflex Agents. A simple reflex agent makes decisions based on current observations, or percepts, and may ignore perception history. Therefore, it has limited intelligence and its autonomy is flawed in the presence of unobservable environment aspects because its rational behavior is somehow precomputed in the design phase.

Model-based reflex agents maintain an internal state that depends on the history of perceptions. This internal state is frequently updated and two types of knowledge of the environment are added to the design of the agent: (1) the evolution of the world regardless of the agent, and (2) the effects of the agent's actions on the environment. This knowledge is called a model of the world and the agent uses it along with condition-action rules to make decisions.

In the architecture of the Simple Network Management Protocol (SNMP) [Presuhn, 2002], an Internet-standard protocol for network management, agents are installed on managed devices and send information to a manager entity. This information may be requested or the agent may generate a trap message (unsolicited and created during an exceptional situation). They may be viewed as simple reflex agents, because they can monitor changes in values of the objects being managed and send a message to a network manager if these values exceed some threshold. The internal structure of these agents use condition-action rules that map perceptions to actions.

Another example of reflex agents is presented by Boutaba et al. [2003], where agents are used for resource management and dynamic tuning of switching parameters. Management policies defined by the network administrator are derived from goals and embedded into agents on network switches. The reasoning is done by means of if-then rules, collecting information, such as maximum service delay and average queue size, which is coded into the agent's model of the world.

Goal-Based Agents. These agents also maintain models of the world, but they use goals, instead of condition-action rules, to derive actions. They are more flexible because their model can be modified and the agent's behavior can be changed simply by specifying new goals.

Goal-based agents that control networks autonomously are presented by Esseghir et al. [2008]. In that architecture agents are deployed in the network with global goals and their behavior is controlled by those goals and an internal knowledge base. The agent consults and updates the knowledge base during its operation and, if necessary, a central entity can modify its goals.

Thottan and Ji [1998] presented an anomaly detection system where a method based on Bayesian Belief Networks is used to represent knowledge about failure alerts. The agent's goal is to combine low-level information from network devices and generate high-level abstractions useful to network managers.

Utility-Based Agents. These agents use a more general performance measure, referred to as utility, which enables the comparison between different states of the world. According to Russell and Norvig [2009], "a rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes — that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome". These agents present advantages in terms of flexibility and learning, and also in cases of conflicting goals, or goals that are difficult to achieve together.

Utility-based agents are used to self-organize resource allocation for networks by Eymann et al. [2003]. Through self-interested maximization of utility, agents subjectively weigh and choose preferred alternatives. In Das et al. [2008] and Tesauro et al. [2004] an agent computes an utility function that uses information specified in the application's service-level agreements. The utility function optimizes the allocation of servers of enterprise networks.

Learning Agents. The learning process in an agent involves the modification of each component of an agent's structure in order to achieve a "closer agreement" with the available feedback information from the environment, improving its overall performance. This process allows the agent to operate in initially unknown environments, and become more competent as it acquires new knowledge.

Dietterich and Langley [2007] make a deep analysis of the application of these agents in networks. They show that the detection of anomalies can be accomplished by modeling the network using Bayesian models and then searching for states with lower probability. After detection, learning methods can also be used by these agents in choosing repair methods. Configuration and optimization tasks, such as parameter selection, compatible parameter selection, and topological configuration are performed usually through the learning of a heuristic function *h(x)* that estimates the quality of the best solution reachable from configuration x by applying repair operators.

Peng et al. [2003b] uses learning in the detection of DDoS attacks by sharing information from multiple reflectors. Agents learn when to send a message alerting attacks within an optimal threshold while minimizing the detection delay. A similar approach is employed by Xu et al. [2007], but using reinforcement learning towards optimized strategies of information exchange.

Execution Environments

Aspects of the agent execution environment determine how the agent is situated in the world. Either it is fixed, in some node or server in the network, or it is moving between different nodes. Depending on the option chosen, it impacts directly the design of how the agent executes and collects information from the network.

Stationary Agents. Stationary agents run only on the system where they have been initialized and cannot move. Their execution environment is built on top of the resource being managed or in a central entity controlling other agents. If the information they want is not within the reach of their sensors, they must interact with other agents or systems using a communication system, usually through remote procedure calls or messages.

An architecture of stationary agents is proposed by Rouhana and Horlait [2001], where agents are used to enhance a congestion management algorithm. Many works already cited in this paper use stationary agents employing some of the agent structures presented, e.g. Boutaba et al. [2003] and Esseghir et al. [2008] for network management and Bullot et al. [2008] for network security.

Bieszczad et al. [1998] claimed that stationary agents for networks are less effective, difficult to deploy and awkward when compared with mobile solutions. However, this argument lacks evidence because several vulnerabilities arise from mobile approaches, yielding more possible security breaches in the mobile implementation.

Mobile Agents. There are software agents that can move between locations. Added to the basic structure, mobile agents have a navigation model which is in charge of the agent transportation [Bieszczad et al., 1998]. When dispatched from their original location, mobile agents travel to different network regions (nodes) and perform tasks in the execution environment provided by those nodes, including assessing their internal resources. The agent can return to its origin with some knowledge about visited nodes, or travel for a long time, trying to accomplish its internal goals.

Towards the goal of creating flexible, adaptable, and intelligent network management solutions without increasing the burden on network resources, Stephan et al. [2004] propose a new management platform architecture based on ontology-driven mobile agents. With the same objective, Chen et al. [2009] adapt an autonomic network architecture with mobile agents in order to reduce the complexity necessary to keep up-to-date information models of managed network elements. Mobile agents travel to managed network nodes translating vendor-specific information, monitoring network resources and communicating with other agents.

DDoS attack detection is performed by mobile agents in a work by Akyazi and Uyar [2008]. Their architecture increases the reliability of the detection system because mobile agents still perform detection tasks even in the failure of a control unit. Satoh [2006] describes an efficient and bandwidth-conscious framework for distributed intrusion detection in wireless networks using mobile agents.

Task Environments

A task environment can be categorized in single-agent or multiagent:

Single-agent. When only one agent is responsible for some task in the network. This type of environment was most widely used in early models of agent application to networks. For instance, network supervision [Esfandiari et al., 1998], where agents help to process a large volume of alarms and event notifications through learning techniques.

Multiagent. When the environment is composed of many agents and there are some problems that depend on distributed information or acting, agents are embedded with some models of other agents, protocols and mechanisms for society interaction (e.g. cooperating, coordinating, negotiating) and some communication infrastructure (e.g. languages, message systems). In this way, they can interact in order to accomplish common or conflicting goals in the network.

Boutaba et al. [2003] propose a cooperative multiagent system that provides a self-regulation network control management by means of automatic adjustment of congestion control parameters. A similar approach is used by Esseghir et al. [2008] in managing a DiffServ network by means of intelligent agents. In its architecture, each agent interacts with a central manager and also interacts with a neighboring agents (one hop away).

Bullot et al. [2008] build a distributed intrusion detection system (IDS) based on local

IDS Snort. Agents are embedded within network elements and their role is to share local and situated knowledge. Peng et al. [2003a] propose a system where network attacks are detected by means of the cooperation between agents by sharing their beliefs about potentially suspicious traffic. The framework combines different agents' beliefs and the goal of the multiagent system is to detect attacks with low traffic overhead.

For network management, a multiagent coordination approach is proposed by Tianfield [2003] where the cooperation among benevolent agents and the market based competition among self-interested entities, can be used in various aspects of network service management, resource management, and operation management. For inter-domain routing adjustment, a multiagent framework called AISLE is presented by Akashi et al. [2006], where cooperating agents exchange messages with analysis results of network routing policies.

Table B.1 summarizes the categorization presented in this section. The most relevant references for each category are provided.

Table B.1: Categorization of Autonomous Networks.

	Agent Program				Execution Env		Taks Env	
	Reflex	Goal	Utility	Learn	Stationary	Mobile	Single	Multi
Presuhn [2002]	X							
Boutaba et al. [2003]	X				X			X
Esseghir et al. [2008]		X			X			X
Thottan and Ji [1998]		X						
Eymann et al. [2003]			X					
Tesauro et al. [2004]			X					
Das et al. [2008]			X					
Dietterich et al. [2007]				X				
Peng et al. [2003b]				X				X
Xu et al. [2007]				X				
Rouhana et al. [2001]					X			
Bieszczad et al. [1998]					X			
Bullot et al. [2008]					X			X
Stephan et. al. [2004]						X		
Satoh [2006]						X		
Akyazi and Uyar [2008]						X		
Chen et al. [2009]						X		
Esfandiari et al. [1998]							X	
Tianfield [2003]								X
Akashi et al. [2006]								X

B.3 Architectures for Autonomous Networks

According to Russell and Norvig [2009], an agent architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the actuators as they are generated. Thus, the union between the program and the architecture forms an agent and the relationship between several of them forms a multiagent system, all controlling some portion of the network. Depending on the agent's program, presence of mobility or the existence of more than a single agent in the environment, a particular architecture may be more appropriated than another.

For autonomous networks, the architecture provides the coupling between the internal structure of the agent and the managed resources, such as network entities and channels, data and control planes, and network policies. Additionally, communication and coordination between agents is provided by the architecture.

This section surveys agent-based architectures and frameworks for autonomous networks taking into account the classification provided in Section B.2.3. The decision to separate the architectures by research area and not classify them by general characteristics is that each research area has slightly different objectives, reflecting directly on the construction of the architecture of the agents.

B.3.1 Directly Applied AI Approach

Directly applied AI refers to the application of AI's agent theory with no mention to autonomic concepts, cognitive networks or knowledge planes. It refers to early proposed agent architectures, stationary agent applications, mobile agents, and more recent applications of multiagent theories to networks. The selected architectures provide singular insights on how to integrate agents into networks in order to accomplish autonomous behavior. Gaiti [2008] provides complementary information on agent application to autonomous networks and communications.

Intelligent Networks

Inspired by interface agents, Esfandiari et al. [1998] aims to build intelligent networks with the incorporation of AI techniques into network management and supervision. The architecture proposed has two main components: The Chronicle Recognition System (RS), a model-based simple reflex agent that processes alarms and triggers corresponding actions, and The Learning System, a learning agent which receives tutoring from the network manager and derives new rules to the RS. Situated in a single-agent environment and having stationary property, the agent can actively monitor the network, detect failures and launch alerts.

Meskaoui et al. [2003]

This architecture integrates the notion of agent and multiagent system to the management of networks, specifically DiffServ networks [Meskaoui et al., 2003]. Agents with behavior designed to reach specific objectives are integrated into network nodes. Agents perform actions to prevent and stop congestion.

Agent's structure is composed of an intelligence core and inter-agent communication language for cooperation. Models of behavior are built to allow agents to "carefully" react to congestion by means of the control of different output queues. Agent cooperative behavior allows it to detect congestion in a distributed way, sharing information with neighbor nodes.

SILO

Taking a holistic view of the network, one of the objectives of SILO (Architecture for Service Integration, controL, and Optimization) [Dutta et al., 2007], is to allow applications to work synergistically with the network architecture to select the most appropriate functional blocks and tune their behavior so as to meet the exact user requirements and optimized performance. Mechanisms for adaptive control along with "cross-layer" interactions for the purpose of optimizing behavior are built into the framework.

In the SILO architecture, a stationary control agent resides within a network node. This goal-based agent is responsible for optimizations within the architecture, taking into account QoS requirements, resource availability and any policies in place at this time. These agents have abstract representations of services and mechanisms for formal reasoning about their properties and interactions. Aiming to optimize its behavior, the control agent must communicate with agents on other nodes in the network in order to share information about the nodes.

DWRED

The Dynamic Weighted Red [Rouhana and Horlait, 2001] employ a multiagent system to enhance the Random Early Detection (RED) congestion management algorithm. Each agent dynamically modifies parameters of classes of traffic in the router. Agents take into account the network congestion and their behavior is reflexive, since they must follow rules for router queue already determined in the design phase.

Agents can request parameter modifications in neighbor routers (without periodic polling) to control network congestion between participating routers and their agents. Agents are implemented using a generic multiagent platform written in JAVA, that manages the basic functions in the platform.

AISLE

The problem of inter-domain routing complexity between more than 34.998 ASs in the Internet is handled by Akashi et al. [2006]. There is an autonomous policy-based adjustment mechanism achieved by means of the AISLE framework, a multiagent-based model designed to cooperatively control BGP (Border Gateway Protocol) routing information.

The stationary agent deployed in an AS performs intra-AS control functions. The agent determines its actions according to the description of policies, monitors the information from the BGP border routers, modifies the BGP information, and sends the information back to the edge routers in order to adapt to changes in the network status. It cooperates with other ASes in order to perform inter-AS control functions.

Each AISLE agent has three components: policy-control-engine, BGP-controller and cooperative-action controller. The policy-control-engine interprets policies, triggering observation and control actions when necessary. The BGP-controller monitors and controls BGP through iBGP sessions. The cooperative-action-controller coordinates inter-AS cooperative actions for inter-AS routing, monitoring, and control. Communication between the agent modules is performed via RPC/SSL/TCP and all primitive functions for distributed environment are provided by the ENCORE platform [Akashi et al., 2002]. An internal system provides capabilities and roles of agents together with a BGP topology map. The policy is represented by rules which describe a set of actions, such as how to acquire and evaluate results.

Lavinal et al. [2009]

The application of the multiagent paradigm is used by Lavinal et al. [2009] in order to build a self-adaptable framework for network management. In that framework, a manager agent (MA) is embedded with management functionalities it can execute. These MAs have reactive behavior and social abilities. The coupling between MAs and managed elements (ME) is made by some standard managed protocol or direct access to the resource.

Agent's capabilities are represented by means of a management role, an abstract description of the agent behavior. With the purpose of sharing knowledge between MAs, the characteristics of MEs are described by means of ontologies which represent the different types of MEs and the relation between them. Ontologies also represent the management actions that each agent is capable of.

Agent's goals are subordinated to a local plan or a request from another MA. In order to accomplish its goals, a MA should carry out a self-management control loop (Observe-Analyze-Plan-Adjust), strictly tied to inter-MA interactions.

Hegazy et al. [2003]

A simple intrusion detection architecture based on a multiagent system is introduced by Hegazy et al. [2003]. The architecture has four main modules: (1) a sniffing module gathers packets from the network. It is composed by a simple reflex sniffer agent that analyzes packets on the wire; (2) an analysis module analyses packets. These model-based reflex agents must keep track of the environment and past packets. Internally, they have signatures of attacks; (3) decision modules take actions according to the severity of the attack. Goal-based or utility agents which know the goals and policies of the network. A common task of these agents is to communicate with the network administrator; (4) alert module implemented using simple reflex agents, generate reports and logs.

All agents are installed in some node that monitors the network. Sniffing agents pass alarms to analysis agents, which then pass results to the decision agents, which then pass decision actions to the alarm agents. The detection occurs in an on-line form and agents have their own thread of control.

An early adoption of intelligent agents to intrusion detection is introduced by Thottan and Ji [1998]. The main difference from prior works is that the sniffing agent does not collect packets directly, instead it collects information contained in MIB (Management Information Base) variables and uses statistical approaches for alarm generation such as Piecewise stationary auto-regressive (AR) models.

Xu et al. [2007]

In the architecture proposed by Xu et al. [2007], DDoS attack detection is performed by means of the cooperation among distributed detection agents. Those agents are also embedded with learning capabilities in order to optimize the communication costs among detection agents.

Multiple detection agents are placed into edge routers and communicate via messages passing. Information combination is employed in order to increase the accuracy of local observation. Based on early observations of normal traffic, Hidden Markov Models (HMMs) are used to model the statistical behavior of normal traffic and these HMMs determine when anomalies occurs.

A reinforcement learning method, based on rewards from the environment, is used to make the agent learn the time to send broadcast messages to other agents in a manner that does not overload the link capacity. The agents of this architecture deployed a simplified version of a Distributed Reinforcement Learning (DRL) algorithm.

Akyazi and Uyar [2008]

In the architecture presented by Akyazi and Uyar [2008], each network host has a stationary agent in charge of monitoring the host. As soon as an anomaly occurs, it warns a main agent, located at a different and secure node in the network. The main agent creates different mobile agents specific to the detected attack and these agents travel into the network in order the increase the reliability of detection results. The implementation of mobile agents uses the JADE platform and they have DDoS datasets to identify anomalies.

An early adoption of the idea of remote code execution is performed by delegated agents [Goldszmidt and Yemini, 1998]. In this environment, "a remote elastic server" accepts instantiation of threads belonging to delegated agents. These agent could also manage network devices dynamically programming SNMP MIB entries.

Gavalas et al. [2009]

A mobile agent platform for network management is proposed by Gavalas et al. [2009]. This platform is composed by agents with narrow intelligence, limited autonomy, acting on behalf of someone, and mobile. These mobile agents travel through the network using a heuristic algorithm called HIP for itinerary planning, reducing network overhead due to agent's transfer. As soon the agent arrives at a destination, it is executed by a JVM running in the network node and a number of methods coded into the agent assist in the interaction with the managed device.

Satoh [2006]

A two-layered mobile agent framework is introduced by Satoh [2006]. In that framework, two types of agents act in the network. A navigator agent is in charge of carrying task agents and can be optimized for a particular sub-network. A task agent is application-specific and performs its management tasks at all nodes it visits. Each agent is globally identified and is implemented using virtual machines and objects. They have limited intelligence, acting more like reflex agents executing pre-defined tasks at each visited node.

B.3.2 Knowledge Plane Based Approaches

In general, the term "knowledge plane" proposed by Clark et al. [2003] refers to two distinct concepts. The first indicates the construction of mechanisms for autonomous control of the Internet based on artificial intelligence tools in order to build more reliable networks. The second indicates that these mechanisms should not be introduced into the planes which already exist on the Internet — data and control plane — but a new plane with more flexible characteristics should be created. As Clark's position paper pointed out that intelligent agents and multiagent systems are the most appropriate tools for the construction of this plane, this section presents the knowledge plane architecture as well as other architectures based on it with the same goals.

The Knowledge Plane (KP)

A new construct in the network that has the following characteristics:

- it requires the edge and core network involvement,
- it provides a global perspective of the location of information necessary for problem diagnosis,
- it has a compositional structure which deals with imperfect and conflicting information,
- it provides a unified approach to knowledge and has a cognitive framework as foundation.

The cognitive framework deals with incomplete, inconsistent, and possibly misleading or malicious information. It has goal conflicting resolution mechanisms and is general enough to accommodate future innovations in the Internet. After constant evolution, the knowledge plane should close the control loop with the network in a recognize-act way. This control loop requires features such as representation, reasoning and learning in order to cope with Internet's constant development.

The knowledge plane architecture is distributed, compositional and multi-scale. Despite the distributed nature, there must be a global perspective to knowledge, combining percepts and assertions from different stakeholders. As asserted by Clark et al. [2003], the core of the architecture must support cognitive computation, what is a challenging problem, due to the highly dynamic and distributed KP environment and the limitations of current knowledge level algorithms and agent architectures.

Situatedness-Based Knowledge Plane

For Bullot et al. [2008], the KP should only be a protocols/algorithms layer to manage knowledge within the network, in order to design new autonomic control/management algorithms. Consequently, the situatedness KP obtains information from the control plane (being embedded on the nodes of the network or through standard management protocols) and gives meaning to the information, converting it into knowledge.

Agents are embedded into the network equipments that can be edge-routers, switches, servers, etc. These network equipments perceive and act on their environment. As each network equipment is considered an agent, the network as a whole can be considered a multiagent system. In this architecture, each agent has a situated vision of the environment and communicate with others to share a part of their internal knowledge, possibly extending it.

The situated KP was implemented to help the self-management of routing optimizations, anomaly alarm correlations and a collaborative intrusion detection system. Two main performance measures are used to evaluate the implementation: computational load overhead in the network equipment and the network load overhead.

Information Planes

A similar approach to the KP is called information planes. These planes supply information about the network and leave the task of adapting to some internal mechanisms, without resorting to AI techniques for autonomous behavior. The importance of them is that they extend the concepts of the KP for new domains, such as the Sophia information plane described by Wawrzoniak et al. [2004], that collects, stores, propagates, aggregates, and reacts to observations about the networks's current condition. The Sophia architecture is based on a distributed system running throughout the network that collects information about network elements, evaluates statements about this state and reacts accordingly to the results. The iPlane [Madhyastha et al., 2006] uses a centralized agent concept to distribute measurement tasks in the network in order to obtain better performance of overlay networks.

