AddreSsing ThReats for virtualIseD services

DELIVERABLE D1.1

STATE OF THE ART, PROJECT CONCEPT AND REQUIREMENTS

Grant Agreement number:	786922
Project acronym:	ASTRID
Project title:	AddreSsing ThReats for virtualIseD services
Start date of the project:	01/05/2018
Duration of the project:	36 months
Type of Action:	Research & Innovation Action (RIA)
Project Coordinator:	Name: Orazio Toscano Phone: +39 010 600 2223 e-mail: orazio@ericsson.com
Due Date of Delivery:	M10 (28/02/2019)
Actual Date of Delivery:	27/02/2019
Work Package:	WP1 – Reference Architecture
Type of the Deliverable:	R
Dissemination level:	PU
Editors:	POLITO
Version:	1.0

List of Authors	
POLITO	POLITECNICO DI TORINO
Fulvio Valenza, Fulvio Risso, Riccardo Sisto, Guido Marchetto	
CNIT	CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI
Matteo Repetto, Alessandro Carrega	
DTU	DANMARKS TEKNISKE UNIVERSITET
Thanassis Giannetsos	
ETI	ERICSSON TELECOMUNICAZIONI
Orazio Toscano	
INFO	INFOCOM S.R.L.
Maurizio Giribaldi	
SURREY	UNIVERSITY OF SURREY
Mark Manulis	
AGE	AGENTSCAPE AG
Benjamin Ertl	
UBITECH	GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS
Anastasios Zafeiropoulos, Eleni Fotopoulou, Thanos Xirofotos	
TUB	TECHNISCHE UNIVERSITAET BERLIN
Tran Quang Thanh, Stefan Covaci	

Disclaimer

Copyright

Table of Contents

1 Executive Summary

This document reports the main outcomes of the preliminary technical activities carried out at the beginning of the ASTRID project in Tasks 1.1, 1.2, 1.3 and 1.4. The collective purpose of these tasks was to depict a general picture of evolving computing paradigms, to point out open challenges in cyber-security management, and to lay the foundations for the definition of a new architecture that goes beyond the limitations of existing approaches.

The rise of new forms of cyber-threats is largely due to massive usage of virtualization paradigms and the growing adoption of automation in the software life-cycle. The analysis of Task 1.1 reveals that such paradigms are progressively eroding the traditional boundaries of the security perimeter model and integrating a growing number of weak devices and applications in existing industrial and commercial processes. Security appliances today benefit from years of experience in fighting cyber-attacks, but often lack the spatial and temporal visibility to properly tackle increasing stealthy, advanced, and persistent threats. Yet their design is deeply tailored to physical infrastructures, hence falling short to meet the flexibility, efficiency, effectiveness, and elasticity levels required by distributed virtual services. The ASTRID concept implies tighter integration of security aspects in software orchestration, hence leveraging security-by-design and easiness in adopting bleeding-edge detection methodologies. A number of relevant application scenarios has been described to show how the ASTRID concept improves current practice, and to point out the main challenges and technical requirements. Most of the technical aspects highlighted by the devised scenarios are expected to merge into the two project's Use Cases.

The concrete design and implementation of the novel concept of cyber-security for virtualized services must take into account existing technologies and their expected evolution. The outcome from Task 1.2 includes a number of alternative technologies for fast and flexible inspection and monitoring of both the network and the software. It also shows that modern software orchestration paradigms have the capability to dynamically create and manage detection frameworks tailored to the specific service and users. In this respect, the prevailing types of security appliances have been briefly analysed, highlighting the main aspects that affect the information and data base for identification of anomalies and known attacks. In addition, best practices and the normative framework have been considered for legal aspects, including traffic interception and forensics analysis.

Based on the Project's concept and target application scenarios, Task 1.3 and 1.4 have initiated the design phase by elaborating a number of functional and architectural requirements. These requirements represent the preliminary guidelines for the ASTRID architecture, and will be further enriched with more concrete implementation requirements in the context of Task 1.5.

2 Introduction

This deliverable provides a summary of the information, data and methods that are preparatory to the design of the ASTRID architecture. Specifically, this document is divided into three main parts, corresponding to the contributions of different Tasks:

- project concepts and current trends in cyber-security elaborated by Task 1.1 (Section 3),
- relevant technologies and approaches identified by Task 1.2 (Section 4), and
- framework requirements devised by Task 1.3 and 1.4 (Section 5).

The first part of the document (Section 3) describes the project concepts and current trends in cyber-security. In the first part, it explains how the massive usage of virtualization has led to remote/distributed computing paradigms, including cloud, edge, and fog infrastructures (Section 3.1). With the growing recourse to public infrastructures, IoT devices, and multi-tenancy for cost and efficiency reasons, the boundaries between different domains fails physical and effective isolation, making the security perimeter models largely ineffective and raising additional cyber-threats (Section 3.2). Then, it recaps the current practice and highlights the main limitations in terms of performance, visibility, security (Section 3.3). The analysis of challenges and emerging trends shows an increasing need for flexibility, programmability, and autonomicity (Section 3.4), which motivate a transition from discrete cyber-security appliances to integrated frameworks, based on composable and interoperable layers corresponding to the main processes: context, detection, awareness (Section 3.5). Based on these premises, the overall vision for virtualized services and the main ASTRID concepts are illustrated (Section 3.6) and applied to a set of possible usage scenarios (Section 3.7).

The second part of this document (Section 4) analyses the relevant State-of-the-Art. Specifically, this analysis focus on: (i) orchestration models and strategies (Section 4.1); (ii) specification and refinement of network security policies (Section 4.2); (iii) programmable data planes for packet processing (Section 4.3); (iv) data collection and abstraction (Section 4.4); (v) distributed detection algorithms and frameworks (Section 4.5); (vi) legal interception and forensics investigation (Section 4.6); (vii) identity management and access control (Section 4.7). For the sake of brevity, all Sections focuses on the relevance and applicability of specific technology to the ASTRID framework. Where relevant, additional details are reported as Annexes.

Finally, the third part (Section 5) analyses the function and architectural requirements that come from the main concept, objectives, and application scenarios previously discussed.

3 Project concepts and current trends in cyber-security

Several market forces, like the need for flexibility, externalization, outsourcing, and cost-effectiveness are driving towards the creation of multi-domain and complex business chains and the large usage of cloud resources, especially in the creation of cyber-physical systems. This approach undoubtedly leads to more agility in service deployment and operation, even though the tight integration among diverse business roles and the need to share infrastructure and data bring additional security and privacy concerns that have not been addressed in a satisfactory way yet.

The general scenario depicted above can be further analysed by distinguishing two main trends. On the one hand, the availability of ever richer and more powerful cloud services has largely pushed the transition towards virtualization solutions, moving to the cloud even core and critical business processes in the name of increased availability, cost-effectiveness, and agility. The advent of 5G technology is expected to further accelerate this transition, by effectively integrating computing, storage, and communication resources in large pervasive environments. On the other hand, evolving business models and the large potential behind cyber-physical systems is fostering the transition from monolithic to modular architectures, spanning multiple administrative and business domains. The success of industry-driven initiatives like FIWARE witnesses the need for common and standard APIs to dynamically compose complex business chains made of software functions and smart things from different vendors.

From a cybersecurity perspective, the rise of virtualization technologies and edge computing is progressively widening the geographical area where valuable assets (servers, applications, virtual services, smart "things") are deployed. As a result, more and more business processes are built on distributed, multi-domain, and heterogeneous environments, stretching well beyond the traditionally safer enterprise's networks and equipment therein. Unfortunately, cyber-security paradigms for network threats have not advanced at the same pace.

3.1 The virtualization wave: cloud, edge, and fog

The **cloud** paradigm provides a cost-effective solution to run elastic applications, but also raises many security concerns due to the hypervisor layer, outsourcing, and multi-tenancy [1]. As a matter of fact, the attack surface is increased by the larger number of components: guest environments (virtual machines), host operating systems (servers), hypervisors, management interfaces, shared storage and networks. Sharing a common infrastructure has the unpleasant side effect that an attack to the infrastructure affects many services and tenants (e.g., DoS on shared physical networks).

Though private clouds can be set up and operated internally by single organizations, the real benefits come from outsourcing, when resources are rent from public infrastructures and there is no issue with hardware management. Tenant isolation should provide independent and secure execution sandboxes, leveraging technologies as hypervisors, network virtualization, and virtual storage. However, the shared infrastructure widens the class of local adversaries, also including other tenants and the infrastructure providers, raising new attack models (i.e., grey boxes, which involve tenants and their cloud providers) in addition to mainstream white (i.e., employees) and black boxes (i.e., external attackers) [1].

Software-based isolation introduces security interdependence in a multi-tenant environment: for instance, DoS attacks against the physical network affect all virtual networks of all tenants, while a compromised hypervisor is a potential source of eavesdropping and alteration for every hosted virtual machine or software container. In any case, full trust in the cloud provider is required, since Trusted Platform Modules are not broadly available yet.

While chasing for interactive and low-latency services, fog and edge computing are usually seen as the cloud extension to support delay-sensitive applications, like autonomous driving, health services, online gaming [2]. Lightweight tasks are run at the network edge, on local devices or network equipment (base stations, radio network controllers, access points), while deep processing is left to large cloud installations.

Fog computing clusters virtual resources from a heterogeneous set of devices deployed in the environment, owned by different entities. Most security issues with fog computing come from the hostile, uncontrolled, and unreliable environment. According to recent industrial efforts towards standardization [3], fog computing needs a management framework for deployment of software and node-to-node communication. This software backplane has a similar role to cloud management software (and interoperability is also expected to allow fog/cloud interaction), and will also be responsible to implement trust, confidentiality, and integrity services (e.g., root-of-trust for trusted execution environments, encrypted communication channels); clearly, it also represents the Achille's heel of a fog infrastructure that, if compromised, directly affects security and trust of all applications and users.

Lacking any form of physical or virtual perimeter, fog nodes are more exposed to tampering, physical damage, spoofing and jamming than cloud servers, similarly to what happens for IoT devices; however, the attack surface is larger for fog nodes, because they are prone to injection of flawed information and malware, service manipulation, data leakage [4]. Yet, compromised fog nodes are far more threatening than IoT devices, since they usually have more resources available, private data and privacy concerns, and trust relationships with a larger number of other nodes and remote instances. Mobile fog nodes are likely to take part into different federations over time, so they are more prone to get compromised and to be used as Trojan horses in multiple infrastructures. Multi-ownership of such devices brings also severe trust issues, introducing addition challenges for privacy and data leakage. Definitely, the nature of the fog paradigm naturally leads to the threat of rogue and untrustworthy infrastructures.

Edge computing targets similar applications as fog computing, but with different architectures and business models (the infrastructure is owned by a single operator and does not include user's devices); in addition, edge computing explicitly leverages telco infrastructures to provide mobile edge services like radio network information, location, bandwidth management [5] Edge computing has security concerns similar to the cloud. However, distributed resources co-located with peripheral network installations usually have less restrictions and control for physical access than traditional data centers, hence the risk of tampering is not negligible. Resources will also be limited and subject to exhaustion, due to space constraints and the cost for capillary installations, so DoS will be more likely than in the cloud. The large number of installations in multiple locations will also complicate management, will probably require more human resources (with different capabilities and security skills) and will increase the risk of wrong, poor, or missing configurations.

Exposing APIs and service access points increase the attack surface and the potential impact of an attack. As a matter of fact, such services give access to sensitive information about the physical and virtual environment, including positioning and network traffic of other users. The integration of edge computing with the legacy Operations Support System of the whole network also brings the risk that successful intrusions and privilege escalations lead to control of large infrastructures and regional or national communication services.

Finally, edge computing is expected to run orchestratable services, by dynamically composing several applications together (e.g., for Network Function Virtualization). Orchestration tools are often designed to dynamically select and load software images from specific repositories. In this case, external software may run inside the security perimeter with all related security risks.

3.2 Increasing threats from cyber-physical systems

High-performance and pervasive wireless connectivity is the key technological enabler for designing cyber-physical systems (CPS), where smart devices and software components are deeply intertwined (e.g., smart grid, autonomous automobile systems, medical monitoring, process control systems, robotics, etc.). In CPS, smart devices (sensors, actuators, robots, etc.) provide interaction with the physical environment, while computing platforms host intelligence to take decisions and react to the evolving context.

Implementation of CPS may leverage flexible and pervasive computing paradigms so as to effectively address challenging performance requirements like latency and availability; hence, they represent typical extensions of the virtualization paradigms already discussed in Section 3.1. However, the presence of smart devices brings additional security concerns.

There are a potential unlimited number of things that can be clustered together to build CPS, both in everyday life (e.g., home network broadband gateways, digital video recorders and smart TVs, smart appliances, implantable and wearable medical devices, connected cars) and industrial applications (e.g., sensors and actuators in SCADA systems and industrial automation). These resource-constrained devices are typically equipped with very simple security services (for instance, password-based authentication), while encrypted communications, integrity, intrusion detection, virtual private networks, and other security measures are often missing. As a matter of fact, the processing overhead to analyze packets, software, behaviours, events, and logs slows down systems and may be unsustainable for simplest devices (smart-phones, sensors and smart things); moreover, the usage of custom embedded operating systems makes portability of security software more difficult than in user terminals. In addition, even when security services are available, they are often disabled, left with default values, or not configured for home appliances, because average users do not have enough skills for proper configuration.

With the growing base of installed things connected to the Internet, cyber-criminals are expected to have an almost infinitely large attack surface and huge availability of resources for large-scale attacks. For instance, recent botnets like Mirai, Brickerbot, and Hajime have demonstrated the vulnerability of IoT as well as the possibility to exploit compromised devices to carry out large DDoS attacks.

Though cyber-physical systems are not an explicit target for ASTRID, their interdependency with cloud technologies is not negligible. In this respect, they should be taken into consideration when designing the ASTRID architecture, and suitable collaboration links should be established with other Projects dealing with this specific domain.

3.3 Current practice and limitations

With the increasing integration of IoT, cloud, edge, and fog resources in complex business chains involving several (untrusted) parties, the security perimeter becomes elastic (since it grows and shrinks according to resource usage) and usually encompasses external devices, software, and infrastructures. The firewall (or similar appliance) at the network boundary inspects incoming and outgoing traffic but does not protect against internal threats. The need to evolve towards distributed and capillary architectures has mainly resulted in the concepts of distributed firewalls and virtual security appliances for the cloud.

The concept of 'distributed firewall' has been proposed for virtualization environments, to integrate packet inspection and filtering in hypervisors, witnessing the importance of pervasive and capillary control. Distributed firewalls for cloud computing build on the concept of micro-segmentation [5], and deploy packet inspection rules in hypervisors, while keeping centralized control. They enable very fine-grained control over security policies, beyond mere IP-based structure [6]. For instance, vCloud Director 8.20 by VMware includes a distributed firewall, while OpenStack Neutron includes the Security Groups feature. A distributed firewall removes the need for traffic steering (all network packets go through the hypervisor, which is part of the firewall) and IP-based rule structures (through the notion of logical "containers" or "security groups").

Despite their common usage in cloud networking, distributed firewalls have some important limitations. First, this approach is currently effective for enforcing filtering rules, but does not have the flexibility to provide deep inspection capability tailored to the specific needs for detecting threats and on-going attacks. Second, they cannot provide the same guarantees of private enterprise networks: external resources lie in third-party infrastructures where trust mechanisms are still missing (i.e., the behaviour of physical hardware and networks cannot be controlled by cloud users). Third, their application in multi- and cross-cloud environments is not straightforward, since their configuration is based on internal communication mechanisms for each infrastructure. This issue will be even more severe in cyber-physical systems, with the integration of smart things in cloud applications, which are expected to be a consistent use case for 5G.

Given the reduced set of security features integrated in virtualization platforms and the increasing needs for cross-cloud deployments, users are generally left most of the burden for protecting their applications against external threats. Since, on first approximation, virtualization environments could be viewed as special instances of physical networks, software-based versions of security middleboxes (Intrusion Prevention/Detection Systems, Firewalls, Antivirus, Network Access Control, etc.) may be integrated in service graph design [7] [8]. We argue that this approach comes with important limitations in the current cyber-security landscape:

- *Performance*: security appliances are traditionally deployed as expensive and proprietary hardware modules. More and more vendors, such as Fortinet, Barracuda Networks, F5 Networks, and CheckPoint, are offering software versions of their security appliances, mostly for data centers and virtualized IT environments, which simplify deployment and re-configuration. However, although they can be deployed on commodity servers, virtualized editions of security appliances typically inherit the management interfaces of their hardware versions, thus prohibiting unified vendor-agnostic management via open APIs. Further, virtualized editions of security appliances do not benefit from hardware acceleration, and this may lead to inefficiency. As a matter of fact, more than 80% of all new malware and intrusion attempts are exploiting weaknesses in applications, as opposed to weaknesses in networking components and services, hence rules have evolved from memoryless simple string matching to stateful automata (such as regular expressions). Also, the increase in the complexity of protocols makes modelling their normal behaviour increasingly difficult; as a result, more computing cycles per packet are required either checking against more elaborate rules or trying to detect sophisticated anomalous behaviours. Performances fall quickly, especially in case of large volumetric attacks.
- *Context-awareness*: the nature and composition of multi-vector attacks requires pervasive monitoring and global view, and the deployment of Security Information Event and Management (SIEM) software for effective detection, which may be too cumbersome and ineffective for small applications and services.
- *Attack surface*: virtual security appliances are more exposed to attacks than their physical counterpart, since they run in the same virtualization environment to protect.
- *Propagation of vulnerabilities*: the growing trend to re-use the same software for multiple applications, often distributed as pre-package images, brings the risk of propagating software, architectural, and configuration vulnerabilities to many applications running in different infrastructures, which can become very dangerous botnets.

In a nutshell, virtual security appliances cannot exploit hardware acceleration, slow down virtual machines, and require additional software instances; eventually, they are seldom used, and the overall service usually results less secure and more prone to incidents than their physical deployments.

3.4 Challenges and emerging trends

Once the surrounding fence is no more able to stem the flow of external attacks, the need arises for new forms of internal techniques that could effectively tackle a larger base of threats and attacks than current solutions, correlate events in both time and space dimensions, feed novel disruptive approaches capable of estimating the risk in real-time, and carry out focused and effective defensive and mitigation actions.

Next generation frameworks for situational awareness are expected to combine fine-grained and precise information with efficient processing, elasticity with robustness, autonomy with interactivity. State of Art of commercial products and research efforts shows some general trends and specific challenges in this respect: the shift from centralized to distributed architectures, the programmability of the infrastructure, the chase to efficiency and performance, to need for robustness and data protection, dynamic adaptation to changing environments and conditions through orchestration, correlation of data in time and space, and suitable representation to humans. In the rest of this Section, we analyze each of these factors in detail.

In recent years, a great effort has been undertaken to increase the programmability of communication networks [9]. This allows fine-grained control over forwarding operation, relying on dumb network equipment and its logically centralized smart controller. Networks are indeed the pervasive infrastructure that connects all devices and smart things, hence they represent the ideal mean for capillary monitoring, inspection, and enforcement.

The combination of packet processing in network devices and computing hypervisors allows a great flexibility in where traffic is analysed and inspected, in both physical and virtual environments. This approach has already been used for many years, collecting flow statistic through protocols as SMTP, NetFlow, sFlow, IPFIX, and, more recently, OpenFlow [10]. Threat identification by exploiting network programmability has already been investigated by research papers [11]. The Defence4All plugin for the OpenDayLight SDN controller is the most popular platform in this context. It monitors packet counters available in OpenFlow devices to quickly detect anomalous and suspicious conditions and, in case, diverts the traffic towards an external scrubbing facility for in-depth analysis and mitigation. Defence4All only works for DoS attacks, and wastes network bandwidth when redirecting the traffic to an external appliance.

Unfortunately, most of existing approaches are essentially based on static, pre-defined, and inflexible filtering and detection rules. SDN controllers (e.g., OpenDayLight, Quake, NOX) require the definition of detailed instructions and/or programs from applications (by using internal APIs or descriptive languages as YANG [12]), and just translate them into OpenFlow or other protocol messages. The "intent framework" in ONOS goes in the direction of more automation towards real Network-as-a-Service (NaaS) [13], but it is still far from a complete and overall abstraction model (it currently only addresses connectivity).

Recently, the interest has started shifting from stateless to stateful operation in network switches, which provides far more programming flexibility and efficient processing in the data plane, while reducing the overhead in the control plane. This would eventually allow more advanced programming models, well beyond static forwarding rules and flow-level reporting available today, which include inspection and detection on flows and/or packets, aggregation and even storing capabilities. OpenState [14] delegates basic state update operations to network switches. This abstraction is rather powerful, but it only allows the switch to run simple Finite State Machines (FSMs), where transitions are limited to state changes, and does not include comparisons or complex computations that would be necessary for detection tasks that compare values against thresholds, which is currently being developed by the same authors [15]. The OpenState framework is already used for detecting DDoS attacks by StateSec [16]. In addition to OpenState, other technologies are being developed that process network packets and I/O events (e.g., FD.io, Snabb switch, IOVisor, XDP, BESS), which may be used to bridge software-defined networking with threat detection algorithms.

It is worth pointing out that enhanced programmability also brings more dynamicity in running detection and monitoring tasks. This means that lightweight processing could be used for normal operation, while reverting to deeper inspection at the early stage of any suspicious anomaly (or upon signalling from some knowledge-sharing framework), with clear benefits on the overall processing load. Further, a distributed and capillary architecture, with inspection capability in each network device and hypervisor, automatically addresses scalability, since the processing resources grow with the system size. This increases efficiency and boost better performance, especially when attacks are complex to detect.

Recent estimations say that user applications are the most attractive target for attacks (more than 80% of attempts). The increased security has led to the elaboration of more complex attacks, which eventually turns into more difficult detection. In addition, the ever-growing number and complexity of protocols and applications makes their traffic and behaviour increasingly difficult to understand and to model, which complicates the detection of anomalies. Accordingly, inspection is evolving from simple memory-less string matching to stateful rules (such as regular expressions). As immediate consequence, more processing power (hence CPU cycles) are required to check packets and instructions against more elaborated rules. Therefore in-line detection is likely to overwhelm software-based implementations of load balancers, firewalls, and intrusion prevention systems, especially in case of large volumetric attacks [17]. It is therefore necessary to consider extensions or (re-)designs that adopt hardware and in-kernel acceleration (e.g., GPU, Intel DPDK, FD.io, Snabb switch, IOVisor, XDP, BESS) to build fast data paths that process packets at nearly line speed.

Trustworthy of the processed information, events, and knowledge is of paramount importance, since an inappropriate response may be more damaging than the original attack. Any loss of integrity in the infrastructural components (i.e., hardware tampering) or control protocols may result in inaccurate, inappropriate, manipulated, or poisoned context information, which gives a forged situational awareness and eventually leads to ineffective, late, or even counterproductive reactions.

Encryption and integrity services are almost always available in control channels, as well as user authentication and access control, but no certification of origin and time of information is usually available. Authentication, authorization, and access control are already present in SDN controllers (e.g., OpenDayLight uses a token-based mechanism). However, trustworthy and integrity of the collected information are also fundamental requirements for maintaining historical evidence with legal validity, to be used for example in forensics investigations. At the same time, privacy issues must be tackled in order to not disclose any personal and sensible information without explicit consent, even to technical and management staff, apart in case of criminal investigation [18]. Specific challenges include collecting and conserving events and traffic patterns in a confidential way, anonymizing data before analyses, and making them accessible in clear form only in case of legally authorized investigation [19]. Cyber-security frameworks have to guarantee the origin and integrity of security events, as well as the integrity of their sources, to keep relevant information in safe, trusted, and secure storage, and to make data available without disclosing sensitive information [20]. Relevant mechanisms include timestamping, symmetric and public key cryptography, PKI infrastructures, anonymization and pseudonymization, digital signing, message integrity codes and hashing functions. One possible solution is the definition of security middleware, which acts as common substrate for all virtual and physical services. In addition, the possible evolution of homomorphic encryption [21] [22] may represent the ground-breaking factor to foster privacy-preserving computation schemes still inconceivable right now.

The progressive introduction of more programmable devices brings more flexibility and dynamicity in processing, but also requires a control plane that exposes device capability, and an orchestration plane that automates the process of on-the-fly building and deploying the configuration/code. In the transition from centralized to distributed architectures for cyber-security systems, it is indisputable that orchestration will play a crucial role in shaping the behaviour of the capillary programmable infrastructure, i.e., to delegate filtering and pre-processing tasks to programmable resources, including network switches, hypervisors, and smart things, with tight coordination with the deployment and life-time management of software. Through orchestration, the granularity, detail, and periodicity of collected information can be tuned dynamically according to specific needs (e.g., increase granularity in a specific area where anomalies have been detected).

The transition to more programmable infrastructures does not only increase their flexibility, but also widens the attack surface [23]. Malicious, wrong, or inaccurate code/configuration may be exploited for both passive and active attacks. In the context of software-defined networking, the importance of formal verification for preventing security violations and other unwanted behaviours of the network (such as forwarding loops) has been widely recognized in the scientific community. Several formal verification techniques have been developed targeting both SDN functionalities (most notably OpenFlow rules [24] [25]) and virtual graphs of network functions [26, 27, 28, 29]. A limitation of all these techniques is that they are not integrated into the orchestration process, but they act either before it (on the user-specified service graph) or as a post-processing step after orchestration. This is not the best solution. An early check, in fact, may miss security problems introduced afterwards, while with a later check, if errors are detected by the verifier, service deployment fails because the orchestrator does not have clues about how to fix the errors or the orchestrator has to iterate through the many possible solutions, which is clearly inefficient.

Advances are needed for the development of formal approaches that, while providing final assurance levels similar to the ones of the state-of-the-art formal verification techniques, are incorporated into the secure orchestration process, which in this way produces network configurations that, once deployed into the underlying infrastructure, are formally guaranteed to satisfy the required security policies. In fact, the problem of how to ensure the correctness of service orchestrators has already been recognized as a critical one in security-critical cloud computing environments, so that the idea of formally verifying orchestration procedures has been recently proposed (e.g., [30, 31, 32]), for verifying cloud-related policies (e.g., verify that payment procedures are properly designed). Further extensions are needed to address similar concerns about orchestrator correctness, but also considering the edge and fog environments.

The dynamicity, sophistication, and speed of attacks require autonomous response to provide timely and effective countermeasures. Sharing and correlating events and anomalies within the same and among different domains is also essential in order to (even proactively) anticipate any emerging or upcoming threat already (partially) detected somewhere. In-deep analytics are required to detect and identify threats from elementary and apparently uncorrelated events. Some tools are already available to this purpose (e.g., the ECOSSIAN platform and the Caesair model [33]).

Pervasive and fine-grained monitoring of ICT installations will produce an impressive amount of data, even if programmability is used to tune the deep of inspection according to the actual need. Big data and machine learning capabilities are required to extract relevant knowledge from the cluttered flow of information, by correlating data from pervasive data sources. The challenge is to add predictive and proactive capabilities to existing security tools and systems, in order to prevent attacks by analyzing the environment, rather than merely react in case of compromise.

The whole process can therefore be split into three tasks:

- Collect and aggregate data from a multiplicity of sources, including all relevant inputs (programmability).
- Correlate inputs in space and time dimension, even in different administrative domains (correlation), in order to promptly detect, classify, and predict multi-vector and interdisciplinary cyber-attacks. The challenge is the real-time elaboration of massive events from a huge number of sources, while maintaining several properties such as scalability, autonomy, usability, fault tolerance, and responsiveness.
- Build the global security assessment of the overall system, including identification of threats, attacks, vulnerabilities (evaluation).

Existing algorithms already make use of flow-level information for network volume anomaly detection [34], though this only represents the crumbs of what may be available tomorrow. New algorithms for vulnerability analysis and threat detection may be based on the ideas of the Attack Graphs [35], Attack Surface analysis [36], Kill Chain definitions [37] and Attack trees models [38] with the support of the deep learning techniques, Petri nets [39], and game theory models [40]. Correlation should also include automatic selection of the algorithms for the analysis of the threats based on the threat potential negative impact, both environment-dependent and environment-independent.

Once the proper knowledge has been built by correlating and understanding data and events, it must be used in the most appropriate manner for reaction and mitigation. Situation awareness must be represented to security staff, must feed smart reaction tools (including service orchestrators), and must be shared in order to boost synergic and coordinated response.

Next-generation visualization tools and interfaces shall include concrete models and specifications for an innovative approach in the field of cyber-security, being able to capture novel aspects entailed by virtualization paradigms (including fog architectures and the IoT), also bearing in mind the need for distributed, pervasive and capillary monitoring techniques. This means that situational awareness shall be related to the system topology and composition through proper data visualization, with clear indication of the main vulnerabilities, flaws, threats, and their position [31].

Existing tools for visualization should be improved with real-time cross-domain information, in order to reach a better response time and to reach the scope of situational awareness. Visualizations must be able to provide enough information to prepare effective response strategies but shall also avoid to reveal sensitive information about other domains (e.g., vulnerabilities, lack of proper defense tools, undergoing attacks, etc.), as well as include proper identification and verification of credentials of authorized users, to avoid propagating useful information to attackers.

The high degree of interconnectedness of communication infrastructures in practice exposes every information system to the same threats. On the other hand, the growing complexity and organization of cyber-attacks, which are often carried out simultaneously against different targets, are drastically reducing the time to learn new threats and to disseminate relevant information. Collective sharing of data, events, and relevant security information looks the only viable way to build large-scale situational awareness and to protect critical infrastructures. Both technical (e.g., data semantics and communication interfaces) and organizational aspects (e.g., privacy, ownership, confidentiality) should be considered in the design of effective security information sharing platforms [41].

3.5 Towards integrated and pervasive situational awareness

New architectures and usage models, which leverage virtualization paradigms and the Internet of Things (IoT), are now revealing the substantial inadequacy of legacy security appliances to effectively protect distributed and heterogeneous systems (including cloud, edge, and fog installations) against cyber-threats. As a matter of fact, the prevalent paradigm in enterprise security is still the "security perimeter" model, which assumes safe isolation of ICT assets by physical or virtual network segmentation, hence concentrating protection at the perimeter only. Running virtual machines in public cloud/edge installations, as well as integration with third party's devices and smart things, blur the boundary between public zones and private domains, hence making hard to apply the security perimeter model in a trustworthy and effective way. Since valuable ICT assets cannot be easily enclosed within a trusted physical sandbox any more, there is an increasing need for a new generation of pervasive and capillary cyber-security paradigms over distributed, multi-domain, and geographically-scattered systems.

The predominant interspersion of lonely valuable resources with unsafe computing and communication infrastructures makes the application of the security perimeter at each site ineffective, because of the overhead to run complex agents in end devices, especially in case of resource-constrained "things". In addition, the growing complexity of cyber-attacks, often based on multi-vector approaches, are urgently demanding more correlation in space and time of (apparently) independent events and logs, and more coordination among different security applications.

We argue that, in relation to network threats, most of the rigidity of current security paradigms comes from two main factors: i) the need for physical isolation of enterprise's assets from the outside world, and ii) the presence of multiple standalone appliances placed at exchange points, each dealing with specific security aspects (e.g., firewalling, intrusion detection/prevention, virtual private networking, antivirus, deep packet inspection), as pictorially depicted in Figure 1. Because of this typical fragmentation, each appliance has only a partial view of the whole context, and enforcement of security policies may also be limited in effectiveness.