B.3.3 Autonomic Based Approaches

The aim of autonomic network architectures is to build networks which can work in a totally unsupervised manner, able to self-configure, self-monitor, self-adapt, and self-heal, terms also referred as self-* properties. According to Dobson et al. [2006], the origins of autonomic networking comes from IBM's autonomic computing initiative [Kephart and Chess, 2003]. In autonomic computing, autonomic elements performing fixed functions interact with other elements, possibly in a very dynamic environment. These autonomic elements close a control loop with the managed elements, entities with functional behavior in the environment. This control loop is dependent on the control theory used, but often has components to collect, analyze, decide, and act over the managed elements. Kephart and Walsh [2004] argue that autonomic components must be designed as rational agents, because agents precisely map the behavior of such components, because agents "perceive and act upon their environment, selecting actions that, on the basis of information from sensors and built-in knowledge, are expected to maximize their objective".

This section presents agent-based architectures within the autonomic network research. For more information about other types of architectures, Dobson et al. [2006] and Samaan and Karmouch [2009] present general surveys on autonomic networks and communications.

Unity

Software architecture proposed by Tesauro et al. [2004] aiming to self-manage distributed computing systems, including networks. In this architecture, agents control resources and deliver services to humans. They have an internal autonomic behavior and perform the self-* operations in the managed elements. Externally, the architecture is organized in a multiagent setting and populated by several types of autonomic elements with specific skills, such as a resource arbiter element, registry elements, policy repository elements and sentinels. These types of agents are used to deploy self-* properties to the system, such as goal-driven self-assembly, self-healing and self-optimization. The latter, for example, uses service-level utility functions to dynamically allocate and manage compute servers within the data center.

Kephart and Das [2007] extend the Unity architecture with an effective utility-function resource-allocation engine based on utility-based agent programs. Das et al. [2008] extend the application areas of the architecture and deploys an agent-based resource allocator. Lubin et al. [2009] apply new market mechanisms to the multiagent setting.

FOCALE

The Foundation Observation Comparison Action Learn rEason (FOCALE) autonomic network management architecture [Jennings et al., 2007] is based on the possibility to adapt the behavior of the network control loop in order to cope with observed changes. In order to accomplish this adaptation two control loops are implemented: a maintenance control loop and an adjustment control loop. The latter is responsible to adapt the maintenance loop to network policy changes.

FOCALE is a distributed architecture, where individual network devices incorporate autonomic management software implementing both already mentioned control loops. These network devices are referred to as managed resources and they are coupled with an autonomic manager (AM) through a model-based translation layer (MBTL). The AM is not vendor-specific and access the resources of the managed device by means of an information model, the system ontology and the set of deployed policies. The MBTL is in charge of information delivery to the AM and the deployment of corresponding actions. The MBTL is a special component because it needs to have in-depth knowledge of the managed resource, translating vendor-specific data to DEN-ng compliant vendor-neutral data. In a multiagent setting, each AM can communicate with other AM in order to coordinate activities and achieve common goals.

Chen et al. [2009] extend the pure autonomic characteristics of the FOCALE architecture with intelligent agent concepts. In this architecture, adaptive mobile agents are added to the managed resources, simplifying the function of the MBTL. These agents translate their own languages to a common language (XML), allowing heterogeneous devices to communicate with each other.

ANA

The Autonomic Network Architecture (ANA) [Bouabene et al., 2010] provides a framework and execution environment for the development and testing of autonomic communication protocols. This architecture supports the dynamic adaptation to environment and network components changes by means of the introduction of generic networking abstractions and communication primitives. Bouabene et al. [2010] indicates that the ANA architecture could be used by autonomic management systems like FOCALE [Chen et al., 2009] as an abstraction layer and flexible machinery for autonomic services operation.

Similar approach was took by Gogineni et al. [2010] when defining an autonomic layer foundation for network management. The aim of that network layer is to guarantee the robust communication between autonomous control agents. Another similar approach is presented by Razzaque et al. [2006] where a cross-layer architecture is depicted.

Esseghir et al. [2008]

In the architecture proposed by Esseghir et al. [2008], agent technology is used to manage network resources. Several types of agents are deployed over a part or entire network equipments according to global objectives defined by a central authority. In a multiagent setting, each agent can communicate directly with the central entity or with their neighboring agents.

Central to this architecture is the creation of two new planes: the governing plane, in charge of controlling the deployed agents, and the knowledge plane, in charge of expanding the agent's knowledge base. The agent internal structure is organized as follows: a knowledge base, a communication module, an action module, an interfacing module with the governing plane, a decision module and an information collecting module. During a multiagent setting, agents can share information about the network devices using inter-agent communication mechanisms.

Generic Approaches

Some works provide generic application of agents and multiagent systems to autonomic networks. Lavinal et al. [2009] proposes a self-adaptive management framework for networks. Its architecture uses multiagent concepts, ontologies and cooperative aspects. Similar approaches for network management can be found in [Cheng et al., 2006], [Schmid et al., 2006] and [Tianfield, 2003]. The latter using distributed intelligent agents.

B.3.4 From Cognitive Networks

Architectures of cognitive networks can be categorized into two objectives: the first centers on using cognition to aid in the operation and maintenance of the network, while the second centers on cognition to solve “hard” problems, problems that do not have a feasible solution other than the use of cognition [Rouhana and Horlait, 2001; Thomas, 2007].

Dietterich and Langley [2007] present learning architectures for cognitive networks. In some of the architectures presented, learning of knowledge is deployed in order to select actions and plans for an agent to carry out in the world. An agent can also learn policies in order to increase network efficiency. A second class is the learning for interpretation and understanding which allows the agent to interpret and understand network events.

B.3.5 Weaknesses of Autonomous Network Architectures

Despite the major consolidation of the theory of agents and multiagent systems, with the creation of robust architectures [d’Inverno et al., 2004], development methodologies [Zambonelli et al., 2003] and security mechanisms [Ramchurn et al., 2004], there are few applications of rational agents to control Internet domains. This section discusses the reasons why applying agent theory to network is a very difficult task and discusses the weaknesses of the works that attempted to do it.

The greatest challenge for the frameworks and architectures described in Section B.3 is the complexity of knowledge representation and reasoning. In order to create a model of the network, intelligent agents must map real entities and relationships to internal representations suitable for efficient reasoning. The problem appears when the agent needs to deal with several types of different network entities, with different models of behavior, distributed knowledge among several nodes or even ASs, conflicting policies and goals.

Due to the highly dynamic characteristic of networking, an agent’s knowledge base must be updated frequently. Even for a small network domain, the knowledge plane can turn into a massive base of information that must be handled by the agent’s reasoning mechanism. Depending on the agent’s reasoning mechanism and the knowledge base size, the time needed for obtaining a result (e.g., the next action to take) can become impractical. Furthermore, the management of information from different sources can incur in uncertain or obsolete knowledge bases that do not reflect the current state of the network.

The works mentioned in Section B.3 fail to provide efficient mechanisms for knowledge base creation and maintenance. They often depend on inaccurate information provided by humans [Esfandiari et al., 1998], information from highly distributed sources which are difficult to synchronize [Lavinal et al., 2009], and very specific information about network resources that are hard to update [Dutta et al., 2007].

The FOCALE architecture [Jennings et al., 2007] tries to overcome some of these limitations by means of a model-based translation layer which is in charge of converting vendor-specific models and presenting to the agent a single model. This is a limited solution, since the agent architecture is still dependent on the frequency of updates of the information model used.

Due to the difficulty to create concise knowledge bases, an agent's reasoning mechanisms are often tightly coupled to the problem being addressed. For example, the Unity architecture [Tesauro et al., 2004] uses utility functions specifically tailored for network resource dynamic allocation. The architecture proposed by Xu et al. [2007] used specific Hidden Markov Models tailored for DDoS attack detection and mitigation. Also, works such as the Autonomic Network Architecture [Bouabene et al., 2010] and the Situatedness Based Knowledge Plane [Bullot et al., 2008] fail to demonstrate how their architectures perform network reasoning. Such approaches seem to be very distant from the objective of building generic agents capable of handling a large number of Internet's problems and evolving over the time.

Another important limitation of autonomous networking approaches is the complexity involved accessing the network. As agents need to access network resources, they must be embedded in a network node or the network node must provide some interface for access. Not all nodes have enough resources to implement agents on top of them, and a general access interface is still very difficult to deploy. Even harder is to try to update closed systems such as proprietary routers/switches, as well middle-boxes in the network. The works of Bieszczad et al. [1998], Stephan et al. [2004], Chen et al. [2009], Satoh [2006] and Akyazi and Uyar [2008] require the existence of an execution environment in the network resources. This type of requirement leads to agent architectures very dependent on low-level network hardware and often prone to security vulnerabilities, since a malicious code can be transported into an agent's knowledge base.

Since there are more than 40,000 ASs in the Internet [CIDR, 2012], and they are conceived to be managed by different stakeholders, the success of some of the applications of agents to networks depends on the large-scale adoption of the solution by the ASs. The approaches of Balasubramaniam et al. [2009] and Xu et al. [2007] require the interaction of agents present in each administrative domain.

This is very difficult to guarantee, due to the complexity involved in managing such a distributed approach, where different network policies are deployed concurrently. Furthermore, the structure of the Internet inhibits large changes in network architecture. Thus, large-scale Internet scalability tests with agents are almost impossible to deploy. Those tests are important to evaluate the long-time behavior of agents, as well as the convergence of multiagent systems cooperative or competitive methods.

Lastly, Clark et al. [2003] pointed out the necessity to create economical incentives for the implementation of cognitive systems in the Internet. The agent architectures lack these mechanisms, which can guarantee the benefits for all participating ASs. Besides, the need to change each router or end-system seems to be a disincentive for agent-based architecture adoption.

B.4 Autonomous Software-Defined Networks

The horizon toward an agent architecture widely deployed on the Internet seems quite distant. The limitations set forth in Section B.3.5 are difficult to resolve in the current Internet architecture, with its emphasis on generality and heterogeneity. A paradigm shift seems to be the new perspective.

The main limitations and weaknesses of agent architectures described in Section B.3.5, such as the complexity of agents to handle knowledge representation and reasoning, the arduous task of deploying agent frameworks, and the complexity to handle the total distribution of control between ASes, seems to prevent the large-scale adoption of agent techniques in the current Internet.

As seen in Section B.1, the key concept of SDN is *abstraction*. The ideas of using abstract data structures which represent network entities and logically centralize them in a network OS are somewhat revolutionary. Many of the complex protocols and algorithms are simplified by adopting such abstraction and even more powerful versions of them can be designed and rapidly tested.

The SDN abstraction seems to be the most promising way to successfully create agent-based architectures which control and manage large-scale parts of the Future Internet. Many of the limitations of autonomous networks today can be fulfilled by features presented by network operating systems. Figure B.3 tries to capture the relationship between the weaknesses of agent frameworks and features of SDN.

Knowledge representation within agent frameworks is totally dependent on the network view maintained by the network OS. This network view allows agents to model different entities in the network such as protocols, packets, routes, access control lists (ACL), users, services. In this way, it is possible to create concise representations without the need to handle low level entity

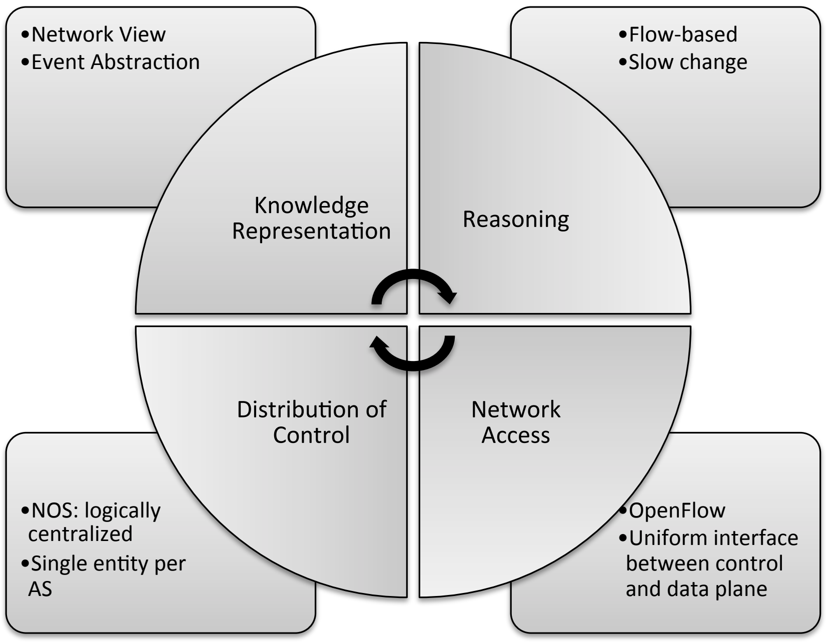

Figure B.3: Relationship between SDN and agent framework weaknesses.

presentations such as IP address, MAC address, protocol packet headers and states. Keeping knowledge about high level abstractions reduces the burden of manipulating an exponential number of facts into the agent's knowledge base. Furthermore, the network OS generates events each time the network changes its state. These events can be used to update the variable states or facts in the knowledge base.

Reasoning mechanisms which generate agent actions can be very efficient with SDN. Instead of performing per-packet reasoning, agents perform flow-based reasoning. It means that the agent only handles flow-initiations (the first packet in a flow) and some events (related to its goals). This greatly reduces the scale and the time needed to entail an action.

In SDN, agents do not directly access network resources, then they do not need to implement interfaces for every network entity they relate with. This interface is the OpenFlow protocol which provides secure and concise access to network entities. Every action which must be performed on behalf of agents is sent through OpenFlow. Every information coming from the network is delivered through the OpenFlow. It is not necessary to implement agent execution code for every network router or host in the network. This reduces the problems with different execution environments or even security breaches introduced by such frameworks.

Finally, agents are deployed on top of the network operating system. Instead of distributing agents to every node in the network, they are logically centralized in the network OS. For large-scale Internet deployment, we can say that each AS has its own network controller supporting agent technology. Furthermore, we can reduce the number of agents from millions to thousands, which corresponds to the number of ASes in Internet. In this way, it is possible to reduce the complexity of the protocols for interaction and negotiation between ASes, as well the reduction of management traffic being carried out in the Internet.

Passito et al. [2010] proposed the redesign of autonomous networking with SDN concepts. That work pointed out the need to enhance network operating systems with agent capabilities, such as reactivity, pro-activity and social ability. This approach enables the building of artifacts for the autonomous control of networks, allowing networks to self-govern their behavior, but only within the constraints of the goals that the system as a whole seeks to achieve. To tackle large scale Internet problems, social abilities like cooperation and negotiation may be used to make agents interact with other network domains.

Using such high-level and centralized abstraction of the network will reduce the complexity of building agents in a complex and often uncertain environment. This feature expressively reduces the burden to construct a translation layer into each agent to cope with different network vendors. From the network operating system perspective, agents are used as an efficient manner to build autonomous network control artifacts. Components, now characterized as agents, can be used to build self-managed networks and exploring autonomous solutions for configuration, optimization, healing and protection.

B.5 Appendix Remarks

This appendix presents the concept of software-defined networks and discuss their limitation when dealing with cooperation between ASes in the Internet. We saw that this problem can be addressed using social abilities provided by autonomous agents.

Agent-based architectures for autonomous networks are presented. A general definition is presented taking into account single agent [Russell and Norvig, 2009] and multiagent [Shoham and Leyton-Brown, 2009] concepts, since many areas of research have a particular definition according to their aims. A categorization based on the characteristics of intelligent agents is depicted. The types of programs for agents, execution environment and characteristics of the environment where agents are situated are presented and related with research in autonomous networks. Agent-based architectures are presented, categorized according to the broad area of research which they belong, such as applied AI and autonomic networks.

Limitations of current approaches are discussed. Among them the most important are the lack of scalability, the complexity of knowledge representation of so diverse environment, and the difficulty of cooperation among the many AS on the Internet. Finally, a new direction for research is presented: the building of autonomous agents and multiagent systems on top of a network operating systems. In this way, autonomous agents could access network resources in a totally different way. More efficiently and scalable architectures can be created, new forms of knowledge representation developed and the interaction between Internet AS deployed with less complexity.

Appendix C

AgNOS: Autonomous Control of SDNs

Intelligence has two parts, which we shall call the epistemological and the heuristic. The epistemological part is the representation of the world in such a form that the solution of problems follows from the facts expressed in the representation. The heuristic part is the mechanism that on the basis of the information solves the problem and decides what to do.
John McCarthy (1927–2011)

Contents

In Appendix B, we saw the concept of a Software-Defined Network. Furthermore, we presented a literature survey of autonomous networks and discussed the limitations and weaknesses in the architectures that have been proposed. Finally, we proposed the extension of SDNs with autonomy capabilities to cope with several open issues, mainly the distributed control of network domains.

In this appendix, we present a *multiagent framework*, called AgNOS, which aims to provide SDN domains with new capabilities. Proactiveness enables them to have goal-directed behavior and social abilities give them a manner to cooperate with other agents in order to satisfy their design principles. We use these new capabilities to address issues that arise when there is a relationship between several SDN domains.

The appendix begins describing an *AgNOS agent*, the building block for the development of a multiagent architecture for the control of networks in the Future Internet composed by SDN-based domains. Then, we describe this architecture, the declarative level of policy network specification and reasoning. We also briefly describe the procedural level and we also discuss some implementation issues.

C.1 A View of Agents in SDN

C.1.1 AgNOS Agent

Based on the definition of autonomous networks in Section B.2.1, an agent is a software entity capable of operating autonomously, perceiving the environment (i.e. the network), and creating effects or changes in it. From this point of view, an agent for the Future Internet, should have capabilities to deal with resources (e.g hosts, links, routers, switches), network policies, network traffic, users, and so forth. An *AgNOS agent* is defined by a set of attributes and processes to represent its behavior. Such attributes can be static or dynamic, and internal or external.

Internal attributes are private, i.e. are not accessed by the external world, but other AgNOS agents may know or ask about them. Whether or not the required information is supplied depends on the AgNOS access rules. On the other hand, external attributes are public, and so visible to others.

The behavior of an AgNOS agent is manifested by the actions it takes upon the environment and also upon its attributes by means of its internal process. An action of an AgNOS agent may succeed or fail and affect the environment through *events* it generates or *messages* it sends to the subject of the action. Events are handled by a network controller which is in charge of applying the low-level commands related to each event. Messages sent to other AgNOS agents are similar to the ones used in multiagent systems communication based on the *speech act* approach [Searle, 1969; Cohen and Perrault, 1979]. The behavior of an AgNOS agent is affected by the environment, depending on the events generated by the network controller and the contents of the messages it receives.

An AgNOS agent may receive and send requests. It also can send information about its internal state. Furthermore, the AgNOS agent can order some action upon other agents. It may or may not honor requests, and may send requests, queries and information which are not necessarily related to the messages it receives, but to its internal knowledge such as beliefs and desires. Even in this case, the agent sends messages to express the effects of its action.

The following sections describe the set of attributes which define AgNOS agents, i.e., the knowledge associated with the AgNOS agent specification, actions of agents, and groups of AgNOS agents.

C.1.2 Knowledge Associated with AgNOS Agents

Although network representation is a NOS responsibility, for reasoning purposes we suggested, in section C.1.1, that AgNOS agents should internally represent the network resources they are able to control or which have influence upon their behavior. Due to performance issues (the solution's search space can grow exponentially), the AgNOS agents must have a partial representation of the system/environment and other agents located in other domains. The next sections describe the elements of this knowledge.

Environment of AgNOS Agents

AgNOS agents are capable of controlling network domains. At a first, we need to define an environment as a network domain where agents will be placed and ran in order to keep the behavior of this domain within desirable states.

Agents are located on top a network controller in charge of this domain and the knowledge they build about the environment is constrained by the interface provided by the network controller. In this way, agents have access to a network view, which is a data structure that represents a set of information maintained by the network controller. Agents also have direct access to events generated by management applications and messages sent by other agents.

Figure C.1 depicts an example of information handled by AgNOS agents to build their knowledge. The low-level attributes of the network are represented on the right side of the figure.

The mid-level attributes represent attributes directly manipulated by a network controller. The high-level attributes are the ones accessed directly by AgNOS agents (*i.e.* maybe embedded in the agent's knowledge base). Note that Figure C.1 only shows a fraction of the information in the environment. Section C.2 extends and formalizes this information in the AgNOS architecture.

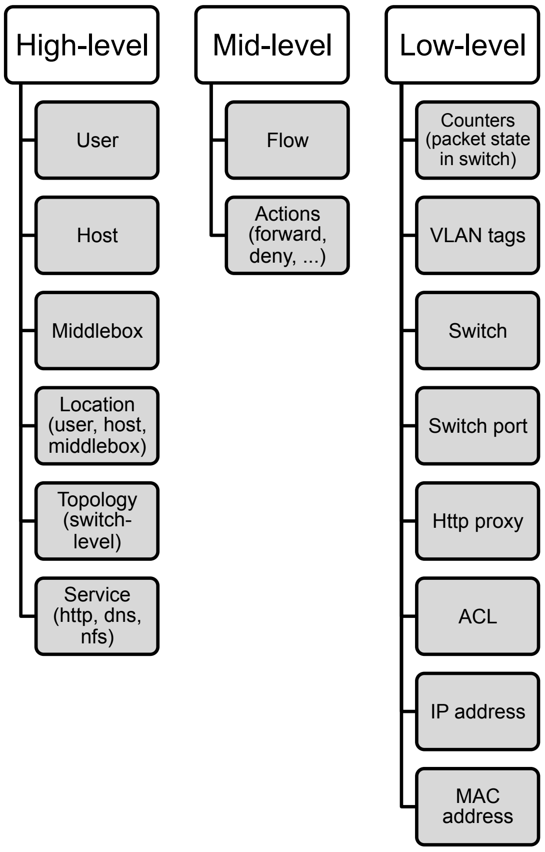

Figure C.1: Example of information involved in AgNOS environments.

Events are another source of information. Controllers may not introduce the information provided by an event directly in the network view. In this way, AgNOS agents must have event handlers in their sensors capable of analyzing the event content and embed that information in the knowledge base. Example of common events which can be handled by AgNOS agents are those related to switch status, for example joining and leaving the network, packet received, switch statistics received. Also related to switch and host level topology, some services provided by the network, users and hosts authentication, network policies, and so forth. Section C.2 provides a definition of how AgNOS agents use this type of information about the environment.

Messages provide information about different network domains from other AgNOS agents. Messages may contain information about network attributes often related to some issue which involves different domains. For example, in a counter DDoS-scenario, it is necessary to provide an agent's peers with information about traffic features and the location of attackers. This information is sent to related AgNOS agents that process it and embed this knowledge in their base for up-to-date decisions. Section C.2 presents the types of messages and their contents in the AgNOS architecture.

An agent's environment is a dynamic structure, generated according to the current status of network entities. Two things are needed to build it: a language to represent such knowledge, and specialized inference mechanisms. The language should be able to represent schemas of reasoning about network entities and their relationships.

C.1.3 Actions of AgNOS Agents

The actions an AgNOS agent can perform are those that change its state, the state of the environment or that cooperate with other agents. Usually, changes are considered as functions which map the value of attributes of an agent from one state to a subsequent one.

Definition C.1 (Action of an AgNOS agent)**:**
Let Λ be the set of possible states of an agent A and Υ the set of information resource of A. An action *of A is a mapping $\mathcal{A}$ from $\Lambda \times \Upsilon$ to Λ, written as $\mathcal{A} : \Lambda \times \Upsilon \rightarrow \Lambda$.*

An action is a composition of sensing the environment, deciding what to do and executing the action. Actions, the outcome of agent's actuators, can be described as events. An AgNOS agent is capable of generating events that are processed by the network controller, which enforces the low-level directives in the switches and updates the network view. AgNOS agents also generate an specific form of action called messages, which are delivered to the interested agents in another domain. The delivery of messages can be interpreted as events.

C.2 An Architecture for AgNOS Agent Systems

In Section C.1, agents were presented from a SDN modeling point of view. In this section, we describe one way of implementing the concept of AgNOS agents.

C.2.1 Basic Assumptions

Symbolic Architecture

The central computational idea behind multiagent systems is to reduce the search space (within an agent) by distributing parts of it. Whenever an agent needs information about the world, and this information does not concern the agent itself, then the agent interacts with the environment. This is always the case, no matter whether the architecture is symbolic or reactive or hybrid. Symbolic architectures are concerned with how the world (the external information of the agent) is symbolically represented, as well as the interactions happening. The agent's point of view of the environment is represented by the sets of messages the agent sends and receives, and events dispatched by the network controller.

Communication

We make the following assumptions about the process of communication between agents in the AgNOS architecture.

- Agents use a peer-to-peer architecture to send and receive messages.
- An agent can only access messages and network events addressed to itself.
- The order in which messages arrive may be different from the order they are sent. In this case, AgNOS relies on a connection-oriented transport layer protocol.
- A message may or may not be delivered. The channel can contain errors and there may be packet loss.
- An event generated by a network controller is always delivered to the agent. Whenever the agent creates an event it is delivered to the network controller.
- Whenever an agent tries to read a message it will eventually succeed, but it does not mean the agent will be locked in a busy-waiting state until the message arrives.

Atomicity of Computation within an Agent

To simplify the design, the level of atomicity of an agent's overall computation is limited to the level of a message or event received from the environment. This means that an agent will read and process just one message or event at a time. Only after processing a message or an event it may read another one, and so on. The complexity of adding parallelism is avoided in this way.

As the order in which the messages and events are received is non-deterministic, then the outcome of an agent's overall computation through the logical time should be similar to the one generated by another agent with real parallel processors. The only difference, as stated above, is that the latter would need a more detailed specification of its parallel processes accessing shared knowledge.

Resource Acquisition and Interaction

An AgNOS agent is able to exchange and negotiate the resources it needs with others. Also, an AgNOS agent does not need to send messages to all individuals within the environment, but only to those which may influence the problem solving. In a counter-DDoS scenario, for example, only agents belonging to attacking domains might receive messages from the victim.

We can separate the computation in two levels. On one level, there are the actions related to changes on the attributes of the AgNOS agents. On another level, there are computations related to interactions between the agent and the members of the environment in order to obtain information or request services. The concept was proposed by Mota [1998], and we adopt it here for architecture state level and AgNOS state level, as depicted in Figure C.2. The state level shows a subset of AgNOS agents (A_1, A_2 and A_3), executing actions, say α_1, α_2 and α_3 that may change their state. Changes at this level may be shown directly at the upper level if they are public, or indirectly when they generate message exchange among the agent and the environment.

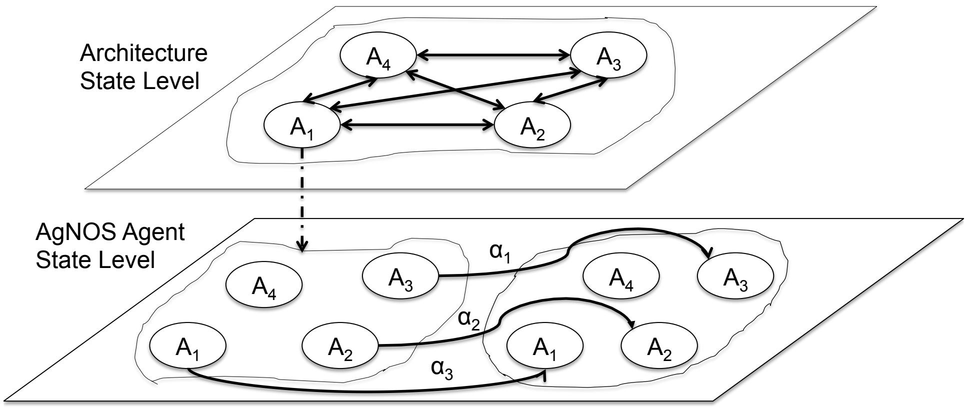

Figure C.2: Architecture and state level of AgNOS.

C.2.2 AgNOS Agent Features and Life Cycle

A formal model for intelligent agents is presented by Wooldridge [2009], based on that given by others [Genesereth and Nilsson, 1987; Russell and Wefald, 1991; Russell et al., 1993]. In this section, the AgNOS agent behavior is specified taking Wooldridge's formal definition of agency as its base. Furthermore, the agent architecture is depicted along with its task environment specification.

First, every agent is situated in some environment. It is assumed that the environment may be in some instantaneous state of a finite set E of discrete states. AgNOS agents are assumed to have a range of possible available actions, which transform the state of the environment. A *run*,

r, of an agent in an environment is a sequence of interleaved environment states and actions:

Definition C.2 (Run)**:**
Let E be a finite set of states $\{e_0, e_1, \ldots, e_n\}$, A be a finite set of actions $\{\alpha_0, \alpha_1, \ldots, \alpha_n\}$. The *run* of an agent is defined as the sequence:

$e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{n-1}} e_n.$

AgNOS environment states represent the current state of a network domain. Every state e_i indicates how the network domain is currently viewed in terms of topology and services. The actions of AgNOS agents are bound to the network controller abstraction layer. All network controller primitives can be modeled as actions, such as the insertion and deletion of flow table entries, and the generation of events. The process of sending messages to other agents through the network controller is also considered an action.

Following Definition C.2, there is a starting state e_0 (*i.e.* after the network controller initialization) and agents choose an action to perform for every new state achieved. The carried out action is defined after the state transformation function execution:

Definition C.3 (State Transformation Function)**:**
Let E be a set of finite states, A be a finite set of actions, R be the set of all possible finite sequences (over E and A), and R_A be the subset of R that end with an action. A *state transformation function* is defined as:

$\tau \colon R_A \to 2^E.$

Definition C.3 shows that a state transformer function maps a run to a set of possible environment states. Let $r_0, r_1, \ldots, r_n$ to stand for members of R, then if $\tau(r_i) = \emptyset$, there are no possible successor states to r_i. In this case, the system has ended its run.

Definition C.4 (Environment)**:**
The environment Env is a triple $Env = \langle E, e_0, \tau \rangle$, where E is a set of environment states, $e_0 \in E$ is an initial state, and τ is a state transformer function.

Following the Definition C.3, an environment Env is defined based on environment states, an initial state and the state transformer function.

Definition C.5 (The AgNOS agent)**:**
Let R_E be a set of runs that end in an environment state, and A a set of actions. The AgNOS agent is defined as a function which maps runs to actions:

$Ag_{AgNOS}\colon R_E \to A.$

From this definition it follows that the decision-making process of the agent depends on the history of environment states and previous actions.

Following Definitions C.4 and C.5, a *system* is denoted as a pair containing an agent and an environment. The system will have associated with it a set of possible runs.

Definition C.6 (The AgNOS Agent Run)**:**
Let $R(Ag_{AgNOS}, Env)$ to denote the set of runs of agent Ag_{AgNOS} in environment Env. A sequence $(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$ represents a *run of an agent* Ag_{AgNOS} in environment $Env = \langle E, e_0, \tau\rangle$, if

1. e_0 is the initial state of Env;

2. $\alpha_0 = Ag_{AgNOS}(e_0)$; and

3. for all $n > 0$,
$$e_n \in \tau((e_0, \alpha_0, \ldots, \alpha_{n-1})),$$
and
$$\alpha_n \in Ag_{AgNOS}((e_0, \alpha_0, \ldots, e_n)),$$

The AgNOS agent life cycle is understood in terms of the following definition.

Definition C.7 (See)**:**
Let Per be a (non-empty) set of percepts. Then *see* is a function:

$see\colon E \to Per.$

Definition C.8 (Next)**:**
The function *next* maps an internal state and percept to an internal state:

$next\colon K \times Per \to K.$

Definition C.9 (Action)**:**
The action-selection function *action* is defined as a mapping:

$action\colon K \to A.$

The agent starts in some initial internal state i_0. It then observes its environment state e, and generates a percept $see(e)$. The internal state of the agent is them updated via the *next* function, becoming set to $next(i_0, see(e))$. The action selected by the agent is then

$action(next(i_0, see(e)))$. This action is then performed, and the agent enters another cycle, perceiving the world via *see*, updating its states via *next*, and choosing an action to perform via *action*.

Agent Task Specification

The specification of tasks to be carried out by agents is done via a predicate specification. A predicate specification is denoted by Ψ, and $\Psi(r)$ indicates that a $r \in R$ satisfies Ψ.

Definition C.10 (Task Environment)**:**
A *task environment* is defined to be a pair $\langle Env, \Psi\rangle$, where Env is an environment, and $\Psi\colon R \to \{0, 1\}$ is a predicate over runs.

A task environment specifies the properties of the network where AgNOS agents execute and also the criteria by which an agent will be judged to have either failed or succeeded in its tasks (*i.e.* the specification Ψ).

Definition C.11 (Satisfiability of Runs)**:**
Given a task environment $\langle Env, \Psi\rangle$, $R_\Psi(Ag_{AgNOS}, Env)$ denotes the set of all runs of the agent Ag_{AgNOS} in the environment Env that satisfy Ψ:

$R_\Psi(Ag_{AgNOS}, Env) =$
$\{r | r \in R(Ag_{AgNOS}, Env)\ and\ \Psi(r)\}$.

Thus, an agent Ag_{AgNOS} succeeds in task environment $\langle Env, \Psi\rangle$ if at least one run of Ag_{AgNOS} in Env satisfies specification Ψ:

$$\exists r \in R(Ag_{AgNOS}, Env)\ such\ that\ \Psi(r). \tag{C.1}$$

For AgNOS control plane, task predicates define the agent behavior during its life cycle. The *ontological commitment* before predicate specification depends on the agent goals in the network. For example, an agent may be in charge of notifying the network administrator in case of general failures.

The AgNOS agent architecture is essentially a map of the internals of the agent (data structures, operations on them and the control flow between these data structures). Agents of AgNOS are *logical* agents, also known as *deliberate* agents [Genesereth and Nilsson, 1987]. In the next sections we present the formal definition of AgNOS logical language to represent its internal state, network policies and properties. Then we give a general intuition of the underlying principles of the inference engine.

Multiagent Interaction

AgNOS agents may have different goals. Interaction is necessary because some tasks towards these goals may depend on dynamic cooperation to be achieved. AgNOS agents engage in this scenario if they have distinct but interrelated expertise. In this case, the language defined in Figure C.7 is used.

AgNOS defines a mechanism for coordination. Following the study of performatives of AgNOS communication language, a coordination protocol is defined based on FIPA [2002]. A Contract Net (CNET) protocol based mechanism for agents's task sharing is defined. In this protocol, agents decompose the problem and allocate the tasks to other agents.

C.3 AgNOS Declarative Level

C.3.1 Logical Language Syntax

AgNOS logical agents have an internal state represented as a knowledge base of logic formulae. Each formula is represented using clausal notation with the standard terminology. Predicates and constant symbols start with lower-case letter, while variables start with capital letter. A predicate represents a relation between terms. A term is either a constant symbol or a variable. With these basic notions we define the elements of the language as follows.

Definition C.12 (Atomic Formula & Literal):
Let `p` *be a symbol representing a relation over the terms* `t_1`*, …* `t_n`*. Then an atomic formula is either*

`p(t_1, ...,t_n)`*, or*

`~ p(t_1, ...,t_n)`

We say that a Literal *is either an atomic formula or its negation. L and ∼ L are called positive and negative literals, and they are complementary to each other.*

In principle, there is no restriction on the number of terms a predicate may have, although it is usually needed just a few to model first-order relations. This is an advantage when compared to FML which restricts this only to *keywords*. AgNOS logic does not impose such a restriction, but we assume policies can be described without functions and recursive terms (like lists). Thus, policies written in AgNOS logic are more expressive then in FML.

Definition C.13 (AgNOS Clausal Rules)**:**
Let `L`$_1$*, ...,* `L`$_n$ *be literals and the disjunction logical connective* `;`*. An* AgNOS Clausal Rule *is defined as the disjunction* `L`$_1$ `;` *... ;* `L`$_1$*.*

AgNOS clausal rules can represent any FML policy, but not vice versa. FML uses a Horn-clause of Logic Programming approach, which is composed of just one postive literal (called the head), and all others are negative literals. Taking the same example described in Section B.1, says that `todd` and `michelle` are superusers, and a superuser has no communication restriction.

```
allow(Us,Hs,As,Ut,Ht,At,Prot,Req) <= superuser(Us).
superuser(todd).
superuser(michelle).
```

To represent in AgNOS Clausal Rule we just need to use the elementary logical equivalence rule which states that $B \Leftarrow A$ is equivalent to $B \vee \neg A$. As in AgNOS notation $\vee$ is ";" and $\neg$ is "∼", then the same knowledge is expressed as

```
allow(Us,Hs,As,Ut,Ht,At,Prot,Req) ; ∼ superuser(Us).
superuser(todd).
superuser(michelle).
```

Definition C.14 (AgNOS Internal State)**:**
Let L be a set of AgNOS clausal rules (or knowledge base), and let $K = 2^L$ be the set of all possible sets of knowledge base. If K is composed of $KB_1, \ldots, KB_n$, then an Internal State *of aAgNOS agent is an element of K.*

Formulas in AgNOS KB express knowledge about the network and other agents:

Events : formulas may represent events in the network controller.

Policy : formulas may represent network policy rules defined in the AgNOS logical language.

Messages : formulas may represent message-based interaction between agents. From the messaging history, it is possible to derive other agents' internal state representation.

C.3.2 Logical Reasoning Engine

AgNOS inference engine is a Clausal-Based Formal System [Vieira, 1987] in which clauses are divided into two categories:

Initial Clauses or knowledge base, say **B**, are those beloging to the set of axioms plus the negation of the query.

Derived Clauses are the ones produced with inference operations.

If S is a sentence or query, in clausal form, and B is the set of initial clause, then a deduction of S from B correponds to derive an empty clause, $\sqcup$, from $\sim \mathtt{S} \cup \mathtt{B}$, or according to Herbrand theorem, to prove that $\sim \mathtt{S}\cup\mathtt{B}$ is *unsatisfiable*. Vieira proposed a new clause to represent derived ones: *structured clauses*. Such clauses are more sophisticated than ordered clauses [C.-L. Chang, 1973] because they are a tree structure rather than an imposed linear ordering over literals. A brief definition of it is given as follows.

Definition C.15 (Structured Clause)**:**
Let L be a literal and $\{\mathtt{L}_1, \ldots, \mathtt{L}_n\}$, with $n \geq 0$, a set of literals. A Structured Clause *(SC), is the ordered pair $\langle L, \{\mathtt{L}_1, \ldots, \mathtt{L}_n\}\rangle$. If we call* C *to this pair, than tree*(C) *is called the* the SC tree *with a diagram*

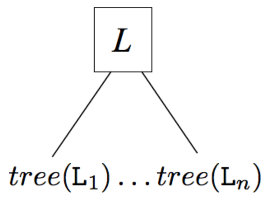

Figure C.3: A tipical SC tree.

If C is a SC, then a *structured sub-clause* (SSC) of C is any SC that occurs in C including C itself. Literal L in Figure C.3 of Definition C.15 is called *root literal*. Root literals of all SSC of C are called *elected* (because they are the ones likely to be chosen for an inference operation), all remain literals are called *candidates*.

To explain the three inference operations we recall, from the brief presentation of FML in Section B.1, that structured clauses can be seen as a “linear form” of Proof Trees [Bruynooghe, 1983]. According to [van Emden, 1984], a proof tree is a data structure which stores the path of a search tree from the root until the current node being processed. AgNOS proof tree is efficiently implemented by means of *shared structures* [R.S. Boyer, 1972].

At every proof step the clause deduced corresponds to the set of candidate literals. This clause belongs to a set of clause instances from **B** which is unsatisfiable. So, AgNOS proof procedure builds a sequence of such clause instances, say $\mathtt{C}_1, \ldots, \mathtt{C}_n$, until the last one, $\mathtt{C}_n$, is equal to $\sqcup$.

The following function is important to help us understand the information within an structured clause.

Definition C.16 (Elected Literal (ELit)):
Let $\mathcal{C}$ be a set of structured clauses, $\mathcal{L}$ a set of literals. The Elected Literal (*Elit*) *function is the mapping $\mathcal{C} \cup \mathcal{L} \rightarrow \mathcal{L}$ defined as, where $X \in \mathcal{C} \cup \mathcal{L}$*

$$Elit(\mathtt{X}) = \begin{cases} \mathtt{X} & L \textit{ when } \mathtt{X} \textit{ is a literal} \\ L & \textit{when } \mathtt{X} \textit{ is a SC and } L \textit{ is its root literal} \end{cases}$$

The interpretation or codification of an SC is a set of clause instances of **B**, the initial clause set or the AgNOS agent knowledge base. Given a SC $\mathtt{C}$, its associated set of instances consists of clauses generated from:

1. the unit clause $\{Elit(\mathtt{C})\}$.
2. for every SSC of $\mathtt{C}$ $\langle L, \{\mathtt{L}_1, \ldots, \mathtt{L}_n\}\rangle$, $n \geq 0$, the clause built with the complementary literal of L plus candidate literals from $\mathtt{C}$, i.e. $\{\sim L\} \cup \{Elit(\mathtt{L}_1), \ldots, Elit(\mathtt{L}_n)\}$

When the second step tries to build a clause with no candidate literals it means the negation of the query, which belongs to $\mathtt{B}$, leads to a contradiction, and so the query is valid. As deduction goes on, derived clauses are build in an ordered way by picking candidates from the knowledge base to apply to resolution. Such candidates are already ordered according to a codification function applied when the base started. The order goes from *ground terms* to more sophisticated ones, with as many terms (variables and constant symbols) as needed.

Inference Operations

The first inference operation is actually used in one situation: to build an SC from the negation of the query, say $\mathtt{X}$. For this, it is used an special "literal", $\top$, only used for this purpose to be the root literal of the first SC.