To effectively tackle multi-vector attacks, a broad range of data from heterogeneous sources should be collected, fused, and processed with fine granularity. The likelihood of detection increases with the deep of knowledge, so raw data would be better than distilled knowledge, but management of large amounts of information may be overwhelming. In this respect, the evolution of the legacy cyber-security paradigms towards more integrated and collaborative frameworks is desirable, where a common and pervasive substrate feeds several detection algorithms in a fine-grained programmable way. At the conceptual level, the most disruptive innovation should come by going beyond the traditional "verticalization", where multiple discrete appliances cope with specific security aspects (e.g., firewalling, intrusion detection/prevention, anomaly detection), in favor of horizontally-layered architectures, which decouple distributed context monitoring from (logically) centralized detection logic, as shown in Figure 1. This visionary perspective somehow aligns to the same evolutionary path already undertaken by software-defined networking. Such evolution would be properly addressed by a multi-tier architecture that decouples a pervasive and shared context fabric, where the environment is monitored and security actions may be enforced in a capillary way, from centralized business logic, where detection and mitigation algorithms are implemented and leverage big data and other advanced techniques. In addition, a presentation layer facilitates the interaction with users and other security systems.

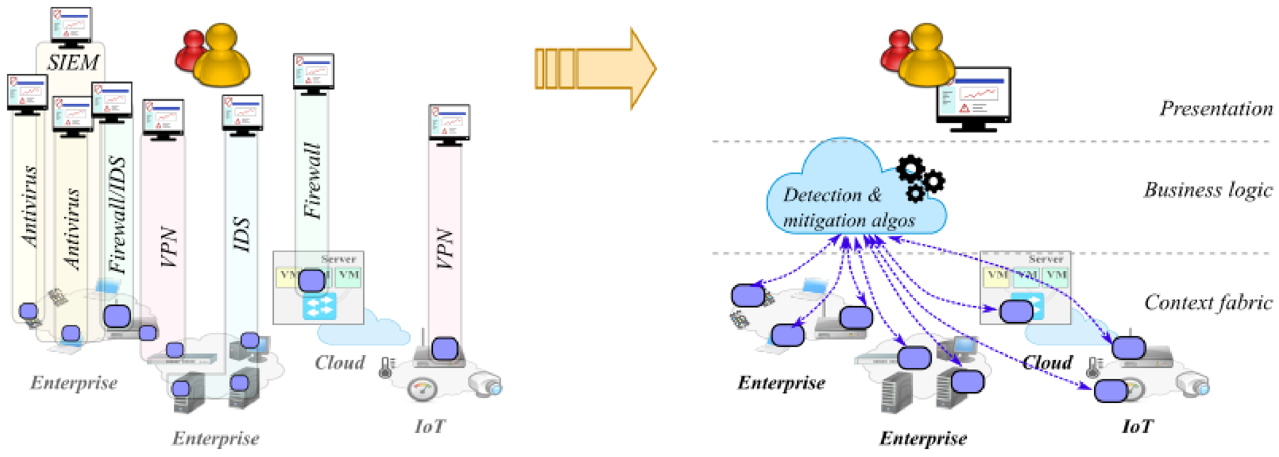


Figure 1. The complexity and multi-vector nature of recent cyber-security threats require a transition from current narrow-scope silos to a more integrated multi-vendor layered and open framework.

A more technical conceptual view of an innovative cybersecurity framework is represented in Figure 2. It shows specific operations and information present at the three layers identified above; the left side concerns data collection and fusion to build wide situational awareness through identification of cyber threats and attacks, while the right side shows the translation of remediation strategies and countermeasures into proper local configurations.

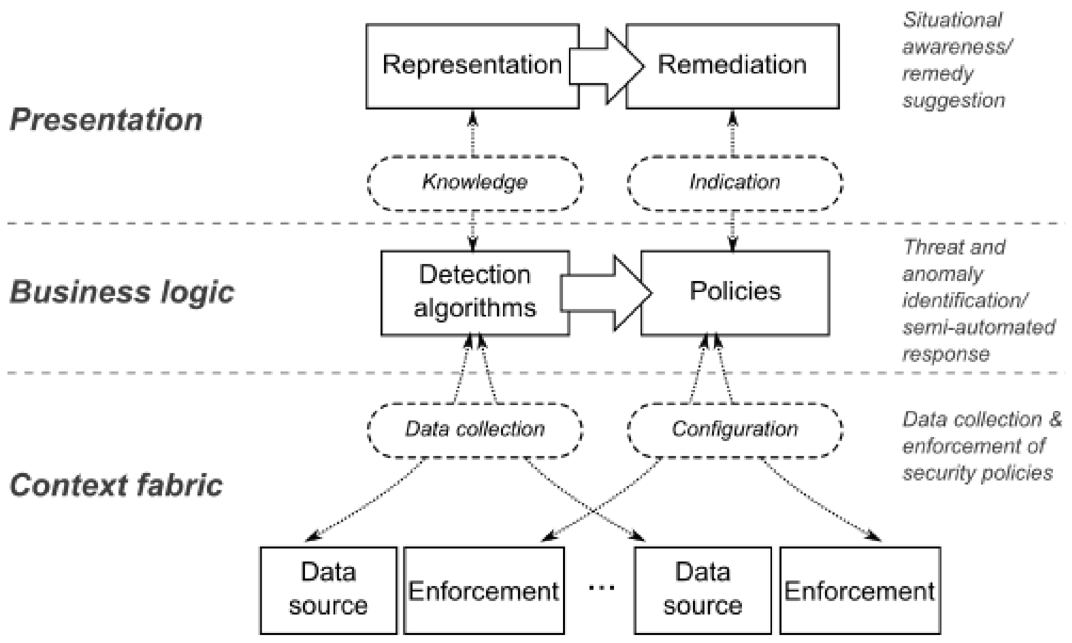


Figure 2. Conceptual information workflow for a distributed cyber-security framework.

3.5.1 Context and enforcement

Efficiency and interoperability build on a common context “fabric” that provides a uniform substrate to collect security events, data, measurements and logs from heterogeneous sources, scattered over a mix of enterprise, cloud, edge, and fog installations.

A single and unified layer for data collection results in far more efficiency, by avoiding duplication of the same analysis and inspection tasks for different applications. Conceptually, the context fabric might seem the same concept as Security Information and Event Management (SIEM), which collects events and logs for centralized analysis and correlation. However, as the same name implies, the context fabric entails a capillary and thick monitoring texture for detecting and collecting security-related information, encompassing network measurements, system calls, daemon and application logs, and security events from heterogeneous sources in (maybe virtual) networking and computing devices, while tuning the detail level according to the current situation and risk.

It exposes advanced programming interfaces, well beyond the flow-level reporting already available today for anomaly detection (e.g., NetFlow, sFlow, IPFIX), to configure the type and depth of inspection, pre-processes information, and provides ‘refined context’ instead of raw data that might flood the network and the detection algorithms. OpenFlow [42] and NetConf [43] interfaces are already available both in open-source (e.g., Open vSwitch) and commercial network devices; though supported operations are just limited to filtering and statistical reporting, the interest in stateful processing [14] [16] is paving the road for a richer and more flexible set of processing capabilities, which promises to push much more intelligence to network devices. Other kinds of interfaces would be required to extend the framework to behavioural analysis, access control, and other relevant tasks for a more general cyber-security framework.

Consequently, the context fabric entails a rich set of traffic filtering, packet and behaviour inspection, processing, aggregating, and, likely, storage functions that are delegated to specific domains for performance and privacy matters. Such functions will no more rely on dedicated hardware appliances or virtual software functions; rather, the challenge is to build on the growing availability of flexible and programmable data planes in network devices, operating systems, and hypervisors. The context fabric may be present in network devices, in hypervisors, in a virtualization containers, or directly into the networking stack of an operating system, so to cope the different virtualization options and deployment scenarios. Hardware and software acceleration for fast packet processing is highly desirable to create fast paths inside switching and routing devices, virtual functions, hypervisors. To this purpose, the implementation of the programmable agent may build on technologies like Intel DPDK, FD.io, Snabb switch, IOVisor, BESS.

Available computing and networking programmable infrastructures often provide both inspection and enforcement capabilities, so that a reduced set of technologies must be deployed to implement fully reactive systems. Enforcement must include packet filtering and redirection but should also cope with typical orchestration functions like starting, stopping, replacing, or migrating virtual functions so to remediate to security breaches and violations.

3.5.2 Detection and policies

Above the shared “context”, business logic includes different algorithms that implement different security functions: identification and prevention of attacks (intrusion, DoS, eavesdropping, replication, etc.), identification of vulnerabilities, new threats, and anomalies. This gives better opportunity to merge and correlate data from different domains, as well as analysis from different applications. A business logic layer allows diversification of security services as well as vendors, under a shared and open framework that avoid technological and commercial lock-ins.

From a conceptual point of view, virtual services constitute a hackable network of services; thus, a continuous internal audit of their security is required. The purpose is to improve detection (and even prediction) of the most complex multi-vector and interdisciplinary cyber-attacks. In this respect, the challenge is the definition of innovative algorithms that define which metrics are needed for each monitored point and correlate them in both time and space dimensions. This represents a substantial improvement over existing detection and prevention algorithms, which currently only work with a limited set of information [34] [44]. Detection should also support effectively signature- and rule-based detection similarly to existing IPS/IDS tools [44, 45]; both static and dynamic analysis of code are required to protect virtual services during their lifecycle. Different techniques may be used for these purposes: regression analysis, predictive and prescriptive analytics, data mining and machine learning. Obviously, a larger base of data and events would increase the processing burden, but this should not be a problem, since the control plane is outside the service graph and could run in dedicated infrastructures with big data techniques. The definition of advanced algorithms will eventually result in more computation complexity than today but will also add predictive and proactive capabilities to existing security tools and systems. The increased complexity will require big data and machine learning capabilities to effectively extract knowledge in nearly real-time. New algorithms must therefore be properly designed to fit parallel and elastic computation provided by such paradigms, in order to be effectively orchestrated by high-level components.

The decoupling between the context and the detection algorithms represents a major difference with respect to current practice and requires common models to gather data from heterogeneous sources; data harmonization is necessary to provide common formats and syntax for data coming from different domains and (possible) different controllers. In addition, the preliminary challenge is to understand which tasks should be offloaded locally and which tasks must be performed centrally. In general, the target should be to run detection algorithms on high-performance, reliable, and protected infrastructures (e.g., private cloud installations), while offloading monitoring, inspection, and filtering tasks to local resources in the cloud, the edge, and the fog. We remark that detection algorithms must be in part re-engineered to fit the distributed structure and deployment model of the specific orchestrator. However, we think that this effort could only be undertaken after the main framework has been outlined, and the design requirements are more clearly defined.

For a large number of well-known attacks (e.g., DoS, port scan), mitigation consists in standard actions, so response may be easily automated by a set of security policies. Security policies may be expressed in terms of very simple ‘if-then-else’ clauses, making the execution of specific actions (e.g., traffic shaping, traffic redirection, etc.) contingent upon occurrence of specific events (e.g., measurements over given threshold).

3.5.3 Awareness and reaction

Finally, the presentation layer concerns the representation and usage of situational awareness built by underlying security applications. The human interface is the interactive tool to draw the current cyber-security picture and to enable quick and intuitive response to attacks. It provides intuitive and easily understandable situational awareness to effectively support the decision process, by proper representation of the risk of possible attacks and the identification of threats and weaknesses (also including origin, positioning, dangerousness, replicability, etc.), and by enabling definition of custom reaction strategies in case of new and unknown threats.

Specific challenges include data and method visualization (e.g., to pinpoint the actual position of attacks and threats in the network topology, to point out the possible correlation between events in different domains), and decision support (e.g., to suggest remediation and countermeasures, to define automatic response to well-known attacks). Also, the presentation layer should provide seamless integration with CERT networks to share information about new threats and attacks among different administrative domains (e.g., with STIX), in order to facilitate continuous update of the attack data base and the elaboration of common reaction and mitigation strategies [41]. Integration with existing risk assessment and management tools is also desired, so to automate most procedures that are currently still carried out manually. This will ultimately speed up the sharing and learning process, reducing reaction times and improving the overall resistance and resilience.

Solutions may rely on multi-layer software architectures and REST-based APIs for accessing threats and attacks databases by multiple devices, flexible graphical layout definition by templates and stylesheets to adapt the representation to heterogeneous devices and platforms, event-driven publish/subscription mechanisms for real-time notification of threats, anomalies, attacks.

Cymerius, a visualization platform from Airbus, has already demonstrated the power and usefulness of threat visualization [33]. However, further research and innovation is required to manage complex multi-vector and multi-domain attacks and to integrate with national and international cyber-security response frameworks.

Presentation entails interaction with humans, to trigger manual reaction in case of complex attacks, or just to set security policies to automatically react to simpler and well-known ones. Semi-automated response is another option today, leveraging orchestration tools that manage life-cycle operations for complex systems and business chains, especially for virtual services.

Orchestration has been a hot topic in cloud/NFV environments for many years, hence different solutions and technologies are already available [46, 47, 48, 49] They mainly differ in the application models (which may be model-driven or data-driven) and management paradigms (which can be centralized or based on local managers), mostly depending on the optimization target and the specific environment (cloud, edge, NFV). A specific task for the orchestrator is automation and abstraction of the underlying configuration; in this respect, an “intent framework” would translate high-level description of monitoring and analysis information into specific instructions; this can be viewed as a sort of Monitoring/Inspection-as-a-Service. A policy framework represents the simplest and most immediate implementation, already available in many architectures [49, 50].

Finally, the user interface shall include tight access control to information to avoid attackers to gain visibility over security breaches and vulnerabilities.

3.5.4 Forensics and legal validity

Even the most reliable system may occasionally be compromised; in this case, it is important to investigate the cause to identify additional protection measures. In this respect, a critical issue is the legal validity of the extracted data to prosecute attackers. Common challenges in this area include: i) storing trusted evidence, ii) respecting the privacy of users when acquiring and managing evidence, iii) preserving the chain of custody of the evidence. We remark that in the proposed framework the problem is not the same as the definition of Cloud forensics [50, 51], since investigation in our case is carried out by the service owner and not by the cloud provider.

A certification process should be responsible for origin, timestamping, digital signing, integrity of relevant information that is used for security audits and legal interception; the solution should be able to capture enough information to trace security attacks in a reliable manner and to interpret the data post-factum. A legal repository should be responsible for secure and trusted storage of data, information, and events (security audit trails) for successive off-line analysis, cyber-crime investigation, and evidence in court. Key features in this case is trustworthiness, availability, integrity, resilience, resistance to attacks, and scalability, in order to prevent alteration or losses of data in case of attack.

The repository should be based on storage solutions specifically designed and implemented to comply with requirements for lawful applications. We believe existing virtual file systems for distributed storage of information (e.g., Ceph or Hadoop), which split information among different storage nodes, may be able to achieve the required reliability, security, availability, and scalability features.

3.6 *The ASTRID concept*

Taking into consideration the main technological gaps and current evolutionary trends, the **main objective** for the ASTRID project is ***i*)** to provide better awareness about cyber-security threats of virtualised services, referred to each single component (i.e., each specific application) as well as the service as a whole (i.e., the entire service graph), and ***ii*)** to facilitate (possibly automate) the detection and reaction to sophisticated cyber-attacks. Specific challenge will be the ability to detect vulnerabilities, threats and attacks not only from the canonical input/output channel of the services, but also internally to the service.

ASTRID explicitly addresses the following security concerns that are often underestimated in current approaches [52]:

- placement and configuration of the security appliances become part of service graph design, so often dealt with by people with no specific skills and experience in cybersecurity aspects;
- security and isolation of the internal (virtual) network rely on third-party segmentation mechanisms, which means that potential vulnerabilities, breaches, and threats of the virtualised resources (including software containers and network links) are not visible to owners of the virtual services;
- security appliances may increase the attack surface: attacks can target any functions of the service, including NAT, firewalls, IPS/IDS, hence vanishing the protection and leading to a misleading perception of security;
- security appliances as antivirus and intrusion detection must be replicated in each virtual function of the service graph, hence yielding excessive overhead and computing requirements;
- legal investigation is usually difficult, because there are no standard mechanisms to inspect exchanged traffic and monitors events and logs.

Riding the wave of the cloud paradigm, a major trend already identified in Section 3.4 is the transition from infrastructure-centric (Figure 4.a) to service-centric frameworks (Figure 4.b), which gives service providers better situational awareness about their deployed services. Indeed, virtual security functions can be easily "plugged" into service graphs when the Infrastructure-as-a-Service model is used, leveraging the large correspondence with physical infrastructures that is present in this case. Software orchestration facilitates the automatic deployment of security appliances, but their placement in the service graph is usually defined at design time by people that might not have the right security skills. In addition, application to other cloud models is not straightforward, especially when some software components are shared among multiple tenants (i.e., Service-as-a-Service), they should run on resource-constrained devices (i.e., the fog or IoT), or the service topology changes over time (e.g., for scaling, discovery of new components, failures). Given the lack of standard APIs and control interfaces, data collection, harmonization, and correlation may be very difficult. ASTRID aims at further improving this evolution; the **main concept** is the disaggregation of security appliances into a set of programmable inspection elements that feed a (logically) centralized detection logic (Figure 4.c).

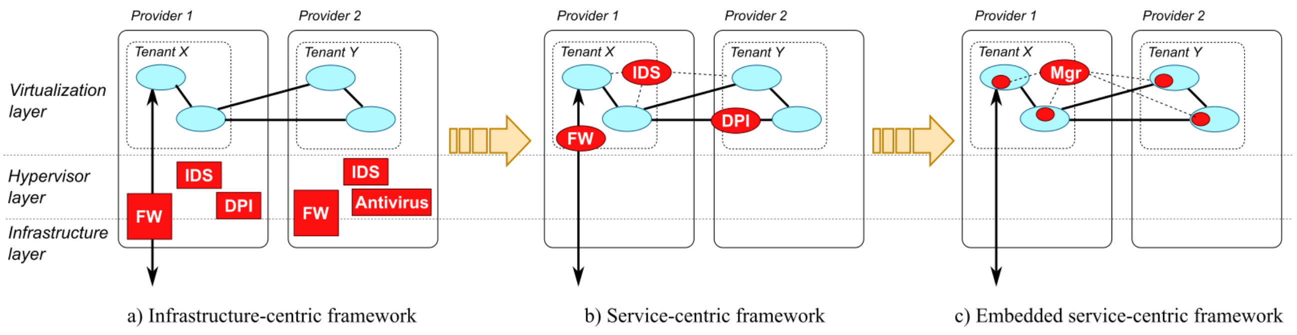


Figure 3. The ASTRID concept pursues a transition from infrastructure-centric to service-centric cybersecurity frameworks.

The general approach of ASTRID is tight integration of security aspects in the orchestration process. The **distinctive characteristic** of ASTRID with respect to other proposals is that no explicit additional instances of security appliances are added to the service graph. Instead of overloading the graph with complex and sophisticated threat detection capabilities, ASTRID will only perform lightweight monitoring and inspection tasks in service graphs and their execution environments, which feed detection algorithms placed outside the graph design, as part of a powerful and overarching awareness logic, as pictorially shown in Figure 4. Hence, orchestration is no more responsible to deploy security appliances (Figure 4.a), rather to programmatically deploy and configure a set of distributed agents and/or hooks with inspection and enforcement capabilities, and to connect them with a (logically) centralized detection logic (Figure 4.b).

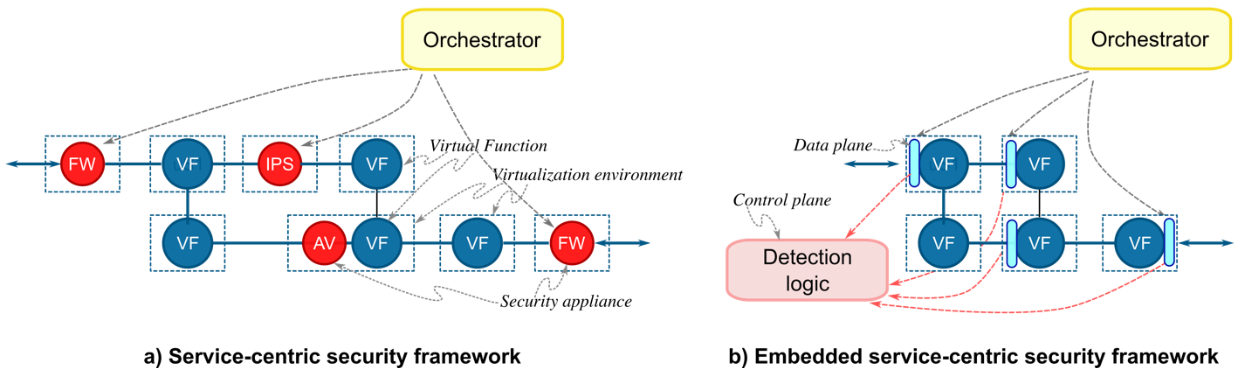


Figure 4. ASTRID concept.

The main enabler for ASTRID is **programmability**, i.e., the capability to change not only inspection parameters (as IP addresses, TCP/UDP ports, application logs) but also the same processes of inspection and aggregation. That means the ASTRID framework will not be constrained by design to a specific set of protocols but will be able to inspect new headers and protocols as defined by the detection logic. The ambition is therefore a very generic inspection and monitoring framework, able to collect heterogeneous security data from multiple sources.

A conceptual representation of ASTRID is shown in Figure 5. The “Security Model” provides a common abstraction for the underlying programmable hooks. It uses specific semantics to describe programming models (e.g., programming languages, availability of compilers, interpreters, or virtual machines) and security-related capabilities (e.g., logging, event reporting, filtering, deep packet inspection, system call interception), as well as provides a common interface for their configuration.

Policies describe in an abstract form various life-cycle management actions; for instance, types of events that should be collected, anomalies that should be reported, actions that should be undertaken upon detection of potential attacks, etc. Policies may be encoded in high-level descriptive languages (e.g., XML, JSON) for requesting specific orchestration services (e.g., setting a packet filter for a given traffic flow, replacing a buggy or

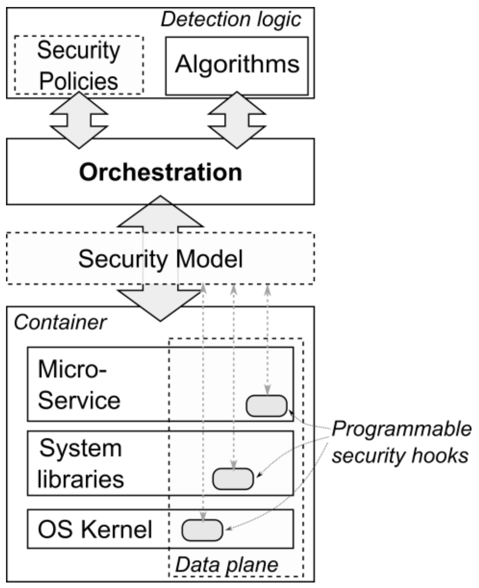


Figure 5. ASTRID multi-tier architecture.

misbehaving function, triggering packet or software inspection). They are agnostic of the underlying data planes, so that they can be used with different (even heterogeneous) programming technologies.

Relying on a common and homogeneous substrate for inspection and monitoring will facilitate the correlation of security data in both the time and space dimension. Algorithms can therefore consist of typical functions (i.e., firewall, IDS/IPS, antivirus, Application-Level Gateway), but can also realize new forms of detection based on correlation of data from different subsystems (disk, network, memory, I/O), leveraging machine learning and other forms of artificial intelligence.

Clearly, the disruptive ASTRID concept does not only bring important benefits, but also poses new **technical and procedural challenges**. As a matter of fact, the need to collect data may overwhelm the network; in addition, the confidentiality and integrity of such information is of paramount importance, especially to support legal and forensics investigation. In this respect, the ASTRID concept entails a more complex framework than the general architecture outlined so far. At the proposal stage, three main macroblocks and a number of logical functions were already identified, as shown in Figure 6.

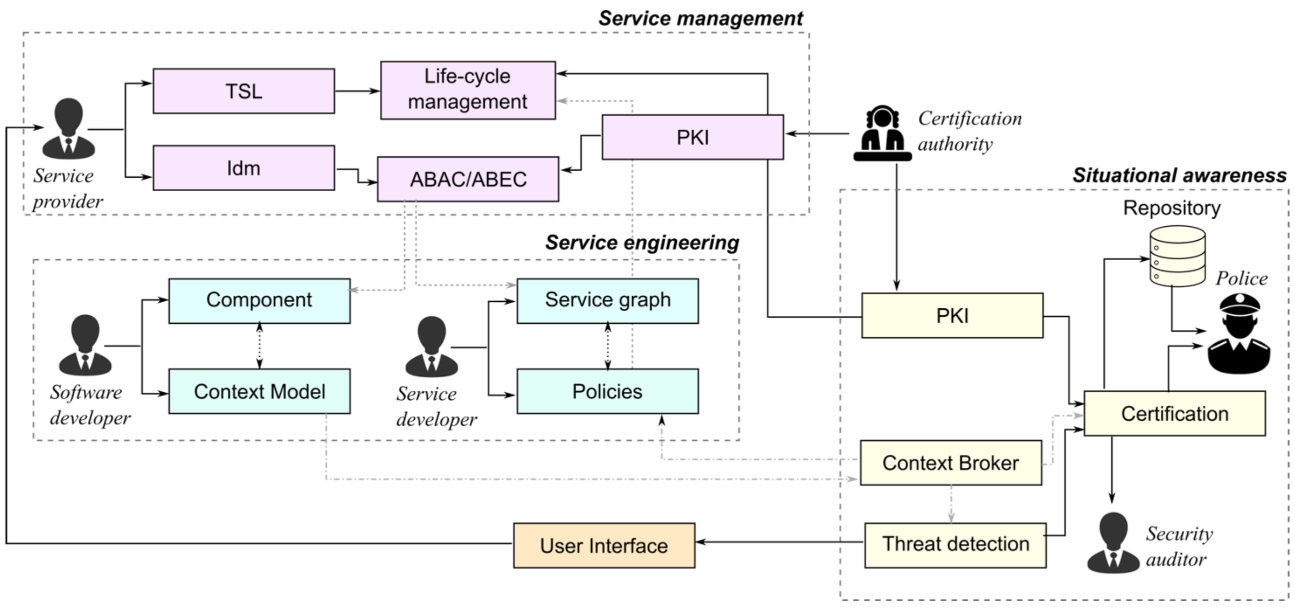


Figure 6. ASTRID framework.

The three macroblocks are [53]:

- *service engineering*, concerning the development and abstraction layers for both software components (micro-services, virtual functions) and service graphs;
- *service management*, dealing with secure deployment and life-cycle management of service graphs;
- *situational awareness*, responsible for detecting threats and certifying data for security audits and court investigations.

Service engineering is based on the usage of a *Context Model*. Indeed, a service graph is a high-level description of the service, which must then be translated in a proper configuration of the underlying virtualization environment and operating conditions. The Context Model is just an abstract representation of the possible environments and conditions, including information about dependencies among components (e.g., web application that needs a database for storage), requirements on computing/networking/storage resources (e.g., number of CPUs, RAM, bandwidth), underloading or overloading conditions, failure conditions, etc. Orchestration makes use of this information for deployment and lifecycle management, hence adapting the service and its components to the changing context. These processes are usually based on the execution of specific hooks for management of single components and the whole graph. Hooks are often provided as scripts, to install and configure the system during deployment, to start/stop/restart the service, to clone the service, to scale the service, to react to failures, to de-provision the service and so on. The definition of behavioural rules for orchestration is typically done by *Policies*, which are described by a condition/action pattern. Policies are specified separately to the system implementation to allow them to be easily modified without altering the system. ASTRID makes use of policies to shape the system behaviour to the evolving context.

Secure management of service graphs entails both authentication and encrypted channels for interacting with the service components. Thus, access to the security context is mediated by ABAC (attribute Based Access Control) and ABEC (attribute Based Encryption Control). It also contains an IdM (Identity Management) component and a PKI infrastructure (rooted at a trusted and public Certification Authority). Secure deployment also entails selection of trusted services (i.e., which have been previously verified by static code analyses and certified to be safe), hence a TSL (Trusted Service List) component is present.

Situational awareness includes all the components to collect, distribute, and process knowledge from the individual components of the service graph. The Context Broker is responsible for collecting context information by a Pub/Sub paradigm; it feeds other engines that take decision based on the current security context. Threat detection process events, data, logs from individual components and identifies threats and attacks. It makes use of Complex Event Processing for carrying out detection tasks (rule-based, big-data, machine learning). Output from the threat detection logic is distributed to several processes:

- the *user interface*, to provide proper visual representation of the current situation to the service provider, and to assist him in taking decision for remediation actions and countermeasures;
- the *certification* process, which is responsible for origin, timestamping, digital signing, integrity of relevant information that is used for security audits and legal interception; the ASTRID solution is able to capture enough information to be able to trace security attacks in a reliable manner and to interpret the data post-factum;
- the *secure repository*, which conserves data with legal validity (security audit trails) for forensics investigation that is initiated after the threat or attack has been identified.

For what concerns threat detection, the target is protection from both software vulnerabilities and network threats, hence involving a mix of source and run-time code analysis, formal verification, network analytics, and packet filtering techniques Specific issues to be tackled:

- Guarantee that the software does exactly what is supposed to do.
- Guarantee that the software has not been tampered, either voluntarily or through a successful attack.
- Guarantee that traffic streams flowing in, out, and across the service are (*i*) clean, (*ii*) do not contain anomalies or suspicious patterns, and (*iii*) are forwarded according the desired security policies.

3.7 Application scenarios

Beyond the mere concept, it is necessary to identify tangible use cases for the ASTRID technology. In this Section, some envisioned scenarios are identified and described, which represent potential business opportunities for the ASTRID Consortium. In this respect, they will be used to identify the set of technical, functional, and procedural requirements for the design and implementation of the ASTRID technology. Most of the described scenarios will then be demonstrated in the two ASTRID Use Cases, as described in the following D4.1.

A common template is used for all application scenario, which allows a systematic presentation. An initial brief description includes the following elements:

- **Name**: short name of the referenced scenario.
- **Objective**: the main objective that should be achieved in the referenced scenario.
- **Scenario**: the description of the specific problem to be solved, including main motivations (technical and/or legal) and challenges.
- **Business cases**: envisioned market segments for commercial exploitation. The description is deliberately limited to a very short reference, because the topic will be elaborated in more details in D5.5/5.8.
- **ASTRID stakeholders**: The Consortium partners that are mostly interested in the referenced scenario, either for scientific or commercial exploitation.

In the operation section, two approaches to solve the problem are described:

- **Current practice**: a short description of available tools and methodologies that are already used in the referenced scenario, together with their weakness and limitations.
- **ASTRID practice**: an indicative description of how the ASTRID framework would be used to solve the same problem, with explicit indication of how high-level configuration and policies will be translated in low-level operations.

Finally, main **innovation and benefits** brought by the ASTRID approach are briefly summarized; an indication is also given about the main technical/procedural **challenges** to be solved by the Project.

All scenarios are characterized by an intrinsic rigidity of current practice in the configuration of security properties, as well as the need for manual operations and high-skilled personnel. ASTRID improves the usability, reliability, and effectiveness of security operation, by bringing more automation and leveraging programmability of the underlying infrastructure.