Definition C.17 (SC Decodify):
Let $\mathtt{C}$ be a clause instance with literals $\mathtt{L}_1, \ldots, \mathtt{L}_n$. Then, decodify $\mathtt{C}$ *creates the SC $\langle \top, \{\mathtt{L}_1, \ldots, \mathtt{L}_n\}\rangle$*

Since this rule is only used one, all other deductions are linear because there will always exist an SC as a premise. The other two inference rules are resolution based. For the purpose of this thesis only *expasion* rule is presented. This rule implements resolution and since network policies do not have the need to use function terms or recursive rules, both rules makes the proof procedure complete according to the Herbrand theorem. But the full engine of AgNOs is covers these other cases too.

Definition C.18 (Expansion)**:**
Let C *an SC in which there is at least one occurrence of candidate literal L. Let* D *some clause with a literal M, complementary to L, such that there is a most general unifier σ between L and M. An* expansion *of* C *via* D *is defined as:*

1. *Let* $\mathtt{C}_1$ *the result of the substitution of L, in* C*, for* $\langle L, \{\mathtt{L}_1, \ldots, \mathtt{L}_n\}\rangle$*, where* $\mathtt{L}_1, \ldots, \mathtt{L}_n$ *are literals of* D*, except M.*
2. *The result of the expansion is* $\mathtt{C}_1\sigma$*, and σ is applied to all elected and candidate literals of* $\mathtt{C}_1$*.*

An agent's decision-making process is then based, at least in part, on this reasoning mechanism. However, since an agent's knowledge base and the engine are in the realm of declarative knowledge, it is necessary to create a way so that actions can recover knowledge or even change it according to Definitions C.9 and C.3.

Definition C.19 (Declarative Knowledge Abstraction)**:**
We call knowledge abstraction *of the declarative level the following operations which allow the storage and recovery of knowledge. In what follow KB is the agent's knowledge base and α a sentence.*

prove(α) *is an action that fires the inference engine and return the result of the deduction of α from KB, i.e. whether $\{\sim \alpha\} \cup KB$ is insatisfiable or not.*

knows(α) *is an action that change the agents knowledge base KB to $KB \cup \alpha$, where α is sentence in clausal form.*

remove(α) *is an action that change the agents knowledge base KB to $KB - \alpha$, where α is sentence in clausal form.*

In what follows we show how to build a KB, how to fire the engine and how all this definitions work together to perform deductions. The example shown makes use only of the first abstraction.

Let the following sentences be agent's KB, that defines `alice` and `toddy` as members of the group of `superusers`, and both are not `attackers`[1], `tcp` and `udp` ports are defined, and then two rules. One defining flow properties and the other defining an `attacker` as a non `valid` user.

```
superuser(alice).
superuser(toddy).
```

[1]There can be restrictions to declare negative axioms, but we just want show the example of the running engine.

$\sim$ `attacker(alice).`

$\sim$ `attacker(toddy).`

`tcp(6).`

`udp(17).`

`allow(Us,Hs,As,Ut,Ht,At,Prot,Req);` $\sim$ `superuser(Us);` $\sim$ `udp(Prot);` $\sim$ `valid(Us).`

`valid(Us) ; attacker(Us).`

$\alpha =$ `allow(alice,Hs,As,Ut,Ht,At,17,Req).`

Now we can ask if `alice` flow is allowed or not by evoking the declarative abstraction `prove`.

`prove(`α`)`

$\sim \alpha =\sim$ `allow(alice,Hs,As,Ut,Ht,At,17,Req).`

A_0 =`decodify(`α`)` $= \langle \top, \{\sim$ `allow(alice,Hs,As,Ut,Ht,At,17,Req).`$\}\rangle$

Select clause instance `C` $= \{$ `allow(Us,Hs,As,Ut,Ht,At,Prot,Req);` $\sim$ `superuser(Us);` $\sim$ `udp(Prot);` $\sim$ `valid(Us)`$\}$

Apply expansion in A_0 via `C` and apply the most general unifier to all other remaining literals.

$A_1 = \langle \top, \langle \sim$ `allow(alice,Hs,As,Ut,Ht,At,17,Req)`, $\{ \sim$ `superuser(alice)`,
$\sim$ `udp(17)`,
$\sim$ `valid(alice)`$\}\rangle\rangle$

Apply expansion in A_1 via `superuser(alice)` and apply the most general unifier, $\emptyset$ in this case, to all other remaining literals.

$A_2 = \langle \top, \langle \sim$ `allow(alice,Hs,As,Ut,Ht,At,17,Req)`, $\{\langle \sim$ `superuser(alice)`$, \{\}\rangle,$
$\sim$ `udp(17)`,
$\sim$ `valid(alice)`$\}\rangle\rangle$

Apply expansion in A_2 via `udp(17)` and apply the most general unifier, again $\emptyset$, to all other remaining literals.

$A_3 = \langle \top, \langle \sim$ `allow(alice,Hs,As,Ut,Ht,At,17,Req)`, $\{ \langle \sim$ `superuser(alice)`$, \{\}\rangle,$
$\langle \sim$ `udp(17)`$, \{\}\rangle,$
$\sim$ `valid(alice)`$\}\rangle\rangle$

Apply expansion in A_3 via `valid(alice)` and apply the most general unifier, again $\emptyset$, to all other remaining literals.

$A_4 = \langle \top, \langle \sim$ `allow(alice,Hs,As,Ut,Ht,At,17,Req)`, $\{ \langle \sim$ `superuser(alice)`$, \{\}\rangle,$

$$\langle \sim \texttt{udp(17)}, \{\}\rangle,$$
$$\langle \sim \texttt{valid(alice)}, \{\}\rangle\}\rangle\rangle$$

Since there is no candidate left $A_4 = \sqcup$, and so `alice`'s flow is allowed.

Another advantage of AgNOS *linear resolution* implementation is to deal with intelligent backtracking Mota [1993] for dealing with clashes among terms in the unification process.

AgNOS inference engine is as efficient as the one used in FML because it is also based on *linear resolution*, which enables linear run-time evaluation of logical formulae. As all Logic Programming languages, FML only allow queries to predicates that are in the head of a sentence.

C.4 AgNOS Procedural Level

This section presents the procedural level of the AgNOS architecture discussed in Section C.2. The procedural level describes the manner of how AgNOS actions/sensors are implemented in a procedural language using the component architecture available in network OS.

C.4.1 AgNOS Agent Properties

An Autonomous Systems in the Internet is controlled by an AgNOS. Figure C.4 depicts an intra-AS view of AgNOS. An AS's *data plane* is composed by nodes, links and switches. Forwarding is based on the OpenFlow switch abstraction [ONF, 2011].

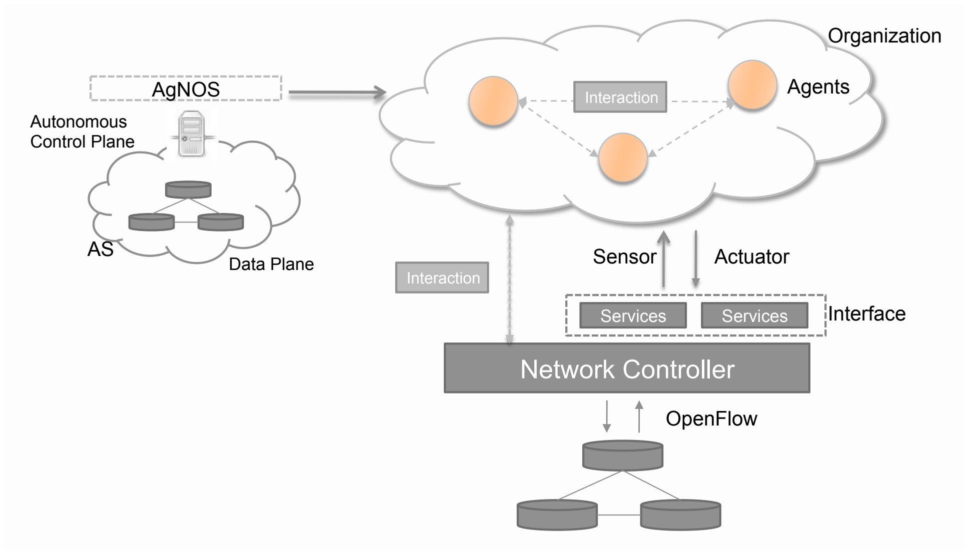

Figure C.4: The vision of AgNOS in an intra-AS setup.

The AS's *control plane* is composed by a network controller and intelligent agents (Ag_{AgNOS})

on top of it. The network operating system run in a network server and is connected to some port in a switch in the network. In this work, we extended and deployed the NOX network controller [Gude et al., 2008], because it is the most comprehensive implementation and adopts the most recent abstractions of SDN.

Intelligent agents are implemented as components. Components execute code on top of the network controller and allow agents to access the services through a set of *interfaces*. Interfaces provide means for agents to access components in charge of controlling network topology, policy definition, link-level access, link/switch status.

Figure C.5 depicts the AgNOS agent internals. The network controller offers to agents' sensors a set of *percepts*. Percepts can come from events or messages. Sensors can obtain, for example, information about flow tables in all network switches (provided by a switch component), information about the evolution of network policies (provided by a policy evaluator) and receive messages from other agents (provided by a messenger component). The network controller also offers implementations of the AgNOS agents *actions*. Actions such as directives to block or rate-limit a network user are provided by the network controller.

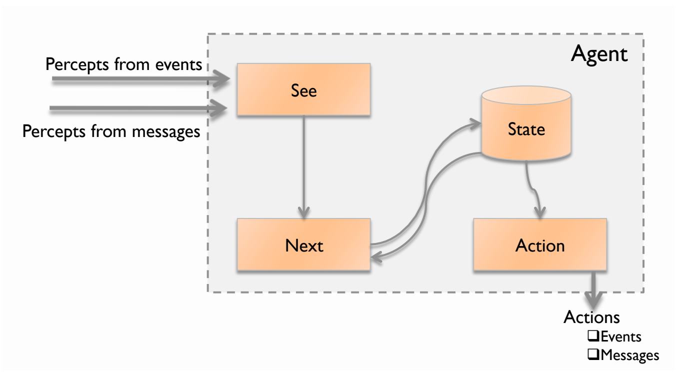

Figure C.5: AgNOS agent architecture.

Agents are implemented in C++ code for efficiency in terms of event processing. The *see* and *action* internal functions are in charge of receiving percepts and dispatching actions. *Next* and *State* functions are in charge of agent's decision-making. As defined in Section C.2.2, the decision on what action to take for each state depends on AgNOS language inference over formulas of the KB.

C.4.2 Multiagent Properties

The AgNOS agents form an organization in charge of control plane management. These agents can interact into a single organization or interact with another AS. In the latter case, interaction messages to/from other agents are handled by the network controller, such as sending, receiving or prioritizing messages (e.g. network controller can give a better path for agent's messages). Figure C.6 depicts an inter-AS view of AgNOS.

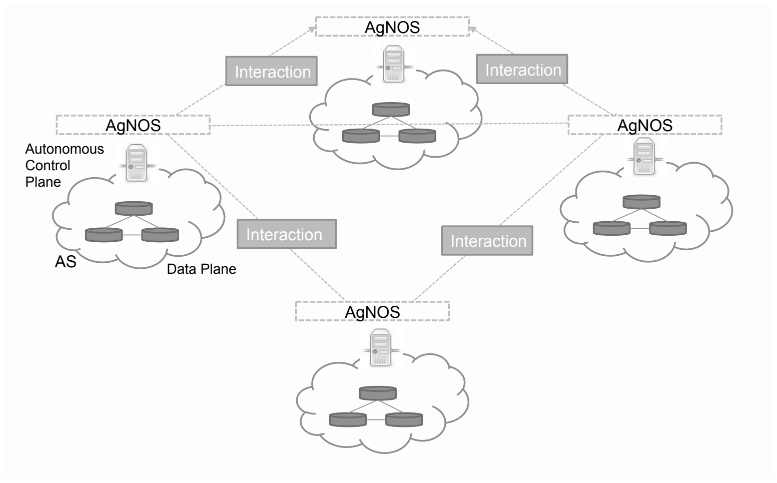

Figure C.6: The vision of AgNOS in an inter-AS setup.

Agents engaged in an organization interact through a communication language and a negotiation protocol. We extended the network controller's messenger component in order to allow the interaction between multiple AS. In this way, the messenger component implements in C++ our AgNOS communication language defined in Section C.4.3. We decided to implement an efficient version of our agent communication languages based on the bit-efficient approach FIPA [2004a]. Instead of string manipulations, the message encoding is made through a more efficient bit representation which states the message fields and contents. Furthermore, messages are created by agents and handled by the messenger component, which encapsulates them and send through a SSL/TLS Dierks and Rescorla [2008] secure channel.

If coordination is necessary, agents deploy a Contract Net protocol with performatives defined in the language denoted in Figure C.7 and implemented by the messenger component.

C.4.3 Communication Language

After distinguishing AgNOS agents internal and external functionalities, the following step is to provide agents with communication skills. AgNOS agents perform *communicative actions*, i.e., they seek to persuade other agents appropriately. They do not force or directly interfere in other agent's internal states. The communication is treated as action following the *speech act theory* [Wooldridge, 2009; Austin, 1975]. In this way, natural languages utterances, also referred to as *speech acts*, could change states e in the environment Env.

The communication in the AgNOS control plane is carried out through a subset of ACL (Agent Communication Language) FIPA [2004b], a speech act based language proposed by FIPA (the Foundation for Intelligent Physical Agents). FIPA's ACL defines a language for messages and

```
ACLCommunicativeAct = Message.
Message = MessageType MessageParameter* EndofMsg.
EndofMsg = EndofCollection.
EndofCollection = 0x01.
MessageType = PredefinedMsgType.
MessageParameter = PredefinedParam.
PredefinedMsgType =         | 0x01  /* accept-proposal */
                                 | 0x02  /* agree */
                                 | 0x03  /* cancel */
                                 | 0x05  /* confirm */
                                 | 0x06  /* disconfirm */
                                 | 0x07  /* failure */
                                 | 0x08  /* inform */
                                 | 0x0b  /* not-understood */
                                 | 0x0d  /* propose */
                                 | 0x11   /* refuse */
                                 | 0x12  /* reject-proposal */
                                 | 0x13  /* request */
PredefinedMsgParam =        0x02 AgentIdentifier   /* sender  */
                                 | 0x03 RecipientExpr  /* receiver */
                                 | 0x04 MsgContent  /* content */
                                 | 0x05 ReplyWithParam  /* reply-with */
                                 | 0x09 Language  /* language */
                                 | 0x0b Ontology  /* ontology */
                                 | 0x0c Protocol   /* protocol */
                                 | 0x0d ConversationID. /* conversation-id */
AgentIdentifier = 0x02 AgentName
                      EndOfCollection.
AgentName = BinWord.
AgentIdentifierCollection  = (AgentIdentifier)* EndOfCollection.
RecipientExpr =  AgentIdentifierCollection.
MsgContent  =  BinString.
ReplyWithParam = BinExpression.
Language = BinExpression.
Ontology =  BinExpression.
Protocol = BinWord.
BinWord  = 0x10 Word 0x00
BinString =  = 0x14 String 0x00
BinExpression = BinExpr
BinExpr = BinWord
Word =   [~ "\0x00" – "\0x20", "(", ")", "#", "0" – "9", "-", "@"]
              [~ "\0x00" – "\0x20", "(", ")"]*.
String = StringLiteral
StringLiteral = "\"" ([ ~ "\"" ] | "\\\"")* "\"".
```

Figure C.7: AgNOS communication language - EBNF syntax definition.

performatives for defining the messages' semantics. The main goal of ACL is to pursue interoperability between different agent architectures. In the case of AgNOS, this is not required since AgNOS agents follow the same architecture. In this way, our subset of ACL provides a more efficient and lightweight version of the language, very suitable to be implemented into the network controller.

The AgNOS communication language is defined through the message syntax which is expressed using standard EBNF format. This definition is depicted in Figure C.7. The main aspects of the language are the definition of *message types* and *message parameters*. Message types denote the performatives related to each sent/received message by agents. These performatives were chosen based on an analysis of the types of messages necessary for proper inter-AgNOS interaction. The two most important performatives are *inform* and *request* and the following 10 performatives can be defined in terms of them. The formal semantics of all performatives in the language were inherited from FIPA's ACL FIPA [2004c].

Message parameters define fields necessary for proper message delivery and conversation controlling, the content of the message (varying according to agent's goals), the ontological commitment and the negotiation protocol which agents are engaged.

Below, we present a very simple example of how agents can communicate using the AgNOS communication language:

```
(inform
  :sender (SecurityAgent 192.168.1.1)
  :receiver (SecurityAgent 10.0.0.1)
  :ConversationID 192.168.1.1-0001
  :content
    "AttackDetected(host,port)"
  :ontology AgNOS
  :language)

(request
  :sender (SecurityAgent 192.168.1.3)
  :receiver (SecurityAgent 10.0.0.1)
  :ConversationID 192.168.1.1-0001
  :content
    "Block(host,port)"
  :ontology AgNOS
  :language)
```

The first message uses an *inform* performative in order to announce a detected attack. The message is targeted to a *SecurityAgent* in the AS of domain 10.0.0.1. The *content* field express what the agent wants to make the other agent to believe. The second message defines a request to blocking traffic in the origin using port and host information.

C.5 Appendix Remarks

This appendix presented the AgNOS framework. First, we presented a vision of AgNOS agents in the Future Internet based on SDN domains. Then, we defined an abstract architecture, formalizing internal and external agent structures.

We then discussed the cooperation mechanisms between AgNOS domains, as well the definition of an specific language for AgNOS agents communication.

Finally, we present the implementation issues involved in AgNOS, mainly its integration with a network controller.

Appendix D

Applying AgNOS in the Future Internet

Being abstract is something profoundly different from being vague...
The purpose of abstraction is not to be vague, but to create a new semantic level
in which one can be absolutely precise.
E. Dijkstra (1930–2002)

Contents

This appendix describes case studies that show how the AgNOS framework can deal with important problems in the Future Internet. The aim of these case studies is to demonstrate that AgNOS provides an efficient control mechanism for SDN when handling problems involving multiple and different domains and cooperative network controllers. We provide two case studies: (1) DDoS attack mitigation, and (2) network traffic management based on inter-domain routing.

Agents are in the core of the AgNOS framework. They access the abstraction provided by network controllers and manage the network entities connected to them. The first step in creating such type of agents is the design of a core feature. The core feature is the method used to manage a functionality in the network. In our case studies, the agents' core features are: (1) the mitigation mechanism used to block attack traffic and (2) the routing stack used to control different types of traffic.

Thereafter, we select the functions and data provided by network controllers that will be used to attach the core feature to the network. Function can provide access to switches in order to control their behavior. Data can be long-term collected information from users and switches, such as location and traffic features. The agents use this function and data, internally modeled following the framework described in Appendix C, and control the network, eventually cooperating with external AgNOS agents.

Finally, the agent must be configured in order to be instantiated at the same time as the network controller and to work cooperatively with other components available in the environment. In the next sections, we extensively discuss the creation of the case studies following the previous design phases.

This appendix is organized as follows: Section D.1 describes how AgNOS agents can deal with DDoS attacks. Section D.2 demonstrates AgNOS agents autonomously handling network traffic management. Section D.3 discuss the experiment design and the methodology used to validate the proof-of-concept implementations. Section D.4 presents the results of the experiments.

D.1 Distributed Denial-of-Service Attacks

Distributed denial-of-service attacks (DDoS) is one of the most important threats against Internet Service Providers [Arbor Networks, 2009]. The attacks grow in a scale of quantity and sophistication of the techniques used. The main characteristic of these attacks is that they occur in a distributed manner, *i.e.*, infected hosts located in different Internet domains are coordinated to simultaneously send undesirable traffic to a target. The objective of these attacks is to overload the target's resources to prevent users to access its services.

Many DDoS resistant architectures have been proposed in the literature [Liu and Yang, 2010; Argyraki and Cheriton, 2009; Liu et al., 2008; Yang et al., 2008; Anderson et al., 2004; Soldo et al., 2012]. These approaches allow DDoS victims to drop or limit undesirable traffic by means of capacity announcements or filters which automatically block attack traffic.

The distributed nature of DDoS attacks complicates the large-scale deployment of these architectures, since each AS should update end-systems or routers in order to offer such protective services. Many of these mechanisms also involve complex management actions, as well as inter-AS negotiation towards the attack mitigation.

Differently from the previous architectures, Peng et al. [2007a] and Xu et al. [2007] propose the use of multiagent systems in the detection and mitigation of DDoS attacks in the Internet. Their main limitation is that they need to deploy agents on each router or switch in the network, what reduces the chance to apply them in large-scale scenarios. Furthermore, they lack cooperation mechanisms for inter-AS negotiation of filtering or capacity announcement mechanisms.