3.7.1 Situational awareness

Description	
Name	Situational awareness for virtualized services.
Objective	Detection of intrusions and advanced persistent threats through correlation of heterogeneous security context.
Scenario	A cloud application or an NFV service is designed as service graph and ready to be deployed by an orchestrator. Virtualization infrastructures partially change the typical threat models of physical installations. Indeed, tampering is a minor issue in this case, but de-coupling software from the underlying shared infrastructure raises new security concerns about the mutual trustworthiness and the potential threats between those two layers. Cloud/NFV applications are vulnerable to network attacks from the Internet (intrusion, eavesdropping, redirection, DoS, …). These vulnerabilities can be largely mitigated by well-known security appliances, but the latter definitively increase the attack surface, as demonstrated by the number of vulnerabilities reported to NIST for security applications from main vendors since 2016 [54]. Virtual services cannot take full trustworthiness of internal resources for granted: indeed, the cloud paradigm raises many security concerns due to the hypervisor layer, outsourcing, and multi-tenancy [1] . As a matter of fact, the attack surface is increased by the larger number of components: guest environments (virtual machines), host operating systems (servers), hypervisors, management interface, shared storage and networks. Sharing a common infrastructure has the unpleasant side effect that an

	attack to the infrastructure affects many services and tenants (e.g., DoS on shared physical networks). Tenant isolation should provide independent and secure execution sandboxes, leveraging technologies as hypervisors, network virtualization, and virtual storage. However, the shared infrastructure widens the class of local adversaries, also including other tenants and the infrastructure providers, raising new attack models (grey boxes, involving tenants and the cloud providers) in addition to mainstream white (employees) and black boxes (external attackers). Timely detection of attacks is today more difficult than ever. More than 80% of all new malware and intrusion attempts are exploiting weaknesses in applications, as opposed to weaknesses in networking components and services, hence rules have evolved from memoryless simple string matching to stateful automata (such as regular expressions). Also, the increase in the complexity of protocols makes modelling their normal behaviour increasingly difficult; as a result, more computing cycles per packet are required for either checking against more elaborate rules or trying to detect sophisticated anomalous behaviours. As a matter of fact, basic in-line DDoS detection capabilities of network devices such as load balancers, firewalls or intrusion prevention systems may have once provided acceptable detection when attacks were smaller but complex volumetric attacks, leveraging on the weak security posture and proliferation of IoT devices, can easily overwhelm these devices since they utilize memory-intensive stateful examination methods [55]. Stealth attacks are now becoming the new norm to circumvent legacy security appliances. Sophisticated, targeted, and persistent cyber-attacks, collectively indicated as Advanced Persistent Threats (APT) [56] are mostly complex and generally involve multiple stages that span over a long period of time. Hence, they occur along both spatial and temporal dimensions. **Spatial dimension**. Typical APT begin by scanning the target system to make an inventory of public resources and identify possible attack vectors (web sites, emails, DNS, etc.). A number of complementary actions are then initiated to gain access, including social engineering and phishing, internal port scanning, malware injections, and so on. All these actions target different subsystems (e.g., public web servers, DNS, users, internal networks, hosts); when taken alone, they might be confused with legal operations by standalone security appliances. **Temporal dimension**. The execution of the different stages may take days, weeks, or even months. For example, fraudulent emails might be sent for days or weeks before a rash or careless user opens the embedded links. Again, an installed malware might be left snooping for days before starting to collect data. It is therefore challenging for any standalone security appliances to store long historical traces, and to correlate events that may have happened weeks or months ago.
Business cases	• Situational awareness for cloud services. • Situational awareness for NFV services and Service Function Chains. • Situational awareness for cyber-physical systems. • Integration with existing risk management tools.
ASTRID stakeholders	UBITECH, SURREY, AGE, POLITO, CNIT, TUB, DTU
Operation	
Current practice	Most cloud management software only provides firewall services, while other security services must be implemented by software versions of legacy security appliances (Intrusion Prevention/Detection Systems, Firewalls, Antivirus, Network Access Control, etc.). Many vendors of security appliances ship integrated solutions for cloud

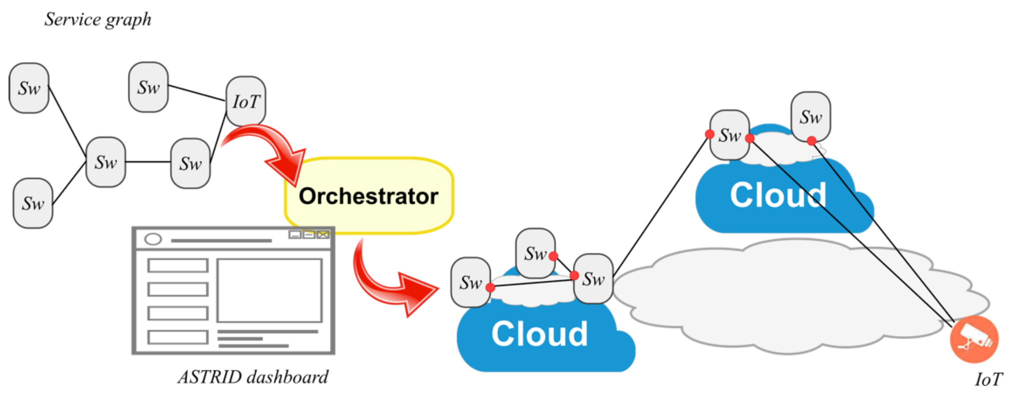
	protection; they often consist of a centralized management and analysis dashboard and many agents that implement conventional security services. These solutions are usually complex to deploy and lead to vendor lock-in. Leveraging novel software orchestration paradigms, security appliances can be integrated in service graph design. This approach usually brings a large overhead on service graph execution: virtual security appliances cannot exploit hardware acceleration, require more resources (CPU, memory) and slow down virtual machines, deploy additional software instances (which increases the cost), and require tight integration between service designers and security staff (which is not always simple to achieve in current development processes). Eventually, they are seldom used and the overall service usually results less secure and more prone to incidents than their physical deployments [44]. Additionally, these components are built in the classic fashion, protecting the system only against outside attacks, while most of the cloud attacks are now done through the compromising of the network functions themselves using tenant networks running in parallel with the compromised system.
ASTRID practice	Service developers design their service graphs without caring about security appliances. Indeed, they are only required to enable the ASTRID framework and provide high-level indication of what security services are required. Once the ASTRID framework is enabled, the service graph is enriched with the necessary functions and hooks, and the graph is shared with the security staff. Security is largely automated by the orchestration process, which adds all required functionalities and performs configurations before deployment or as part of life-cycle management operations. The ASTRID dashboard provides an easy and intuitive mean to select security features from a list (which may include intrusion detection, APT detection, DoS mitigation, DNS attacks, firewalling). Each detection service uses a default detection algorithm, but additional properties may be present to select among alternative algorithms. Though it is possible to select all available detection features, the more algorithms are run, the more data are likely to be generated and the more overhead on service execution. The right set of features has therefore to be decided by the security manager according to the service criticality and a risk/cost analysis. Anyway, adding/removing features is a very simple and intuitive operation that can be carried out at runtime and does not require manual reconfiguration or re-deployment of the whole graph. Before applying the new configuration, formal verification techniques are applied to ensure that the run-time service graph satisfies the required security policies. During service operation, any relevant security event is reported by the ASTRID dashboard to the security manager. Detected attacks are graphically pinpointed on the service graph. The dashboard also enables to set notifications (by email, SMS or other messaging protocol, phone call) with tailored messages to the intended recipients (security manager, service developer, management or legal staff, etc.).

Innovation and benefits

- **De-coupling monitoring and inspection tasks from the detection logic.**
- **Formal verification of correctness for security policies.**
- **Spatial and temporal correlation of data for a single or multiple graphs.**
- **Heterogeneous security context (network packets, logs, system calls).**
- **Application of machine learning and other artificial intelligence techniques.**
- **Automation of deployment and re-configuration operations.**

Challenges
• **Continuous operation in detached mode or loss of connectivity.**

3.7.2 Distributed firewalling

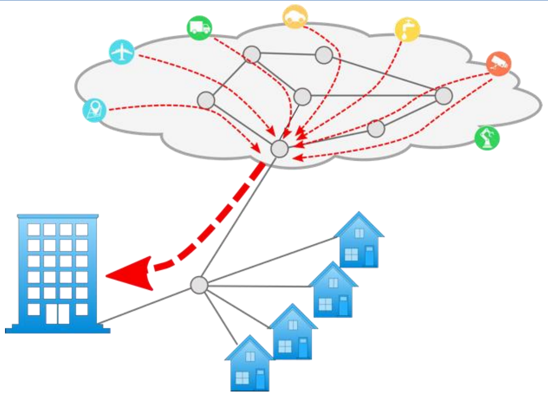
Description	
Name	Distributed firewall for cross-cloud applications and cyber-physical systems.
Objective	Control network traffic to/from virtual services and physical devices (IoT) deployed in different technological and administrative domains.
Scenario	Service providers are increasingly interested in leveraging the capillary availability of cloud installations for delocalize their services. There are several reasons that motivate this approach: increased resilience to failures and natural disasters, compliance with regulations on privacy and data locality, reduced end-to-end latency and bandwidth usage, interaction with physical objects. The lack of a physically-isolated environment implies the need to monitor all packets exchanged and to enforce communication rules on the network flows. Depending on the specific service, different firewalling paradigms may be necessary: • *Stateful packet inspection* is the simplest operation mode that must be supported to allow communication sessions initiated by "internal" hosts; inspection is usually limited to a few IP/TCP/UDP header fields. • *Circuit gateways* allow to dynamically open specific ports, usually upon authentication; this paradigm is often used with Voice-over-IP applications and other services that set up multiple communication sessions. • *Application gateways* perform deep packet inspection in packet bodies and need to understand the specific application's syntax and protocol.  Service graph Sw IoT Orchestrator ASTRID dashboard Cloud Cloud IoT
Business cases	• Cloud applications deployed on multiple heterogeneous (public/private) infrastructures. • Cyber-physical systems (IoT applications, Smart Cities, Smart Grid, etc.) • Fog/edge computing in 5G verticals (energy, automotive, multimedia and gaming, smart factory).
ASTRID stakeholders	UBITECH, POLITO, CNIT

Operation	
Current practice	There are currently different approaches for enforcing filtering rules in the scenario under consideration. **Distributed firewalls**. Distributed firewalls for virtualization environments and cloud computing build on the concept of micro-segmentation [57], and deploy packet inspection rules in hypervisors, while keeping centralized control. They enable very fine-grained control over security policies, beyond mere IP-based structure [58]. vCloud Director 8.20 by VMware includes a distributed firewall [59], while OpenStack Neutron includes the Security Groups feature [60]. **Micro-firewalls**. In the IoT domain, recent botnets like Mirai, Brickerbot, and Hajime have not only demonstrated the vulnerability of IoT installations, but also the possibility to exploit compromised devices to carry out large DDoS attacks (1 Tb/s and above). Micro-firewalls are therefore available to protect single or a small cluster of devices with minimal hardware installation and cost, in application scenarios as apartments and dorm rooms, commercial-control applications (SCADA), as well as more traditional small office, home office deployments [61] [62]. Some of these products are based on a mix of open-source technologies such as pfSense, Linux/FreeBSD. **Virtual security appliances**. Security appliances are traditionally deployed as expensive and proprietary hardware modules. More and more vendors, such as Fortinet, Barracuda Networks, F5 Networks, and CheckPoint, are offering software versions of their security appliances, mostly for data centers and virtualized IT environments, which simplify deployment and re-configuration. This simplify the integration in virtualized environments and the automatic management by software orchestration tools.
ASTRID practice	Virtual services are designed by selecting micro-services or virtual network functions (depending on the specific application domain), dragging and dropping them on the design canvas, and by connecting them in the desired topology. Neither the software developer nor the service developer has to care about network ports to open: the service developer is only required to select the “Firewalling service” among the list of security features provided by ASTRID. Then, IP addresses and TCP/UDP ports used by virtual functions are already declared in the associated metadata, so the ASTRID security orchestrator can infer the right rules for intra-service communication. The orchestrator also sets proper restrictions on source addresses if the service components are deployed in different infrastructures, as well as if they are connected to external IoT devices or services. By default, access to the service from external networks is only allowed to the public interface (for instance, it may be a web server or REST API); the security manager can then limit the access to a restricted group of users/subnetworks or opening additional pinholes for specific features. Filtering rules are then set by the ASTRID orchestrator on the security hooks deployed in the virtualization environment (virtual machines, containers). A common security understanding is that enforcement may be ineffective without awareness. The ASTRID dashboard also includes monitoring and alerting functions, which report measurements and traffic analysis. Examples of available indicators include the number of packets and amount of traffic seen to/from open ports, number of dropped packets on closed ports, number of concurrent connections. This information can be directly monitored and interpreted by the security manager or, more likely, feed some detection and analysis engine that provides alerts in case of anomalies or suspicious usage patterns.

Innovation and benefits
• **Automatic definition of firewalling rules, based on specific communication needs inferred by the graph topology.** • **Refinement of firewalling rules by the security manager.** • **Multiple firewalling models are supported, independently of the specific features of the underlying infrastructure.** • **Network statistics are collected in a capillary way.**
Challenges
• **Fast yet lightweight filtering technologies in resource-constrained environments and devices.**

3.7.3 Network monitoring

Description	
Name	Programmable network traffic monitoring for DoS protection.
Objective	Flexible definition of monitoring features and statistics.
Scenario	Network operators are constantly concerned with service availability. Telecommunication networks are often complex infrastructures, with hundreds (if not thousands) of points of presence, where large and small customers connect and are aggregated, interconnected by many redundant links. In addition, many appliances are needed to implement authentication and authorization, multimedia services (i.e., Voice-over-IP, video broadcasting), mobility, virtual private networking, xDSL services, etc. One major problem for network operators is the presence of large malicious flows, intended to carry out Denial-of-Service (DoS) attacks. Though telco operators are not committed to provide security services to their customers, such flows overwhelm their infrastructures and jeopardize the connectivity of their customers, so they are strongly motivated to detect and mitigate DoS attacks. The impact of DoS attacks has scaled with the growing size of the Internet, also extending to very large *Distributed* Denial-of-Service (DDoS), which exploits compromised devices and services to amplify the size of the attack (e.g., the DNS amplification attack). The proliferation of *smart* (yet not *hardened*) devices connected to the Internet is drastically worsening the problem: recent botnets like Mirai, Brickerbot, and Hajime have not only demonstrated the vulnerability of IoT installations but also the possibility to exploit compromised devices to carry out large DDoS attacks (1 Tb/s and above). Though large deviations from historical traffic patterns are easy to detect, the presence of a huge number of distinct rivulets of a DDoS are far more complex to detect, correlate, and consequently mitigate. Individual organizations are therefore likely to be subjected to (D)DoS attacks, which the telecom operator cannot (or is not willing to) mitigate. In this case, detection of DoS attack at the network perimeter is usually possible, so to avoid overwhelming the internal networks and devices, but the access link remains overloaded and so Internet connectivity is unavailable.

	
Business cases	• Monitoring and detection for virtual network operators. • Monitoring and detection features in network management systems. • Protection of enterprise cloud services.
ASTRID stakeholders	ETI, CNIT, TUB
Operation	
Current practice	Current protection of telco operators from DDoS attacks is largely based on traffic statistic collected at the network edge. Many protection tools already collect traffic information and measurements from network devices, by using protocols like OpenFlow, NetFlow, sFlow, IPFIX [63, 64]. These protocols have been available for many years, and are mostly based on static, pre-defined, and inflexible filtering and detection rules. When anomalies are detected, alerts are issued, and human staff takes care of heavy manual re-configuration (tunnels and VPNs) that steers the traffic towards scrubbing centers, where hardware appliances carry out in-depth analysis of traffic flows and discard malicious traffic. Deployment of such hardware appliances at each point of presence is considered costly and inflexible. For the enterprise market, many vendors are now offering cloud-based DoS mitigation services in addition to selling traditional hardware-based appliances [65] [66] [67] [68]. Such solutions are based on big scrubbing centers deployed in the cloud, that replace hardware appliances at the customer's premise. Two main architectural solutions are usually implemented. An Always-On architecture permanently diverts all traffic to/from the customer across the scrubbing center, which is therefore responsible to only deliver clean traffic. In this case, the DoS attack stops at the scrubbing center, which has enough bandwidth to not become a bottleneck for Internet communication. With an On-Demand service, the traffic is normally delivered to the customer's site. Network traffic is monitored through specific protocols; in case of anomalies, the traffic is diverted through the cloud scrubbing center that performs in-depth analysis and cleans the traffic until the attack ceases. In both cases, the redirection of the network traffic may use BGP diversion methods (smaller prefix, path prepend, advertisement and withdrawal, anycast) or DNS diversion methods.
ASTRID practice	A cloud service or a NFV network slice [69] is designed by the service developer according to the domain-specific model. At deployment time, the service provider selects the "DoS protection" feature from the ASTRID security dashboard. The security orchestrator enriches the service graph with additional functions that may be

	required to monitor and inspect traffic, then the graph is returned to the main orchestrator for deployment. The additional functions inserted by ASTRID include: • programmable monitoring hooks at the service perimeter; • analytics engine in a separate VM. The analytics engine continuously monitors traffic statistics and builds historical usage patterns. When the traffic volume grows behind the normal profile, the analytics engine requires finer-grained statistics to identify the malicious traffic and isolate it; it also installs filtering rules to drop unwanted packets. Finally, it alerts the security manager of the on-going attack, so that additional countermeasures or re-configuration actions can be taken. After the traffic volume gets back to the normal profile, monitoring hooks are re-configured to report less detailed statistics, so to save CPU time and network bandwidth.
Innovation and benefits	
• **Network statistics are defined dynamically, tailoring the depth of inspection and the amount of reported information to the actual analysis needs.** • **Reaction to anomalies can be triggered automatically after detection, removing the need for manual operations and reducing the unavailability of the service.** • **Human intervention is limited to supervision of the whole process.**	
Challenges	
• **Programmability and monitoring capabilities well beyond the flow-level reporting already available today for anomaly detection (e.g., NetFlow, sFlow, IPFIX).** • **Smart analytics that work with variable network statistics.**	

3.7.4 Trusted and safe execution

Description	
Name	Trusted software and safe execution at run-time.
Objective	Static and run-time analysis of software integrity and safe behaviour.
Scenario	The description of virtual services is usually done in abstract terms, by linking together several abstract functions (e.g.: database server, firewall, web server). Such functions are then linked to specific software versions (including the operating system to use) at deployment time. Software orchestration is usually implemented as a dynamic process that retrieves software from public or private market places, deploys them in a virtualization infrastructure, and configures them according to the graph topology and metadata[1]. Software available in the marketplace includes disk images that can be directly loaded and booted in a virtual machine as well as packages that must be installed in already running operating systems. In both cases, the dependability of the whole service heavily relies on the integrity of the software. However, such a static analysis cannot provide adequate guarantees on the trustworthiness of the software services after deployment. After the service is started, software integrity cannot be taken as granted forever. There may be replacements or

[1] Software orchestration also includes the provisioning of virtual resources (virtual machines, virtual networks, virtual storage, etc.), but this is not relevant for the specific scenario.

	alterations by legitimate users or intruders, both in the virtual environment and at the hypervisor level. It is therefore of paramount importance to continuously verify the correct behaviour of the software.
Business cases	• Static validation and certification of virtual functions. • Dynamic verification and validation of software behaviour.
ASTRID stakeholders	SURREY, DTU, UBITECH
Operation	
Current practice	Verification of the origin and integrity of the software is a common problem since the advent of file sharing over the Internet. Integrity is usually checked by checksumming the software binaries with hash algorithms (usually, MD5 or SHA-X). Clearly, this is useful to detect any alteration in the communication path but does not guarantee the origin of the information. Most OS vendors by now use certificates and public-key infrastructures to certify their repositories and the software updates. The weak link in this procedure is, as usual, the distribution of the authentication information. A pre-installed key can safely be trusted when the original installation of the OS comes from an OEM product or a trusted digital medium. Alternatively, the usage of X.509 certificates is a common practice in many fields. In other cases (as often happens for open-source products), the vendor's key must be retrieved from the Internet during the first installation; it is usually available on well-known web sites that use X.509 certificates. Cloud-ready images to be booted in VMs are usually shipped with the same procedures described above. Beyond certification of the origin, there may be concern about the safety of operations performed by third-party software. Till now, the decision to trust or not to trust a given vendor is left to the user, who usually takes decisions based on the reputation or the credibility of the supplier. When more automation is used, some technical mean is required to evaluate the reputation, similar to the mechanism already used by PGP for mail users (the usage of blockchain technologies is currently under investigation). However, software orchestration is still in its infancy, and there is no common established practice in this field for software distribution.
ASTRID practice	Software images and packages for deploying virtual services are retrieved by public and private repositories. The software may be accompanied by checksum or other integrity checks and digitally signed; however, this is not enough for the more conscious ASTRID user. Reputation and certification of the vendor are important, but the software may still be buggy due to inaccurate quality control or unknown vulnerabilities. To overcome this kind of issues, the ASTRID framework performs source code analysis before deploying it as part of service graphs. Current existing solutions for statically analyzing the behaviour (i.e., signature-based intrusion detection, antivirus software solutions, etc.) of a software image, prior to deployment, are not suitable for the monitoring of microservices in the context of ASTRID due to significant differences in the underlying architecture. The vision is to explore the integration of more advanced analysis systems, namely dynamic and concolic analysis systems. Dynamic analysis systems, such as "fuzzers", monitor the native execution of an application to identify flaws. When flaws are detected, these systems can provide actionable inputs to trigger them. They require the generation of adequate "test cases" to drive the execution of the vulnerability analysis process. Concolic execution engines utilize program interpretation and constraint solving

techniques to generate inputs to explore the state space of the binary, in an attempt to reach and trigger vulnerabilities. In ASTRID, the goal is to combine such techniques so as to design a hybrid vulnerability analysis tool which leverages fuzzing and concolic execution in a complementary manner to find deeper bugs. The innovation lies exactly in this synergy; by combining the strengths of the two techniques, we mitigate their weaknesses, avoiding the path explosion inherent in concolic analysis and the incompleteness of fuzzing.

Once the software has been validated, a digitally-signed validation mark can be added as metadata. The more the validation marks from different validation systems, the more likelihood the software is safe. However, static validation cannot include all possible execution scenarios and conditions, so the ASTRID framework also offer a run-time validation service. The purpose of run-time validation is to continuously verify the execution, and promptly alert in case something goes wrong during the execution of a (micro-)service

Towards this direction, we will identify critical attack points for such (micro-)services (e.g. memory regions, libraries, ports and network interfaces) and devise appropriate software "hooks" (through the use of eBPF) for their real-time monitoring. Towards this direction, information of interest includes the monitoring and identification of attacks subverting machine code executions that are typically mounted by taking control over the execution stack, e.g. by exploiting buffer overflows. Especially, return-oriented programming is considered as one of the most harmful attacks of this type allowing the attacker to reuse the code and obtain Turing-complete functionality without code injection by maliciously combining and executing a chain of short code sequences. These attacks can bypass many mitigation measures of modern operating systems, incl. data execution prevention through non-executable memory pages and address space layout randomization.

When anomalies or a dangerous behaviour is detected, the whole virtual function is isolated for further analysis and appropriate mitigation measures will be explored (e.g., replace by the originally deployed software with a trusted state that has already been verified).

Innovation and benefits

- **Verification and validation of the software behaviour, in addition to certification of integrity and origin.**
- **Software is verified according to the expected operational environment and the real operational environment.**
- **Run-time analysis of the software behaviour.**
- **Real-time monitoring and collection of software execution memory residue for further analysis.**
- **Combination of dynamic and concolic analysis systems for enhanced software behavioural analysis during run-time while bypassing the limitations of these approaches when employed in a standalone manner.**

Challenges

- **Remote verification of the software behaviour at run-time.**
- **Disconnected operation.**
- **Replacement of *stateful* virtual functions (e.g.: databases).**

3.7.5 Response to Attacks and Enabling Forensic Investigation

Description	
Name	Response to Attacks and Threat Enabling Forensic Investigation
Objective	Response to attacks, save relevant data for forensics, enable specific operation for threat investigation, share cyber-threat intelligence.
Scenario	Even the most reliable system may occasionally be compromised. This situation has some important implications: • the system loses trustworthiness, so it should be removed from production as soon as possible; • a safe copy should be restored as soon as possible, to reduce the unavailability especially in case of critical services; • software, and recent execution traces should be analysed to identify any loss or alteration data; • a "screenshot" of current status should be saved, together with any relevant history, to investigate the causes, the attack methodology, and the attack source, and to identify possible responsibilities; • cyber-threat intelligence, including the identification of vulnerabilities and attack vectors, should be shared with relevant national and international bodies, to avoid the propagation of the threat.
Business cases	• Forensics and lawful investigation. • Honeypots and threat investigation. • Cyber-threat intelligence and automation of information sharing processes with national and international CSIRTs/CERTs.
ASTRID stakeholders	INFO, UBITECH, AGE, DTU, SURREY
Operation	
Current practice	Recovery of compromised systems has required manual processes for long, including operations like shut down of applications and servers and network reconfiguration (even with partial re-cabling). Historically, *honeypots* have been used to draw attackers and to study their behaviour. Honeypots are dedicated installations, including networks and servers, that fully emulate the behaviour of real production systems, but are typically less protected. They are specifically designed for monitoring and inspection, so that investigation can be carried out about the attack techniques and the attacker behavioural patterns. Today, network segmentation (i.e., virtual private LANs and more recent software-defined technologies) allows to isolate malicious or suspicious users and to easily redirect them in fake systems (i.e., honeypots). By using cloud infrastructure, the deployment of honeypots is far less laborious, quicker, and more cost-effective than before. However, most operations are still carried out manually and the delay due to technical and administrative provisioning procedures is not suitable for nearly to real-time reaction. As a matter of fact, it usually takes from minutes to days to have a VM ready, depending on the automation level.

	As part of the procedures that are triggered following a security attack, 'forensic investigations' aimed at providing Law Enforcement Agencies with evidences suitable to the identification and prosecution of culprits, are of particular importance. Forensic investigations should be promptly triggered since the detection of the incident, so as to speed up the acquisition of the digital raw data from the attacked system and to minimize the possible loss of evidences because of the intrinsic volatility of data (e.g. process cache and circular logs). Forensic investigations may regard two types of data: a) logged data regarding service functionalities and service's users' activities, which may be routinely and lawfully collected by the service provider and b) users' private data, whose access is allowed only following an order from a court or a law enforcement agency. Data collection, storage and handling must be realized in a way to guarantee that data authenticity cannot be disputed: data collection must be performed in accordance to formal processes and by exploiting sound technologies that provide assurances about data authenticity. The increasing digitalization in all sectors has created a large dependency on secure and trustworthy ICT, and the impacts may be extremely severe when the compromised system is a critical infrastructure. Attacks propagate quicker than in the past and the risk that a specific vulnerability is exploited in different context increases proportionally. To effectively counter cyber-attacks, it is therefore necessary to share information about vulnerabilities and threats; this process is normally mediated by national or international security centers (i.e., CERTs/CSIRTs). These organizations manage public repositories of vulnerabilities, known threats, countermeasures and foster active participation by the largest number of organizations. Despite of massive digitalization in all economical and business sectors, sharing of information about cyber-threats and security incidents is still largely based on paperwork and emails. National CSIRTs provide support at the procedural level, foster cooperation and sharing of best practice, but do not cover technical issues [70]. This will certainly delay the knowledge of new threats and attacks, as well as the elaboration of remediation actions and countermeasures for every different context.
ASTRID practice	During service execution, the ASTRID framework collects and processes the security context. The latter may include network traffic analysis, network statistics, application logs, system call traces, with the exclusion of any access to users' private contents. These data are processed by detection and analysis algorithms, which trigger an alert whenever suspicious activity or known attacks are detected. The security context is routinely saved as possible evidence in future investigations. This process also includes *certification*, which is responsible for origin, timestamping, digital signing, integrity. In this respect, the ASTRID framework works according to the following principles: i) storing trusted evidence, and ii) preserving the chain of custody of the evidence. As soon as a security alert is raised, if the security manager has selected the "Forensic Service", the ASTRID framework autonomously reacts according to the specific policies. Typical measures include replacement of compromised functions, isolation of the service, archiving for legal investigation. **Removal of compromised functions**. Almost any intrusion begins from one host and can then propagate along the whole service. ASTRID timely breaks off any intrusion at its early stage, by replacing each compromised virtual function with a clean copy. **Service isolation**. ASTRID can *freeze* a compromised service to make it unusable. The image will be stored and subject to

	to the same technical provisions that guarantee the origin and the integrity of the security context; access to such image, which contains users' private data, will be strictly limited to the security manager that will have to guarantee, within a defined time period, either the handover of data to a proper Law Enforcement Agency or the deletion of such data. Alternatively, all sensitive and private data are detached, and the access is restricted to suspicious users only; at the same time, the number and verbosity of the security context is increased, so that detailed information can be collected to investigate the attack and its evolution. Basically, the original instance of the service is turned into a honeypot, while legitimate users are redirected towards a clean instance of the service. The honeypot will constitute the first target for more intrusive investigations that can affect also users' private data and that may be conducted following the intervention of the relevant authority. Finally, if a new threat is discovered, the framework automatically prepares incident reports according to common templates and imports this knowledge into some CSIRT database[2]. Automation of this procedure reduces the risk for human mistakes and negligence, as well as inaccurate or incomplete information.
Innovation and benefits	
• **Automated response to attacks.** • **Reconfiguration of the service graph.** • **Certification of security audits to be used as evidence.** • **Transformation of compromised service instances into honeypots.** • **Compliance to GDPR and other national regulations on privacy.**	
Challenges	
• **Saving, verification, and restoration of the session status in case of replaced instances.** • **Replacement of *stateful* virtual functions.**	

3.7.6 Lawful interception

Description	
Name	Lawful interception.
Objective	Support crime investigation by intercepting traffic when requested by the court.
Scenario	The massive softwarization and the introduction of NFV technologies are radically changing the business models of telecom operators. Originally, wide area telecommunication infrastructures were mainly designed to carry voice services, and the transmission of data packets represented an additional service. Today, the design principles have totally reversed: communication infrastructures are mainly packet-oriented networks, and voice communication represents just one (often not the main) among many other services. In legacy telco networks, the infrastructure was typically owned (or just "operated", in case of public ownership) by a single or a few operators, architectures typically followed a hierarchical model, all traffic was unencrypted, and interception could be

[2] Though automated reporting of incidents is a natural complementary feature for ASTRID, it is not envisioned by the Project.

	done by duplicating communication channels at specific switching points (e.g., by demultiplexing and extracting SDH/SONET circuits). With the introduction of software-based communication services built over the Internet, interception becomes far more difficult. As a matter of fact, the service may be operated by a provider in a foreign country, the topology and routing of the traffic is not always known, and end-users may use additional software to encrypt their data. Many alternative communication services are today already available, which leverages the Internet connectivity, for both voice and messaging: Skype, Telegram, GoToMeeting, Webex, What's Up, just to mention a few among the most popular. With the progressive introduction of Voice-over-IP protocols, telco operators are shifting to similar paradigms; the main difference is that they also give physical connectivity and own the infrastructure. People are often attracted towards alternative services by a number of factors: price, availability of different platforms (mobile phones, PCs), confidentiality and anonymity. The implementation of communication services must properly balance privacy rights of individuals with the need for a secure society; the specific objective is to avoid virtual services to become a catalyst for criminal and illegal activities.
Business cases	• Forensics and lawful investigation.
ASTRID stakeholders	INFO, UBITECH
Operation	
Current practice	Providers of public telecommunications networks and services are legally required to make available to law enforcement authorities information from their Retained Data (RD) and the ability to perform Lawful Interception (LI) which is necessary for the authorities to be able to monitor telecommunications traffic as part for criminal investigations. Typically, this functionality is supported at nodes within the core network. However, when telecommunication services are easily deployed for small clusters of users, the usage of encrypted channels make it impossible for providers to accomplish legal obligations. Service owners have in this case similar responsibilities than telecommunication providers, but with the additional issue that their software might not be designed with legal obligations in mind.
ASTRID practice	ASTRID services are designed with security and privacy in mind. Public communication services (both based on voice, text, or other media) make uses of encrypted channels when exchanging data among users' devices and virtual functions. To be compliant with specific legislation on privacy, ASTRID services always use encryption between users and the virtual functions, and between the different internal virtual functions. In addition, records of any activity are kept about the time, duration, and participation of each communication session, according to relevant regulations on cyber-crimes and privacy. When a law enforcement authority is authorized and make request to intercept communications, the service provider just enables the "Lawful interception" feature and set policies to identify the involved sessions (for example, communication between two specific users, all communications of a single user, communication among a group of users, communications that happen within a given region). The

ASTRID framework automatically duplicates the session traffic in the most appropriate point and makes it available to the authorized agents by delivering it to an external application; the confidentiality is guaranteed by using encryption channels between the service and the user and by an access control framework that verifies the identity and authorization of the user before giving access.