In previous work [Braga et al., 2010], we also showed how a network controller is susceptible to failures caused by DDoS attacks. Due to its centralized characteristics, all attack traffic is directed to an AS and overloads the network controller which controls it with a great amount of connections request. If the receiving buffer of the network controller is compromised, the whole system (including agents and management applications) can crash. We addressed this problem using a self-recovery mechanism for network controllers [Fonseca et al., 2012], but if the DDoS attack is not mitigated, the replication of the controller state is not enough to keep the network running.

D.1.1 DDoS Scenario

Figure D.1 depicts a DDoS scenario with SDN domains. Since SDN does not change the datapath between autonomous systems, the attack follows the same steps as in the current Internet design.

In a DDoS scenario, an attacker on host H_i has access to multiple hosts H_s belonging to different ASes. Those hosts are connected by OpenFlow switches/routers. The attack traffic is generated from those sources. The lower part of Figure D.1 shows the aggregated attack traffic sent by several H_s reaching the edge router of H_d. This traffic overloads the network or the router handling it. *AgNOS* denotes the autonomous network controller, *Agent* the agent in charge of detecting and mitigating attacks. Finally, denotation $M(s, d)$ means the exchange of messages from source s to destination d used for cooperation between two instances of AgNOS.

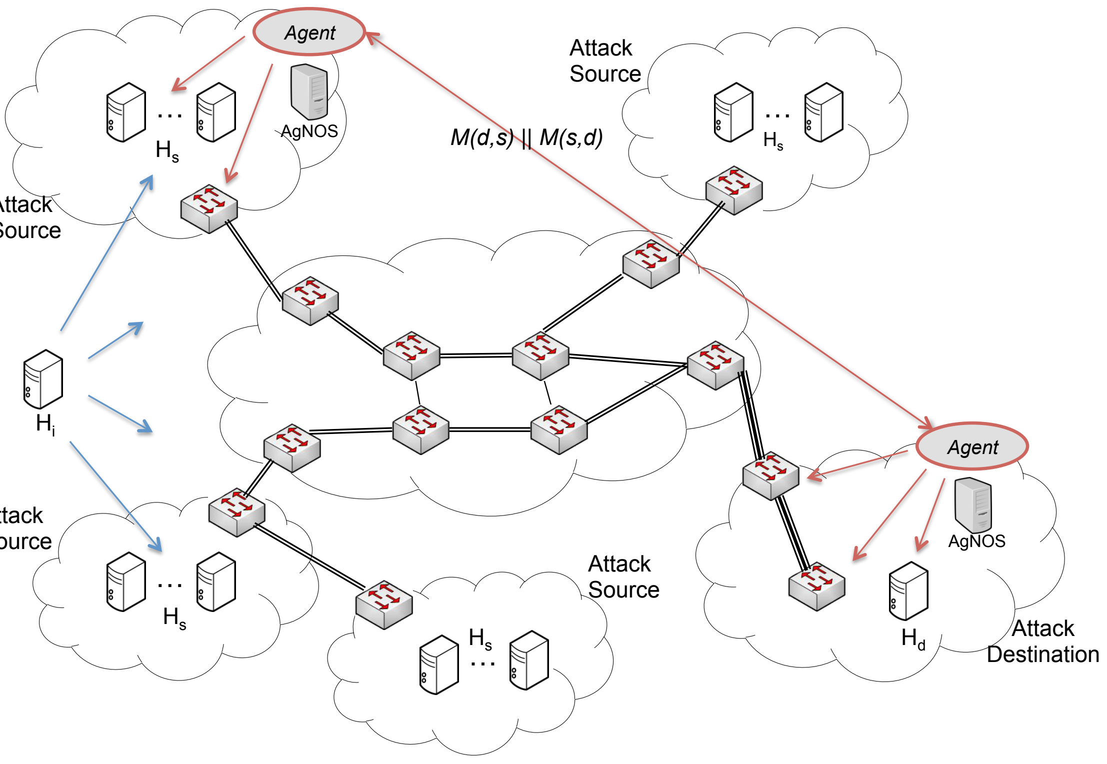

Figure D.1: DDoS scenario with SDN domains.

Any H_s in every domain must have an attachment point to the datapath. This attachment point must be an OpenFlow switch port. For any connection an H_s initiate, the network controller receives the first packet and defines the actions that must be carried out. At this point, the network controller updates the network view with the features of this connection. If the authenticator application decides to grant access to host H_s, the controller updates the flow-table in each OpenFlow switch in the network.

It is possible to observe in Figure D.1, that one or more hosts initiate a connection with H_d. Actually, current DDoS attacks in the Internet can have more than millions of malicious nodes, also known as *bots*, with peak loads around 100 Gbps [Abliz, 2011].

D.1.2 DDoS Mitigation with AgNOS

For an AgNOS agent to be able to mitigate DDoS attacks, it needs to block or, at least, rate-limit multiple flows. The core feature (blocking attack traffic) provides the way AgNOS agents are able to perform this task in an SDN network. AgNOS agents perform packet blocking by means of the relationship between its actions and a switch application. Every network controller in this SDN scenario controls the packet forwarding between hosts by means of a switch application.

This switch is implemented in software and is able to control the delivery of packets modifying the flow-tables in OpenFlow switches. It can be the case that an authentication application is deployed in the network controller. Then, it is also possible to track the location of users in the network.

The data generated by these applications, and also by the network controller is kept in the network view. This network view is updated for every new event in the datapath that is related to switching and authentication.

Sensing the Environment

As described in Section C.1.2, the percepts that arrive from the environment to the AgNOS agents can be of two types: events and messages. Table D.1 summarizes the type of events an AgNOS agent may be interested in order to generate sentences to its knowledge base in the DDoS scenario. Message events that can be received by AgNOS agents are summarized in Table D.2.

Table D.1: Events in the DDoS scenario.

Events of AgNOS agent interest	**Description**	**Source**
Host_event	A change in the host state: arrival or departure.	Authenticator
Packet_in_event	New packet received by the controller.	Controller
Flow_mod_event	Addition or modification of a flow.	Controller
Flow_removed_event	Flow expiration or removal.	Controller
Attack_detected	Raised when a DDoS attack is detected in the network.	Attack detector
Attack_target	Raised when a DDoS target is identifiable in the network.	Attack detector
Attack_source	Raised when a DDoS source is identifiable in the network.	Attack detector
Attack_mitigated	Raised when a DDoS attack is mitigated in the network.	Attack detector

Two important things should be noted from Table D.1. There are events generated directly by the network controller and events generated by auxiliary management applications, such as the Authenticator. AgNOS agents do not directly handle the events, but updates their knowledge bases through the event's effect in the network view.

Events directly handled by AgNOS agents are the ones generated by an attack detector. An attack detector is a component in charge of analyzing the network traffic and deciding if the network is under a DDoS attack. We designed this network component in a related work described by Braga et al. [2010]. The attack detector uses a self-organizing map to detect patterns of attacks based on traffic features. The component does not analyze per packet information, but uses information about flows available in the OpenFlow switches which is accessed by the network controller.

Table D.2: Message events in the DDoS scenario.

Communicative Act	**Summary**	**Description**	**Main Message Contents**
Agree	The AgNOS agent agrees to perform an action of blocking the attacking host.	Agreement to a previously submitted request to perform an action of blocking attacking host.	agent-identifier, content, language, in-reply-to
Failure	The AgNOS agent tells another agent that it tried to block an attacking host but it failed.	Informing that an action of blocking attacking host was considered feasible by the AgNOS agent, but was not completed for some given reason.	agent-identifier, content, language
Inform	The AgNOS agent informs another agent that it is under attack.	The agent believes it is under attack and expects the other AgNOS agent come to believe it.	agent-identifier, content, language
Refuse	The AgNOS agent refuses to perform an action of blocking an attacking host. It may explain the reason.	Does not accept to perform an action of blocking a host.	agent-identifier, content, language
Request	The AgNOS agent requests that another agent blocks an attacking host.	The content of the message is a description of the action to be performed.	agent-identifier, content, language

Message are also directly handled by the AgNOS agents. It is important to observe that messages in Table D.2 can be received by agents at any time if the receiver is an attacker or being attacked. For example, the *request* and *inform* messages are received only by AgNOS agents controlling networks that originate a DDoS attack. Furthermore, the *refuse*, *agree*, *failure*, and *inform* (in case of an accomplished task) messages are received only by AgNOS agents at an attacked network.

AgNOS Agent Decision

The AgNOS agent lifecycle determines that for each received percept from the network, it should update its knowledge base with this new fact. Every AgNOS agent have an initial state which is determined by the initial knowledge in its KB. In the AgNOS architecture the initial state is based on the policies of the network domain controlled by the AgNOS agent.

The following fragment from a KB can be considered a network policy in a SDN domain:

```
allow(Us,Hs,As,Ut,Ht,At,Prot,Req); ~authenticated(Us); ~udp(Prot).
udp(1).
tcp(2).#
```

In the fragment above, the only flows allowed in the network are those where users (Us) are authenticated and the transport layer protocol ($Prot$) is the UDP (User Datagram Protocol). Then, if the AgNOS agent receives an event from the network (*i.e.* from the authenticator) which incur in the following statements:

```
authenticated(user1).
authenticated(user2).
authenticated(user3).
```

only $user1$, $user2$, $user3$ with UDP will be allowed to forward packets to the network.

As soon the AgNOS agent starts its lifecycle and events happen in the network, the KB is update with new facts. In the DDoS scenario the main facts that can occur from the point of view of an attacked domain are the ones generated by the attack detector. These facts may be, for example:

```
attackdetected(user1).
attacksource(200.129.156.1).
(...)
attacksource(10.0.0.1).
```

where a DDoS attack towards $user1$ is detected and the locations of the attack sources are written in the KB.

Once the AgNOS agent proves that a DDoS attack is happening in the network, it makes a decision to mitigate the attack. In order to make a decision, AgNOS agents use the meta-interpreter described in the Section C.2.

Acting on the Environment

The AgNOS framework states that the agents act on the environment through two ways: generating events that are handled by management applications or the network controller, and sending messages to other AgNOS agents.

From the point of view of an attacked network, the best action to take is to request the origin of the attack to block a flow or rate-limit it. In this way, AgNOS agents generate message events described in Table D.2. The main event is the *request* message that has in its *content* parameter the location of the attack origin. If needed (e.g. to improve the mitigation mechanism), the AgNOS agent may use an *inform* message to advertise to their peers that it is under a DDoS attack.

Both *inform* and *request* messages arrive at the destination in the form of events. These events are used to update the KB of the AgNOS agent at the attacker domain. The autonomy mechanism of this agent allows it to autonomously decide to cooperate. This decision often depends on the knowledge in the KB. For example, if the network policy allows it to cooperate with the peer agent or even if the request can be executed by the AgNOS agent in the source domain, such as described in the following KB fragment:

```
cooperate(Us);~peer(Us);
peer(agnos_agent1).
(...)
peer(agnos_agent2).#
```

If the agents decides to act and block the flow at the origin, it must generate *flow_mod_events* or directly manipulate the network view in order to block the flow in the datapath. Note that the agent does not access the datapath directly, but uses the network controller to enforce the cooperation.

When the block action is performed, the agent can communicate this new fact generating an *inform* message with the contents referencing the prior request. As soon as this message arrives at the attacked domain, the AgNOS agent updates its KB.

Eventually, the attack is mitigated or its rate is limited to an accepted value. This is determined by the attack detector, which then can generate an *attack_mitigated* event that is used to update the AgNOS agent's KB.

D.2 Network Traffic Management

Despite the use of SDN, it is anticipated that the Internet will continue to be organized around domains [Koponen et al., 2011]. This assumption requires the implementation of inter-domain routing for packet delivery between ASes.

The Border Gateway Protocol (BGP) [Rekhter et al., 2006] is considered the interdomain routing protocol of the Internet. In BGP, reachability information is exchanged between ASes which are autonomous to decide how to express routing policies considering, for example, rankings of routes or, filtering undesirable paths.

Nonetheless, many issues with BGP remain largely unsolved [Yannuzzi et al., 2005]. Convergence to stable routing [Suchara et al., 2011], security [Butler et al., 2006], and complexity are representative examples of such issues. Regarding complexity, BGP is a very complex protocol and its totally distributed nature elevates this aspect to an extreme, causing it to negatively impact the performance of the protocol in the Internet. An example of this case occurs when a configuration fault completely disrupt the protocol's operation [Feamster and Balakrishnan, 2005].

AgNOS poses a new opportunity to build autonomous interdomain routing, based mainly on BGP, distinct from the *status quo* of the Internet routing. In fact, some researchers favor to redesign routing, such as Koponen et al. [2011], but our work tries to capture the basic functionality of BGP and apply it directly to AgNOS. In this way, its is possible to autonomously control routing using AgNOS agents.

This proof-of-concept implementation involves inter-domain routing and the management of network traffic by means of routing decisions. It stands on top of our previous work [Bennesby et al., 2012] in routing for SDN. This work builds a component system for SDN controllers called *Interas*. This system is in charge of providing routing features between network domains based on the BGP protocol. BGP is implemented using SDN concepts such as events and flows and, runs as a management application on top of a network controller.

D.2.1 Traffic Management Scenario

A scenario involving inter-domain routing and traffic management is depicted in Figure D.2. It has six autonomous systems interconnected: $AS1, AS2, AS3, AS4, AS5, AS6$. There is a host H_s at $AS1$ sending traffic to a host H_d at $AS6$. The first stablished path between $AS1$ and $AS6$ passes only through $AS2$. The *Interas* component uses the BGP to update the routing tables managed by each network controller in each autonomous system.

Eventually, there is a bottleneck link between $AS2$ and $AS6$, what causes the transmission rate between H_s and H_d to decrease to an unacceptable value. In this case the AgNOS agent functionality is to perform a new routing configuration to enforce this flow to follow the path $AS1$–$AS2$–$AS4$–$AS5$–$AS6$. This task is accomplished by means of cooperation between multiple AgNOS agents and their internal relationship with the *Interas* component. Denotation $M(s, d)$ means the exchange of messages from source s to destination d used for cooperation between AgNOS.

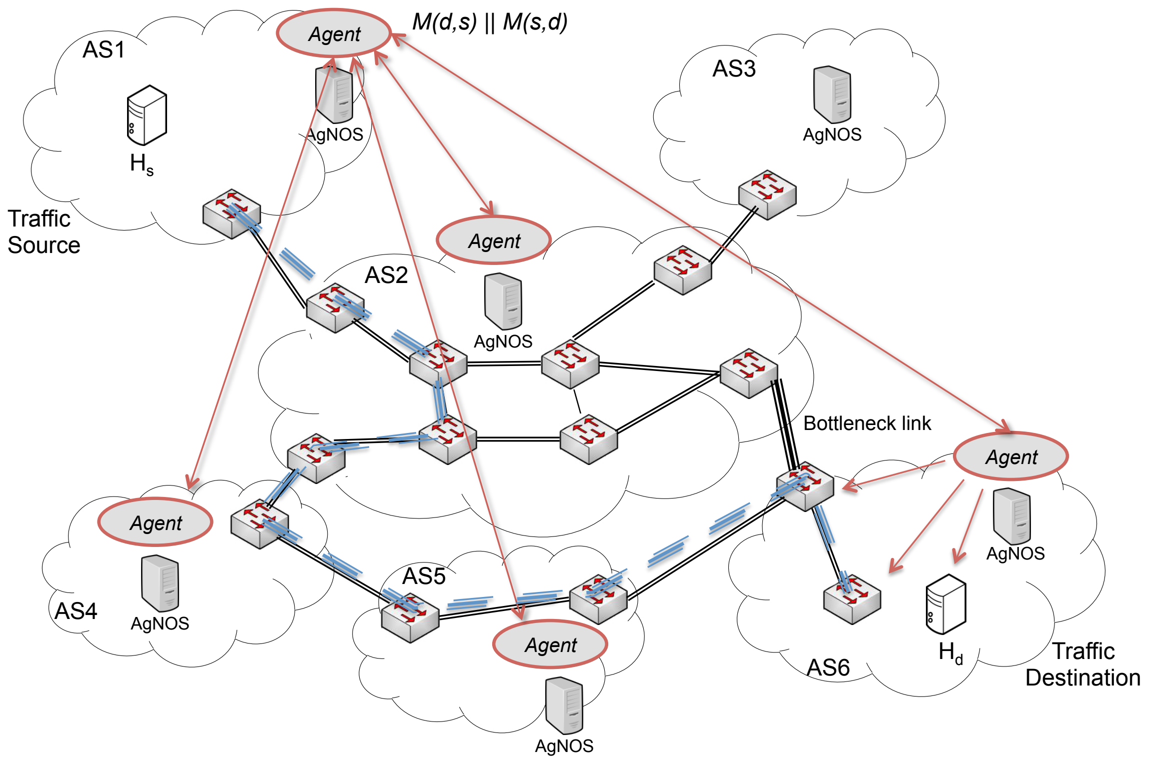

Figure D.2: Traffic management scenario with SDN domains.

The same manner as in the DDoS scenario, any H_s in every domain must have an attachment point to the datapath. This attachment point must be an OpenFlow switch port. For any connection an H_s initiate, the network controller receives the first packet and defines the actions that must be carried out. At this point, the network controller updates the network view with the features of this connection. Prior to forwarding the packets of this flow to an outside datapath, the *Interas* component must create the inter-domain routing table.

D.2.2 Management of Network Traffic with AgNOS

The AgNOS agent's core feature for the management of network traffic is the ability to control the path between a source host and a destination host. For traditional scenarios of enterprise and campus networks, this task is straightforward with SDN. The network controller can enforce the path of switches where packets will be forwarded. In an inter-AS scenario this task is complex and depends on mechanisms for large-scale routing. This core feature is offered to AgNOS agents by means of our BGP-based routing component called *Interas*. Since *Interas* is available to the agents, they can receive constant updates about the external connections with multiple ASes.

The data generated by this component, and also events generated by the network controller are used to update the network view. AgNOS agents use this information to keep track of the network status.

Sensing the Environment

Most of the events of interest of an AgNOS agent in the traffic management scenario are handled by the *Interas* component. Figure D.3 depicts this routing architecture and further details can be consulted elsewhere [Bennesby et al., 2012].

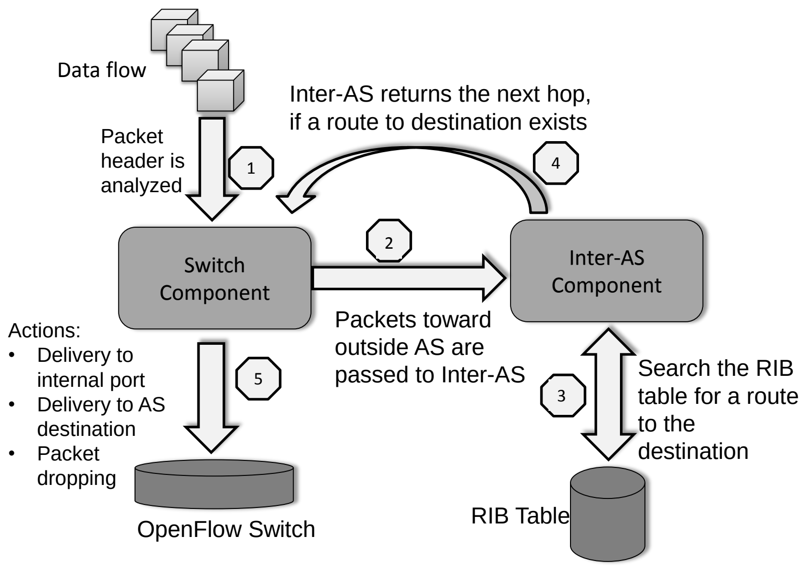
Data flow
Packet header is analyzed
1
Inter-AS returns the next hop, if a route to destination exists
4
2
Switch Component
Inter-AS Component
Packets toward outside AS are passed to Inter-AS
Actions:
• Delivery to internal port
• Delivery to AS destination
• Packet dropping
5
3
Search the RIB table for a route to the destination
OpenFlow Switch
RIB Table

Figure D.3: Routing architecture for packets arriving in the network controller [Bennesby et al., 2012].

Table D.3 presents the most important events for AgNOS in the traffic management scenario. The Next_hop is generated by the *Interas*. This event is used to keep the KB updated with the AS relationship with other domains. The Host_status event is a special type of event not yet described in this thesis. The origin of this event is the host that suffers the network constraint. It is the host sending information to the AgNOS about its current state. Such type of communication is based on the work by Handigol et al. [2008] where a host manager sends information to a load balancing system in the network controller.

Table D.3: Events in the network traffic management scenario.