Software orchestration helps in this situation to quickly reconfigure the service graph so to i) duplicate the sessions at relevant interception points and forwards the duplicated traffic towards the authorized authority, by deploying additional virtual functions or performing re-configuration; ii) deploy additional virtual interception appliances to be used by enforcement authorities; iii) enforce access control on external users that want to access intercepted sessions; iv) encrypt duplicated traffic before sending to the external entity.

Innovation and benefits

- **Encrypted channels for traffic flowing between different domains.**
- **Reconfiguration of the service graph.**
- **Certification of communication services to privacy and law enforcement regulations.**
- **Traffic analysis tools and analytics available to law enforcement authorities.**

Challenges

- **Manage end-to-end quality of service while duplicating the traffic.**

4 Relevant technologies and approaches

The set of relevant technologies for ASTRID has been selected according to the main concept and challenges described in Section 3.

The common denominator of different virtualization paradigms is the high degree of automation in provisioning. The ability to dynamically create virtual machines, containers, networks, storage areas, and software functions has brought additional dimensions to plain replication and synchronization tools (e.g., based on ANSIBLE, CHEF, PUPPET scripts). Software orchestration can effectively manage virtual services, from the creation of an execution environment to deployment, configuration, and life-cycle management of the software. Different models and tools have been proposed for software orchestration in the recent years, mainly targeting configuration- or data-driven approaches. An analysis of orchestration models and tools is therefore important to drive the definition of the ASTRID architecture, seeking the more general and technology-agnostic solutions so to ensure applicability and portability to both cloud and NFV environments (see Section 4.1). More automation will eventually turn into simpler instantiation and configuration of security aspect, hopefully by mean of high-level policies (see Section 4.2).

The preliminary indication of a multi-layer architecture that decouples context management from the detection algorithms suggests the need for fast and effective monitoring and inspection technologies, as well as a framework for data fusion and aggregation. In this respect, recent frameworks for building programmable network data planes promise packet inspection at nearly line speed, and full flexibility to adapt the deep of inspection to the actual needs; some technologies even extends the analysis to system calls (see Section 4.3). On the other hand, many behavioural aspects of the applications are not directly reflected on network packets or system calls, so the analysis of system and application logs remains a desirable feature. In this case, the issue is not the access to the logs, but rather the pragmatic collection, organization, and abstraction of heterogeneous information through a common interface.

Many frameworks are already available to this purpose (see Section 4.4), and the specific challenge is their extension to also cover for network and syscall data.

Though the elaboration of advanced and innovative detection algorithms is not the primary objective for the Project, the usefulness and effectiveness of the ASTRID framework must be demonstrated in that direction. Under this perspective, a thorough analysis of the literature about algorithms for intrusion and attack detection is not worth. Rather, the focus is on the identification of the main different methodologies and their specific requirements on the context to process. The ambition is to consider both software and network analysis, as well as some basic enforcement capability that could be integrated as automatic mitigation/response actions (see Section 4.5).

Apart the direct usage for detection, data and events collected by the monitoring/inspection process must also be conserved for possible investigation and prosecution in court. Relevant normative and research frameworks for maintaining information with legal validity must therefore be kept into account. Specific aspects pointed out by the selected application scenarios include legal interception and forensics investigation (see Section 4.6).

Least but not last, privacy and ownership must be ensured for the security context. Data collected from the different functions that compose the virtual service must be exposed to authorized entities only. For legal validity, it is also important to consider non-repudiability and certification of the origin. In this respect, identity management and access control will represent an inescapable element of the ASTRID framework (see Section 4.7).

4.1 Orchestration models and strategies

In this section, a short overview of mechanisms applied for orchestration of cloud applications and network services is provided. The objective is to present relevant technologies in both areas, taking into account that they are related to the scenarios considered within ASTRID. Cloud orchestration mechanisms are applied for deployment and runtime management of cloud applications over virtualised infrastructure. In this case, application requirements are mostly considered without strict guarantees for the reserved networking infrastructure and the fulfilment of network-oriented requirements (network-agnostic applications deployment). Network service orchestration mechanisms are applied for the deployment and runtime management of virtualized network functions that are provided to application service providers on behalf of telecom operators. It should be noted that evolving activities are also in progress with regards to the design and implementation of APIs facilitating the interaction among these orchestration domains.

4.1.1 Cloud Orchestrators

Given the emerging trend for the design and management of cloud-native applications, similar trends exist in the design and development of orchestration solutions that can be applied over such applications. Orchestrating cloud-native components is an extremely hot topic. The reason for that is that cloud computing unleashed new capabilities to application service providers as far as managing their workload is concerned. The struggle to minimize both CAPEX and OPEX has transformed the DevOps ecosystem in such a way that cloud-based deployment and management is considered a de-facto approach. However, even under this perspective, application providers are continuously seeking to minimize costs through the maximum utilization of purchased resources. These efforts have paved the way to different virtualization strategies.

At present, there are three distinct ways for an application to be cloud native. These are a) the usage of Virtual Machines; b) the usage of Unikernels and c) the usage of Containers. Virtual machines are widely used the last ten years. They are extremely mature and well isolated since hypervisors offer a strong sandboxing layer. Their disadvantage is the actual size since each application that is executed in a VM has to inherit the footprint of its base operating system. Unikernels [71] is a new approach to deploying cloud services via applications written in high-level source code. Unikernels are single-purpose appliances that are compile-time specialized into standalone kernels and sealed against modification when deployed to a cloud platform. In return they offer significant reduction in image sizes, improved efficiency and security, and should reduce operational costs. There are several dominant unikernel frameworks. The most dominant ones include OSV.io and Mirage . While Unikernels are extremely small in terms of footprint, their disadvantage is the lack of generalization during their creation. In practice, each unikernel necessitates the compilation of a custom kernel. Finally, Containers provide isolation of an application from the operating system and the physical infrastructure in which it resides using kernel-based functionality (namespaces and cgroups). Such isolation provides means for the safe execution of applications; hence applications cannot contaminate the operating system and furthermore they cannot contaminate resources used by other applications. The advantage of containers is their small footprint and their disadvantage is the weak (i.e. kernel based) isolation scheme. The effort to maximize the utilization of cloud resources provided significant momentum to containers.

When it comes to orchestrating cloud native components there is no silver bullet for orchestrating containers and VMs simultaneously. It could be argued that Docker Swarm, and Kubernetes are well-proven, production-grade systems for deployment, scaling, and management of containerized applications while OpenStack Heat is a VM-based orchestrator. These orchestrators offer, out of the box, cluster management, static scaling, desired state reconciliation, load balancing and rolling updates.

Moreover, a set of cloud-native applications orchestration have been provided by H2020 research projects. ARCADIA [72] orchestrator is able to orchestrate both VMs and containers offering the advanced feature of automatic instrumentation and dynamic policy management. Finally, MATILDA [73], based on the ARCADIA code base has released an orchestrator that is able to deploy containerized apps on top of a modified telco OSS. A set of intelligent orchestration mechanisms are introduced and developed related to optimal deployment of cloud applications over programmable infrastructure, runtime policies enforcement, as well as mechanisms interacting with services provided by telecom operators through Northbound APIs. Such a feature can be proven very helpful for ASTRID given the examination of scenarios that tackle both cloud computing and telecom operators' worlds.

4.1.2 NFV Orchestrator

NFV is considered as the key technology for the 5G networks. NFV will transform the way that networks are built and operated by leveraging standard IT virtualization technology and consolidating network equipment types onto industry standard servers. So far, many of the NFV and orchestration vendors are considering the first ETSI NFV and NFV MANO specifications [74] as the standard way to describe and develop their products. The purpose of MANO is to facilitate the management of the resources in the network to meet expected demand patterns, maintaining desired QoS & security levels, while making efficient use of resources and minimizing costs. This could be achieved through interactions with the various layers of the MANO stack including three main manager components: Virtual Infrastructure Manager (VIM), Virtual Network Function Manager (VNFM) and NFV Orchestrator (NFVO). There are also interfaces to external functional blocks such as the traditional Element Management System (EMS) and the legacy Operation Support System / Business Support System (OSS/BSS) or NFV Infrastructure (NFVI). NFVI is the NFV Infrastructure that includes physical (server, storage, etc.), virtual resources (Virtual Machines) and software resources (hypervisor) in an NFV environment.

Two main prominent data modeling languages are adopted and being updated for specifying NFV applications/services: Topology and Orchestration for Cloud Applications (**TOSCA**) and Yet Another Next Generation (**YANG**). **TOSCA** is a data modeling standardization effort led by OASIS for cloud orchestration environments and applications. It is a hybrid information/data-model based on Yet Another Multicolumn Layout (**YAML**) with a number of variants that commercial NFV management and orchestration (MANO) vendors have introduced based on their own YAML Domain Specific Language (DSL). TOSCA targets the orchestration of cloud applications, and it can also be used to configure NFV applications. The TOSCA NFV profile specifies a Network Functions Virtualization (NFV) specific data

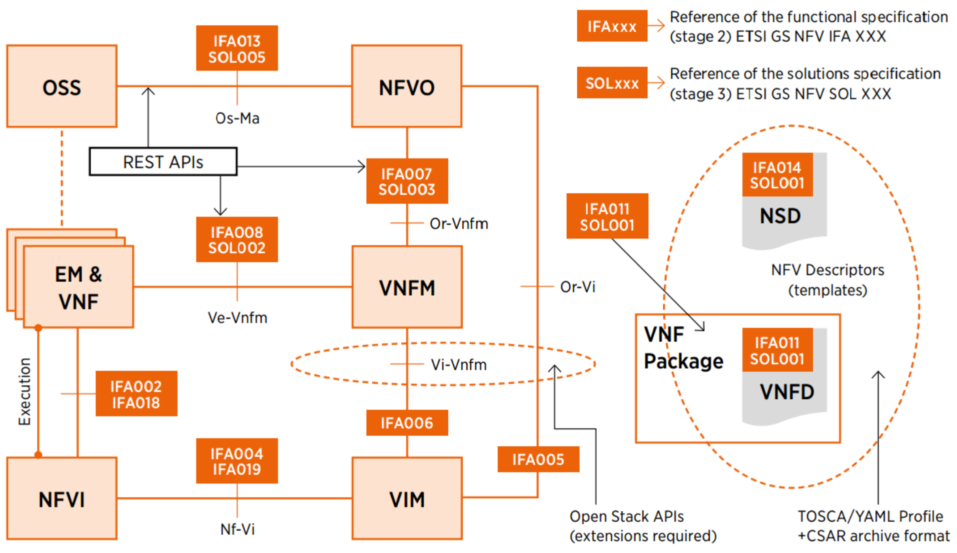

OSS
IFA013 SOL005
Os-Ma
NFVO
REST APIs
IFA007 SOL003
Or-Vnfm
IFA008 SOL002
EM & VNF
Ve-Vnfm
VNFM
Vi-Vnfm
IFA002 IFA018
Execution
IFA006
IFA004 IFA019
IFA005
NFVI
Nf-Vi
VIM
Or-Vi
IFA011 SOL001
IFAxxx Reference of the functional specification (stage 2) ETSI GS NFV IFA XXX
SOLxxx Reference of the solutions specification (stage 3) ETSI GS NFV SOL XXX
IFA014 SOL001
NSD
NFV Descriptors (templates)
VNF Package
IFA011 SOL001
VNFD
Open Stack APIs (extensions required)
TOSCA/YAML Profile +CSAR archive format

Figure 7. The ETSI NFV MANO architecture framework and specifications.

model using TOSCA language. **YANG,** used by the Internet Engineering Task Force (IETF), provides a semantically precise data modeling of both configuration and state change information. It is also possible to compile YANG specifications into a YAML DSL with associated data model parsers/validation checkers. Both are following the ETSI NFV reference architectures, interfaces, and building blocks and supported by current two major open source orchestration frameworks: Open Network Automation Platform - **ONAP** (TOSCA) and Open Source MANO – **OSM** (YANG). ONAP is an open-source software platform that enables the design, creation, and orchestration of services on an infrastructure layer on top of individual virtual network functions or SDN—or even a combination of the two. ONAP was formed as a combination of the Linux Foundation's OPEN-Orchestrator Project (OPEN-O), and the AT&T ECOMP (Enhanced Control, Orchestration, Management, and Policy) platform to form one unique code. OSM is an ETSI-hosted initiative for the development of open-source NFV Management and Orchestration software stacks. The initiative is fully aligned with the ETSI-developed NFV reference architecture. The OSM architecture integrates open source software initiatives such as Riftware as Network Service Orchestrator and GUI, OpenMANO as Resource Orchestrator (NFVO), and Juju Server as Configuration Manager (G-VNFM).

Common NFV orchestration solutions including commercial products are briefly discussed in a recent survey [74]. Many of them are often extensions of proprietary platforms provided by big vendors such as Nokia **CloudBand**, Ericsson **Network Manager**, **Cisco Network Service Orchestrator**, **Oracle Communication**, **ZTE Cloud Studio** , etc.

There are few details publicly available, mostly marketing material. For instance, Cisco offers a product named Network Services Orchestrator enabled by Tail-f. The platform deploys some management and orchestration functions such as NSO, Telco cloud orchestration, NFVO, and VNFM. Nokia CloudBand is an ETSI NFV MANO system with commercially proven reliability, automation, repeatability, and security. It is flexibly deployed for any combination of NFV Infrastructure / Virtualized Infrastructure Manager (NFVI/VIM), generic VNF Manager (G-VNFM), and NFV Orchestrator, and serves VNFs from Nokia and other suppliers. ZTE Cloud Studio NFV Orchestrator product provides network design, sandbox test, E2E orchestration, agile deployment, open integration, intelligent operation and maintenance, 5G network slicing and other functions. The Oracle Communications Network Service Orchestration orchestrates, automates, and optimizes VNF and network service lifecycle management by integrating with BSS/OSS, service portals, and orchestrators. The list of commercial solutions is not

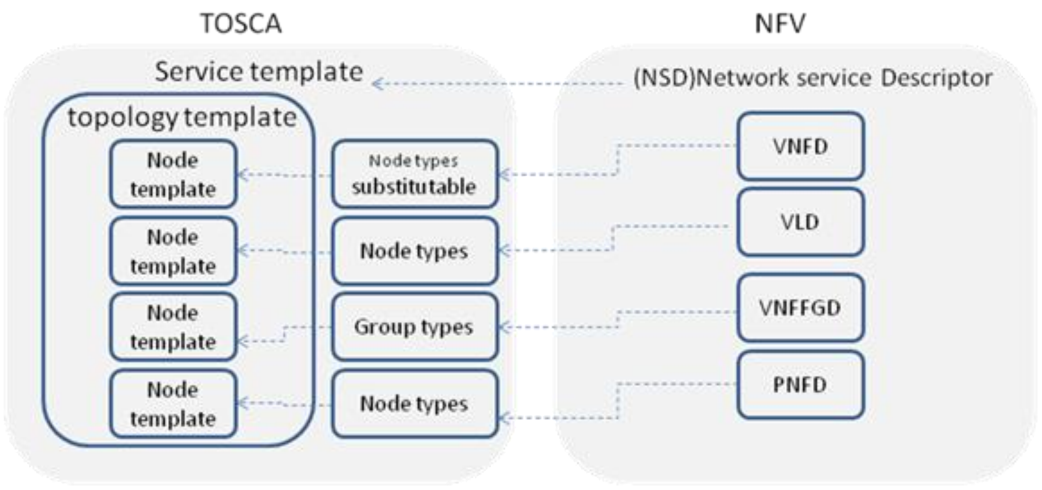


Figure 8. General mapping between TOSCA and NFV.

exhaustive and will undoubtedly become outdated. In addition to that, there are many open source solutions available. Beside current two aforementioned main stream solutions, OSM and ONAP, other open source solutions have been developed and are ongoing updated such as:

OpenBaton: an open source reference implementation of the ETSI NFV MANO by Fraunhofer FOKUS and Technical University Berlin. It is easily extensible, integrates with OpenStack, and provides a plugin mechanism for supporting additional VIM types. It supports Network Service management either using a generic VNFM or interoperating with VNF-specific VNFM. It uses different mechanisms (REST or Pub/Sub) for interoperating with the VNFMs. The first release was launched in 2015, and the current version is 6.0. OpenBaton supports TOSCA and its components are implemented using Java on top of Spring Boot framework. OpenBaton is one of the main components of the 5G Berlin initiative and supported by different European research projects: NUBOMEDIA, Mobile Cloud Networking, and SoftFIRE.

Tacker: an official OpenStack project building a Generic VNF Manager (VNFM) and an NFV Orchestrator (NFVO) to deploy and operate Network Services and Virtual Network Functions (VNFs) on an NFV infrastructure platform like OpenStack. It is based on ETSI MANO Architectural Framework and provides a functional stack to Orchestrate Network Services end-to-end using VNFs. Tacker components are mostly developed in Python and it supports TOSCA.

Cloudify: an orchestration-centric framework for cloud orchestration focusing on optimization NFV orchestration and management. It provides a NFVO and Generic-VNFM in the context of the ETSI NFV and can interact with different VIMs, containers, and non-virtualized devices and infrastructures. Cloudify is aligned with the MANO reference architecture but not fully compliant. Besides, Cloudify provides full end-to-end lifecycle of NFV orchestration through a simple TOSCA-based blueprint following a model-driven and application-centric approach.

TeNOR: developed by the T-NOVA project, a multitenant/multi NFVI-PoP orchestration platform. Its architecture follows micro-service design pattern and is split into two main components: Network Service Orchestrator, responsible for NS lifecycle and associated tasks, and Virtualized Resource Orchestrator, responsible for the management of the underlying physical resources. Both descriptors follow the TeNOR's data model specifications that are a derived and extended version of the ETSI NSD and VNFD data model. TeNOR is currently used in other European projects such as 5G SESAME, 5G Ex.

X-MANO: developed in the VITAL project under a permissive APACHE 2.0 license (available to researchers and practitioners), an open–source multi–domain NFV network service orchestrator. To address the multi-domain life-cycle management requirement, it introduces the concept of programmable network service based on a domain specific scripting language allowing network service developers to use a flexible programmable Multi-Domain Network Service Descriptor (MDNS), so that network services are deployed and managed in a customized way. X-MANO supports TOSCA.

4.1.3 Northbound APIs

Lot of work is evolving the latest years towards the specification of open APIs supporting the interaction among the telecom operators and the cloud/edge application providers. Such work is targeting on reducing the gap related to the bidirectional interaction among these stakeholders and the design and development of mechanisms enabling both of them to take advantage of each other offerings. Following, short reference to the most well-known initiatives is provided.

- **3GPP Initiates Common API Framework Study** - Provide set of APIs for exposing network functions on behalf of telecom operators to vertical business sectors.
- **Open Mobile Alliance (OMA) API**.
- **ETSI MEC API**.

4.2 Specification and Refinement of network security policies

As reported in the survey [75], in literature have been defined different types of policies, such as confidentiality, integrity, role base access control, military security, management and networking. In this project, we focus our activities on Network Security Policies (NSPs), where an NSP was defined as "a set of rules to administer, manage, and protected to network communication" [76].

The following subsections summarize the most significant works on NSP specification and refinement.

4.2.1 NSP Specification

In literature, several policy languages have been developed and proposed during the last years, unfortunately most of these languages are not related to network security (the most significant ones are related to access control [77] [78] [79]) and are defined only for one specific domain of network security, such as filtering [80] [81] or channel protection [82].

To the best of our knowledge, the most significant work that proposes a compressive language to manage, analyze and configure several network security devices is defined by Basile et Al. [83, 60, 84]. The first version of policy language proposed by Basile, was proposed in the context of the European project POSITIF, and then it was enriched and improved during the European project PoSecCo and SECURED.

The last release of this work proposes two policy language Hight Security Policy Language (HSPL) and Medium Security Policy Language (MSPL). The HSPL is suitable for expressing the general protection requirements of typical non-technical end-users, such as "Alice is not authorized to access Internet after 20 pm". While, the MSPL expresses specific configurations by technically savvy users in a device-independent format, such as "deny *.sex", "deny src 192.168", or "inspect image/* for malware".

Currently the ANASTACIA H2020 European project use both the HSPL and MSPL as network security policy language [85].

4.2.2 NSP Refinement

Policy refinement is the process "to determine the resources needed to satisfy policy requirements, to translate high-level policies into operational policies that may be enforced by the system, to verify that the set of lower level policies actually meets the requirements of the high-level policy" [86].

Theoretically, the configurations derived by the policy refinement are correct by construction. For this reason, typically, the refinement solutions use formal approach (like Ontologies, SAT Solver, model checking) that during the computation of low-level configuration verify their correctness. If is not possible to generate a correct solution, the refinement solutions must advise the network/security administrators, and, in some cases, it may propose alternative solutions (i.e. support only a subset of policies).

Even if many models and methods have been proposed to automatically refine firewalls configuration, very few works model and refine the configuration on different security controls (i.e, firewall, VPN, IDS, monitor, parental control, etc…). The most significant work on firewall policy refinement are:

- *Bartal et al.* proposed a solution, named Firmato, for the refinement of high-level filtering requirements into packet filter configurations [80]. Firmato uses a knowledge base, formalized with an entity-relationship model that describes high-level filtering requirements and network topologies, and a translator, which refines high-level filtering requirements into packet filter rules. Unfortunately, Firmato has been validated on a network with a single border firewall, hence its applicability to large and heterogeneous scenarios has not been proven.
- *Verma et al* used a similar approach to develop FACE, a firewall analysis and configuration engine [81] FACE takes as inputs the network topology and a global security policy written in a high-level language, and outputs the packet filter configurations. Moreover, authors propose an iterative process (that includes topological analysis) to identify non-enforceable policies and suggest how to make them implementable.
- *Adao et al.* develop a tool, named MIGNIS, that translates, abstract firewall specifications into real Netfilter configurations [87]. MIGNIS supports Network Address Translation (NAT), i.e., it allows the translation of the source and destination addresses of a packet while it crosses the firewall, and it applies the translation consistently to all packets belonging to the same connection. The limitations of MIGNIS are that uses only a blacklist approach and it does not permit administrators to manually add some configurations.
- *Adi et al.* formally modelled systems and their interactions using ambient calculus approach. Security policies are specified with the aid of a dedicated modal logic [88]. The tool verifies whether the specification meets the security policy if not, the optimal enforcement for the system is automatically generated using enforcement operator. If a process already satisfies the policy, there is no need for an enforcement.
- Casado et al [89] take a different approach with their solution named SANE. Instead of generating concrete packet filter configurations by refinement, authors proposed a centralized architecture where a central server (the policy decision point) controls all decisions made by the network devices (the policy enforcement point). Refinement is implicitly performed when answering from the enforcement points. However, such a centralized architecture, which is interesting from the research point of view, is not very appreciated by Internet or Network Service Provider.

To the best of our knowledge, the only two works that support the refinement of several network security policies are proposed by *Garcia-Alfaro et al.* [90] and *Basile et al.* [83, 60]:

- *Garcia-Alfaro et al* proposed MIRAGE, a tool for the analysis and deployment of configuration policies [90]. MIRAGE supports packet filter configuration, as Firmato and FACE, but it also configures intrusion detection systems (IDS) and VPN gateways. {Since it relies on general capability and refinement models, our approach greatly improves over these refinement models, as it supports different types of security controls.}
- The refinement approach Basile et al. [83, 60], firstly identify the security function needed to correctly enforce the HSPL policies, then refine by a rules engine the HSPL in the MSPL policy and finally translate the MSPL in the low-level configurations for the selected functions. This approach was also integrated into Open MANO a NFV Architecture developed by Telefonica.

Finally, in the literature, several works use the refinement process to fix some error or misconfiguration using a formal and automatic approach. Two significative examples are:

- *Souayeh et al.* propose a formal and fully automatic approach for correcting error in firewall configurations using Yices a SMT solver [58].
- *Gember-Jacobson et al*. develop a CPR (Control Plane Repair) tool, to automatically repair distributed control plane configurations [59]. It takes as input a networks existing configurations and desired security and availability policies (i.e., a specification) and using a SMT solver, takes in outputs configuration patches.

4.3 Programmable data planes for packet processing

The analysis of network traffic is an important aspect for attack detection and threat identification. It consists of inspecting packet headers, comparing their fields against given values, updating statistics, and sometimes modifying (or dropping) the packet. The main requirement is to perform all these operations without delaying packet transmission and reception and without overwhelming the inspecting device; this may be challenging particularly in case of large volumes of traffic, as typically happens for NFV services.

Traffic analysis is indeed part of the more general packet processing that is performed in a communication network. Typical operations include filtering, classification, queueing, switching, routing, natting, and deep packet inspection; they represent the "data plane" of network architectures, which performs the operations requested by the control and management planes. Packet processing in the data plane is usually based on simple logical structures (decision trees, finite state machines) that are easy to realize in hardware, which is the best option to effectively address performance issues. Performance is measured as number of packets processed per unit of time, and is mostly affected by high-speed links, small packet sizes, and large protocol states. Even recent software-defined paradigms make systematic use of hardware-based switching technologies (usually FPGA or Network Processors) for building fast data planes, while software applications are only developed for the control and management planes.

However, the interest in NFV technologies has fed the interest for efficient packet processing in software. This effort has resulted in several technologies for fast and programmable data planes, which look interesting for the project purposes. For this reason, this Section analyze and compares alternative technologies, and elaborates on their appropriateness for traffic monitoring and analysis in ASTRID.

There are two main elements that hinder effective packet processing in software:

- limited performance (bus capacity, memory access) in commodity hardware (PC-like);
- frequent memory copies in the processing pipeline.

Hardware performance of modern general-purpose servers has largely improved in the last decade, with the introduction of high-bandwidth internal buses (i.e., PCI-express, PCI-X), Direct Memory Access (DMA). Memories are improving as well (i.e., DDR3/4), but unfortunately more in terms of bandwidth than in terms of access time, which has a huge impact on network processing particularly when the data cannot be found in the CPU cache. Overall, hardware is today a secondary performance issues than software. This is especially true for virtualization hardware, which is designed to bear the traffic of multiple VMs.

Performance limitations of software mostly come from the generality and complexity of modern operating systems, which are designed to support many different protocols (in many cases, just for backward compatibility with outdated solutions). As a matter of fact, typical operating systems are organized in multiple architectural tiers, which often results in long internal pipelines. The most relevant structure is certainly the dichotomy between *kernel* and *user* space. Processing packets in user-space is far simpler and more secure than in kernel space, but it requires the packet to stay in the user space most of the time, otherwise it suffers excessively from the overhead and latency introduced by the kernel/user space context switching. In addition, the network stack implementation within the kernel usually follows common protocol layers (i.e., TCP/IP and ISO/OSI), so the pipeline length changes accordingly to the specific layer where packets are handled. Real-time operating systems have bare structures that limit the internal overhead and enable more efficient processing; however, we argue that they cannot be used in virtualized environments, where multi-tenancy and strict resource isolation are inescapable requirements. Definitely, the networking stack of standard operating systems is not suitable for processing large streams of data, so a number of alternative solutions have been designed to overcome this limitation.

Before elaborating on the alternative solutions, it is worth recalling that data plane programmability entails two main aspects. The first aspect concerns the design of the processing pipeline, i.e., the type and sequence of operations to be performed on network packets. Lightweight processing usually requires pipelining inspection, modification, and processing operations according to software and hardware architectures (bath processing, zero-copy, no context switches, and so on). The processing pipelines may be fully written in high-level languages, it may consist of a fixed yet programmable structure, or it may be defined according to specific low-level modelling languages. The second aspect concerns the specific rules to classify packets, update statistics, and perform actions. In some cases, this configuration is explicitly included in the implementation of the processing pipeline, but there are situations where external entities (i.e., the control plane) are expected to push the configuration through specific protocols. Since both aspects are relevant for programming inspection and monitoring operations, we consider available technologies for both the data and the control planes.

4.3.1 Technologies for building fast data planes

Table 1 compares the main technologies for developing fast data planes that may be of interest for ASTRID. They are grouped according to the overall approach and scope. For the sake of brevity, we only elaborate on their main concepts and applicability in this Section, while a more detailed description is reported in Annex A.

Table 1. Comparison of main data plane technologies.

	Technology	Target	Programmability	Main features	Status
Kernel hooks	Raw socket	Network socket	–	Direct access to packets (no kernel networking stack). Packet filtering (BPF).	Active
	PF_RING	Network socket	BPF-like packet filters	Packet filtering in kernel space, direct access from user-space. Dynamic NIC Access. Zero-copy support. Hardware filtering in NIC where available.	Active

	Netmap	Raw packet I/O	–	Direct access to packets (no kernel networking stack). Memory-mapped region with pre-allocated packet buffers.	Active
	OpenOnload	Acceleration middleware	–	Hybrid kernel/user-space stack; dual kernel/user-space operation. Memory-mapped kernel region.	Active
Data planes in user space	DPDK	Development framework	C/C++ libraries	Zero-copy packet access, ring buffer, memory management, Environment Abstraction Layer	Active
	VPP	Fast data plane	Memory-mapped message bus. Bindings for C, Java, Python, Lua	L2 switch, IP router, MPLS, Tunnelling, Packet telemetry, Filtering and classification, Lawful interception.	Active
	ODP	APIs for hardware acceleration	–	Reference implementation for Linux (odp-linux).	Active
	BESS	NFV dataplane	Python/C++ Script/CLI	Packet processing pipelines. Kernel bypass or legacy operation.	Active
	SNABB	NVF dataplane	Lua/C	Kernel bypass. Service graph design. High-speed communication with VMs in user-space. SIMD offloading. Just-in-time compiler.	Active
	OpenState	SDN dataplane	OpenFlow extensions	Stateful packet processing. Based on OpenFlow Software Switch.	Inactive

Data paths in kernel space	eBPF	General in-kernel event processing	C (data plane); C++, Python, LUA (control plane)	Direct access to packets; integration with Linux networking; possibility to intercept any system call (hence not limited to networking tasks).	Active
	Open vSwitch	Software dataplane	OpenFlow/ OVSDB	Support hardware- or software-based datapaths. Fast and slow paths.	Active
Programmable DPs	P4	Programmable hardware data plane	P4	High-level language to access hardware features.	Active

4.3.1.1 Kernel hooks

A first group is represented by kernel hooks that give userland applications direct access to networking hardware or the internal kernel networking stack (Section A.1); they are usually designed to provide direct access to low-level drivers or network interface cards, hence bypassing in part or completely the kernel stack. These hooks are not network data plane per se but can be used to implement custom processing pipelines.

Kernel hooks deliberately just provide basic input/output functions, because they can be used for different applications (monitoring, forwarding, replication). In ASTRID, they may be used for monitoring purposes, assuming traffic replication is available; however, the implementation of enforcing actions requires to by-pass the kernel stack, and to re-write a lot of code to manage network protocols. Indeed, the main benefit of these frameworks, namely acceleration, is partially vanished in virtual environments, where access to the real hardware is usually mediated by the hypervisor (except when SR-IOV is used). That means the DMA and similar features will not be available, even if performance still benefits from zero-copy and lightweight protocol operations. In this respect, Section 5 will clarify whether high packet I/O rate is a technical requirement or not for the ASTRID framework.

4.3.1.2 Data planes in user space

The second group consists of full data planes implemented in user-space (Section A.2); they are generally conceived as lightweight and high-performance replacements for the standard networking stack, which poor performance are often due to the large number of heterogeneous protocols supported.

Several frameworks are available for creating data planes in user-space. They often leverage kernel hooks previously described to provide fast access to raw packet I/O, while hiding the heterogeneity and complexity of the hardware. These frameworks are specifically conceived to build networking functions, bypassing the standard kernel networking stacks. Therefore, they provide a set of built-in functions for queuing, pipelining, forwarding, encrypting/decrypting, and timestamping operations on network packets. In some cases, they explicitly provide support for connection of Network Functions. The programming abstraction is often limited to define the packet pipelines in terms of processing modules; there are standard modules, but additional ones can be also developed by users. Applications can be specifically developed to directly connect to these frameworks via specific libraries; legacy applications may also connect through the standard networking stacks. However, performance benefits are only evident in the first case, where all the duplication and inefficiency of kernel implementations are totally bypassed. Data planes in user-space are a very effective solution to build software-based implementation of network devices, or to create dedicated environments for network function virtualization and service function chaining. The applicability of these frameworks in ASTRID might not be straightforward: the innovative cyber-security concept of ASTRID, focused on the service rather than on the underlying infrastructure, does not require full networking capabilities in all VMs. Even more, the impossibility to modify the hypervisor software layer, which is common in virtualized environments (e.g., public datacenters) prevent the usage of the above technologies. Finally, (re-)engineering applications to use specific networking libraries represents an important hindrance to applicability in the largest number of contexts. OpenState and OPP are specifically conceived to enhance the programmability of the hardware, hence they do not fit well virtual machines. Compared to OpenFlow, they allow the creation of "state machines", which are far more powerful than stateless configuration. However, the processing pipeline is fixed (i.e., the conceptual schemes shown in Figure 43 and Figure 44), unlikely P4 where programmers have the possibility to declare and use own memory or registries. However, the flexibility of P4 leverages more advanced hardware technology, namely dedicated processing architectures or reconfigurable match tables as an extension of TCAMs, even if P4 programs may also be compiled for eBPF in the Linux kernel.

4.3.1.3 Data planes in kernel space

The third group is represented by built-in kernel filtering and processing capabilities (Section A.3), which is essentially limited to the restricted family of Berkeley Packet Filters. The BPF framework provides a kernel virtual machine to run user programs in a controlled environment. The latest "extended" version gives access to a broad range of system-level resources (network sockets, traffic schedulers, system calls, kernel functions), without the need to deploy additional processing frameworks. The BPF implements a very effective monitoring solution: its directed acyclic control flow graph (CFG) was shown to be 3 to 20 times faster than the Boolean expression tree used by the filtering module in the DLPI, depending on the complexity of the filter. Another difference is that BPF always makes the filtering decision before copying the packet, in order to not copy packets that the filter will discard. Interestingly, there have been attempts to use eBPF programs to write extensible datapaths, for example for Open vSwitch. We point out that any inspection/enforcement solution based on the eBPF implicitly makes use of the standard kernel stack, which may be less efficient than existing data paths in user-space depending on the operating conditions. However, the main advantage of eBPF is the capability to run on an unmodified software stack, encompassing existing Linux kernels, which is a rather common scenario also in virtualized environments (and hypervisors). As stated before, the analysis of the application scenarios and technical requirements will clarify whether the performance level is acceptable or not for the Project's objectives.

4.3.1.4 Fully programmable data planes

The fifth group considers actual programming languages for packet processing (Section A.3.4). In this case, the pipeline structure is defined by the program, hence it is conceptually possible to carry out any kind of operation on any packet header, provided this is consistent with performance requirements.

Fully programmable data planes are unlike to be suitable for software-based implementation. Indeed, they are mainly conceived for hardware targets, to overcome the traditional difficulty of low-level languages (e.g., VHDL). There is only one technology available in this field, namely P4, which is actually conceived for re-programming the hardware and so appears to fall outside the main ASTRID scope. However, P4 programs can also be compiled for the eBPF, so they could be used to build kernel data paths for VMs.

It is hard at this stage to say whether a P4 program might be more convenient than a native eBPF program. In general, we point out that P4 can be used to describe protocol headers, and this may be useful when writing eBPF programs. Writing eBPF programs in P4 could be useful in case the same program is used for both software and hardware targets (e.g., in NFV environments).

4.3.2 Programming abstractions

Beyond processing technologies, it is also important to understand what abstractions are available to program or, in some cases, just configure the data plane. Programming abstractions mark the boundary between the data and control planes, and basically defines what kind of computation can be performed locally and what need to be done centrally.

Despite the large number of acceleration technologies and frameworks for data plane, the number of programming abstraction is rather limited (Section A.4). Usually, these technologies leverage the concept of Software-Defined Networking to steer the pipeline through configuration of matching rules and processing actions.

We briefly compare two standard protocols for SDN and one interesting proposed extension in Table 2. In the context of the ASTRID framework, while programmable data planes can be used to attach specific hooks to relevant security events (including packets and system calls) and to define processing pipeline, control protocols can be used to push matching rules and specific actions (drop, block, allow, alter, etc.) in the pipeline.

Table 2. Comparison of control and configuration protocols.

Protocol	Target	Language	Main features	Status
OpenFlow	SDN/ hardware data plane	OpenFlow	Flexible packet processing; available on (selected) physical network devices.	Active
OpenState/ OPP	SDN/ hardware data plane	OpenFlow extensions	OpenFlow extensions for stateful packet processing.	Inactive
NETCONF/ RESTCONF	SDN	Yang	Flexible abstraction for network devices and services, overcoming the limitations of SNMP. Template-based description	Active

OpenFlow pushes matching rules and processing/forwarding actions in flow tables. It expects a specific structure for the data plane, based on multiple tables and fixed processing pipelines. OpenFlow supports a limited (yet rather representative) set of protocol headers; vendor-specific extensions are possible, but at the cost of reduce interoperability. The main limitation of the OpenFlow abstraction is stateless operation, which prevents the possibility to manage a large number of common network protocols (and security inspection operations too). OpenState extends OpenFlow with the abstraction of the extended finite state machine, which have been used for DDoS protection in SDN [14]. Unfortunately, this project has never evolved to a more mature state, including the implementation of an efficient software dataplane.

RESTCONF is more flexible, since it does not assume any specific structure for the underlying dataplane. Indeed, the template-based approach of NETCONF allows to write Yang models for very different devices, although the counterpart of this flexibility is some more procedural complexity, due to the need to exchange and agree on the specific models.

Currently we are not aware of any standard abstraction that applies to the eBPF technology and that enables its reconfiguration from a remote controller.

4.4 Data collection, fusion, abstraction

Remote collection of logs is already a well-established practice, with many frameworks available for this purpose with different levels of features and programmability.

4.4.1 Log Collectors

Apache Hadoop facilitates data intensive tasks over big data while maintaining the properties of volume, velocity and varieties. It has two logical modules: HDFS (Hadoop Distributed File System) as storage and Map Reduce for computational workflow. HDFS provides high availability and fault tolerance by compromising on the data redundancy supporting horizontal scalability. MapReduce [91] is conceived for writing applications which process vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner.

Apache Hive is a data warehouse infrastructure for providing data summarization, query, and analysis. It provides an SQL like language called HiveQL with schema on read and transparently converts queries to map/reduce, Spark jobs, etc.

Logstash is a monitoring tool concerned with the collection of logs and event data. It is responsible for parsing and filtering log data. It supports and event processing pipeline allowing chains of filter and routing functions to be applied to events.

Kafka is a streaming platform capable to publish and subscribe the records which are in the form of streams. It is like a messaging system of an enterprise and it is used in storing streams of records without any fault helping in processing record streams as they come.

The Unix syslogd daemon supported cross-network logging. It lacks support for failure recovery, for throttling its resource consumption, or for recording metadata. Messages are limited to one kilobyte, inconveniently small for structured data. Instead, **rsyslog** is used on UNIX and Unix-like computer systems for forwarding log messages in an IP network. It implements the basic syslog protocol, extends it with content-based filtering, rich filtering capabilities, flexible configuration options and adds features such as using TCP for transport. Finally, **syslog-ng** is the replacement for backward compatible with syslog. Some of the features include supporting IPv6 protocol, capability to transfer log messages with reliability using TCP, and filtering the content of logs using regular expressions. However, it does not protect against log data modifications when it is placed at an endpoint.

Chukwa represents a design point in between two existing classes of systems: log collection frameworks on the one hand, and network management systems on the other. It combines the abundance of data display tools of existing Network Management System (NMS) systems, with the high throughput and robustness expected of log collection frameworks.

Scribe is a service for forwarding and storing monitoring data. The meta data model is much simpler than that of Chukwa: messages are key-value pairs, with both key and value being arbitrary byte fields. This has the advantage of flexibility; with the disadvantage of requiring developing its own metadata standard, making it harder to share code.

Artemis processes logs on the machines where they are produced [92]. The advantage is that it avoids redundant copying of data across the network and enables machine resources to be reused between the system and the relative analysis. The disadvantage is that queries can give the wrong answer if a node crashes or becomes temporarily unavailable.

Flume was developed after Chukwa and has many similarities: both have the same overall structure, and both do agent-side replay on error. There are some notable differences as well. Whereas Chukwa does elaborations end-to-end, Flume adopts a more hop-by-hop model.

NIDS is a scalable log analysis framework using cloud computing infrastructure [93]. The primary objective is to efficiently handle large volume of logs from servers by using Hadoop and cloud infrastructure.

Fluent is an open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data. It unifies all facets of processing log data: collecting, filtering, buffering, and outputting logs across multiple sources and destinations.

4.4.2 Log Storage

Cassandra is fault tolerant, decentralizes and gives the control to developers to choose between synchronous and asynchronous data replication. It offers rich data model, to efficiently compute using key and value pairs. It supports third party applications, but it is not entirely secure.

MongoDB was designed to support humongous databases. It is a document database designed for ease of development and scaling. It also has a full index support, replication and high availability.

ElasticSearch is a distributed, RESTful search engine. Designed for cloud computing, to achieve real-time search, stable, reliable, fast, easy to install. Support for data using JSON over HTTP index.

4.4.3 Graph Databases

Based on the above considerations, graph databases as Neo4j, OrientDB, and ArangoDB looks the best solution for building our abstraction layer. Indeed, unlike tabular databases like Cassandra, they support fast traversal and improve look up performance and data fusion capabilities.

Neo4j is a disk-based transactional graph database It supports other language like Python for graph operations. It scales to billions of nodes and relationships in a network. It manages all the operations that modify data in a transaction. Both nodes and relationship can contain properties.

Sparksee is a very efficient and bitmaps-based graph database and written in C++. It makes graph querying possible in different networks like social network analysis and pattern recognition.

InfiniteGraph is a distributed graph database based on a graph like structure. It is designed for to handle very high throughput. A lock server which handles lock requests from database applications.

Infogrid is a graph database with additional software components, whose functions are framed for web applications. It gives an abstract interface to store technology like SQL.

Hypergraph is different from the normal graph because in this edge is points to the other edges. In various fields, it is used in the modelling of the graph data. It supports online querying with an API.

Trinity is a distributed graph system over a memory cloud with a key value store [94].It provides fast data access, graph exploration, parallel computing, and high throughput for larger datasets.

Titan is its scaling feature. It provides support to very large graphs and scales with the number of machines in a cluster, concurrent users and the size of graph.

OrientDB is a 2nd Generation Distributed Graph Database. It can traverse parts of or entire trees and graphs of records in a few milliseconds. It combines the flexibility of document databases with the power of graph databases, while supporting features such as ACID transactions and fast indexes.

ArangoDB is a NoSQL system that combines an aggregate oriented and graph database. The schema is determined automatically and is not provided by the user while allowing to model the data as either documents, key-value pairs or graphs.

4.5 Detection and enforcement

The number of types and variants of attacks to networked systems is extremely high. While limiting the access from the outside was enough in the early days of the Internet, soon the increasing complexity of the software, protocols, and architectures has demanded for advanced forms of inspection, detection, and enforcement encompassing both network devices and end systems. New usage models, including remote connectivity, bring-your-own-device, and virtualization (see Section 3) and new threat vectors have progressively blurred the distinction between internal and external users, hence hindering the application of simple enforcement policies based on the network topology.

Modern security systems are usually complex architectures, including several appliances and services for detection of unauthorized usage and enforcement of security policies: firewalling, intrusion detection and prevention systems, antivirus, virtual private networks, authentication and authorization, network access control, and identity-based networking services. Despite the quite short list of service types, the number of alternative architectures, protocols, algorithms and policies is almost unlimited.

According to the Project's objectives, the ASTRID architecture should:

- guarantee the integrity and correctness of the software, which is usually selected from public repositories;
- detect anomalies in network and service usage, since cloud/NFV applications are typically used for public services.

For the specific purposes of the Project, it is worth analyzing an extended taxonomy, pointing out the main characteristics that should be taken into consideration in the design of the system architecture.

4.5.1 Software analysis

ASTRID will include the provision of secure, robust and efficient run-time source code analysis and behavioural attestation and verification methods to check the internal state of a (potentially) untrusted application service (micro-service) towards establishing its security and trustworthiness. This is considered as one of the main goals towards **"security by design"** solutions, including all methods, techniques and tools that aim at enforcing security at software and system level from the conception and guaranteeing the validity of these properties.

In general, there exist several approaches for vulnerability analysis and assessment for a wide gamut of systems and devices that are partially related to the project; these fall mainly into two categories each featuring different types of advantages and disadvantages: dynamic and concolic analysis systems. Dynamic analysis systems [95], such as "fuzzers", monitor the native execution of an application to identify flaws. When flaws are detected, these systems can provide actionable inputs to trigger them. They require the generation of adequate "test cases" to drive the execution of the vulnerability analysis process. Finally, concolic execution engines [96] [97] (including also remote attestation protocols) utilize program interpretation and constraint solving techniques to generate inputs to explore the state space of the binary, in an attempt to reach and trigger vulnerabilities. However, such techniques, although conceptually related, cannot be directly applied in the micro-service domain due to significant differences in the architecture of the systems to be checked; they have been proposed mainly for applications following the classic application design methodology that is monolithic in nature, i.e., everything is bundled into a single – or perhaps several – virtual machines. Nonetheless, many applications today are being moved to containers running multiple micro-services. Each container has its full complement of packages and dependencies. Therefore, to reap the benefits of moving to such a micro-service-oriented architecture, ASTRID aims at customizing existing VA techniques for applications that have been designed as a suite of small micro-services with many internal dependencies.

Instead of applying these techniques in a standalone manner, the vision in ASTRID is to combine them so as to design a hybrid VA tool which leverages the properties of both approaches in a complementary manner [98]to find deeper bugs not only during the creation of the micro-service (design time) but also after its deployment (run-time). In what follows, we give more details on the specifics of the most prominent mechanisms that have been proposed for each of the aforementioned families of software threat analysis.

4.5.1.1 Software Threat Modelling & Source Code Analysis

All common software threat modelling techniques (Appendix A.6) apply to the context of micro-services, which are divided into two main categories: **white-box** and **black-box**. In the white-box scenario, we have access to the source code of the implementation, and the prime time of this technique is to verify the design, i.e., to assess whether the software to be deployed does what is designed to do. On the other hand, the black-box approach, is suited for the "*attacker's perspective*" scenario, where all we have is access to the compiled and running software, and our target is to enumerate all the vulnerabilities that we can find with this in hand. These approaches remind of the penetration testing process, but threat modeling is nothing more than a systematic method for the identification and the assessment of vulnerabilities to a system, while penetration testing is a systematic process for just the identification (only) of the vulnerabilities.

In the case of the envisioned types of micro-services, since we will have access the source code and a working implementation, both approaches can be applied. With the use of standard source code analysis and threat modelling tools, we are going to pinpoint all the possible threats and with the results of this process we will attempt to detect its vulnerabilities. The state-of-the-art techniques that will be employed are code analysis and fuzzing. The main techniques that we are going to use are code analysis and fuzzing. The white box approach will be comprised of both automatic and manual code analysis, which are the basic methodologies for vulnerability detection. We are going to pass the code through automatic code analyzers that will produce a number of possible vulnerabilities, then, we will assess these results and identify whether they are actual vulnerabilities or not. After this process, we will compare the vulnerabilities we identified with the results of the threat modeling process, and if we find threats that were not covered by the code analyzers, we are going to manually inspect these points of interest to assess if there are any missed vulnerabilities.

There are four basic techniques for code analysis, all of these techniques are used in automatic code review tools, but some of them could also be applied to manual code reviews too:

- **Control Flow Graph** [99]**:** These graphs are the representation of each possible path that an application might take during its execution. More specifically, each node of the graph represents a block of code that does not contain any out of order execution commands (jumps). All the jumps of the code are represented as directed edges that point to the target address of the command to be executed after the jump. This way, we can fully map a piece of software and better understand its architecture.
- **Data Flow Analysis** [100]**:** This technique uses control flow graphs, in order to better understand the values that a variable might get. It is a data centric technique that aims to simplify the code by aggregating the entirety of the commands that act on a variable and produce a smaller subsets of these commands with the same result. This method is mainly used in the optimization process of compilers, but it also helps in static code analysis, as it helps to better understand what changes are applied to data and pinpoint possible vulnerabilities or bugs.
- **Taint Analysis** [101]**:** This method, once again expands on the previous one (Data Flow Analysis), with the target of finding how untrusted sources of data are propagated to the system. It utilizes the results from the data flow analysis, in order to "taint" the data from untrusted sources and follow the path of this data throughout the system. This way we have a visual representation of how far an attacker can reach in the system and what components he can access. Taint analysis can be used to identify both information disclosure vulnerabilities and tainted flow vulnerabilities, with the latter being the scenario we described, and the former is the

scenario where sensitive information can "sink" into outputs that an attacker might have access to, i.e. the reverse process of the aforementioned scenario.

- **Lexical Analysis** [102]**:** This procedure is applied by automated tools, in order to tokenize source files. With the tokenization, it is easier to spot vulnerable patterns and/or commands that should not be used. This method is mainly used by automated tools.

In the case of the black-box scenario, we do not have access to the source code and our main focus is to try and identify vulnerabilities just by a running implementation of the software. The basic and most widely adopted method for such a black-box scenario is **fuzzing**. Fuzz testing is an effective technique for finding security vulnerabilities in a running software. Fuzzers have initially been developed for testing protocol implementations on possible security flaws due to improper handling of malicious input. In [103] the authors present a model-based fuzz framework for systematic automated testing of a Software Stack implementation. This framework is based on black box fuzz testing methods, integrated with target profiling, data modeling and test algorithm etc. However, what makes fuzzers difficult to use is the fact that a fuzzer by design cannot be general–purpose. [104] is another work that applied fuzzing techniques with a tool named Flinder that specializes in discovering typical security-related programming bugs by offering custom test suite development capabilities, Linux support and it can be used for both "black-box" and "white-box" fuzzing.

4.5.1.2 Attestation Protocols during Run-Time

In addition to traditional source code analysis techniques that provide detailed software analysis during design-time (prior to deployment), ASTRID will build on-top of existing remote attestation techniques towards ensuring the internal state of a deployed (and potentially untrusted) application service in order to enhance its security posture. These methods will be used for achieving trustworthiness of the command execution flows which can be used in combination with Berkeley Packet Filters, discussed in Appendix A.3, for the detection of any suspicious activities.

The reason behind employing attestation mechanisms as a means of operational assurance is twofold: First of all, one of the main challenges in managing device software security in today's heterogeneous and scalable infrastructures is the lack of adequate containment and sufficient trust when it comes to the behaviour of a remote software system that generates and processes mission-critical and/or sensitive data. An inherent property in ASTRID is the codification of trust among computing entities that potentially are composed of heterogeneous software components, are geographically and physically widely separated, and are not centrally administered or controlled. By leveraging the artefacts of traditional source code analysis and threat modelling coupled with advanced primitives (such as run-time property-based attestation) and building upon emerging trusted computing technologies and concepts, ASTRID will convey trust evaluations and guarantees for each deployed micro-service.

This high level of trustworthiness which will not only include integrity of system software but also the correctness and integrity of the generated data flows will, in turn, reduce the overall attack vector and allow for the more effective operation of the ASTRID framework. This will allow the secure configuration, deployment and operation of a variety of micro-services offering different functionalities.

In order to cover the ever-increasing attack surface targeting the dynamic execution of remote software systems, the concept of behavioural-based attestation services has been introduced, reflecting the identified (and configured during run-time) preventive, access control, information flow and functional safety policies to be enforced. To this end, the properties (of interest) to be attested will vary depending on the type of functionality that each such (micro-) service offers.

There exist different kinds of attestation, particularly **static attestation** and **dynamic attestation** [105]. Static attestation allows the attestation of static properties and configuration of a remote software system. The most prominent example is the attestation of the integrity of binaries [106]. As the name implies, dynamic attestation deals with dynamic properties of the run-time. For instance, it is concerned about the execution and data flow of programs, and not the static integrity of binaries.

Naturally, attesting dynamic properties is significantly more challenging than attestation of static (already known) properties. Hence, the majority of research has focused on static attestation including industry effort in the Trusted Computing Base introducing secure and authenticated boot loading mechanisms of operating systems [107]. However, given the continuous attacks on dynamic properties such as zero-day exploits which corrupt program's control flows, static attestation alone cannot be considered a viable security solution in the long-run, and needs to enhance with advanced dynamic attestation mechanisms.

The most prominent approach in this context is **Control Flow Attestation** [108]. Control-flow attestation is one of the most important dynamic properties at the software layer since it captures diverse instantiations of software exploits that hijack a program's control flow. Such attacks tamper with state information in the program's data memory area, e.g., the stack and the heap. Software bugs allow an attacker to arbitrarily alter state information and hijack the program flow of applications to induce malicious operations. While traditional attacks require the attacker to inject malicious code [109], state-of-the-art attacks such as return-oriented programming leverage code that is already present in the vulnerable application thereby bypassing modern mitigation strategies [110, 111]. In other words, the attacker resembles malicious codes through a combination of already existing benign code pieces. In contrast to traditional PC platforms and mobile phones, software exploits against cloud services (as the ones envisioned in the Reference Scenarios) can have severe safety consequences. Consider a modern network which features a vast amount of heterogeneous hardware and software components with hundreds of millions line of code. A common theme of such composable infrastructures is that all of them are pushing the envelope with respect to how many application instances can be packed efficiently onto a certain physical infrastructure footprint. This co-existence of multiple micro-services, multiple applications, or even multiple tenants enables a variety of Advanced Persistent Threats (APTs) to be exploited by adversaries.

The general approach of control-flow attestation includes a cyber-physical component (enhanced with some notion of trusted computing technology), acting as the verifier, that first receives the necessary security policies containing the specifics of the properties to be attested. Based on the interpretation of these policies, it then computes all legitimate control-flow paths of an application and stores its measurements in a database. To trigger the run-time attestation, as dictated by an already defined security policy, the verifier sends a request to the component which acts as the prover. The prover component executes the software that the verifier desires to attest and a trusted component measures the taken control-flow paths. For instance, this can be achieved through a hash function. Finally, the attestation result is sent back to the verifier for validation. In the case of a failed attestation about a software's integrity, the information might not be sufficient to understand the software's behaviour. Thus, in this case, a more in-depth investigation of the overall behaviour is needed to detect any cheating attempts or if any type of (non-previously identified) malware is resident to the system. The goal of this functionality is to be able to dynamically define new attestation policies against newly identified attack vectors.

To achieve all the functionalities to be offered by the Control Flow attestation mechanisms to be developed, **sampling** and **tracing** techniques will also be employed. Sampling is the process of collecting information regarding the state of a system: values of some variables, stacks of threads, etc. at unspecified moments of time. Tracing is the process of installing probes at specific places of software. Profiling is the most prominent example of tracing. Sampling is very helpful when you do not know where issue happens, but it hardly helps when you try to know why it happened. With tracing we can install a probe to that function, gather information on lists and collect cumulative execution of function, and then cross-reference them, searching for a pattern in lists whose processing costs too much CPU time. Tracing is used for statistics collection and performance analysis, dynamic kernel or application debug, system audit. Tracing is very powerful, but it can also be cumbersome for whole system analysis due to the volume of trace information generated. Unlike other approaches, dynamic tracing tool embeds tracing code into working user program or kernel, without need of recompilation or reboot.

Since any processor instruction may be patched, it can virtually access any information you need at any place. Dynamic tracing system logic is quite simple: you create a script in C-like language which is translated to a probe code by a compiler. Modern kernel versions have Extended Berkeley Packet Filter (eBPF) integrated (Appendix A.3), and there is experiment on using it as a platform for generating probe code [110]. This is something that will be investigated within the context of ASTRID.

4.5.2 Attack detection

The detection of unexpected or anomalous usage patterns is often symptomatic of a broad range of attacks, including Denial of Service (DoS), unauthorized access, privilege escalation, port scanning, backdoor access, etc. Historically, the detection of attacks has been based on three main methodologies:

- *known signatures*: a signature is a set of rules used to recognize known attack patterns.
- *anomaly detection*: an anomaly is a meaningful deviation from statistical patterns.
- *stateful analysis*: a protocol state machine is used to characterize the usage patterns.

A more detailed elaboration on the three detection methodologies is reported in Appendix B.2.

Different modelling and mathematical approaches can be applied for each methodology on multiple data sources, leading to many possible combinations. Table 3 reports a thorough classification taken from the literature; references to scientific papers for each approach can be found in the original article [44]. The information in Table 3 may be very useful in the design of the ASTRID architecture. As a matter of fact, depending on the complexity of the approach and desired properties (indicated respectively by the "Approach" and "Comments" columns), it indicates the types of data that should be collected (i.e., "Data" column). This may be useful to identify suitable technologies to collect such data at design time, as well as to give indication about the supported detection mechanisms at exploitation time.

Table 3. Classification of multiple IDS techniques [44].

Type	Approach	Methodology			Attacks	Data	Comments
		AD	SD	SPA			
Statistic-based	Statistics	✓	✓		B	Audit data, user profiles, usage of disk and memory	Simple, but less accuracy
	Distance-based	✓			U	Audit data, network packets	Real-time and active measurement
	Bayesian-based	✓	✓		B	Audit data, priori events, network traffic, user profiles	Optimal statistical (probabilistic) model
	Game Theory	✓			U	System's events on incidents, log events, byte sent	Self-study, control is poor
Pattern-based	Pattern Matching		✓		K	Audit records, signatures of known attacks	Simple but less flexible
	Perti Net		✓		K	Audit records, user defined known intrusion signatures	Simple concept and graphic depiction
	Keystroke monitoring		✓		K	Audit records, user profiles, keystroke logs	Using user's typing pattern
	File system checking	✓	✓		B	System, configuration, users files, log files, applications	File integrity checking

Rule-based	Rule-based	✓	✓		B	Audit records, rule patterns from user profiles and policy	Not easily created and updated
	Data Mining	✓	✓		B	Audit data, knowledge base for association rule discovery	Automatically generated models
	Model/Profile-based	✓			U	Audit records, user profiles, network packets, AP profiles	Varied modeling/profiling methods
	Support vector machine	✓	✓		B	Limited sample data, binary data	Lower false positive rate, high accuracy
State-based	State-Transition Analysis		✓		K	Audit records, state-transition diagram of known attacks	Flexibility, detect across user sessions
	User intention identification	✓			U	Audit records, user profiles	High-level task pattern
	Markov Process Model	✓			U	Audit data, sequence of system calls or commands	Probabilistic, self-training
	Protocol analysis	✓	✓	✓	T	Audit records, log files, normal usage (Model) of a protocol	Low false positive rate, less effective
Heuristic-based	Neural Networks	✓	✓		B	Audit data, sequence of commands, predict events	Self-learning, fault tolerant
	Fuzzy logic	✓			U	Audit records, network traffic (TCP/UDP/ICMP)	Configurable, scalable, flexible
	Genetic algorithms		✓		K	Audit data, known attacks expressed as binary patterns	Heuristic and evolutionary learning
	Immune systems	✓	✓		B	Audit data, sequence of system calls	Distributed, high overall security
	Swarm intelligent	✓			U	Network connection data, log file data	Bio-inspired computing intelligence

[a] SD=Signature-based detection, AD=Anomaly-based Detection, SPA=Stateful Protocol Analysis

[b] K=Known attacks, U=Unknown attacks, B=both known and unknown attacks, T=tripartite of AD, SD, and SPA.

Regarding the scope of detection, we distinguish *Host-based IDS* (HIDS), *Network-based IDS* (NIDS), and *Compound IDS* (CIDS). HIDS monitor operating systems and application, whereas NIDS monitor network segments; CIDS are a combination of HIDS and NIDS. The main characteristics of the three types of IDS are briefly summarized in Table 4, with explicit indication of their applicability in cloud environments. A more exhaustive discussion of their characteristics and architecture is included in Appendix B.2.

Table 4. Comparison among different types of IDS.

	HIDS	NIDS	CIDS/SIEM
Processing localization	Local/distributed	Remote	Remote
Architecture	Standalone	Centralized	Centralized/ Distributed
Detection scope	Single host	Network segment(s)	Network segments + multiple hosts
Detection technology	Probes/agents	Probes	HIDS/NIDS
Security context	File system events, system calls, I/O events, application and system logs.	Network packets (header values), traffic statistics (cumulative packet/byte counts per port, per host, per flow, total).	Distilled events from HIDS/NIDS (anomalies, detected attacks, selected events/statistics).
Infrastructure requirements	No extra hardware required.	Physical or virtual network switches.	HIDS, NIDS.
Strengths	Full visibility on system logs, system calls, and internal I/O channels. Visibility on encrypted network traffic. Ability to collect binary dumps. Can detect both internal and external attacks.	Wide scope (multiple hosts). Full visibility on the traffic exchanged by all hosts.	Inherits benefits of both HIDS/NIDS. Correlate events happening on different systems. Improve accuracy and likelihood of detection.
Limitations	Restricted scope (host only). Needs to install on each host.	Cannot inspect encrypted traffic. Analysis is often limited to statistics. Heavy computational load for deep inspection. Only detect external attacks.	Overhead to run multiple detection tools. Bottleneck in the central server. Overhead of communication with HIDS/NIDS. Additional management complexity.
Detection methodology*	SD, AD (even combined).	SD (major), AD, SPA.	SD, AD.
Position in cloud	VM or hypervisor.	Physical switch, virtual switch, hypervisor.	VM or server (in addition to deployment of HIDS/NIDS).

* SD=Signature-based detection, AD=Anomaly-based Detection, SPA=Stateful Protocol Analysis

Concerning the relevance for the Project, we argue that NIDS in their current form is not of interest, because one major assumption of ASTRID is to exclude access of infrastructural components (i.e., hypervisors and network switches). The application of HIDS/CIDS in their current form falls outside the Project's objectives as well, since this brings too much overhead in service deployment and execution. Instead, a centralized architecture with distributed processing is the best option. In this respect, there are two relevant aspects that may be useful for the Project. The first is the definition of what can be processed locally and what should be analysed remotely, with different implication on overhead, likelihood of detection, and accuracy (see Section 4.5.4). The second concerns the collection, aggregation, fusion, and abstraction of security-related data from heterogeneous sources. Several software frameworks are already available for this purpose, both open-source and commercial, as discussed in Section 4.4.