Events of AgNOS agent interest	**Description**	**Source**
Next_hop	Output port indication to the switch component if a route exists	Interas
Host_status	Indicates the condition of a host connection bandwidth	Host

D.3 Experimental Design and Methodology

This section describes the experiment design and methodology used to demonstrate the validity of the proof-of-concept implementations presented in Sections D.1 and D.2.

During the experiments we followed two principles pointed out by Carvalho [2011]: reproducibility and experiment planning. The former refers to "the ability of an experiment to be accurately reproduced by someone else working independently". We tried to keep the experimental environment in a controlled and scalable way. The latter aims to build experiments in manner that it helps to "confirm or reject the hypothesis made about the artifact under evaluation". The experiment planning can help to isolate errors and to estimate the contribution of controlled parameters over the outcome.

D.3.1 Basic Terminology

In order to assist the reader, we define the terms related to the experimental design [Carvalho, 2011; Dodge, 2008; Jain, 1991]:

Response variable. It is the output value that is measured as the input values are changed. Examples of response variable are the total time to send a file during a DDoS attack or the total execution time for an agent to accomplish a task.

Factors. They are the input variables of an experiment that can be controlled or changed by the experimenter. The factor of an experiment can include, for example, the number of hosts participating in a DDoS attack or network delay, and loss.

Levels. The levels of a factor are the specific values to which it may be set. For example, the number of attack hosts in a DDoS scenario can be set to 25.000 to 200.000.

Treatment. It is a particular combination of levels of various factors.

Replication. Replicating an experiment means rerunning it completely, with all the same input levels. Since the measurements of the response variable are subject to random variations, replications of an experiment are used to determine the effect of measurement error on the response variable.

Interaction. An interaction among factors occurs when the effect of one factor depends on the level of another factor.

The *measurement* technique [Jain, 1991] used in this work to validate the proof-of-concept implementations of AgNOS was selected because we wanted to compare its behavior to other architectures used for the mitigation of DDoS attacks and the management of network traffic.

D.3.2 Experimental Setup based on SDN

A complete experimental network can be deployed using virtualization. Virtualization can provide the benefit of flexibility and fast prototyping and testing of network protocols. Early versions of network controllers and management applications were tested using software implementations of OpenFlow switches deployed on top of virtualized machines. One could increase the topology complexity adding more host machines to the testbed. Our early work on DDoS attack detection testbed used this approach [Braga et al., 2010].

We used in this thesis a more recent approach called Mininet [Lantz et al., 2010a]. Mininet is a new architecture for network emulation using OS-level virtualization features. It allows the prototyping of SDN topologies with links, hosts, switches and controllers. This architecture proved to be very flexible, scalable and realistic when prototyping and evaluating our approaches.

Figure D.4 highlights the major elements of a testing scenario using Mininet. Inter-AS communication can be deployed extending the typical usage of Mininet by adding an interface to the Internet, which allows the composition of multiple single-machine ASes. Figure D.4 depicts four network entities: 1. controller or network OS (c); 2. OpenFlow switches (s); 3. linux-based hosts (h); and 4. Ethernet links.

In this scenario, AgNOS agents can be instantiated in a set of controllers $\{c_1, \ldots, c_n\}$, performing actions for every packet from $h1$ toward an external IP address or received from the interface to the Internet. If a packet received is directed to a local IP address, the packet is forwarded to the switch directly connected to the destination host. If not, the packet is forwarded to the port directly connected to the next hop, i.e. another AS. Network traffic generated in the network or service status modifications are monitored by the network controller. Any packet event from another ASes is also treated by this network controller.

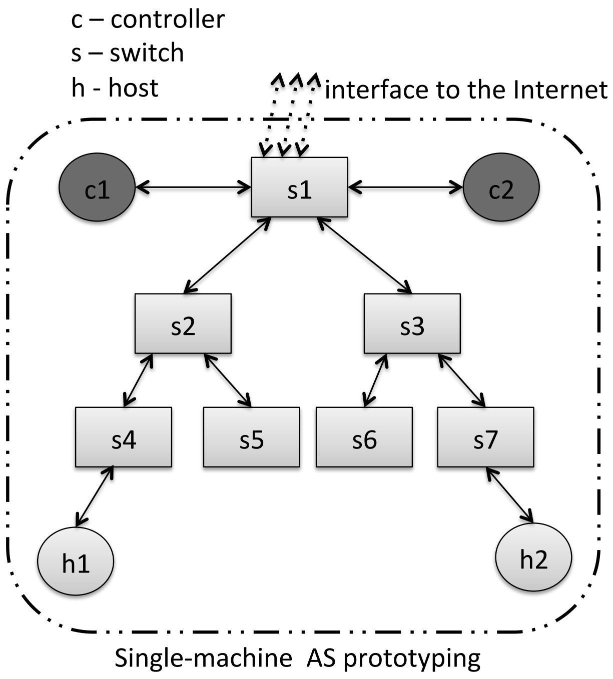

Figure D.4: Mininet: virtualization-based experiments.

D.3.3 Topology

In this thesis, we built an SDN topology based on important related works in DDoS [Liu and Yang, 2010; Yang et al., 2008; Argyraki and Cheriton, 2009]. The reason for choosing such a similar topology was to test our approaches in scenarios similar to those in the literature and to perform reliable comparisons between SDN domains and non-SDN domains. Even the scenario used for the network traffic management using AgNOS could be an extension of the DDoS setup.

Figure D.5 depicts the topology used in the evaluation of AgNOS in a DDoS scenario. In order to build this scenario we followed the rationale presented by Liu and Yang [2010] based on Yang et al. [2008]. The authors intended to simulate attacks in which thousands to millions of attackers flood a well provisioned link. However, they were not able to scale their simulations to support beyond several thousand nodes. We were also unable to support it with network emulation using mininet due to resource constraints. Then, we decided to follow the same approach: "to fix the number of nodes, but scale down the bottleneck link capacity proportionally to simulate the case where the bottleneck link capacity is fixed, but the number of attackers increases" [Yang et al., 2008].

The topology used is a dumb-bell organization where 10 source ASes connect to a destination AS through a transit AS. Each source AS has 100 source hosts connected to a single access router. The transit AS has two routers and the destination AS has one victim destination host. We built the scenario by using the mininet architecture discussed in Section D.3.2. Then

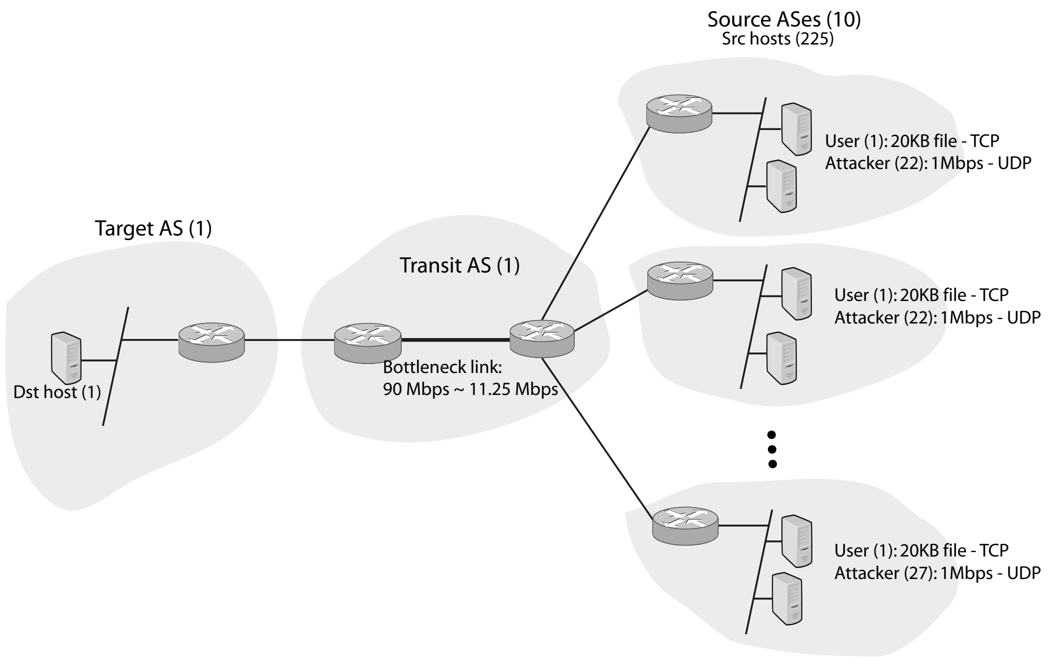
Source ASes (10)
Src hosts (225)
User (1): 20KB file - TCP
Attacker (22): 1Mbps - UDP
Target AS (1)
Transit AS (1)
User (1): 20KB file - TCP
Attacker (22): 1Mbps - UDP
Dst host (1)
Bottleneck link:
90 Mbps ~ 11.25 Mbps
User (1): 20KB file - TCP
Attacker (27): 1Mbps - UDP

Figure D.5: DDoS scenario topology.

every AS is a computer executing the mininet, where each router is emulated by OpenFlow virtual switches. Each host is a mininet-based host (*i.e.* a linux machine) running on top of the architecture. For every AS there is a network controller running the AgNOS system.

The link in the transit AS is the bottleneck link, and all other links have sufficient capacity to avoid congestion. We vary the bottleneck link capacity from 90Mbps to 11.25Mbps to emulate the scenario where 25K $\sim$ 200K senders (both legitimate and malicious) share a 10Gbps link approximately. Each sender's fair share bandwidth varies from 400Kbps $\sim$ 50Kbps. The propagation delay of each link is 10ms. To stress-test the scenario, each source AS have only one legitimate user that repeatedly sends a 20KB file to the victim using TCP. We let each attacker send 1Mbps constant-rate UDP traffic to the victim.

Figure D.6 depicts the topology used in the evaluation of AgNOS in a network traffic management scenario. This scenario topology is slightly different with a total of fourteen ASes. We divided the bottleneck link between two ASes, AS2 and AS3. There are a destination AS, AS1, and a source AS, AS4. Other ASes generate traffic enough to saturate the bottleneck link. The source AS has two paths that can connect to the destination AS. A source host (Src host) can generate TCP traffic addressed to the a destination host (Dst host).

Every AS is implemented as a mininet machine with virtual hosts and virtual OpenFlow-based switches performing the packet forwarding. Every AS has a network controller with the AgNOS system. In this scenario, the routing between ASes is managed by the *Interas* component running on top of the network controller.

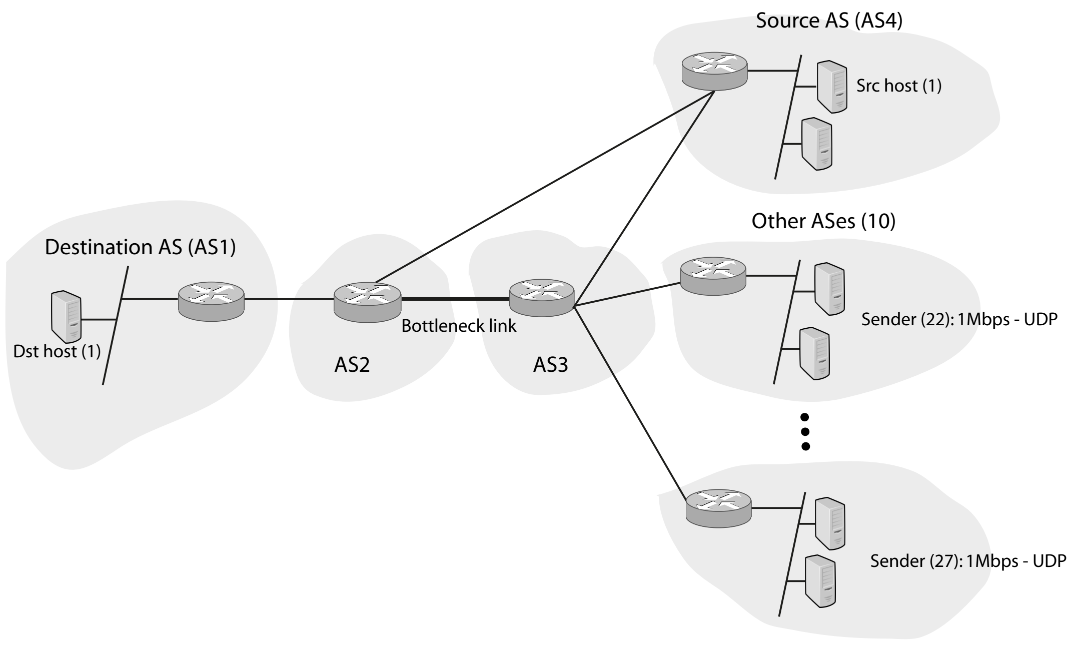

Figure D.6: Network traffic management scenario topology.

D.3.4 Experiments Reliability and Pilot Study

There are two manners to determine the number of experiment replications larger enough to provide result reliability. One way is to follow Dodge's statement about the number of replications to achieve the correct reliability of the results [Dodge, 2008]: a sample size n larger than 30.

We decided to follow a second approach using pilot studies [Chadha, 2006] for every experiment carried out to evaluate the proof-of-concept implementations of AgNOS. In order to determine the number of samples used in our experiments, we performed pilot tests for each scenario. For a *Pilot Sample Size (n) = 10*, with a *Current Estimate = 132.5*, *Current Variance Estimate = 1.67*, an *Acceptable Significant Level (a) = 0.05* and an *Acceptable Absolute Error = 0.5*, then calculating the sample size with acceptable absolute precision, based on the work by Chadha [2006], we entailed 26 samples. For a *Pilot Sample Size (n) = 10*, with a *Current Estimate = 219.14*, *Current Variance Estimate = 1.89*, an *Acceptable Significant Level (a) = 0.05* and an *Acceptable Absolute Error = 0.5*, then calculating the sample size with acceptable absolute precision, we entailed 30 samples. Since we already replicated the experiment 33 times for each treatment under evaluation, we believe this is a sufficient number to obtain the measurements needed for statistically checking the AgNOS performance using both techniques.

D.3.5 Performance Evaluation

DDoS Mitigation

There are many works proposing DDoS mitigation architectures in the Internet [Abliz, 2011; Liu and Yang, 2010; Yang et al., 2008; Liu et al., 2008; Yang et al., 2005]. We aim to compare the performance of AgNOS against the most prominent filter-based DDoS architectures available in the literature. The objective is twofold: to demonstrate that AgNOS agents can act in the network and to show how its performance compares to those of other architectures. We choose the StopIt [Liu et al., 2008] and Fair Queuing (FQ) [Liu and Yang, 2010] systems for that.

The StopIt architecture is a filter-based DDoS defense system. In this way, the victim can install network filters to stop the attack traffic. The Fair Queuing throttle attack traffic to consume no more than its fair share of the bandwidth.

We performed two kinds of experiments in the DDoS scenario:

Unwanted Traffic Flooding Attacks. Attackers directly flood a victim, but the victim can classify the attack traffic, and uses the provide DDoS defense mechanism.

Colluding Attacks. In this type of attack, malicious sender-receiver pairs collude to flood the network.

For the first kind of experiment, the response variable is the average time it takes to complete a successful file transfer. The factor is the number of DDoS attackers. As described earlier the levels of this factor can be: 25K, 50K, 100K, and 200K emulated hosts.

For the second kind of experiment, the response variable is the throughput ratio, *i.e.* the ratio between the average throughput of a legitimate user and that of an attacker. The factor is the number of DDoS attackers with levels of 25K, 50K, 100K and 200K emulated hosts.

Network Traffic Management

For the network traffic management, a host at $AS4$ establishes a continuous TCP connection to a destination host at $AS1$. There is a bottleneck link between $AS2$ and $AS3$. Other 10 ASes also connect to $AS1$ and send traffic through the bottleneck link, as well $AS4$.

The experiment consists in reducing the bottleneck throughput and emulating a network congestion. For $AS4$, it is possible to route its flows directly to $AS2$, but this depends on changes in the routing table in specific OpenFlow routers in that scenario. For comparison we performed the tests with only the Interas component and AgNOS with autonomous routing capabilities.

The response variable was the throughput of the TCP connection between *src host* and *dst* *host*. The factor is the throughput at the bottleneck link (emulated). The level of this factor varies from 90 Mbps to 11.25 Mbps.

D.4 Analysis of the Results

This section presents the performance of AgNOS in the DDoS mitigation and network management scenarios.

D.4.1 DDoS Mitigation

In this section, we demonstrate the functional behavior of the AgNOS agents in the presence of DDoS attacks. The goal of the agent is to mitigate the attack traffic and keep the service in the attack destination running. Figure D.7 presents the average transfer time of a 20KB file from a legitimate host and destination host under attack. We also plot the results for the StopIt architecture and the Fair Queuing mechanism. The topology and experiment methodology are the same for all architectures, as described in Appendix D. The result is related to the unwanted traffic flooding attacks.

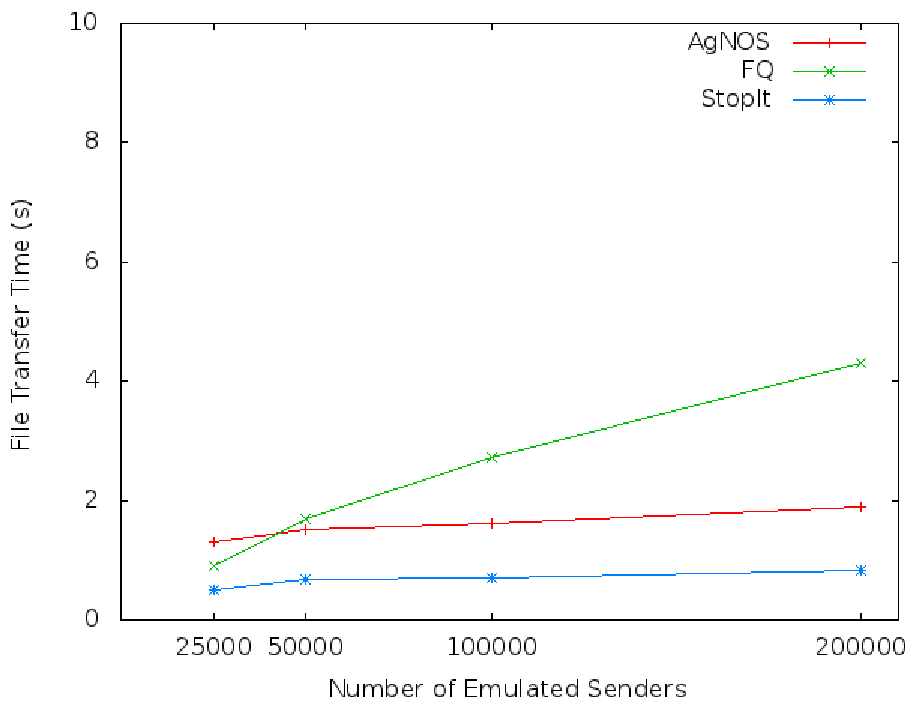

Figure D.7: Average transfer time during DDoS attacks.

Figure D.7 shows that the worst architecture is the FQ, as already demonstrated by Liu and Yang [2010], because it increases linearly with the number of emulated senders when the the packets need to compete with attack traffic for the bottleneck bandwidth. StopIt has the best performance.

The AgNOS architecture provides an acceptable average file transfer time. We consider as an acceptable average file transfer time if this value is lower than a FQ value for the same configuration. Despite its file transfer rate is inferior to that of StopIt, the transfer time slightly increases with the levels of attack hosts factor number. The graph also demonstrate that AgNOS agents work toward the goal of keeping the service in the destination host running.

There is an important difference between AgNOS and StopIt in Figure D.7. This difference is because StopIt deploys a robust traffic policing control loop that guarantees each sender its fair share of bandwidth. AgNOS does not provide yet control for fair share bandwidth (a better optimization approach) but tries to block the flow as soon as possible in the attack source. We do not consider this as an AgNOS limitation, since SDNs can easily provide rate-limiting of flows, but implementing such task is out of the scope of this thesis.

Figure D.8 depicts the throughput ratio between attack and normal traffic in colluding scenarios. The aim of this number is to detected if a legitimate user receives the same average bottleneck throughput as the malicious node. As pointed out by Liu and Yang [2010], FQ and StopIt perform similarly because they use per-sender fair queuing to protect a legitimate user's traffic.