4.5.3 Attack prevention and mitigation

The detection of attacks and investigation of threats are the main technical processes for building local situational awareness. However, a complete cyber-security framework must also account for proper prevention, mitigation, and enforcement actions. The main concept of ASTRID is more automation in security management, hence a quick review of available solutions and paradigms may help in the definition of architectural and functional requirements.

Intrusion Prevention Systems (IPS) are used to dynamical respond to intrusions by blocking users, network packets, applications. IPS automate (at least in part) the response after detection of anomalies, hence avoiding that preliminary scanning or deception attempts turn into successful attacks. IPS may have their own detection capabilities or, as usual in practice, be integrated with IDS tools. In the last case, they are commonly indicated as Intrusion Detection and Prevention Systems (IDPS).

The enforcement of security policies leverages complementary methods. Access control and application management are already covered in other Sections (see Section 4.7), so here we only focus on the network layer.

Enforcing security policies on the network is the main task for firewalls; a short comparison among the most common paradigms is shown in Table 1 (see Appendix B.3.1).

Table 5. Comparison of different firewalling paradigms.

	Stateless packet filters	Stateful packet filters	Circuit-level gateways	ALG
Granularity	Packet	Flow	Connection	Application
Services	Packet filtering	Packet filtering	Connection filtering Packet filtering User authentication Encryption Privacy	Application filtering (allow specific functions only) Data filtering (data exchanged by the applications) User authentication Encryption Privacy
OSI layer	Network	Network	Transport	Application
Implementation	Kernel or userspace	Kernel or userspace	Kernel or userspace	Userspace

Pros	Simple implementation Very fast packet processing	Dynamic behaviour Support both TCP and UDP connections Fast packet processing in hardware and software	Support dynamic network addresses/transport ports Enable identity-based network connectivity Fast packet processing (in case packet filtering is not used) Can inspect encrypted traffic in non-transparent mode	Most secure firewalling paradigm Detailed logging on endpoints and data exchanged Deep visibility Can inspect encrypted traffic in non-transparent mode Automatically infer additional rules
Cons	Lack semantics to dynamically identify logical flows Cannot inspect encrypted traffic	More resources are needed No control on users/applications Cannot inspect encrypted traffic	Latency in setting up each new connection No transparency in case of authentication	Need specific design and implementation for each application Very high processing overhead and low performance

Stateful packet filters appear the most meaningful paradigm for ASTRID, based on the consideration that they can be easily configured by the orchestration process. Inferring ports to be opened dynamically could be done by inspecting the network traffic, but at this stage it appears simpler to rely on the orchestration metadata. In other words, orchestration may play a role similar to circuit-level gateways but checking what is allowed and what is not allowed, and then configuring stateful packet filters accordingly. No authentication is required in ASTRID, since the whole service is under the control of a single administrator.

According to the Project description, no additional appliances should be deployed in the service graph for efficiency reasons. That means that a distributed firewall paradigm should be implemented by embedding packet filtering capabilities into each Virtual Function. The Linux kernel already implements the Netfilter/Iptable framework for packet filtering, address (and port) translation, and packet mangling. Netfilter is a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack. Iptables is a generic table structure for the definition of rulesets. Each rule within an IP table consists of a number of classifiers (iptables matches) and one connected action (iptables target). Other operating systems provide their own packet filtering facilities; for example, the BSD family uses the Packet Filter (PF) framework as firewall.

When talking about distributed firewalls, it is worth mentioning OpenStack Security Groups and similar services in VMware vCloudDirector, AWS, and other cloud management software. The nice thing in these frameworks is the possibility to specify policies beyond the mere IP-based structure. For instance, Security Groups allows to open communication to all VMs in the same project, without specifying their IP addresses. This kind of capability should be explicitly considered in ASTRID.

4.5.4 Distributed frameworks for attack detection

When moving detection tasks from the local scope to a broader system scope, the amount of data and information to be stored and processed increases exponentially, so the primary challenge is scalability. In this respect, the majority of approach explicitly targets cloud deployments and/or big data techniques (see Appendix B.2.3). Consequently, the main issue is to find an optimal balance between local processing overhead and the amount of data to be transferred over the network.

The most common attitude is to have no local processing on the local side, just limiting to detection and inspection tasks. For example, Oberheide et al. [112] propose N-version protection by running multiple antivirus software in parallel at a central location. They deploy an agent that only detects new files and sends them for analysis to the central location. The use of N different engines, each with its own malware signatures, improves the detection capability, but also raises the number of false positives. Therdphapiyanak and Piromsopa [113] collect raw logs from Apache and use the Hadoop Map-Reduce framework to process large amount of Apache logs.

The opposite approach is to leave full inspection and detection on the local side. Zargar et al [114] designed a collaborative framework for cloud environments. Their work is based on common repositories for attack models, reaction policies, sharing rules, and audit logs. The repositories are used by IDPS deployed in the different domains of a cloud environment: the infrastructure, the hypervisor, and VMs. The availability of fine-grained information enables the correlation of events and attacks occurring in the different domains, and the application of coordinated mitigation actions (e.g., by dropping packets in the physical network to protect VMs from a DoS attack). However, the replication of security information both at local and global level generates a high level of redundancy and large communication overhead; further, their approach also requires federation schemes beyond simple authentication (i.e., also including access control) that is not trivial to implement in a real environment. Vieria et al. [115]deployed a standalone IDS for each node in a Grid, sharing the same data for detection (messages exchanged between nodes and logs from the middleware). Though the initial idea was a cooperative system where each node alerts the other in case of attacks, they conclude this is not necessary in their environment.

An alternative approach is based on the concept of mobile agents. Dastjerdi et al. [116]implemented a sort of local programmability, which delivered detection agents on demand for investigation. Their concept was to move code towards data, instead of data towards code. Locally, they need an execution environment (*agency*) to run multiple agents in parallel, to migrate them to other VMs, and to protect from unauthorized access.

4.6 Legal interception and forensics investigation

According to the European Telecommunications Standards Institute (ETSI) **Lawful Interception (LI)** can be defined as a security process in which a service provider or network operator collects and provides law enforcement officials with intercepted communications of private individuals or organizations [117]. Tools for lawful interception including traffic from All-IP networks are designed to detect unlawful activities, identify fraudulent calls, data retention, targeted surveillance, and study usage patterns [118].

With emerging applications that facilitate **encrypted communication** law enforcement more often faces challenges in lawfully intercepting and decrypting traffic to gain insight into criminal activities. Intercepting traffic by **wiretapping** communication lines, especially encrypted messaging communication, requires additional steps and appliances for proxying and decrypting traffic.

Computer **Forensics** is a newer form of criminal **investigation** to collect digital evidence. Forensics refers to "scientific knowledge and methods applied to the identification, collection, preservation, examination, and analysis of information stored or transmitted in binary form"; such methods must be applied "in a manner acceptable for application in legal matters" [119] [120].

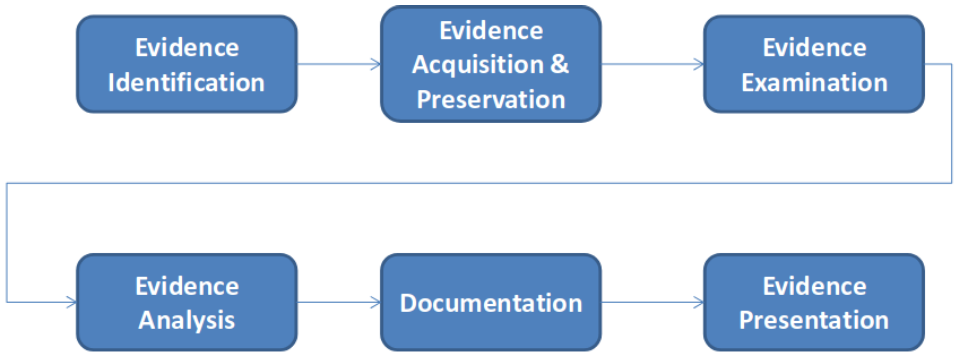


Figure 9. Digital Forensic Multi-Staged Process [120].

The two aforementioned actions are extremely critical as they imply many legal and technical issues.

In fact, legislations strictly regulate interception and access to private data, with the aim to balance the protection of people's privacy rights while ensuring societal security by preventing and fighting crimes. Unlawful access to private information constitutes in itself a crime and data collected by unlawful practices would be invalid from a legal standpoint and could not be used as an evidence in a trial.

Furthermore, from a technical standpoint, interception and digital forensic actions require the usage of a large and diverse amount of techniques and tools to collect, preserve and analyze relevant data [121]. In fact, the diversity of cyber criminal actions (such as, fraud, hacking, equipment misuse, cyber stalking, sabotage, security violations, illegal spreading of illicit materials, phishing) and of the technical characteristics of the platforms they either exploit or target (telecom networks, data centers, user terminal equipment such as smartphone or personal computers, companies' servers) may hugely complicate the research and analysis of evidences. Figure 9 illustrates a digital forensic multi-staged process.

A further important aspect is related to the care that must be applied to the collection and preservation of collected data, in order to guarantee their trustworthiness and, consequently, their legal admissibility as evidence.

Among the set of security measures foreseen in the ASTRID project in response to security incidents and attacks, there are actions that fall under the definition of 'interception of telecommunications' and 'digital forensics', which are aimed to provide evidences to law enforcement agencies for subsequent investigations,

By "interception" it is meant the "action of providing access and delivery of a subject's telecommunications and call associated data" [122], while "telecommunications" is defined as "any transfer of signs, signals, writing, images, sounds, data or intelligence of any nature transmitted in whole or in part by a wire, radio, electromagnetic, photo-electronic or photo-optical system" [122].

4.6.1 Legal framework

Legislations that protect confidentiality of digital data and telecommunications and that regulate telecommunication interception and seizure of digital evidences largely differ from country to country and may vary over time to adapt to the evolution of threats and of legislators' political orientations. For the scope of the ASTRID project, it is worth taking as references the basic provisions of two of the most advanced legal systems in the world, the United States' (US's) and the European Union's (EU's),

Both EU and US have legislations that protect the confidentiality of digital data and telecommunications. In the US, the Electronic Communications Privacy Act (ECPA) of 1986 [123] protects wire, oral, and electronic communications while those communications are being made, are in transit, and when they are stored on computers (the Act was mainly thought to apply to email, telephone conversations, and data stored electronically). Similarly, directive 2002/58/EC of the European Parliament and Council requires Member States to ensure the confidentiality of communications and the related traffic data through national legislation.

At the same time, US's and EU's legislations rule telecommunication interception and access to digital evidences. UE Council Resolution 96/C 329/01 [122] and US's Communications Assistance for Law Enforcement Act (CALEA) [124]and Stored Communication Act (SCA) [125] state that telecom and digital service providers must support access to communications and stored digital data, but only pursuant to court orders or other lawful authorizations. Similarly, with reference to the fight against digital criminality, countries participating in the Council of Europe have signed the Convention on cyber-crime [126], where they agreed to adopt legislative measures to establish the powers and procedures for the purpose of specific criminal investigations or proceedings, including the collection of electronic evidences of criminal offences. Such measures must, anyway, include safeguards for the adequate protection of human rights and liberties, such as judicial or other independent supervision, grounds justifying application and limitation of the scope and the duration of established powers or procedure.

While access to private data is strictly regulated by law and possible only following the authorization of a competent authority, EU legislation requests telecommunication providers to retain some data about network users' activity in order to allow investigation, detection, and prosecution of possible criminal offences. In fact, the EU Directive 2006/24/EC [127] requires providers to retain data related to users' usage of infrastructure resources, such as, for instance, date, time and duration of telephone calls, calling and dialled numbers, date and time of the registration in the Internet access service and the allocated Internet addresses; requested data do not include communication contents, whose interception is subject to aforementioned legal provisions. Retained data may be requested only by national competent authorities; it is up to the single Country to define such authorities and the procedures to request data.

US legislation does not force providers to retain users' activity data; yet, US telecom and digital service providers may retain users' activity logs and, in accordance to the Stored Communication Act (SCA) [125], government agencies can require access to such logs.

Users' activity logs, intercepted communications and seized private digital data need to be properly managed in order to guarantee both users' privacy right and that such data, when lawfully requested, are admissible in evidence.

As far as routinely acquired users' activity logs are concerned, both US's and EU's telecom and digital service providers cannot intentionally disclose relevant information other than to relevant authorities, following lawful procedures. In fact, such data are protected, in the EU, under the General Data Protection Regulation (GDPR) [128] and, in the US, by the same SCA [125], which prevent from any disclosure, unless there is the explicit and informed consent of involved customers. Furthermore, EU's GDPR [128] and Directive 2006/24/EC [127] also require service providers to put in place organizational and technical measures to protect data from accidental loss or alteration, or unauthorized or unlawful destruction, storage, processing, access or disclosure.

Furthermore, data that providers deliver to relevant authorities via lawful procedures must comply with determined legal requirements in order to be admissible as evidences: such requirements may (slightly or substantially) vary from country to country, depending on local legislation.

Within the EU, whilst the Convention on Mutual Assistance in Criminal Matters [129] defines a framework for the collaboration among member States for the search and seizure of evidences, including telecommunication interceptions, such Convention does not establish provisions to guarantee that evidences collected in a country (according to local laws) are admissible in another country's Court. A research funded by the European Commission [130], about the admissibility of electronic evidence in European courts, showed that EU countries' legislations include country-specific regulations that are applicable to electronic evidences, even if electronic evidences are often not explicitly mentioned. In fact, from the point of view of legal practice, in most European countries electronic evidences are considered equivalent to traditional evidences and, more in particular, to documentary evidences. With respect to the requirements for an evidence to be admitted in a trial, two models coexist in Europe: some countries establish few very general criteria for admitting evidences and a broad discretion is given to the judge; other countries regulate in a more restrictive manner the admissibility of evidences in accordance with a series of requirements established by law. In both cases, anyway, as depicted in Figure 10, taken from [130], the main criteria for the admissibility of electronic evidences refer to their 'legitimacy', 'utility', 'relevancy' and 'effectiveness'. By 'legitimacy' it is meant that the evidence must be obtained in compliance with law and with due respect of fundamental human rights, such as freedom of expression and privacy. 'Utility' and 'relevancy' requirements refer to the fact that collected evidences must be strictly related to the scope of the legal proceeding and their collection must have been done following principles of necessity and proportionality. By 'effectiveness' it is meant that evidences must be sound and capable to prove an allegation. It is important noting that the 'effectiveness' criterion is strictly correlated to techniques and procedures utilized to gather, store, analyze and provide electronic evidences; in fact, in order to prove evidences' effectiveness, it has to be proven that evidences are treated by following well documented processes and by using technically sound techniques and that they are protected from accidental or voluntary damages or unlawful modification.

In the US, the admissibility of evidences in trial is regulated by the Federal Rules of Evidence [131]. Such Rules establish that, to be admissible within a legal action, an evidence must be 'relevant' to the same action, i.e. "(a) it has any tendency to make a fact more or less probable than it would be without the evidence; and (b) the fact is of consequence in determining the action". Still, the admissibility of a 'relevant' evidence is also subject to possible limitations provided by the US Constitution, federal statutes and Supreme Court's prescriptions and when its probative value is "substantially outweighed by a danger of one or more of the following: unfair prejudice, confusing the issues, misleading the jury, undue delay, wasting time, or needlessly presenting cumulative evidence". Evidences must be clearly 'authenticated' and 'identified'; such requirement is fulfilled if, when exhibiting an item of evidence, "the proponent" produces "evidence sufficient to support a finding that the item is what the proponent claims it is"; as examples of "auto-authenticated" evidences, the Rules list, among others, "Certified Records Generated by an Electronic Process or System" (i.e. "a record generated by an electronic process or system that produces an accurate result, as shown by a certification of a qualified person") and "Certified Data Copied from an Electronic Device" (i.e. "Storage Medium, or File. Data copied from an electronic device, storage medium, or file, if authenticated by a process of digital identification, as shown by a certification of a qualified person"). The Rules also establish the role in legal action of a "witness who is qualified as an expert by knowledge, skill, experience, training, or education"; such witness "may testify in the form of an opinion or otherwise if: (a) the expert's scientific, technical, or other specialized knowledge will help the trier of fact to understand the evidence or to determine a fact in issue; (b) the testimony is based on sufficient facts or data; (c) the testimony is the product of reliable principles and methods; and (d) the expert has reliably applied the principles and methods to the facts of the case".

As a summary of the aforementioned legal provisions, we can set some preliminary general precepts about telecom and digital service providers' rights and obligations related to the access, storage and management of clients' contents and activity data:

- Access to users' private data (transmitted, stored or runtime processed) is forbidden in absence of a lawful authorization issued by a relevant authority (a law enforcement agency or a court, depending on local legislation); yet, telecom and digital service providers are obliged to guarantee access to such data in response to relevant authorities' requests;
- Users' activity may (or, depending on local legislation, must) be monitored and logged: logs automatically collected may concern users' usage of provider's physical and logical resources, they may (or, depending on local legislation, must) be retained and provided to relevant authorities to support crime investigations;
- Users' activity logs are not at free disposal of providers: disclosure and sharing of such data is subject to involved customers' explicit and informed consent; providers must put in place organizational and technical measures to protect collected data from accidental loss or damages and from unauthorized access, cancellation or modification;
- Technical and procedural measures have to be taken in order to guarantee that the collected raw data (logs or users' private data) as well the results of their analysis can constitute 'legitimate' and 'effective' evidences; particular care has to be taken in order to guarantee the 'legitimacy' of data collection procedures and the reliability and soundness of techniques and procedures related to data collection, storage and analysis as well as to the delivery of evidences to the relevant authorities.

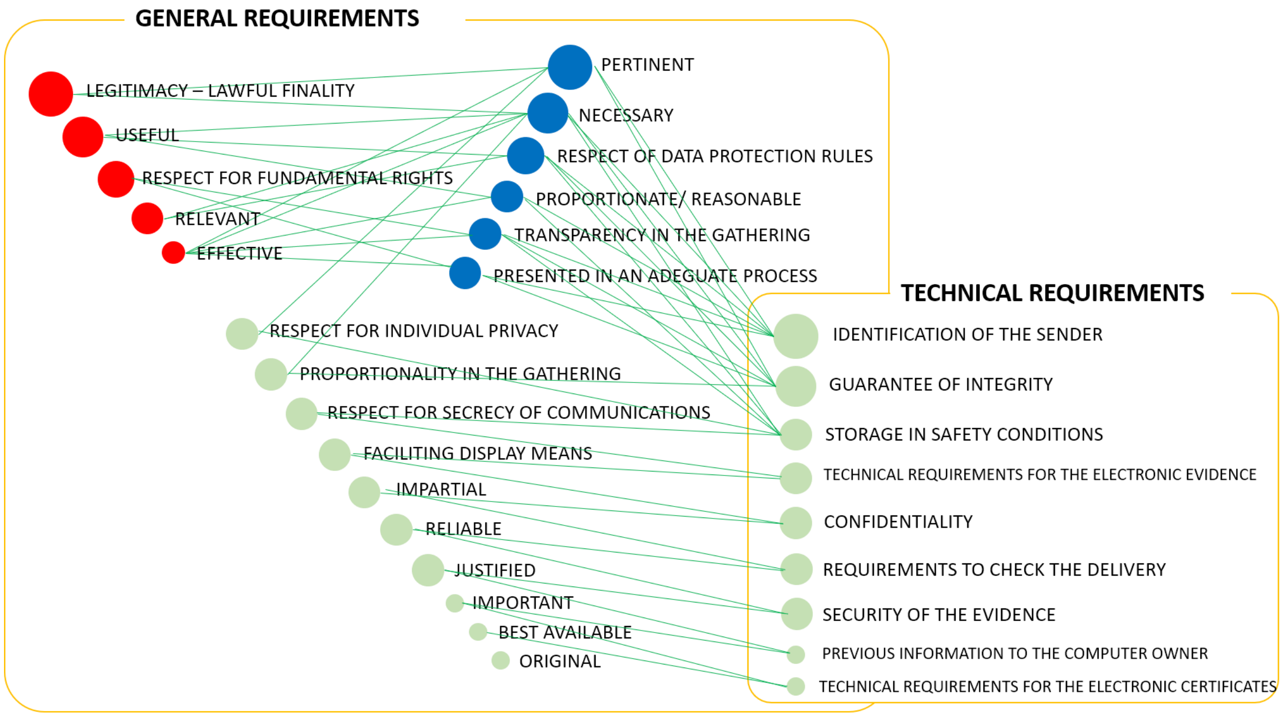

GENERAL REQUIREMENTS
LEGITIMACY – LAWFUL FINALITY
USEFUL
RESPECT FOR FUNDAMENTAL RIGHTS
RELEVANT
EFFECTIVE
PERTINENT
NECESSARY
RESPECT OF DATA PROTECTION RULES
PROPORTIONATE/ REASONABLE
TRANSPARENCY IN THE GATHERING
PRESENTED IN AN ADEQUATE PROCESS
RESPECT FOR INDIVIDUAL PRIVACY
PROPORTIONALITY IN THE GATHERING
RESPECT FOR SECRECY OF COMMUNICATIONS
FACILITING DISPLAY MEANS
IMPARTIAL
RELIABLE
JUSTIFIED
IMPORTANT
BEST AVAILABLE
ORIGINAL
TECHNICAL REQUIREMENTS
IDENTIFICATION OF THE SENDER
GUARANTEE OF INTEGRITY
STORAGE IN SAFETY CONDITIONS
TECHNICAL REQUIREMENTS FOR THE ELECTRONIC EVIDENCE
CONFIDENTIALITY
REQUIREMENTS TO CHECK THE DELIVERY
SECURITY OF THE EVIDENCE
PREVIOUS INFORMATION TO THE COMPUTER OWNER
TECHNICAL REQUIREMENTS FOR THE ELECTRONIC CERTIFICATES

Figure 10. Legal Requirements for Electronic Evidences to be admitted in Trial in Europe (from [130]).

4.6.2 Standard references

Lawful interception (LI) and forensic investigation (FI) have been subject to standardization from some of main Standard Development Organizations (SDOs).

The European Telecommunications Standards Institute (ETSI) has mainly focused on LI, following the evolution of European legislation.

Moving from the European Council Resolution 96/C 329/01 [122], ETSI has defined in TS 101 331 [132] a set of high level requirements relating to handover interfaces between Law Enforcement Agencies (LEAs) and Communication Service Providers (CSPs) in order to cover lawful interception set up and relevant data exchange. Similarly, in TS 102 656 recommendation, ETSI has fixed the requirements for the compliance to the EU Parliament and Council Directive 2006/24/EC [127], related to the handover of retained users' activity logs.

ETSI has also defined, in TS 103 280 [133] document, a dictionary of parameters that are commonly exchanged between CSPs and LEAs to set up interceptions and hand over relevant data.

The overall architecture of an interface for the exchange of LI-related data between LEAs and CSPs is defined in the ETSI TS 133 108 [134], which transposes the content of 3GPP's (Third Generation Partnership Project) TS 33.108 technical document. Such handover interface (HI), as depicted in Figure 11, adopts a three port structure such that administrative information (HI1), intercept related information (HI2), and the content of communication (HI3) are logically separated; the recommendations also foresees specific mediation functions between to interface the CSP network to LEAs' domains.

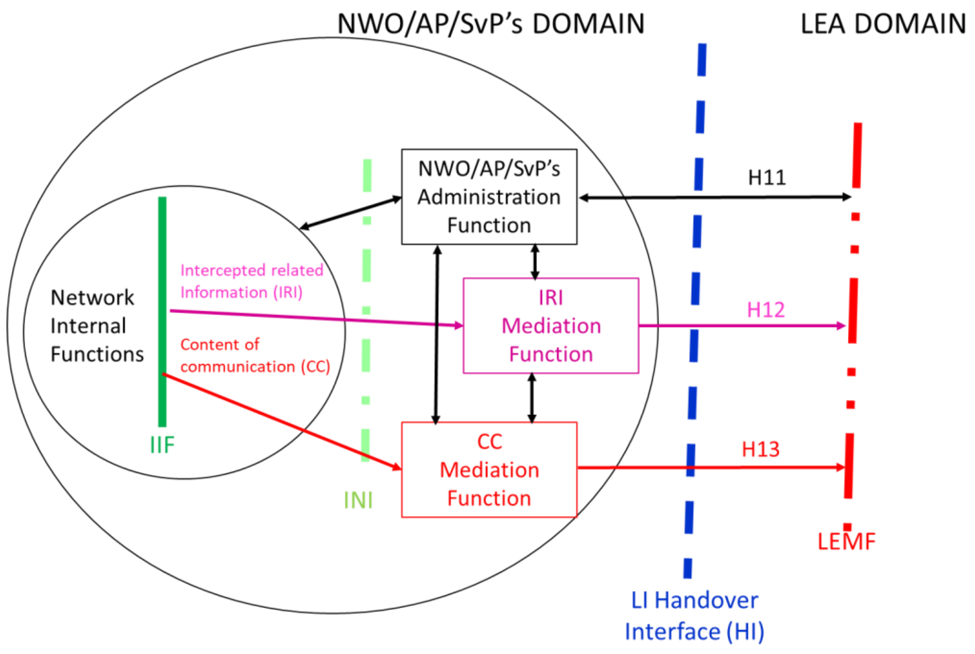

NWO/AP/SvP's DOMAIN
LEA DOMAIN
NWO/AP/SvP's Administration Function
H11
Network Internal Functions
Intercepted related Information (IRI)
Content of communication (CC)
IRI Mediation Function
H12
CC Mediation Function
H13
IIF
INI
LEMF
LI Handover Interface (HI)

Figure 11. CSP-LEA handover interface (from [135]).

Moving from aforementioned specifications, ETSI has developed a set of recommendations that define an IP-based HI interface. ETSI's TS 102 232 standard series, which is composed by seven documents, includes the high level definition of a generic interface for the handover of information related to LI from a CSP to a Law Enforcement Monitoring Facility (LEMF) [136] as well as of the specific information exchanged through such interface when the interception concerns e-mail messages [137], customers' access to Internet services [138] or to layer 2 communication services [139], IP based multi-media services [140], Public Switched Telephone Network and Integrated Services Digital Network (PSTN/ISDN) services [134] and, finally, mobile services [134]. Similarly, with respect to the handover of retained users' activity logs, ETSI's TS 102 657 [141] specification defines the relevant interface between CSPs and LEAs.

ETSI is standardizing the CSP LI-related internal interfaces. In fact, ETSI's TS 103 221 standard series defines the structure of the Internal Network Interface (INI) between the CSP mediation functions depicted in Figure 11and the devices that, being embedded in the CSP network, perform the interception: the interface has been subdivided into three sub-interfaces, which are devoted to the exchange of administrative data (sub-interface X1), of interception related information (X2) and of content of communication (X3); ETSI has officially released the standard document related to X1 sub-interface [134] and has produced an early draft of the further document referring to X2 and X3 sub-interfaces.

With reference to the need to ensure the effectiveness as an evidence of intercepted and retained data, ETSI TS 103 307 [142] lists a set of suitable techniques; such techniques relate to hashing and digital signature technologies to certify the integrity and origin of data.

As far as LI-related US standards are concerned, the web-site of the National Domestic Communications Assistance Center (NDCAC) (a National center, organized under the Department of Justice aimed to facilitate the sharing of solutions and know-how among law enforcement agencies, and strengthen law enforcement's relationships with the communications industry) provides a comprehensive lists of references [143].

Most noticeably, the J-STD-025 standard, jointly developed by the Alliance for Telecommunications Industry Solutions (ATIS) and the Telecommunications Industry Association (TIA), defines the interfaces between a Telecommunication Service Provider (TSP) and a LEA to assist the latter in conducting lawfully authorized electronic surveillance on circuit-switched voice and packet traffics transported over wireline, CDMA (Code Division Multiple Access) mobile and wireless telecommunication networks in compliance with CALEA provisions. The standard defines the TSP's and LEA's networks' interface functions [144] (TSP's Delivery Function – DF - and LEA's Collection Function - CF) used for the hand-over of call-identifying information and call content via specific data channels. J-STD-025 standard is complemented by the ATIS-1000678-v3 standard [145], which covers the definition of the interface between TSPs and LEAs for wireline Voice-over-IP (VoIP) services and by ATIS1000042 [146], a technical report that addresses LI for wireline multi-party conferences based on VoIP.

LI on cable-based networks, operated by cable operators that provide VoIP and Video-over-IP services, is regulated by Cable Television Laboratories' (CableLabs) standards, which, in two major documents, have specified the general architecture for the implementation [147] and the interfaces between operators and LEAs [148].

The diffusion of cloud-based communication (voice and messages), storage and computing services as well as the upcoming extensive usage of virtualization techniques for the implementation of telecommunication services (NFV – Network Function Virtualization), particularly relevant in the fifth generation (5G) mobile network, are pushing SDOs to review and update LI-related standardization. In fact, such technological evolution implies new challenges because of the peculiar technical characteristics of cloud and new network facilities, and because of the novel deployment and administrative practices applied to such facilities and supported services.

Regarding cloud facilities and services, the US NIST's (National Institute of Standards and Technology) SP 800-144 [149] document provides guidelines for the management of security and highlights the different domains of responsibly for cloud service providers and consumers in dependence from the relevant service models (Infrastructure as a Service - IaaS, Platform as a Service - PaaS and Software as a Service - SaaS). Furthermore, NISTIR 8006 document [150] summarizes the results of NIST's research about the forensics challenges faced by experts when responding to incidents that have occurred in a cloud-computing ecosystem.

With a higher focus on LI, ETSI TR 101 567 [117] provides with a high-level overview of the new challenges related to the support of LI in cloud and NFV environments and, more specifically for NFV environment, while ETSI GR NFV-SEC 011 identifies the overall architecture and the set of capabilities, interfaces, functions and components that can be utilized to support LI for Virtualized Network Functions (VNFs).

Similarly, ATIS 1000075 [151] technical study paints a picture of where LI technology must move in order to cope with the rise of most widespread cloud services, such as email, social networking and file sharing, and identifies the need for upgrades of the consolidated ATIS LI-related standards.

FI has been subject to broad set of standards within the International Organization for Standardization (ISO).

Though not strictly focused on LI, ISO/IEC 17025 [152], 17020 [153] and 27001 [154], which define general requirements for the operation of testing and calibration laboratories (the first) and inspection laboratories (the second) and for organizations' information security management systems (the third), constitute sound references for building reliable FI processes. Whilst usually not requested by local legislations, forensic laboratories worldwide are more and more seeking for accreditation of compliance to aforementioned standards in order to assure clients that they own valuable expertise and apply internationally recognized good practices [155] [156].