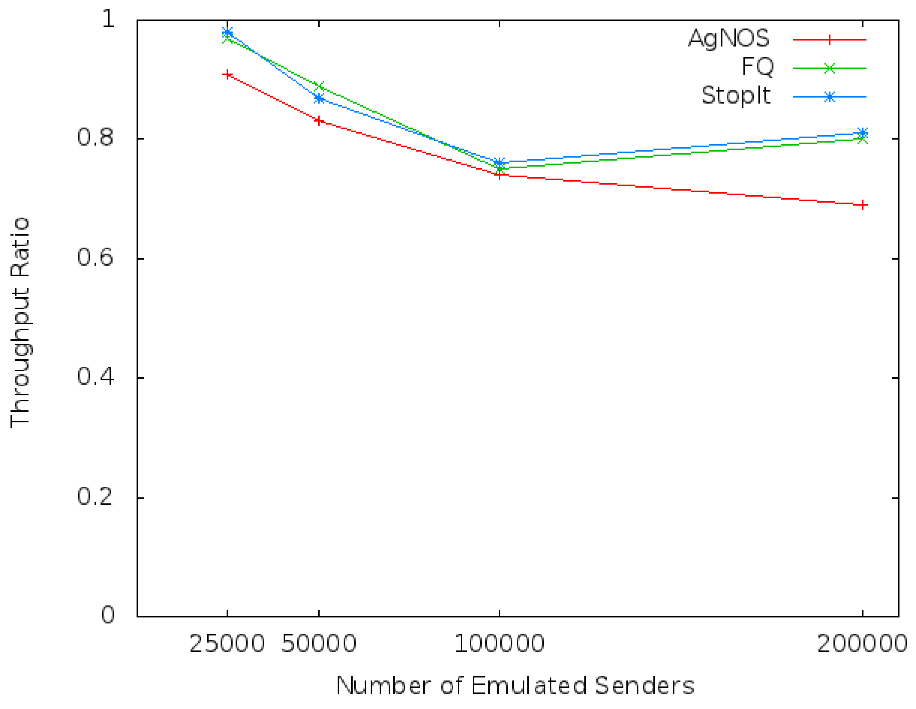

Figure D.8: Throughput ratio between attack and normal traffic .

The AgNOS architecture provides a behavior similar to FQ and StopIt, except for 200K attack hosts whereas AgNOS behavior decreases throughput ratio. Yet, AgNOS behavior is acceptable because it keeps the service running at the target host. This behavior difference between AgNOS and FQ/StopIt for 200K hosts is because after a certain number of senders the traffic policy-based mechanism stabilizes and, even improves their performance in colluding attacks. AgNOS must keep sending messages to others domains requesting to block packets and informing about attacks.

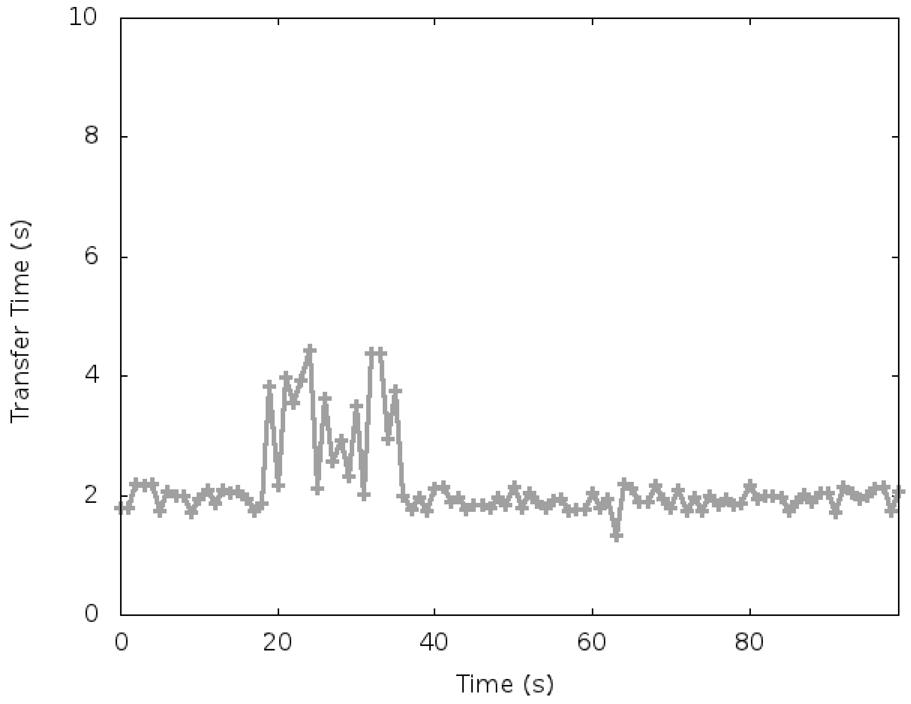

Figure D.9: Transfer time during a DDoS attack.

Figure D.9 demonstrates the exact point where AgNOS begins to reason and cooperate with SDN domains. The experiment time starts at time 0s and attackers start to send the UDP traffic to the victim. At some point, the file transfer time begins to increase, but it soon decreases to an acceptable value. The graph shows that AgNOS agents need some time to communicate between SDN domains about the attack and to request the blocking or rate-limiting of flows.

For that scenario of Figure D.9, the time duration of the cooperation is approximately 18 seconds. It is important to note that it takes about 20 seconds for the attack to fully begin and impact in the file transfer file. Eventually, the detection component sends an *Attack_detected* alarm to the intelligent agent.

D.4.2 Network Traffic Management

The aim of this case study is to show that AgNOS agents can deploy different tasks in networks with SDN domains. Furthermore, the tasks deployed are much less complex in such domains if comparing with the current Internet design. As stated in Appendix D, this proof-of-concept implementation is capable of defining routing tables in order to provide BGP-like routing for SDN [Bennesby et al., 2012]. Each AgNOS agent senses its domain and evaluates if problems exist for some host. In this scenario, as depicted in Figure D.2, a bottleneck link suffers congestion and decreases the overall throughput for every host's flow that passes through this path.

Table D.4 presents the mean of the throughput values of the TCP connection between the tested ASes. When the scenario is deployed only with Interas, the average throughput is about $139Kbps$. When AgNOS agents are deployed in the network, they detect a performance degradation and uses Interas to re-route the flows, then the average TCP throughput increases to approximately $222Kbps$.

Table D.4: Mean of throughput values of the TCP connection.

	Value	Standard Error	Samples
Only Interas	139.28 Kbps	1.43	33
AgNOS	222.72 Kbps	1.67	33

We selected a fragment of the 33 replications deployed for this scenario. Figure D.10 shows the throughput value of the TCP connection between source and destination along the duration of the connection.

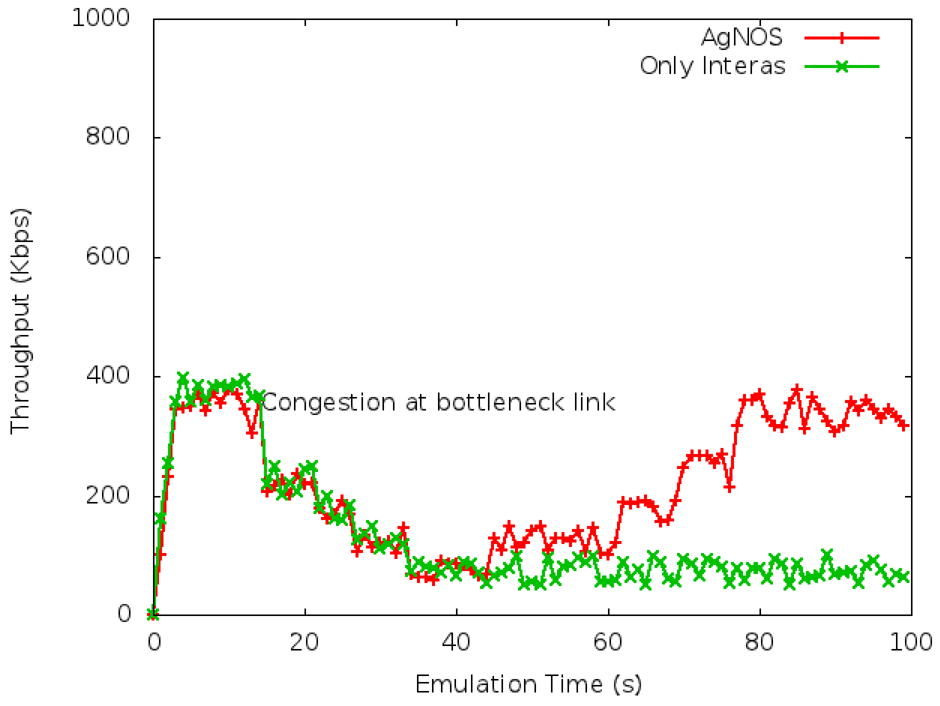

Figure D.10: Throughput of the sender during a bottleneck link congestion.

Based on the these two connections we can draw some important conclusions:

- At some point, the AgNOS agent detects a performance degradation in a host connected to the OpenFlow switch (we set up the threshold to about $50Kbps$).
- The agent reasons about the flow representing this connection and the routing information available in the KB. It generates several events, including messages events between AgNOS agents in other domains. Then, it request a change in the routing table using the Interas component and also sending requests to the peer ASes.
- As expected, after some time the new route increases the TCP throughput and the AgNOS agent receives a positive feedback from the host.

D.5 Appendix Remarks

This appendix presented an overview of two case studies constructed aiming to validate AgNOS novel functionalities. The problem of mitigating DDoS attacks poses the most challenging tasks since it involves the cooperation of agents in different domains and the negotiation of filtering or blocking request. Routing and network traffic control aim to provide a broader application of AgNOS to Internet's problem set.

As a proof-of-concept implementation of the AgNOS architecture, we handled the problems in very realistic scenarios based on important related works. AgNOS agents used cooperation and working with other agents in different SDN domains, they kept the network running within acceptable bounds.

Appendix E

Final Considerations

Chance favors the prepared mind.
Louis Pasteur (1822–1895)

Contents

SECTION B.3.5 PRESENTS a discussion about the limitations and weaknesses of autonomous networking architectures. The main drawback of current architectures is the complexity to handle knowledge about the network. As we saw in Appendix C, that AgNOS agents models the network entities using a logic based on a subset of structured first order clauses. As described in Appendix D, we could model complex scenarios using this logical framework. This method is much more efficient than implementing a model-based translation layer between autonomous systems and the network, such as in Jennings et al. [2007].

In a flow-based management scenario, we achieved better performance than the autonomous networking deploying per-packet analysis, such as in Dutta et al. [2007]. Actually, this performance is difficult to compare with current autonomous networking literature, since most of them are just conceptual and abstract ideas, different from AgNOS that has a complete implementation with reasoning, acting and sensing mechanisms. More details about the difference between per-flow and per-packet intelligent reasoning can be consulted in our related work in Braga et al. [2010].

Furthermore, with a fraction of knowledge and property implementation of the AgNOS agent's sensors with SDN, we have built a complex mechanism for DDoS mitigation, and also traffic management. Such attempt in current autonomous systems is a costly and complex task, such as in Tesauro et al. [2004] and Xu et al. [2007]. The aim of AgNOS is to reduce the complexity to manage networks, with an agent logic closer to the daily tools used by network administrators. Furthermore, it brings into SDN paradigm a new road to experiment with different mitigation procedures, e.g. when conflicting policies from different ASs need to reach an agreement through negotiation.

From the point of view of accessing networking resources, the acting phase of autonomous systems, AgNOS provides a much better and improved mechanism. Instead of accessing low-level primitives, such as in [Bouabene et al., 2010], AgNOS agents generate events which are handled by high-level network applications. For example, we showed an agent able to perform modifications to routing tables. Using only specific events, the agent can re-route packets in the network. Such process in current autonomous system is a very complex task since these agents could have to control different switch/routers APIs and different constrains, probably increasing the time to specify and build local knowledge bases.

A critical difference between AgNOS and current autonomous systems is the execution environment of the agents. To the best of our knowledge, this thesis is the first to implement intelligent agents in the same environment of the network controller. Agents run as network management applications and are served directly by the network view with improved performance since they are in the core of the network controller running in C++. The work of Mattos et al. [2011] also proposes the use of multiagent systems in SDN domains, but their approach is totally different and inefficient. Deploying agents for each switch in the network is a costly operation, and even impossible for some switches. Furthermore, agents does not have completely access to all controller's features. This approach is similar to the ones in autonomous networking [Bieszczad et al., 1998; Stephan et al., 2004; Chen et al., 2009; Satoh, 2006; Akyazi and Uyar, 2008], and its dependency on an execution environment in the switch, leads to a scalability problem that SDN tries to overcome with virtually centralized controllers.

Deploying intelligent agents for each network switch/router is an unreliable design decision. As already stated, the Internet has more than 40,000 ASes. We believe that the possibility of creating and deploying intelligent agents on top of network controllers can motivate network stakeholders to adopt large-scale cooperative solutions. This realizes one the goals of Clark et al. [2003], providing incentives for cognitive tools in the Internet. Instead of embedding agents for each router, they just update the software running centrally in the network controller. Furthermore, agent-based solutions can be tested using real testbeds [Tavakoli et al., 2009] or, as done in this thesis, through high efficient network emulation tools [Lantz et al., 2010b].

This is the reason we consider the results in Section D.4 very acceptable, despite the fact of a slightly poorly behavior. DDoS filter-based mechanisms such as Liu et al. [2008] and capability-based such as Yang et al. [2008], need the update of every router in the Internet, or a large subset of it. This thesis showed that by implementing such mechanisms as AgNOS features, we are able to expand the applicability of such solutions that are very hard to see in large-scale Internet deployments. The vision of SDN allows this, we the possibility to incrementally update portions of the Internet with SDN capabilities and providing ways to create innovations.

Cooperation between SDN domains is an important aspect of the Future Internet. The possibility to exchange information and request actions that can not be accomplished locally is a feature needed to address important problems.

We showed in this thesis that DDoS mitigation and inter-AS routing for traffic management can be accomplished from the cooperation between AgNOS domains. Instead of proposing a new application-level protocol, we deployed a subset of a very efficient communication language for agents. Furthermore, this language is implemented using SDN primitives. AgNOS agents can cooperate on top of a common knowledge and efficiently deliver and receive messages. This is very different from *status quo* and network controllers can benefit from decades of agent language research. With a small and efficient subset of this AgNOS's communication language we handled two important problems in the Internet.

This work stands on the assumption that SDN will gradually be adopted by network domains. SDN is not a toy concept. It is part of a world-wide consortium trying to innovate in network research, and its deployment is reaching large scale infra-structures like GENI and Internet2, as well as being adopted by some of the most important network operators, such as Google [Google, 2013]. Additionally, a new proposal of IP and transport unification through OpenFlow increases its possibility for future large scale adoption [Das et al., 2010]. As long this assumption holds, this research shows how intelligent scalable architectures can be used to build the Future Internet based on SDN.

SDN is a promising technology because it offers flexibility in the development of new services and innovation for networks. This is the case of AgNOS. With AgNOS we demonstrated the following statements:

Knowledge and Reasoning. The process of building autonomous entities in SDN is less complex because of the centralized approach and the abstraction provided by a network controller. In this thesis, agents perform efficiently, in a way not yet described, or demonstrated, in the literature of autonomous networks. AgNOS agents do not need to perform per-packet processing. They follow SDN's *flow* definition and represent and reason using knowledge which changes less frequently. In Appendix C, we presented the details of this architecture.

Cooperation for SDN domains. AgNOS agents do cooperate. This is an important feature for SDN in the Future Internet. Most of the problems of the Internet exist due to the lack of inter-domain cooperation. Appendix D shows AgNOS handling these problems.

Deployment. AgNOS can be smoothly deployed in legacy networks. As long the network becomes more based on the OpenFlow's switches abstraction, agent's *percepts* and *actions* become more efficient. This is a great improvement from current autonomous systems that are restricted to small scale testing. AgNOS is easily deployable in any SDN domain with smoothly configuration. AgNOS agents keep track of the most needed knowledge.

E.1 Future Works

In Section B.4 we propose a paradigm shift in autonomous networking research, as well in the way we build SDN intelligent agent-based applications. This marriage opens up new roads to explore and develop new techniques such as:

Noncooperative network environments. In this work we developed the framework to build intelligent agents for SDN. This is not an easy task since SDN provides new concepts and abstractions. AgNOS implements the foundations of this approach. We decided to reduce the scope to cooperative agents, and the study of the relationship between noncooperative domains is an interesting problem to explore. Game theory applied to multiagent systems [Shoham and Leyton-Brown, 2009] can be used to build agents that behave autonomously in more complex SDN scenarios.

Conflicting inter-AS policies. In this work we used a very simple policy for the mitigation in the case of DDoS (Section D.1.2). In Section D.4.1, it was pointed out that new mitigation techniques could be used when conflicting policies from different ASs need to reach an agreement through negotiation. There are many possibilities in AI literature to explore conflict resolution attempts, such as collaborative plans for group action.

Large scale deployment. Mininet is a robust tool for SDN prototyping. As a future work, AgNOS agents can be tested in large scale scenarios with production traffic, such as the one provided by GENI (Global Environment for Network Innovations) [GENI, 2012].

Extension to different problems. Since the foundation of the AgNOS architecture is established, AgNOS agents can be extended to work with different problems. To handle these problems, we need to model the knowledge involved in the environment and also to develop extensions to the sensors and actuators of the agents using SDN primitives.

AgNOS resilience. We stand on our own mechanism based on backup techniques for the architecture resilience [Fonseca et al., 2012]. Future work can extend this resilience mechanism and delegate to AgNOS agents the task of synchronizing the backup process.

E.2 Comments on Publications

The following list presents the papers published during this research and papers under peer-review by the community:

- A. Passito, E. Mota, and R. Braga. Towards an Agent-based NOX/OpenFlow Platform for the Internet. In Workshop on Future Internet Experimental Research, may 2010. [Passito et al., 2010] – This paper presents the AgNOS's seminal idea. To the best of our knowledge, this is the first publication to state the application of autonomous networks to the SDN paradigm.
- R. B. Braga, E. M. Mota, and A. P. Passito. Lightweight DDoS Flooding Attack Detection using NOX/OpenFlow. Local Computer Networks, Annual IEEE Conference on, 0:408?415, 2010. [Braga et al., 2010] – In this paper, we built the basic framework of DDoS attack detection for AgNOS. The attack mitigation mechanism described in this paper complements the effective DDoS defense mechanism provided in AgNOS.
- R. Bennesby, P. Fonseca, E. Mota, and A. Passito. An Inter-AS Routing Component for Software-Defined Networks. In IEEE/IFIP Network Operations and Management Symposium, 2012. [Bennesby et al., 2012] – In this paper, we created the basic framework to handle inter-domain routing in the Future Internet. With this framework, AgNOS agents can effectively control flow routing.
- P. Fonseca, R. Bennesby, A. Passito, and E. Mota. A Replication Component for Resilient Openflow-Based Networking. In IEEE/IFIP Network Operations and Management Symposium, 2012. [Fonseca et al., 2012] – In this paper, we developed a mechanism for SDN resilience. This feature is important since AgNOS agents are deployed on top of controllers susceptible to network failures or security attacks. With this component, agents can be replicated in different backup servers.
- E.Mota, A. Passito, P. Fonseca, R. Bennesby, R. Braga. Experimenting Software-Defined Networking Applications in OpenFlow-based Virtualized Testbeds. Elsevier Computer Networks, 2012. [Article under review].
- A.Passito, E.Mota, N. Souza. Agent-based Autonomous Networking and a New Perspective on Future Internet. ACM/IEEE Computing Surveys, 2012. [Article under submission process].
- A.Passito, E.Mota. Autonomous Control in the Future Internet. ACM/IEEE Transactions on Networking, 2012. [Article under submission process].

Bibliography

Abliz, M. (2011). Internet Denial of Service Attacks and Defense Mechanisms. Technical report.

Akashi, O., Fukuda, K., Hirotsu, T., and Sugawara, T. (2006). Policy-based BGP Control Architecture for Autonomous Routing Management. In *INM '06: Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management*, pages 77–82, New York, NY, USA. ACM.

Akashi, O., Sugawara, T., K., M., Maruyama, M., and Koyanagi, K. (2002). Agent System for Inter-AS Routing Error Diagnosis. *IEEE Internet Computing*, pages 78–82.

Akyazi, U. and Uyar, A. (2008). Distributed Intrusion Detection using Mobile Agents Against DDoS Attacks. In *Computer and Information Sciences, 2008. ISCIS '08. 23rd International Symposium on*, pages 1 –6.

Anderson, T., Roscoe, T., and Wetherall, D. (2004). Preventing Internet Denial-of-Service with Capabilities. *SIGCOMM Comput. Commun. Rev.*, 34(1):39–44.

Arbor Networks (2009). Worldwide Infrastructure Security Report, Volume V. http://www.arbornetworks.com.

Argyraki, K. and Cheriton, D. R. (2009). Scalable Network-Layer Defense Against Internet Bandwidth-Flooding Attacks. *IEEE/ACM Trans. Netw.*, 17(4):1284–1297.

Austin, J. (1975). *How To Do Things with Words*. Harvard Univ Pr.

Balasubramaniam, S., Botvich, D., Jennings, B., Davy, S., Donnelly, W., and Strassner, J. (2009). Policy-Constrained Bio-Inspired Processes for Autonomic Route Management. *Computer Networks*, 53(10):1666 – 1682. Autonomic and Self-Organising Systems.