ISO has also developed a wide set of standards related to IT security, which go under the ICS (International Classification of Standards) 35.030 code (corresponding to the area of 'Information Technology - IT security') [157]; some of these recommendations specifically focused on FI, mainly covering procedural issues:

- ISO/IEC 27037 [158] provides guidelines for specific activities in the handling of digital evidence that can be of evidential value; such activities include identification, collection, acquisition and preservation of evidences and are applicable to a set of device such as digital storage media used in standard computers like hard drives, floppy disks, optical and magneto optical disks, mobile phones, Personal Digital Assistants (PDAs), Personal Electronic Devices (PEDs), memory cards, mobile navigation systems, standard computer with network connections;
- ISO/IEC 30121 [159] provides a framework for organizations on the best way to prepare for digital investigations before they occur; this International Standard applies to the development of strategic processes (and decisions) relating to the retention, availability, access, and cost effectiveness of digital evidence disclosure;
- ISO/IEC 27041 [160] defines guidance on mechanisms for ensuring that methods and processes used in the investigation of information security incidents are "fit for purpose"; it includes best practice on defining requirements, describing methods, and providing evidence that implementations of methods can be shown to satisfy requirements.;
- ISO/IEC 27042 [161] provides guidance on the analysis and interpretation of digital evidence in order to address issues of continuity, validity, reproducibility, and repeatability; it defines best practice for selection, design, and implementation of analytical processes and recording sufficient information to allow such processes to be subjected to independent scrutiny when required; it also provides guidance on appropriate mechanisms for demonstrating proficiency and competence of the investigative team.
- ISO/IEC 27043 [162] provides general rules to build common incident investigation processes across various incident investigation scenarios involving digital evidence.

4.6.3 Technological overview

Because of the sensitivity and importance of Lawful Interception (LI), several telecom equipment and solution providers include LI capabilities in their offer; for example, CISCO Catalyst 6500 switch effectively support packet duplication for LI purpose and Ericsson provides its customers with a full solution to manage LI. Other companies, such as, among many, BAE System and ELBIT, provide LI specific components and solutions, spanning mediation hardware devices (e.g., routers, intercept access points, gateways, switches), LI related software (for instance, to enable CSPs to manage the whole LI process and LEAs to analyze collected data) and infrastructures (for instance, processing and storage devices to run LI management software and securely store relevant data). All the aforementioned solutions are, of course, not for free and covered by industrial property rights.

On the other hand, some free tools to capture and analyze traffic within IP, Ethernet and Wi-Fi networks, which can be utilized as building blocks for a LI infrastructure, are available; these tools, some samples of which are listed in Annex C, implement solutions for intruding wireless networks and for filtering, collecting and analyzing IP traffic.

As far as digital forensic investigation is concerned, a broad set of tools, both free of charge and for sale, are available.

US' NIST provides an on-line search engine for forensic tools [163] classified into 28 functional categories. NIST also runs the Computer Forensic Tool Testing (CFTT) project [164], which aims to test digital forensic tools, measure their effectiveness, and certify them. Similarly, the European Union Agency for Network and Information Security (ENISA), which acts as the center of expertise for cyber security in Europe and supports EU stakeholders (EU states and private subjects) in the development and implementation of security policy, has provided a list of tools for the response to security incidents, including the support of following forensic investigations [165].

Most forensic tools cover most traditional digital forensic activities, which apply to the collection and examination of data from hardware devices such as personal computers and smartphones, servers, network storage systems and networking devices; such tools, can be classified on the basis of their scope, as per the following list [164].

- Physical Media Analysis: The analysis of physical media; it can, for instance, pertain the analysis of the layout of digital storage devices such as hard disks and memory chips; the analysis of this layer includes processing the custom layout and even recovering deleted data after it has been overwritten, for example;
- Media Management Analysis: The analysis of media management layer; examples of this activity includes examining and reconstructing hard disk partitions, the combination of multiple disks into a volume, and the integration of multiple memory chips into memory space.
- File System Analysis: The analysis of the file system, which translates the bytes and sectors of the partition to directories and files; the analysis in this layer includes, for instance, examining directory and file contents and recovering deleted files.
- Application Analysis: The analysis of the application, which translates data, typically returned from the file system, into the custom format needed by the application; analyses in this layer includes viewing log files, configuration files, images, documents and reverse engineering executables.
- Network Analysis: The analysis of the network layer, which translates the lowest level data from a physical network or wireless network to the data that is used by an application; analyses in this layer focus on network packets and logs generated by network devices.

Annex C provides some examples of free tools and toolkits that can be exploited for digital forensic investigations.

Besides the above listed practices, because of the diffusion of cloud-based services, a new discipline, usually denominated as Cloud Forensic, is taking root. With respect to such matter, most consolidated tools now available are specifically focused on cloud storage services [164], while researchers are, on one hand, exploring the capability of existing tools to effectively cope with the new scenario [166] [167]and, on the other hand, identifying requirements and solutions to enable cloud infrastructure to effectively support forensic investigation [168] [169] [170] [171].

4.7 Identity management and access control

Identity management (**IdM**) is often used interchangeably with Identity and Access Management (**IAM**). It is an umbrella term for all of the core logic around identity in a corporate environment that enables the right individuals to access the right resources at the right times and for the right reasons. IAM often refers not just to identification but to the whole suite of practices including authentication (a given user is the user they identify as), authorization (a given user has the proper permissions to access a certain piece of data) and identity federation (users can use the same identification data to access resources on related domains). IAM is becoming important in the fight to protect users/customers from security threats (e.g., identity theft, fraud, and privacy abuses) and is considered as a new perimeter [172].

4.7.1 Development Trends

Organizations are adopting emerging technologies like cloud computing, the Internet of Things (IoT) and APIs to support digital transformation. Such technologies will directly impact the next generation IAM solutions. In ASTRID, following key trends/requirements for the IAM service will be taken into account when designing the framework architecture.

Cloud-based IAM: In the world of "Everything moving to Cloud," many organizations are turning to cloud-delivered IAM solutions to provide faster time to value, reduced operational costs and flexible scaling to support millions of personal identities and things. The migration to the cloud-based IAM currently follows two fundamental approaches: cloud-native and transition. In the former cloud-native IAM providers develop from scratch all required functionalities/services. In the latter, traditional IAM solutions are updated and improved to work with cloud infrastructure. It is predicted that in the next five years, new "full-service Identity-Management-as-a-Service (IDaaS)" [173] solutions that leverage all advantages of current approaches (ease of use, scalability and full-featured) will be developed and deployed in large enterprises.

IoT-supported IAM: IAM will soon become, if not already, an integral part of each and every IoT solution [174]. According to Gartner, by 2020, the IoT is estimated to reach $2 trillion in revenue. Users, devices, gateways, applications/services all have a role to play in IoT solutions and need IAM solution capable of managing them as well as their complex digital relationships (the owners, the manufacturers, special privacy constraints, etc.). To support such requirements, next generation IAM solutions need to extend, leverage/integrate with other related systems such as device management, asset management.

Microservice and API-Based IAM: When applying to identity management, "microservice pattern design" breaks up monolithic Identity and Access Management (IAM) applications into smaller services and provides them to a variety of applications and other consumers as needed. Microservices enable an "abstraction layer" that can dramatically simplify application development, integration, and operational support [175]. In this model, IAM services and functions that are enabled in a secure, easy-to-consume manner [176].

Context and Privacy-aware IAM: Identity context is critical to effective threat detection. This includes information about user, device, environment, accessed resource, and past behaviour. Such context information can help to predict identity behaviours, to provide valuable data to detect security threats without negatively impacting the end user experience. Moreover, as managing of many sensitive data (e.g., PII – Personal Identification Information), supporting privacy concerns (e.g., how to conform to the new GDPR and other privacy regulations) will become very important for any IAM solutions.

4.7.2 Related Standards

There are supported identity and access management standards that can be used in IAM solutions. *Security Assertion Markup Language (SAML)* 2.0 is an OASIS standard for federated identity management that supports both authentication and authorization. It uses XML to make assertions between an identity provider and a relying party. Assertions can contain authentication statements, attribute statements, and authorization decision statements. SAML is very widely supported by both enterprise tools and cloud providers but can be complex to initially configure. **OAuth** [177] an open authorization protocol, is the evolving standard solution to secure API access. OAuth allows users (resource owner) to grant third-party applications (client) accessing user data (resource server) without sharing credential (e.g., password). The client could be a native mobile application, a desktop application, a server-side or browser-based web application. OAuth has widespread adoption by many service providers (e.g., Google, Twitter, Facebook, GitHub, etc.). The current version (OAuth2) keeps the overall architecture but is not compatible with previous versions (OAuth 1.0 and OAuth 1.0a). **OpenID Connect** defines a simple identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner. The current version of Open ID Connect is 1.0, and it is not compatible with previous OpenID versions. User-Managed Access (**UMA** is an OAuth-based access management protocol standard designed to give an individual a unified control point for authorizing who and what can get access to multiple sources of digital data, content, and services, no matter where all those things live. eXtensible Access Control Markup Language [178] (**XACML**) is developed by OASIS to standardize the authorization decisions in enterprise applications. XACML defines an XML-based policy language, a request / response scheme and access control architecture. The latest version is 3.0. This is a fine-grained access control standard as different parameters can be taken into account when giving authorization decision. This is achieved by adopting Attribute Based Access Control (ABAC) model, and different attributes are utilized when defining access policies in order to describe "who has access to what resources under which conditions." This simple but powerful solution also allows decoupling authorization decisions from application point of use as access policies are specified separately from where they are enforced. The System for Cross-domain Identity Management - **SCIM** (former as Simple Cloud Identity Management) defines a RESTful protocol for identity account management operations. It is designed to make managing user identities in cloud-based applications and services easier. Current version 2.0 is built on an object model where a resource is a common denominator, and all SCIM objects are derived from it.

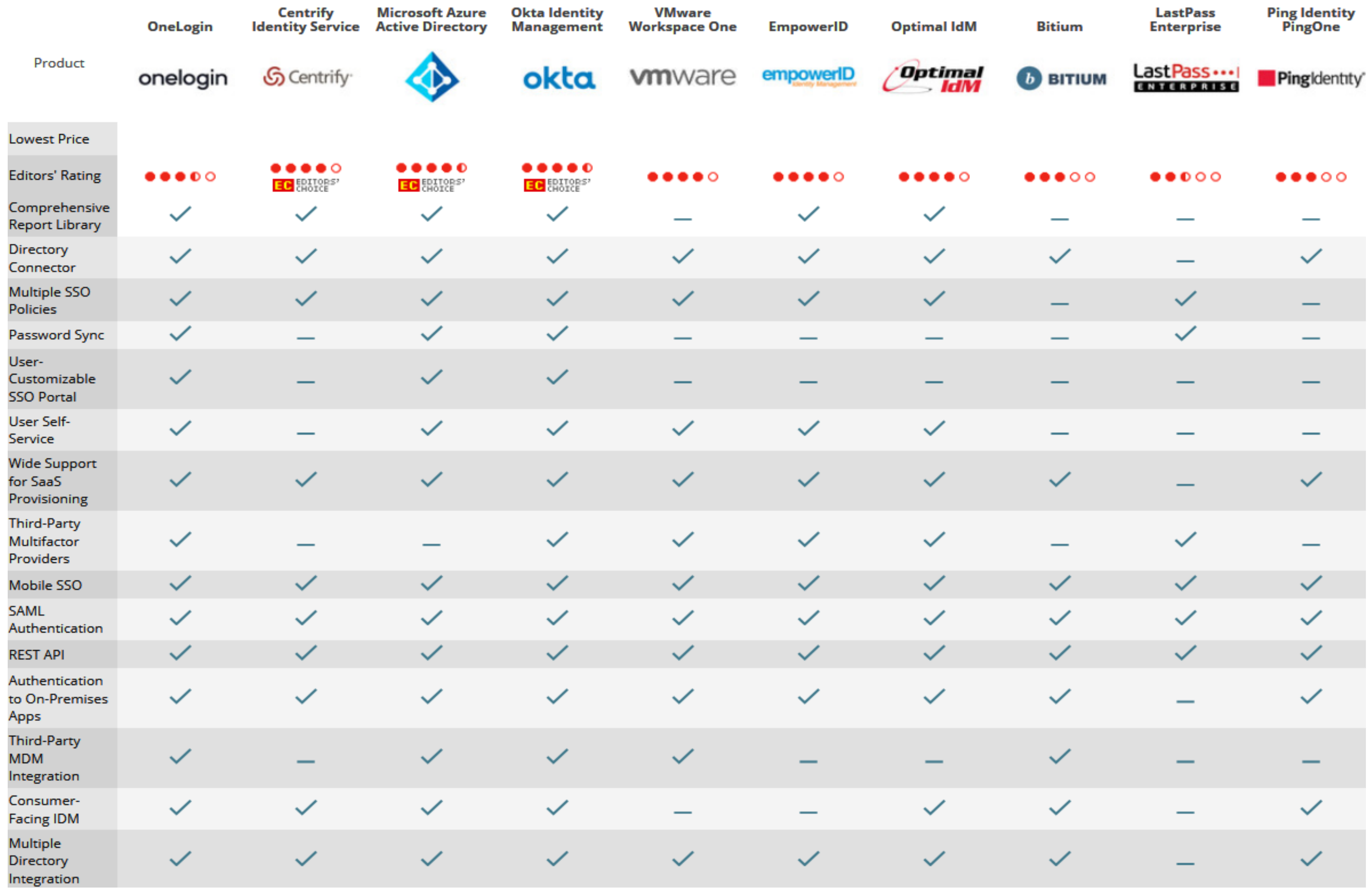
4.7.3 Related Solutions

There are many IAM tools and software solutions, and the number is still growing. The current list of top commercial IAM providers in the market can be found at [179] including company overview and contact information. They are: Avatier, Axiomatics, Beyond Trust, CA Technologies, Caradigm (Impravada), Centrify, Core Security (Courion), CyberArk, Digital Guardian, Duo, ForgeRock, Gemalto, Gigya (SAP), Hitachi ID, IBM, Lieberman Software (Bomgar), MicroFocus (NetIQ/Novell), Microsoft, Netskope, Okta, Omada, Oracle (includes Sun & Passlogix), One Identity (Quest), PingIdentity, RSA Aveksa, Radiant Logic, SailPoint, Saviynt, SecureAuth, StealthBITS and Thycotic. An interesting comparison for the list of best IAM in the year 2018 [180] by PCMag UK is presented at Figure 12.

As aforementioned, the current IAM market is growing with various solutions from various vendors and cloud-based service providers. **Axiomatics** provides externalized authorization through attribute and policy-based access control for databases, applications, and APIs. Its solutions are ideal for enterprises and government agencies that must securely share information while complying with complex and ever-evolving regulations. Axiomatics is a leader in dynamic access control through its suite of industry leading products – the Axiomatics Policy Server and the Axiomatics Data Access Filter. Axiomatics helps our global customers within healthcare, finance, manufacturing, insurance, and government agencies tackle their access control challenges with a fine-grained ABAC approach. Identity

Management from **CA Technologies** provides the ability to manage and govern user identities as well as the tools to gain control over your privileged users~~-~~across physical, virtual and cloud environments. The CA Identity Management and Governance solution includes CA Privileged Identity Manager and CA Identity Suite. **ForgeRock** identity platform makes it possible to build unique customer profiles, share consistent data company-wide, and personalize customer engagement on any device. They offer an open source identity platform architected to work as a unified system. With a single REST API for invoking any identity service, this cohesive stack is purpose-built to handle the complexity of any digital channel or application. ForgeRock's identity platform includes Access Management, Identity Management, Identity Gateway, and Directory Services. **Microsoft** Forefront Identity Manager (FIM) provides self-service identity management for users, automated lifecycle management across heterogeneous platforms for administrators, a rich policy and workflow framework, and detailed audit capabilities. Its Azure Active Directory is a comprehensive identity and access management cloud solution that provides a robust set of capabilities to manage users and groups and help secure access to on-premises and cloud applications including Microsoft online services like Office 365 and a world of non-Microsoft SaaS applications. **Okta** is an integrated identity management and mobility management service that securely and simply connects people to their applications from any device, anywhere, at anytime. The Okta service provides a deeply integrated experience across directory services, single sign-on, strong authentication, provisioning, mobility management, and reporting. It runs in the cloud on a secure, reliable, extensively audited platform and integrates with on premises applications, directories, and identity management systems. **One Identity** solutions eliminate the complexities and time-consuming processes often required to govern identities, manage privileged accounts and control access. These solutions enhance business agility while addressing IAM challenges with on-premises, cloud, and hybrid environments. **Oracle** Identity Management offers a suite of identity management solutions that allow organizations to simplify identity lifecycle management and secure access from any device for all enterprise resources – both within and beyond the firewall. Products include Oracle Identity Manager (OIM), Oracle Access Manager (OAM), Oracle Identity Governance (OIG), Oracle Directory Services (including OUD, OID & ODSEE), and Oracle Mobile Security Suite (OMSS). **Ping Identity** provides identity and access management (IAM) solutions that give customers and employees one-click access to any application from any device. Capabilities include Sign-on (SSO) and Federated Identity, Multi-Factor Authentication, Web Access Management, User Management, and Cloud Directory, User Provisioning and Deprovisioning, Mobile and API Access and Application Integrations. PingIdentity IAM products include PingOne®, PingID™, PingAccess®, and PingFederate. Identity and Access Management is the must-have service offered by all cloud-based service providers (e.g., Amazon Cognito, Google Identity, Facebook Parse). **Amazon Cognito** lets users add sign-up, sign-in, and access control to their web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0, OAuth and OpenID Connect. Amazon Cognito is HIPAA eligible and PCI DSS, SOC, ISO/EIC 27001, ISO/EIC 27017, ISO/EIC 27018, and ISO 9001 compliant.

Users can also take the benefit from available open source IAM solutions. **KeyCloak** is an open source Identity and Access Management solution aimed at modern applications and services. It makes it easy to secure applications and services with little to no code. KeyCloak provides different important features such as single-sign on, identity brokering, user federation and social login. Keycloak is based on standard protocols and provides support for OpenID Connect, OAuth 2.0, and SAML. **WSO2 Identity Server** is an open source IAM product that specializes in Access management, comprising of Identity federation, SSO, Access control, Identity Governance administration, and API security. It's highly extensible, offering the flexibility to write extensions with its rich set of connectors, gives the ability to connect to other third party and cloud systems and to support complex IAM use cases that fit any identity requirement. WSO2 IS implements all the aforementioned standards. **OpenUnison** provides open source identity management. OpenUnison quickly integrates into user continuous integration / continuous deployment (CI/CD) pipeline with a build process that generates an easily patched web application that can be run in almost any Java Servlet container (such as Tomcat, Wildfly, or Jetty) or can be run in an embedded Undertow server.



Product	OneLogin	Centrify Identity Service	Microsoft Azure Active Directory	Okta Identity Management	VMware Workspace One	EmpowerID	Optimal IdM	Bitium	LastPass Enterprise	Ping Identity PingOne
Lowest Price										
Editors' Rating	●●●◐○	●●●●○ EC EDITORS' CHOICE	●●●●◐ EC EDITORS' CHOICE	●●●●◐ EC EDITORS' CHOICE	●●●●○	●●●●○	●●●●○	●●●○○	●●◐○○	●●●○○
Comprehensive Report Library	✓	✓	✓	✓	—	✓	✓	—	—	—
Directory Connector	✓	✓	✓	✓	✓	✓	✓	✓	—	✓
Multiple SSO Policies	✓	✓	✓	✓	✓	✓	✓	—	✓	—
Password Sync	✓	—	✓	✓	—	—	—	—	✓	—
User-Customizable SSO Portal	✓	—	✓	✓	—	—	—	—	—	—
User Self-Service	✓	—	✓	✓	✓	✓	✓	—	—	—
Wide Support for SaaS Provisioning	✓	✓	✓	✓	✓	✓	✓	✓	—	✓
Third-Party Multifactor Providers	✓	—	—	✓	✓	✓	✓	—	✓	—
Mobile SSO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SAML Authentication	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REST API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Authentication to On-Premises Apps	✓	✓	✓	✓	✓	✓	✓	✓	—	✓
Third-Party MDM Integration	✓	—	✓	✓	✓	—	—	✓	—	—
Consumer-Facing IDM	✓	✓	✓	✓	—	—	✓	✓	—	✓
Multiple Directory Integration	✓	✓	✓	✓	✓	✓	✓	✓	—	✓

Figure 12. The Best Identity Management Solutions of 2018.

Various IAM-related solutions have been developed in the context of several European research projects. In FIWARE, different security generic enablers were implemented. The Keyrock Identity Management Generic Enabler [181] brings support to secure and private OAuth2-based authentication of users and devices, user profile management, privacy-preserving disposition of personal data, Single Sign-On (SSO) and Identity Federation across multiple administration domains. The Wilma proxy Generic Enabler [182] brings support of proxy functions within OAuth2-based authentication schemas. It also implements PEP functions within an XACML-based access control schema. The AuthZForce PDP/PAP Generic Enabler [183] brings support to PDP/PAP functions within an access control schema based on XACML. The AAA Fine-grained authorization enabler [184] has been specified and implemented in the 5G ENSURE project. The IAM in the SONATA project [185] was developed based on existing open source solution KeyCloak.

5 Framework requirements

5.1 Methodology

In order to steer the design of the ASTRID framework, this Section analyses the architectural requirements that come from the main concept, objectives, and application scenarios discussed in Section 3. The methodology is largely based on the FURPS+ approach [186] which classifies the architectural requirements in:

- *Functional*: they represent the main system features;
- *Usability*: they are concerned with characteristics such as aesthetics and consistency in the user interface;
- *Reliability*: they include availability, accuracy of system calculations, and the system's ability to recover from failure;
- *Performance*: they set constraints on throughput, response time, recovery time, start-up time, and shutdown time;
- *Supportability*: they are concerned with characteristics such as testability, adaptability, maintainability, compatibility, configurability, installability, scalability, and localizability;
- *others (+)*, which mainly include:
  - *Design*: they specify or constraint architectural elements and their relationship;
  - *Implementation*: they specify or constraint the coding, software, and applications used in the realization of the system;
  - *Interface*: they specify the paradigms and protocols to interact with external systems;
  - *Physical*: they put constraints on the hardware used to house the system, e.g., shape, size, weight, capability, resources.

For the correct design, implementation, and evaluation of the whole framework, it is also important to identify the group of stakeholders that each requirement comes from. Based on the project concept outlined in Section 3.6, we recall that the Project scope is to facilitate management of security aspects for virtualized services deployed by service providers, either for their own use or for external end users. In this context, it is possible to identify several internal and external stakeholders, with different roles and different implications on security aspects:

- *Cyber-security staff*: they are responsible for safe and secure operation of applications and services, including detection of security incidents, mitigation and response to attacks, investigation of new threats and security incidents. They are the primary users of the ASTRID technology, because the growing adoption of virtualization paradigms is radically changing IT environments and the legacy security models they have used for years.

- *Service developers*: they design the service graph, by translating service requirements into an application topology. Service are often produced as templates, which facilitate the re-use of the same graph in different operating environments.
- *IT operations staff*: they include systems engineers, system administrators, release engineers, DBAs, network engineers; together, they are commonly in charge of delivering services to internal and external users. With virtualized services, many provisioning tasks can now be automated by CMS; the need for agile software development and deployment is also pushing new *devops* paradigms that automate configuration and management tasks through software orchestration. In this scenario, IT operations staff is mostly absent, and this role is basically merged with service development.
- *Software developers*. They are programmers that design and develop software modules that can be chained together into graph topologies.
- *End users*: they are the final users of virtual services. They are not directly involved in any technical aspect, but they can put tight requirements on service continuity and availability in case of critical services for their business.

At this stage, only the architectural requirements that are necessary to design the ASTRID architecture have been identified. The definition of implementation-specific requirements will be reported after the design of the ASTRID architecture, in D1.2.

5.2 Taxonomy

For the purpose of homogeneity, the requirements are organized in the following template:

ID	Priority	Usage Scenario
R#-USER-TYPE	Top/Medium/Low	Application ID
Title		
Brief name of the requirement.		
Description		
This field contains the specification of the requirement (description of the purpose and goals to be fulfilled), written in a preferably concise, yet clear way.		
Notes		
This field provides useful additional information, definitions, specific benefits, limitations and references helping in the full apprehension of the requirement description.		

The **ID** field uniquely identifies the requirement within the project and ease tracking its fulfilment in the next steps of the project. The classification is based on the methodology illustrated in Section 5.1. It is encoded as follows:

- R# is an incremental identifier.
- USER indicates the target stakeholder group that the requirement comes from:
  - CS – Cyber-security staff
  - SD – Service developers/DevOps
  - SW – Software developers/programmers
  - EU – End user

- TYPE classifies the requirement according to the specific implementation issue:
    - DSG: Design requirement;
    - FUN: Functional requirement;
    - IMP: Implementation requirement;
    - IFA: Interface requirement;
    - USB: Usability requirement;
    - RLB: Reliability requirement;
    - PRF: Performance requirement;
    - SUP: Supportability requirement;
    - PHY: Physical requirement.

The **Priority** field indicates whether the requirements should be considered mandatory or not:

- T (Top) means a mandatory requirement that must be fulfilled in order to achieve the project objectives, as identified by the Description of Action, Project concept, and application scenarios;
- M (Medium) means an expected requirement that should be considered in the design, but it is not essential to demonstrate the framework (hence, it might not be present in the implementation of Use Cases);
- L (Low) identifies an optional feature that enhances the value of a commercial product.

The **Usage scenario** which application scenario (Section 3.7) the requirement is relevant to:

- SAW – Situational awareness
- DFW – Distributed Firewalling
- MON – Network monitoring
- SAF – Trust and safe execution
- RAF-Response to Attacks and Enabling Forensic Investigation
- LAW – Lawful interception

5.3 List of requirements

Low Priority

R006-CS-DSG	**Privacy and sensitive data**
R019-CS-FUN	**Autonomous operation**
R023-CS-SUP	**Cyber-Threat Intelligence (CTI)**
R025-CS-USB	**Notification of security events**
R029-CS-PRF	**Detection rate**
R030-CS-PRF	**Precision of detection**
R031-CS-PRF	**Performance of attestation services**
R038-SW-DSG	**Digital right management**
R041-LAW-FUN	**Automatic encryption of interception channels**

Medium Priority

R002-CS-DSG	Data correlation over multiple service graphs
R005-CS-DSG	Historical data
R015-CS-FUN	Distributed firewalling
R017-SD-FUN	Automatic firewall configuration
R018-SD-FUN	Dynamic modification of the service topology
R024-CS-USB	Graphical User Interface
R021-SD-FUN	Automatic enhancement of the service graph
R036-SD-DSG	Lightweight operation and small impact on the service graph
R039-SW-DSG	Integration with existing logging facilities

Top Priority

R001-CS-DSG	Data correlation over the whole graph
R003-CS-DSG	Local processing and programmability
R004-CS-DSG	Secure communication
R007-CS-DSG	Access control
R008-CS-DSG	Repository of trusted security programs
R009-CS-DSG	Collection of detection algorithms
R010-SD-DSG	Integration with orchestration tools
R011-CS-FUN	Hybrid software vulnerability analysis covering both pre- and after-deployment of software services
R012-CS-FUN	Support Vulnerability Analysis Protocol and Algorithms Agility
R013-CS-FUN	Access to heterogeneous security context
R014-CS-FUN	Packet inspection
R016-CS-FUN	Behavioural analysis through attestation techniques
R020-SD-FUN	Recovery from compromised graphs
R022-SD-IFA	REST API
R026-CS-PRF	Average time to detect compromised software
R027-CS-PRF	Average time to detect network anomalies
R028-CS-PRF	Average time to respond to attack
R032-SD-PRF	Average time to replace a single function
R033-SD-PRF	Average time to re-deploy the service graph
R034-SD-PRF	Average time to switch between redundant deployments
R035-SD-PRF	Average time to change the forwarding rules in a service chain
R037-SD-RLB	The attack surface does not increase
R040-LAW-FUN	Certification and legal validity

6 References

[1] G. Pék, L. Butty´an and B. Bencsáth, “A survey of security issues in hardware virtualization,” *ACM Computing Surveys,* vol. 45, no. 3, pp. 1-34, 1 6 2013.

[2] “5G empowering vertical industries 5G VERTICAL SECTORS”.

[3] “OpenFog Reference Architecture for Fog Computing,” 2017.

[4] R. Roman, J. Lopez and M. Mambo, “Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges,” 1 2 2016.

[5] W. Holmes and C. Day, “Foreword by Tom Corn, Senior Vice President, VMware Security Products VMware NSX Micro-segmentation,” 2017.

[6] E. Al-Shaer, H. Hamed, R. Boutaba and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” *IEEE Journal on Selected Areas in Communications,* 2005.

[7] Tran Quang Thanh, S. Covaci, T. Magedanz, P. Gouvas and A. Zafeiropoulos, “Embedding security and privacy into the development and operation of cloud applications and services,” in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, 2016.

[8] D. Montero, M. Yannuzzi, A. Shaw, L. Jacquin, A. Pastor, R. Serral-Gracia, A. Lioy, F. Risso, C. Basile, R. Sassu, M. Nemirovsky, F. Ciaccia, M. Georgiades, S. Charalambides, J. Kuusijarvi and F. Bosco, “Virtualized security at the network edge: a user-centric approach,” *IEEE Communications Magazine,* vol. 53, no. 4, pp. 176-186, 4 2015.

[9] “Software-Defined Networking: The New Norm for Networks ONF White Paper,” 2012.

[10] Limoncelli and T. A., “OpenFlow: A Radical New Idea in Networking,” *Queue,* vol. 10, no. 6, p. 40, 2012.

[11] S. Shirali-Shahreza and Y. Ganjali, “Efficient Implementation of Security Applications in OpenFlow Controller with FleXam,” in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, 2013.

[12] M. Bjorklund, “Internet Engineering Task Force (IETF) A. Bierman Request for Comments: 8040 YumaWorks Category: Standards Track,” 2017.

[13] “Software Defined Networking (SDN) Introducing ONOS-a SDN network operating system for Service Providers”.

[14] G. Bianchi, M. Bonola CNIT, U. Roma Tor Vergata, A. Capone and C. Cascone, “OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch”.

[15] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone and C. Cascone, “Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing,” 6 5 2016.

[16] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet and V. Conan, “Statesec: Stateful monitoring for DDoS protection in software defined networks,” in *2017 IEEE Conference on Network Softwarization (NetSoft),* 2017.

[17] “DDoS Detection | Kentik,” [Online]. Available: https://www.kentik.com/kentipedia/ddos-detection/.

[18] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury and V. Kumar, “Security and Privacy in Fog Computing: Challenges,” *IEEE Access,* vol. 5, pp. 19293-19304, 2017.

[19] Y. Wang, T. Uehara and R. Sasaki, “Fog Computing: Issues and Challenges in Security and Forensics,” in *2015 IEEE 39th Annual Computer Software and Applications Conference*, 2015.

[20] Y. Sun, J. Zhang, Y. Xiong and G. Zhu, “Data Security and Privacy in Cloud Computing,” *International Journal of Distributed Sensor Networks,* vol. 10, no. 7, p. 190903, 16 7 2014.

[21] D. Micciancio and Daniele, “A first glimpse of cryptography’s Holy Grail,” *Communications of the ACM,* vol. 53, no. 3, p. 96, 1 3 2010.

[22] K. Gai and M. Qiu, “Blend Arithmetic Operations on Tensor-Based Fully Homomorphic Encryption Over Real Numbers,” *IEEE Transactions on Industrial Informatics,* vol. 14, no. 8, pp. 3590-3598, 8 2018.

[23] R. Kloti, V. Kotronis and P. Smith, “OpenFlow: A security analysis,” in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013.

[24] M. Kang, E.-Y. Kang, D.-Y. Hwang, B.-J. Kim, K.-H. Nam, M.-K. Shin and J.-Y. Choi, “Formal Modeling and Verification of SDN-OpenFlow,” in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013.

[25] A. Hussein, I. H. Elhajj, A. Chehab and A. Kayssi, “SDN verification plane for consistency establishment,” in *2016 IEEE Symposium on Computers and Communication (ISCC)*, 2016.