Bennesby, R., Fonseca, P., Mota, E., and Passito, A. (2012). An Inter-AS Routing Component for Software-Defined Networks. In *IEEE/IFIP Network Operations and Management Symposium*.

Bieszczad, A., Pagurek, B., and White, T. (1998). Mobile Agents for Network Management. *Communications Surveys Tutorials, IEEE*, 1(1):2 –9.

Bouabene, G., Jelger, C., Tschudin, C., Schmid, S., Keller, A., and May, M. (2010). The Autonomic Network Architecture (ANA). *Selected Areas in Communications, IEEE Journal on*, 28(1):4 –14.

Boutaba, R., Iraqi, Y., and Mehaoua, A. (2003). A Multi-Agent Architecture for QoS Management in Multimedia Networks. *Journal of Network and Systems Management*, 11:83–107. 10.1023/A:1022497125456.

Braga, R. B., Mota, E. M., and Passito, A. P. (2010). Lightweight DDoS Flooding Attack Detection using NOX/OpenFlow. *Local Computer Networks, Annual IEEE Conference on*, 0:408–415.

Bruynooghe, M. (1983). *The Memory Management of Prolog Implementations*. Number 16 in APIC Studies in Data Processing. Academic Press.

Bullot, T., Khatoun, R., Hugues, L., Gaiti, D., and Merghem-Boulahia, L. (2008). A Situatedness-based Knowledge Plane for Autonomic Networking. *International Journal of Network Management*, 18(2):171–193.

Butler, K., McDaniel, P., and Aiello, W. (2006). Optimizing BGP Security by Exploiting Path Stability. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 298–310, New York, NY, USA. ACM.

C.-L. Chang, R.-T. L. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Carvalho, L. (2011). *Gerenciamento Adaptivo da Qualidade da Fala entre Terminais VoIP*. PhD thesis, Federal University of Amazonas, Manaus, Brazil.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking Control of the Enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications*, SIGCOMM '07, pages 1–12, New York, NY, USA. ACM.

Chadha, V. (2006). Sample size determination in health studies. Technical report, NTI.

Chen, H., Zhou, W., and Liu, L. (2009). An Approach of Agent-Based Architecture for Autonomic Network Management. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, pages 1 –5.

Chen, Y., Hwang, K., and Ku., W.-S. (2007). Collaborative Detection of DDoS Attacks over Multiple Network Domains. *IEEE Transactions on Parallel and Distributed Systems*, 18:1649–1662.

Cheng, Y., Farha, R., Kim, M. S., Leon-Garcia, A., and Hong, J. W.-K. (2006). A Generic Architecture for Autonomic Service and Network Management. *Computer Communications*, 29(18):3691 – 3709.

CIDR (2012). CIDR Report. http://www.cidr-report.org/as2.0/.

Cittadini, L., Battista, G. D., and Rimondini, M. (2012). On the Stability of Interdomain Routing. *ACM Comput. Surv.*, 44(4):26:1–26:40.

Clark, D. (1988). The Design Philosophy of the DARPA Internet Protocols. In *SIGCOMM '88: Symposium Proceedings on Communications Architectures and Protocols*, pages 106–114, New York, NY, USA. ACM.

Clark, D. D., Partridge, C., Ramming, J. C., and Wroclawski, J. T. (2003). A Knowledge Plane for the Internet. In *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 3–10, New York, NY, USA. ACM.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a Plan-based Theory of Speech Acts. *Cognitive Science*, 3(3):177 – 212.

Das, R., Kephart, J. O., Lefurgy, C., Tesauro, G., Levine, D. W., and Chan, H. (2008). Autonomic Multi-agent Management of Power and Performance in Data Centers. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, AAMAS '08, pages 107–114, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Das, S., Parulkar, G., Singh, P., Getachew, D., Ong, L., and McKeown, N. (2010). Packet and Circuit Network Convergence with OpenFlow. In *Optical Fiber Conference (OFC/NFOEC'10)*.

Das, S., Sharafat, A., Parulkar, G., and McKeown, N. (2011). MPLS with a Simple OpenFlow Control Plane. In *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*, pages 1 –3.

Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878.

Dietterich, T. G. and Langley, P. (2007). *Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges*, pages 97–120. John Wiley & Sons, Ltd.

d'Inverno, M., Luck, M., Georgeff, M., Kinny, D., and Wooldridge, M. (2004). The dMars Architecture: A Specification of the Distributed Multi-Agent Reasoning System. *Autonomous Agents and Multi-Agent Systems*, 9(1):5–53.

Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., and Zambonelli, F. (2006). A Survey of Autonomic Communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259.

Dodge, Y. (2008). *The Concise Encyclopedia of Statistics*. Springer Reference. Springer.

Dovrolis, C. and Streelman, J. T. (2010). Evolvable Network Architectures: What Can We Learn from Biology? *SIGCOMM Comput. Commun. Rev.*, 40(2):72–77.

Drutskoy, D., Keller, E., and Rexford, J. (2013). Scalable Network Virtualization in Software-Defined Networks. *IEEE Internet Computing:Special Issue on Virtualization*.

Dutta, R., Rouskas, G., Baldine, I., Bragg, A., and Stevenson, D. (2007). The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1899 –1904.

Erickson, D., Gibb, G., Heller, B., Underhill, D., Naous, J., Appenzeller, G., Parulkar, G., McKeown, N., Rosenblum, M., Lam, M., Kumar, S., Alaria, V., Monclus, P., Bonomi, F., Tourrilhes, J., Yalagandula, P., Banerjee, S., Clark, C., and McGeer, R. (2008). A Demonstration of Virtual Machine Mobility in an OpenFlow Network. In *SIGCOMM '08: Proceedings of* *the 2008 conference on Applications, technologies, architectures, and protocols for computer communications (Demo)*, New York, NY, USA. ACM.

Esfandiari, B., Deflandre, G., and Quinqueton, J. (1998). An Interface Agent for Network Supervision. *IATA - IOS Press*, pages 21–28.

Esseghir, M., Ghamri-Doudane, S., and Haddadou, K. (2008). First Steps Towards an Autonomic Management System. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 602 –614.

Eymann, T., Reinickke, M., Ardaiz, O., Artigas, P., Freitag, F., and Navarro, L. (2003). Self-Organizing Resource Allocation for Autonomic Network. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 656 – 660.

Feamster, N. and Balakrishnan, H. (2005). Detecting BGP Configuration Faults with Static Analysis. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 43–56, Berkeley, CA, USA. USENIX Association.

Feldmann, A. (2007). Internet Clean-Slate Design: What and Why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64.

FIPA (2002). FIPA Contract Net Interaction Protocol Specification. *Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00029/.*

FIPA (2004a). FIPA ACL Message Representation in Bit-Efficient Encoding. *Foundation for Intelligent Physical Agents, www.fipa.org/specs/fipa00069/SC00069G.pdf.*

FIPA (2004b). Fipa ACL Message Structure Specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html.*

FIPA (2004c). FIPA Communicative Act Library Specification. *Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00037/PC00037F.html.*

Fonseca, P., Bennesby, R., Passito, A., and Mota, E. (2012). A Replication Component for Resilient OpenFlow-based Networking. In *IEEE/IFIP Network Operations and Management Symposium.*

Gaiti, D. (2008). *Autonomic Networks.* Wiley.

Gavalas, D., Tsekouras, G., and Anagnostopoulos, C. (2009). A Mobile Agent Platform for Distributed Network and Systems Management. *Journal of Systems and Software*, 82(2):355–371.

Genesereth, M. and Nilsson, N. (1987). *Logical Foundations of Artificial Intelligence*, volume 9. Morgan Kaufmann Los Altos, California.

GENI (2012). Global Environment for Network Innovations.

Gogineni, H., Greenberg, A., Maltz, D., Ng, T., Yan, H., and Zhang, H. (2010). MMS: An Autonomic Network-Layer Foundation for Network Management. *Selected Areas in Communications, IEEE Journal on*, 28(1):15 –27.

Goldszmidt, G. and Yemini, Y. (1998). Delegated Agents for Network Management. *Communications Magazine, IEEE*, 36(3):66 –70.

Google (2013). Research at Google. http://research.google.com/pubs/Networking.html.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38:105–110.

Handigol, N., Seetharaman, S., Flajslik, M., McKeown, N., and Johari, R. (2008). Plug-n-Serve: Load-Balancing Web Traffic using OpenFlow.

Hayzelden, A. and Bigham, J. (1999). Agent Technology in Communications Systems: an Overview. *The Knowledge Engineering Review*, 14(4):341–375.

Hegazy, I., Al-Arif, T., Fayed, Z., and Faheem, H. (2003). A Multi-Agent based System for Intrusion Detection. *Potentials, IEEE*, 22(4):28 – 31.

Hinrichs, T. L., Gude, N. S., Casado, M., Mitchell, J. C., and Shenker, S. (2009). Practical Declarative Network Management. In *WREN '09: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, pages 1–10, New York, NY, USA. ACM.

Jain, A., Hellerstein, J., Ratnasamy, S., and Wetherall, D. (2004). A Wakeup Call for Internet Monitoring Systems: The Case for Distributed Triggers. In *Proc. Third ACM SIGCOMM HotNets Workshop*. Citeseer.

Jain, R. K. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons.

Jennings, B., van der Meer, S., Balasubramaniam, S., Botvich, D., Foghlu, M., Donnelly, W., and Strassner, J. (2007). Towards Autonomic Management of Communications Networks. *Communications Magazine, IEEE*, 45(10):112 –121.

Kamienski, C. A. and Sadok, D. (2004). The Case for Interdomain Dynamic QoS-based Service Negotiation in the Internet. *Computer Communications*, 27(7):622 – 637.

Kephart, J. and Chess, D. (2003). The Vision of Autonomic Computing. *Computer*, 36(1):41–50.

Kephart, J. and Das, R. (2007). Achieving Self-Management Via Utility Functions. *IEEE Internet Computing*, pages 40–48.

Kephart, J. and Walsh, W. (2004). An artificial Intelligence Perspective on Autonomic Computing Policies. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 3 – 12.

Koponen, T., Shenker, S., Balakrishnan, H., Feamster, N., Ganichev, I., Ghodsi, A., Godfrey, P. B., McKeown, N., Parulkar, G., Raghavan, B., Rexford, J., Arianfar, S., and Kuptsov, D. (2011). Architecting for Innovation. *SIGCOMM Comput. Commun. Rev.*, 41:24–36.

Lantz, B., Heller, B., and McKeown, N. (2010a). A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 19:1–19:6, New York, NY, USA. ACM.

Lantz, B., Heller, B., and McKeown, N. (2010b). A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 19:1–19:6, New York, NY, USA. ACM.

Lavinal, E., Desprats, T., and Raynaud, Y. (2009). A Multi-Agent Self-Adaptative Management Framework. *International Journal of Network Management*, pages 217–235.

Liu, X. and Yang, X. (2010). NetFence: Preventing Internet Denial of Service from Inside Out. In *SIGCOMM '10: Proceedings of the 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (Demo)*, New York, NY, USA. ACM.

Liu, X., Yang, X., and Lu, Y. (2008). To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-Node Notnets. In *SIGCOMM '08: Proceedings of the 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 195–206, New York, NY, USA. ACM.

Lubin, B., Kephart, J., Das, R., and Parkes, D. (2009). Expressive Power-based Resource Allocation for Data Centers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 11–17.

Madhyastha, H. V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., and Venkataramani, A. (2006). iPlane: an Information Plane for Distributed Services. In *OSDI '06: Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 367–380, Berkeley, CA, USA. USENIX Association.

Magedanz, T., Rothermel, K., and Krause, S. (1996). Intelligent Agents: an Emerging Technology for Next Generation Telecommunications? In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 2, pages 464 –472 vol.2.

Mattos, D. M. F., Fernandes, N. C., da Costa, V. T., Cardoso, L. P., Campista, M. E. M., Costa, L. H. M. K., and Duarte, O. C. M. B. (2011). OMNI: OpenFlow MaNagement Infrastructure. In *2nd IFIP International Conference Network of the Future.*

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74.

Menezes, C. S. (1989). *Ambientes para Processamento de Conhecimento.* PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, PUC-Rio, Brasil.

Meskaoui, N., Merghem, L., and Kablan43, K. (2003). Diffserv Network Control Using a Behavioral Multi-Agent System. In *Network control and engineering for QoS, Security, and Mobility II: IFIP TC6/WG6. 2 & WG6. 7 Second International Conference on Network Control and Engineering for QoS, Security, and Mobility (Net-Con 2003), October 13-15, 2003, Muscat, Oman*, volume 133, page 51. Springer Netherlands.

Mota, E. (1993). Backtracking Inteligente em Árvore de Prova. Master's thesis, Universidade Federal de Minas Gerais.

Mota, E. (1998). *Time Granularity in Simulation Models within a Multi-Agent System.* PhD thesis, University of Edinburgh.

ONF (2011). OpenFlow Switch Specification. Version 1.1.0. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

Passito, A., Mota, E., and Braga, R. (2010). Towards an Agent-based NOX/OpenFlow Platform for the Internet. In *Workshop on Future Internet Experimental Research.*

Peng, T., Leckie, C., and Ramamohanarao, K. (2003a). Detecting Distributed Denial of Service Attacks by Sharing Distributed Beliefs. In Safavi-Naini, R. and Seberry, J., editors, *Information Security and Privacy*, volume 2727 of *Lecture Notes in Computer Science*, pages 217–217. Springer Berlin / Heidelberg.

Peng, T., Leckie, C., and Ramamohanarao, K. (2003b). Detecting Reflector Attacks by Sharing Beliefs. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 3, pages 1358 – 1362 vol.3.

Peng, T., Leckie, C., and Ramamohanarao, K. (2007a). Information Sharing for Distributed Intrusion Detection Systems. *Journal of Network and Computer Applications*, 30(3):877 – 899.

Peng, T., Leckie, C., and Ramamohanarao, K. (2007b). Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Comput. Surv.*, 39(1).

Presuhn, R. (2002). Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416 (Standard).

Ramchurn, S., Huynh, D., and Jennings, N. (2004). Trust in Multi-Agent Systems. *The Knowledge Engineering Review*, 19(01):1–25.

Razzaque, M., Dobson, S., and Nixon, P. (2006). A Cross-Layer Architecture for Autonomic Communications. In Gaiti, D., Pujolle, G., Al-Shaer, E., Calvert, K., Dobson, S., Leduc, G., and Martikainen, O., editors, *Autonomic Networking*, volume 4195 of *Lecture Notes in Computer Science*, pages 25–35. Springer Berlin / Heidelberg.

Rekhter, Y., Li, T., and Hares, S. (2006). A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard).

Rexford, J. and Dovrolis, C. (2010). Future Internet Architecture: Clean-Slate Versus Evolutionary Research. *Commun. ACM*, 53(9):36–40.

Rouhana, N. and Horlait, E. (2001). Dynamic Congestion Avoidance Using Multi-Agents Systems. In Pierre, S. and Glitho, R., editors, *Mobile Agents for Telecommunication Applications*, volume 2164 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin / Heidelberg.

R.S. Boyer, J. M. (1972). The sharing of structure in theorem-proving programs. In Bernadrd Meltzer, D. M., editor, *Annual Machine Intelligence*, volume 7, pages 101–116. Edinburgh University Press.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach.* Prentice hall.

Russell, S., Subramanian, D., and Parr, R. (1993). Provably Bounded Optimal Agents. In *International Joint Conference on Artificial Intelligence*, volume 13, pages 338–338. Citeseer.

Russell, S. and Wefald, E. (1991). *Do The Right Thing: Studies in Limited Rationality*. The MIT Press.

Samaan, N. and Karmouch, A. (2009). Towards Autonomic Network Management: an Analysis of Current and Future Research Directions. *Communications Surveys Tutorials, IEEE*, 11(3):22 –36.

Satoh, I. (2006). Building and Selecting Mobile Agents for Network Management. *Journal of Network and Systems Management*, 14:147–169. 10.1007/s10922-005-9018-1.

Schmid, S., Sifalakis, M., and Hutchison, D. (2006). Towards Autonomic Networks. *Autonomic Networking*, pages 1–11.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.-Y., Kazemian, P., Kobayashi, M., Naous, J., Seetharaman, S., Underhill, D., Yabe, T., Yap, K.-K., Yiakoumis, Y., Zeng, H., Appenzeller, G., Johari, R., McKeown, N., and Parulkar, G. (2010a). Carving Research Slices out of your Production Networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40:129–130.

Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.-Y., Kazemian, P., Kobayashi, M., Naous, J., Seetharaman, S., Underhill, D., Yabe, T., Yap, K.-K., Yiakoumis, Y., Zeng, H., Appenzeller, G., Johari, R., McKeown, N., and Parulkar, G. (2010b). Carving Research Slices out of your Production Networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40:129–130.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge Univ Pr.

Soldo, F., Argyraki, K., and Markopoulou, A. (2012). Optimal Source-based Filtering of Unwanted Traffic. In *IEEE/ACM Transactions on Networking*.

Stanford University (2012). Clean Slate Program. http://cleanslate.stanford.edu.

Stephan, R., Ray, P., and Paramesh, N. (2004). Network Management Platform based on Mobile Agents. *Int. J. Netw. Manag.*, 14:59–73.

Strassner, J., Agoulmine, N., and Lehtihet, E. (2006). FOCALE: A Novel Autonomic Networking Architecture. In *In: Latin American Autonomic Computing Symposium (LAACS)*.

Suchara, M., Fabrikant, A., and Rexford, J. (2011). BGP Safety with Spurious Updates. In *INFOCOM, 2011 Proceedings IEEE*, pages 2966 –2974.

Tavakoli, A., Casado, M., Koponen, T., and Shenker, S. (2009). Applying NOX to the Datacenter. In *Eighth ACM Workshop on Hot Topics in Networks (HotNets-VIII)*.

Tesauro, G., Chess, D. M., Walsh, W. E., Das, R., Segal, A., Whalley, I., Kephart, J. O., and White, S. R. (2004). A Multi-Agent Systems Approach to Autonomic Computing. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pages 464–471, Washington, DC, USA. IEEE Computer Society.

Thomas, R. W. (2007). *Cognitive Networks.* PhD thesis, Virginia Polytechnic Institute and State University.

Thottan, M. and Ji, C. (1998). Proactive Anomaly Detection using Distributed Intelligent Agents. *Network, IEEE*, 12(5):21 –27.

Tianfield, H. (2003). Multi-Agent based Autonomic Architecture for Network Management. In *Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on*, pages 462 – 469.

Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., and Sherwood, R. (2012). On Controller Performance in Software-Defined Networks. In *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services.*

van Emden, M. H. (1984). *An interpreting algorithm for Prolog programs.* Ellis Horwood Series Artificial Intelligence. Ellis Horwood.

Vieira, N. J. (1987). *Máquinas de Inferência para Sistemas Baseados em Conhecimento.* PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro.

Wawrzoniak, M., Peterson, L., and Roscoe, T. (2004). Sophia: an Information Plane for Networked Systems. *SIGCOMM Comput. Commun. Rev.*, 34(1):15–20.

Wooldridge, M. (2009). *An Introduction to Multiagent Systems.* Wiley, 2 edition.

Xu, X., Sun, Y., and Huang, Z. (2007). Defending DDoS Attacks using Hidden Markov Models and Cooperative Reinforcement Learning. In *Proceedings of the 2007 Pacific Asia conference on Intelligence and security informatics*, PAISI'07, pages 196–207, Berlin, Heidelberg. Springer-Verlag.

Yalagandula, P., Sharma, P., Banerjee, S., Basu, S., and Lee, S.-J. (2006). S3: a Scalable Sensing Service for Monitoring Large Networked Systems. In *INM '06: Proceedings of the 2006 SIGCOMM workshop on Internet network management*, pages 71–76, New York, NY, USA. ACM.

Yang, X., Wetherall, D., and Anderson, T. (2005). A DoS-Limiting Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 35:241–252.

Yang, X., Wetherall, D., and Anderson, T. (2008). TVA: A DoS-Limiting Network Architecture. *Networking, IEEE/ACM Transactions on*, 16(6):1267 –1280.

Yannuzzi, M., Masip-Bruin, X., and Bonaventure, O. (2005). Open Issues in Interdomain Routing: a Survey. *Network, IEEE*, 19(6):49 – 56.

Yap, K.-K., Sherwood, R., Kobayashi, M., Huang, T.-Y., Chan, M., Handigol, N., McKeown, N., and Parulkar, G. (2010). Blueprint for Introducing Innovation into Wireless Mobile Networks. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, VISA '10, pages 25–32, New York, NY, USA. ACM.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing Multiagent Systems: The Gaia Methodology. *ACM Trans. Softw. Eng. Methodol.*, 12:317–370.