[26] A. Khurshid, X. Zou, W. Zhou, M. Caesar and P. B. Godfrey, “VeriFlow: Verifying Network-Wide Invariants in Real Time”.

[27] P. Kazemian, G. Varghese and N. McKeown, *Header Space Analysis: Static Checking for Networks*, 2012, pp. 113-126.

[28] R. Stoenescu, M. Popovici, L. Negreanu and C. Raiciu, “SymNet,” in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM ‘16*, New York, New York, USA, 2016.

[29] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto and R. Sisto, “Formal Verification of Virtual Network Function Graphs in an SP-DevOps Context. Schahram Dustdar; Frank Leymann,” *Lecture Notes in,* pp. 253-262, 2015.

[30] W. Chareonsuk and W. Vatanawood, “Formal verification of cloud orchestration design with TOSCA and BPEL,” in *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2016.

[31] H. Yoshida, K. Ogata and K. Futatsugi, “Formalization and Verification of Declarative Cloud Orchestration,” Springer, Cham, 2015, pp. 33-49.

[32] F. Amato and F. Moscato, “Exploiting Cloud and Workflow Patterns for the Analysis of Composite Cloud Services,” *Future Generation Computer Systems,* vol. 67, pp. 255-265, 1 2 2017.

[33] G. Settanni, F. Skopik, Y. Shovgenya, R. Fiedler, M. Carolan, D. Conroy, K. Boettinger, M. Gall, G. Brost, C. Ponchel, M. Haustein, H. Kaufmann, K. Theuerkauf and P. Olli, “A collaborative cyber incident management system for European interconnected critical infrastructures,” *Journal of Information Security and Applications*, vol. 34, pp. 166-182, 1 6 2017.

[34] H. Kasai, W. Kellerer and M. Kleinsteuber, “Network Volume Anomaly Detection and Identification in Large-Scale Networks Based on Online Time-Structured Traffic Tensor Tracking,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 636-650, 9 2016.

[35] X. Liu and Z. Liu, “Evaluating Method of Security Threat Based on Attacking-Path Graph Model,” in *2008 International Conference on Computer Science and Software Engineering*, 2008.

[36] P. K. Manadhata and J. M. Wing, “An Attack Surface Metric,” *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371-386, 5 2011.

[37] M. Mohsin and Z. Anwar, “Where to Kill the Cyber Kill-Chain: An Ontology-Driven Framework for IoT Security Analytics,” in *2016 International Conference on Frontiers of Information Technology (FIT)*, 2016.

[38] X. Lin, P. Zavarsky, R. Ruhl and D. Lindskog, “Threat Modeling for CSRF Attacks,” in *2009 International Conference on Computational Science and Engineering*, 2009.

[39] W. Moody, H. Hu and A. Apon, “Defensive Maneuver Cyber Platform Modeling with Stochastic Petri Nets,” in *Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2014.

[40] Y. Li, D. E. Quevedo, S. Dey and L. Shi, “A Game-Theoretic Approach to Fake-Acknowledgment Attack on Cyber-Physical Systems,” *IEEE Transactions on Signal and Information Processing over Networks,* vol. 3, no. 1, pp. 1-11, 3 2017.

[41] F. Skopik, G. Settanni and R. Fiedler, “A problem shared is a problem halved: A survey on the dimensions of collective cyber defense through security information sharing,” *Computers & Security,* vol. 60, pp. 154-176, 1 7 2016.

[42] “OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06) for information on specification licensing through membership agreements,” 2015.

[43] J. Schoenwaelder, E. Jacobs, U. A. Bierman and E. Brocade, “RFC 6241 - Network Configuration Protocol …NETCONF—,” 2011.

[44] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications,* vol. 36, no. 1, pp. 16-24, 1 1 2013.

[45] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel and M. Rajarajan, “A survey of intrusion detection techniques in Cloud,” *Journal of Network and Computer Applications,* vol. 36, no. 1, pp. 42-57, 1 1 2013.

[46] P. Gouvas, C. Vassilakis, E. Fotopoulou and A. Zafeiropoulos, “A Novel Reconfigurable-by-Design Highly Distributed Applications Development Paradigm over Programmable Infrastructure,” in *2016 28th International Teletraffic Congress (ITC 28),* 2016.

[47] P. Bellavista, L. Foschini, R. Venanzi and G. Carella, “Extensible Orchestration of Elastic IP Multimedia Subsystem as a Service Using Open Baton,” in *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud),* 2017.

[48] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu and M. Rovatsos, “Fog Orchestration for Internet of Things Services,” *IEEE Internet Computing,* vol. 21, no. 2, pp. 16-24, 3 2017.

[49] K. Tsakalozos, C. Johns, K. Monroe, P. VanderGiessen, A. Mcleod and A. Rosales, “Open big data infrastructures to everyone,” in *2016 IEEE International Conference on Big Data (Big Data),* 2016.

[50] K. Ruan, “Cloud forensics: An overview,” *Proceedings of the 7th IFIP International Conference on Digital Forensics.*

[51] D. Barrett and G. Kipper, Virtualization and forensics : a digital forensic investigator’s guide to virtual environments, Syngress/Elsevier, 2010, p. 254.

[52] S. Covaci, M. Repetto and F. Risso, “A New Paradigm to Address Threats for Virtualized Services,” in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC),* 2018.

[53] A. Carrega, M. Repetto, F. Risso, S. Covaci, A. Zafeiropoulos, T. Giannetsos and O. Toscano, “Situational Awareness in Virtual Networks: The ASTRID Approach,” in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet),* 2018.

[54] Andrisoft, “Carrier-grade DDoS detection and mitigation software”.

[55] P. Bhatt, E. T. Yano and P. Gustavsson, “Towards a Framework to Detect Multi-stage Advanced Persistent Threats Attacks,” in *2014 IEEE 8th International Symposium on Service Oriented System Engineering,* 2014.

[56] “Cisco Stealthwatch - Cisco,” [Online]. Available: https://www.cisco.com/c/en/us/products/security/stealthwatch/index.html.

[57] “What’s New in VMware vCloud Director 8.20. Technical white paper,,” 2017.

[58] N. B. Y. B. Souayeh and A. Bouhoula, “A Fully Automatic Approach for Fixing Firewall Misconfigurations,” in *2011 IEEE 11th International Conference on Computer and Information Technology,* 2011.

[59] A. Gember-Jacobson, A. Akella, R. M. Intentionet and H. H. Liu, “Automatically Repairing Network Control Planes Using an Abstract Representation CCS CONCEPTS •Networks → Network management,” 2017.

[60] A. Basile, Cataldo; valenza, fulvio; Lioy, Antonio; Lopez, Diego; Pastor Perales, “Adding Support for Automatic Enforcement of Security Policies in NFV Networks,” *IEEE/ACM Transactions on Networking,* vol. inpress, 2019.

[61] “Security groups - OpenStack Networking Guide - current,” [Online]. Available: http://docs.ocselected.org/openstack-manuals/kilo/networking-guide/content/section_securitygroups.html.

[62] “Security Groups for Your VPC - Amazon Virtual Private Cloud,” [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html.

[63] “On-Demand, Always-on, or Hybrid? Choosing an Optimal Solution for DDoS Protection - Whitepaper - Computerworld,” [Online]. Available: https://www.computerworld.com.au/whitepaper/373159/on-demand-always-on-or-hybrid-choosing-an-optimal-solution-for-ddos-protection/?type=other&arg=0&location=featured_list.

[64] “F5 Silverline DDoS Protection”.

[65] “RFC 7296 - Internet Key Exchange Protocol Version 2 …IKEv2—,” 2014.

[66] “OpenVPN | The World’s Most Trusted Virtual Private Network,” [Online]. Available: https://openvpn.net/.

[67] K. Hamzeh, “RFC 2637 - Point-to-Point Tunneling Protocol …PPTP—,” 1999.

[68] “Cloud DDoS Protection Service: Attack Lifecycle Under the Hood,” [Online]. Available: https://www.radware.com/pleaseregister.aspx?returnurl=6442458442.

[69] J. Thomas, J. Supervisor and H. Kinkelin, “State of the Art Analysis of Defense Techniques against Advanced Persistent Threats”.

[70] “CSIRT ITALIA - HOME,” [Online]. Available: https://www.csirt-ita.it/nis.html.

[71] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand and J. Crowcroft, Unikernels: Library Operating Systems for the Cloud.

[72] “ARCADIA - A Novel Reconfigurable by Design Highly Distributed Applications Development Paradigm Over Programmable Infrastructure,” [Online]. Available: http://www.arcadia-framework.eu/.

[73] P. Gouvas, A. Zafeiropoulos, C. Vassilakis, E. Fotopoulou, G. Tsiolis, R. Bruschi, R. Bolla and F. Davoli, “Design, Development and Orchestration of 5G-Ready Applications over Sliced Programmable Infrastructure,” in *2017 29th International Teletraffic Congress (ITC 29),* 2017.

[74] R. Mijumbi, J. Serrat, J.-l. Gorricho, S. Latre, M. Charalambides and D. Lopez, “Management and orchestration challenges in network functions virtualization,” *IEEE Communications Magazine,* vol. 54, no. 1, pp. 98-105, 1 2016.

[75] R. Boutaba and I. Aib, “Policy-based Management: A Historical Perspective,” *Journal of Network and Systems Management,* vol. 15, no. 4, pp. 447-480, 14 11 2007.

[76] “RFC 3198 - Terminology for Policy-Based Management,” 2001.

[77] N. Damianou, N. Dulay, E. Lupu and M. Sloman, “The Ponder Policy Specification Language,” Springer-Verlag LNCS, 2001.

[78] “OASIS eXtensible Access Control Markup Language (XACML) TC | OASIS,” [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[79] M. Y. Becker, C. Fournet and A. D. Gordon, “SecPAL: Design and Semantics of a Decentralized Authorization Language,” 2006.

[80] Y. Bartal, A. Mayer, K. Nissim and A. Wool, “Firmato: a novel firewall management toolkit,” in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344).*

[81] P. Verma and A. Prakash, “FACE: A Firewall Analysis and Configuration Engine,” in *The 2005 Symposium on Applications and the Internet*.

[82] F. Valenza, C. Basile, D. Canavese and A. Lioy, “Classification and Analysis of Communication Protection Policy Anomalies,” *IEEE/ACM Transactions on Networking,* vol. 25, no. 5, pp. 2601-2614, 10 2017.

[83] C. Basile, A. Lioy, C. Pitscheider, F. Valenza and M. Vallini, “A novel approach for integrating security policy enforcement with dynamic network virtualization,” in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[84] F. Valenza and A. Lioy, “User-oriented Network Security Policy Specification”.

[85] “Policy-based Definition and Policy for Orchestration Initial Report”.

[86] J. Moffett and M. Sloman, “Policy hierarchies for distributed systems management,” *IEEE Journal on Selected Areas in Communications,* vol. 11, no. 9, pp. 1404-1414, 1993.

[87] P. Adao, C. Bozzato, G. D. Rossi, R. Focardi and F. Luccio, “Mignis: A Semantic Based Tool for Firewall Configuration,” in *2014 IEEE 27th Computer Security Foundations Symposium*, 2014.

[88] K. Adi, L. Hamza and L. Pene, “Automatic security policy enforcement in computer systems,” *Computers & Security,* vol. 73, pp. 156-171, 1 3 2018.

[89] M. Casado, T. Garfinkel, A. Akella, M. F. -. U. S. … and u. 2006, “SANE: A Protection Architecture for Enterprise Networks.,” *usenix.org.*

[90] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia and S. Preda, “MIRAGE: A Management Tool for the Analysis and Deployment of Network Security Policies,” Springer, Berlin, Heidelberg, 2011, pp. 203-215.

[91] J. Dean and S. Ghemawat, “MapReduce,” *Communications of the ACM,* vol. 51, no. 1, p. 107, 1 1 2008.

[92] M. B. a. M. G. G. F. Creţu-Ciocârlie, “"Hunting for problems with Artemis,",” in *in First USENIX conference on Analysis of system logs, San Diego, CA, USA, 2008*.

[93] M. Kumar and M. Hanumanthappa, “Scalable intrusion detection systems log analysis using cloud computing infrastructure,” in *2013 IEEE International Conference on Computational Intelligence and Computing Research*, 2013.

[94] H. W. a. Y. L. B. Shao, “"Trinity: a distributed graph engine on a memory cloud,",” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013.*.

[95] I. Haller, A. Slowinska, M. Neugschwandtner and H. Bos, “Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations Dowsing for overflows: A guided fuzzer to find buffer boundary violations”.

[96] S. K. Cha, T. Avgerinos, A. Rebert and D. Brumley, “Unleashing MAYHEM on Binary Code,” 2012.

[97] V. Chipounov, V. Kuznetsov and G. Candea, “S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems,” 2011.

[98] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, G. V. Uc and S. Barbara, “Driller: Augmenting Fuzzing Through Selective Symbolic Execution,” 2016.

[99] F. E. Allen, “Control flow analysis,” in *Proceedings of a symposium on Compiler optimization* -, New York, New York, USA, 1970.

[100] M. Abi-Antoun, D. Wang and P. Torr, Checking Threat Modeling Data Flow Diagrams for Implementation Conformance and Security, 2007.

[101] W. Enck, P. Gilbert, L. P. Cox, J. Jung, P. Mcdaniel and A. N. Sheth, “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones”.

[102] [184] SAFECode EU Project, “Tactical Threat Modeling,” 2017.

[103] Y. Yang, H. Zhang, M. Pan, J. Yang, F. He and Z. Li, “A Model-Based Fuzz Framework to the Security Testing of TCG Software Stack Implementations,” in *2009 International Conference on Multimedia Information Networking and Security*, 2009.

[104] G. Tóth, G. Kőszegi and Z. Hornák, “Case study: automated security testing on the trusted computing platform,” in *Proceedings of the 1st European workshop on system security - EUROSEC ‘08*, New York, New York, USA, 2008.

[105] K. E. Defrawy, G. Holland and G. Tsudik, “Remote Attestation of Heterogeneous Cyber-Physical Systems: The Automotive Use Case (Extended Abstract)”.

[106] T. J. Watson, R. Sailer, T. Jaeger, L. Van Doorn and X. Zhang, “Design and Implementation of a TCG-based Integrity Measurement Architecture,” 2002.

[107] A. Segall and Institution of Engineering and Technology, Trusted platform modules : why, when and how to use them, p. 363.

[108] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik and C. Wachsmann, “SEDA: Scalable Embedded Device Attestation”.

[109] Aleph One, “Smashing the Stack for Fun and Profit,” 1996.

[110] H. Shacham, “The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86),” ACM Press, 2007.

[111] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham and M. Winandy, Return-Oriented Programming without Returns, 2010.

[112] E. C. F. J. Jon Oberheide, ““CloudAV: N-version antivirus in the network cloud,”,” in *in Proceedings of the 17th conference on Security symposium (SS ’08), San Jose, CA — 28th July – 1st August, 2008, pp. 91-106*.

[113] J. Therdphapiyanak and K. Piromsopa, “ “Applying hadoop for log analysis toward distributed ids,”,” in In Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC ‘13), Kota Kinabalu, Malaysia, Jan. 1.

[114] H. T. J. B. J. Saman Taghavi Zargar, “ “DCDIDP: A distributed, collaborative, and data-driven intrusion detection and prevention framework for cloud computing environments,”,” in *In 7th International Conference on Collaborative Computing 2011*.

[115] A. S. C. W. C. W. Kleber Vieira, ““Intrusion Detection for Grid and Cloud Computing,”,” *IT Professional, Volume: 12 , Issue: 4 , pp. 38-43, July-Aug. 2010.*.

[116] K. A. B. S. G. H. T. Amir Vahid Dastjerdi, ““Distributed Intrusion Detection in Clouds Using Mobile Agents,”,” in *In Third International Conference on Advanced Engineering Computing and Applications in Sciences 2009*.

[117] “ETSI - TR 101 567 - Lawful Interception (LI); Cloud/Virtual Services for Lawful Interception (LI) and Retained Data (RD) | Engineering360,” [Online]. Available: https://standards.globalspec.com/std/9983240/etsi-tr-101-567.

[118] “Monitoring Tools for Lawful Interception, Data Retention, Fraud Detection, Voice Quality,” [Online]. Available: https://gl.com/telecom-test-solutions/fraudulent-call-recording-and-monitoring-system.html.

[119] M. Weiser, D. Biros and G. Mosier, “Development of A National Repository of Digital Forensic Intelligence,” *The Journal of Digital Forensics, Security and Law*, vol. 1, no. 2, 1 1 2006.

[120] “Legal Issues in Computer Forensics,” [Online]. Available: https://www.webpages.uidaho.edu/wegman/JerryWegmanPapers/Computer%20Forensics%20AA%202004.htm.

[121] S. Raghavan, “Digital forensic research: current state of the art,” *CSI Transactions on ICT*, vol. 1, no. 1, pp. 91-114, 13 3 2013.

[122] “EUR-Lex - C:1996:329:TOC - EN - EUR-Lex,” [Online]. Available: https://eur-lex.europa.eu/legal-content/SK/ALL/?uri=OJ%3AC%3A1996%3A329%3ATOC.

[123] "Electronic Communications Privacy Act of 1986," [Online]. Available: https://it.ojp.gov/PrivacyLiberty/authorities/statutes/1285.

[124] "[5] United States Congress, "Communications Assistance for Law Enforcement Act (CALEA)", 47 U.S.C § 1001-10, 1994".

[125] U. S. Congress, " "Stored Communication Act (SCA)", 18 U.S.C. §§ 2701–12," 1986.

[126] C. o. Europe, ""Convention on cybercrime", Budapest, 23, November, 2001".

[127] "EUR-Lex - 32006L0024 - EN - EUR-Lex," [Online]. Available: https://eur-lex.europa.eu/legal-content/GA/TXT/?uri=CELEX:32006L0024.

[128] "EUR-Lex - 32016R0679 - EN - EUR-Lex," [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32016R0679.

[129] "EUR-Lex - 32000F0712(02) - EN - EUR-Lex," [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32000F0712%2802%29.

[130] C. Lazaro et al., ""The admissibility of electronic evidence in court", Directorate General for Justice, Freedom and Security of the European Commission,," 2006.

[131] U. S. Code, ""Federal rules of Evidence", 1, December, 2018".

[132] Li, "TS 101 331 - V1.5.1 - Lawful Interception (LI); Requirements of Law Enforcement Agencies," 2017.

[133] Li, "TS 103 280 - V2.1.1 - Lawful Interception (LI); Dictionary for common parameters," 2017.

[134] Li, "TS 102 232-7 - V3.5.1 - Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 7: Service-specific details for Mobile Services," 2018.

[135] TSGS, "TS 133 108 - V9.4.0 - Universal Mobile Telecommunications System (UMTS); LTE; 3G security; Handover interface for Lawful Interception (LI) (3GPP TS 33.108 version 9.4.0 Release 9)," 2010.

[136] Li, "TS 102 232-1 - V3.7.1 - Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 1: Handover specification for IP delivery," 2014.

[137] Li, "TS 102 232-2 - V3.11.1 - Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 2: Service-specific details for messaging services," 2017.

[138] Li, "TS 102 232-3 - V2.3.1 - Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 3: Service-specific details for internet access services," 2011.

[139] Li, "TS 102 232-4 - V3.4.1 - Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 4: Service-specific details for Layer 2 services," 2017.

[140] Li, "TS 102 232-5 - V3.7.1 - Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 5: Service-specific details for IP Multimedia Services," 2017.

[141] Li, "TS 103 221-1 - V1.1.1 - Lawful Interception (LI); Part 1: Internal Network Interface X1 for Lawful Interception," 2017.

[142] ETSI, " TS 103 307 V1.3.1 (2018-04), "CYBER; Security Aspects for LI and RD Interfaces"".

[143] "Lawful Intercept Standards — NDCAC," [Online]. Available: https://ndcac.fbi.gov/file-repository/listandardscip-1.pdf/view.

[144] M. Sherr, G. Shah, E. Cronin, S. Clark and M. Blaze, "Can they hear me now?," in *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09,* New York, New York, USA, 2009.

[145] ATIS, " 1000678.v3.2015 "Lawfully Authorized Electronic Surveillance (LAES) for Voice over Internet Protocol in Wireline Telecommunications Networks, Version 3"," July 2015.

[146] ATIS, " 1000042 "Support for Lawfully Authorized Electronic Surveillance (LAES) of Advanced Voice over Packet (VoP) Conferencing"," March 2012.

[147] CableLabs, " PKT-SP-ES-INF-C01-140314, "Electronic Surveillance Intra-Network Specification°," March 2014.

[148] CableLabs, “PKT-SP-ES-DCI-C01-140314, “PacketCable Electronic Surveillance - Delivery Function to Collection Function Interface Specification”,” March 2014.

[149] W. Jansen and T. Grance, “Guidelines on Security and Privacy in Public Cloud Computing Special Publication 800-144”.

[150] “Forensic Science Challenges”.

[151] ATIS, “ 1000075, “Cloud Services Impacts on Lawful Interception Study”,” December 2016.

[152] “ISO/IEC 17025 - General requirements for the competence of testing and calibration laboratories,” [Online]. Available: https://www.iso.org/publication/PUB100424.html.

[153] “ISO/IEC 17020:2012 - Conformity assessment -- Requirements for the operation of various types of bodies performing inspection,” [Online]. Available: https://www.iso.org/standard/52994.html.

[154] “ISO/IEC 27001 Information security management,” [Online]. Available: https://www.iso.org/isoiec-27001-information-security.html.

[155] P. Bencivenga, ““Which Standards Are Standard? Differences between ISO/IEC 17025 and 17020 for forensic agencies”,,” *Forensic Magazine.*

[156] T. Alcock, ““Changes To Forensic Laboratory Accreditation Requirements – ISO/IEC 17025”,” *Forensic Focus.*

[157] “35.030 - IT Security,” [Online]. Available: https://www.iso.org/ics/35.030/x/.

[158] “ISO/IEC 27037:2012 - Information technology -- Security techniques -- Guidelines for identification, collection, acquisition and preservation of digital evidence,” [Online]. Available: https://www.iso.org/standard/44381.html.

[159] “ISO/IEC 30121:2015 - Information technology -- Governance of digital forensic risk framework,” [Online]. Available: https://www.iso.org/standard/53241.html.

[160] “ISO/IEC 27041:2015 - Information technology -- Security techniques -- Guidance on assuring suitability and adequacy of incident investigative method,” [Online]. Available: https://www.iso.org/standard/44405.html.

[161] “ISO/IEC 27042:2015 - Information technology -- Security techniques -- Guidelines for the analysis and interpretation of digital evidence,” [Online]. Available: https://www.iso.org/standard/44406.html.

[162] “ISO/IEC 27043:2015 - Information technology -- Security techniques -- Incident investigation principles and processes,” [Online]. Available: https://www.iso.org/standard/44407.html.

[163] “Computer Forensics Tool Catalog - Home,” [Online]. Available: https://toolcatalog.nist.gov/.

[164] “Computer Forensics Tool Testing Program (CFTT) | NIST,” [Online]. Available: https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt.

[165] ENISA, ““Incident Handling Management Handbook, Document for Teachers 1.0”,” December 2016.

[166] H. Chung, J. Park, S. Lee and C. Kang, “Digital forensic investigation of cloud storage services,” *Digital Investigation,* vol. 9, no. 2, pp. 81-95, 1 11 2012.

[167] J. Dykstra and A. T. Sherman, “Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques,” *Digital Investigation,* vol. 9, pp. S90-S98, 1 8 2012.

[168] S. Simou, C. Kalloniatis, E. Kavakli and S. Gritzalis, “Cloud Forensics: Identifying the Major Issues and Challenges,” Springer, Cham, 2014, pp. 271-284.

[169] S. Alqahtany, N. Clarke, S. Furnell and C. Reich, “A Forensic Acquisition and Analysis System for IaaS: Architectural Model and Experiment,” in *2016 11th International Conference on Availability, Reliability and Security (ARES),* 2016.

[170] S. Zawoad and R. Hasan, "Trustworthy Digital Forensics in the Cloud," *Computer,* vol. 49, no. 3, pp. 78-81, 3 2016.

[171] N. H. Ab Rahman, N. D. W. Cahyani and K.-K. R. Choo, "Cloud incident handling and forensic-by-design: cloud storage as a case study," *Concurrency and Computation: Practice and Experience,* vol. 29, no. 14, p. e3868, 25 7 2017.

[172] "Identity Is the New Perimeter — But Where's Its Firewall?," [Online]. Available: https://securityintelligence.com/identity-is-the-new-perimeter-but-wheres-its-firewall/.

[173] "The Future of Identity Management (2018-2023) - TechVision Research," [Online]. Available: https://techvisionresearch.com/project/future-identity-management-2018-2023/.

[174] "3 Market Forces Drive Adoption of Identity and Access Management in IoT - Smarter With Gartner," [Online]. Available: https://www.gartner.com/smarterwithgartner/3-market-forces-drive-adoption-of-identity-and-access-management-in-iot/.

[175] A. Cser and M. Maxim, "The Future Of Identity And Access Management Vision: The Identity And Access Management Playbook," 2017.

[176] "How Microservices Can Improve Your IAM Strategy - TechVision Research," [Online]. Available: https://techvisionresearch.com/project/how-microservices-can-improve-your-iam-strategy/.

[177] "RFC 6749 - The OAuth 2.0 Authorization Framework," 2012.

[178] "eXtensible Access Control Markup Language (XACML) Version 3.0," [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[179] "Identity and Access Management Solutions Directory,," [Online]. Available: https://solutionsreview.com/identity-management/identity-management-solutions-directory/ .

[180] "The Best Identity Management Solutions of 2018," [Online]. Available: https://uk.pcmag.com/cloud-services/71363/guide/the-best-identity-management-solutions-of-2018 .

[181] "Identity Management - KeyRock | FIWARE Catalogue," [Online]. Available: https://catalogue-server.fiware.org/enablers/identity-management-keyrock.

[182] "PEP Proxy - Wilma | FIWARE Catalogue," [Online]. Available: https://catalogue-server.fiware.org/enablers/pep-proxy-wilma.

[183] "Authorization PDP - AuthZForce | FIWARE Catalogue," [Online]. Available: https://catalogue-server.fiware.org/enablers/authorization-pdp-authzforce.

[184] "Security Enablers | 5G ENSURE," [Online]. Available: http://5gensure.eu/security-enablers.

[185] "A Holistic, Innovative Framework for Design, Development and Orchestration of 5G-ready Applications and Network Services over Sliced Programmable Infrastructure," [Online]. Available: http://www.matilda-5g.eu/.

[186] R. B. Grady and R. B., Practical software metrics for project management and process improvement, Prentice Hall, 1992, p. 270.

[187] Symantec, "Internet Security Threat Report, Volume 23," 2018.

[188] S. Mccanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," 1992.

[189] W. R. Stevens, UNIX network programming, Prentice Hall PTR, 1998.

[190] r. repson, *PF_RING User Guide Linux High Speed Packet Capture.*

[191] L. Rizzo, *netmap: A Novel Framework for Fast Packet I/O,* 2012, pp. 101-112.

[192] S. Han, K. Jang, A. Panda, S. Palkar, D. Han and S. Ratnasamy, "SoftNIC: A Software NIC to Augment Hardware," 2015.

[193] "BESS: A Virtual Switch Tailored for NFV - ppt download," [Online]. Available: https://slideplayer.com/slide/12738444/.

[194] D. Scholz, P. Emmerich and D. Raumer, "Diving into Snabb".

[195] "Exploring eBPF, IO Visor and Beyond - IO Visor Project," [Online]. Available: https://www.iovisor.org/blog/2016/04/12/exploring-ebpf-io-visor-and-beyond.

[196] "eBPF, part 1: Past, Present, and Future," [Online]. Available: https://ferrisellis.com/posts/ebpf_past_present_future/.

[197] "eBPF maps — Prototype Kernel 0.0.1 documentation," [Online]. Available: https://prototype-kernel.readthedocs.io/en/latest/bpf/ebpf_maps.html.

[198] "eBPF - extended Berkeley Packet Filter — Prototype Kernel 0.0.1 documentation," [Online]. Available: https://prototype-kernel.readthedocs.io/en/latest/bpf/.

[199] "Clarifying the differences between P4 and OpenFlow," [Online]. Available: https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.html.

[200] "P4 16 Language Specification version 1.0.0".

[201] "P4 language- eBPF Backend," [Online]. Available: https://github.com/p4lang/p4c/tree/master/backends/ebpf.

[202] W. Stallings., "Software-Defined Networks and OpenFlow.," *The Internet Protocol Journal, Vol. 16, No. 1,,* March 2013.

[203] O. Ben-Kiki and C. Evans, "YAML Ain't Markup Language (YAML™) Version 1.2," 2001.

[204] P. G. a. L. Kohnfelder, " "The threa to our products,"," 1999.

[205] A. Shostack, In "Threat Modeling: Designing for Security", 2014.

[206] Osterman, Threat Modeling Again, STRIDE, 2007.

[207] V. Saini, Q. Duan and V. Paruchuri, "THREAT MODELING USING ATTACK TREES *," 2008.

[208] Y. Chen, B. Boehm and L. Sheppard, "Value Driven Security Threat Modeling Based on Attack Path Analysis," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007.

[209] A. K. Sood and R. Enbody, "Targeted Cyber Attacks - A Superset of Advanced Persistent Threats," *IEEE Security & Privacy Magazine,* pp. 1-1, 2012.

[210] United States Congress, ""Communications Assistance for Law Enforcement Act (CALEA)", 47 U.S.C § 1001-10,," 1994.

[211] C.-C. Lo, C.-C. Huang and J. Ku, "A Cooperative Intrusion Detection System Framework for Cloud Computing Networks," in *2010 39th International Conference on Parallel Processing Workshops*, 2010.

[212] N. Nong Ye and T. Farley, "A scientific approach to cyberattack detection," *Computer,* vol. 38, no. 11, pp. 55-61, 11 2005.

[213] "2017-2018 Global Application &amp; Network Security Report," [Online]. Available: http://global.radware.com/APAC_2018_ERT_Report_EN.

[214] M. Roesch, "Snort-lightweight intrusion detection for networks,," in *in: Proceedings of the 13th USENIX conference on System administration, Seattle, Washington, 1999, pp. 229–238.*

[215] J. Ng, D. Joshi and S. M. Banik, "Applying Data Mining Techniques to Intrusion Detection," in *2015 12th International Conference on Information Technology - New Generations*, 2015.

[216] K. A. Garcia, R. Monroy, L. A. Trejo, C. Mex-Perera and E. Aguirre, "Analyzing Log Files for Postmortem Intrusion Detection," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 42, no. 6, pp. 1690-1704, 11 2012.

[217] J. Therdphapiyanak and K. Piromsopa, "Applying hadoop for log analysis toward distributed ids,," in *in In Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC '13), Kota Kinabalu, Malaysia, 2013*.

[218] F. S. a. R. F. Ivo Friedberg, "Cyber situational awareness through network anomaly detection: state of the art and new approaches.," *Elektrotechnik und Informationstechnik, Volume 132, Issue 2, March 2015, pp 101–105..*

[219] C. T. J. S. C. Y. W. Y. Z. &. R. K. Abad, "“Log correlation for intrusion detection: A proof of concept.”," in Proceedings of the 19th Annual Computer Security Applications Conference. 2003, Las Vegas, .

[220] A. a. C. A. Kott, “"The Promises and Challenges of Continuous Monitoring and Risk Scoring."," *Security & Privacy, IEEE 11.1 (2013): pp. 90-3.*

[221] T. M. T. a. U. A. Mahmood, “ "Security Analytics: Big Data Analytics for Cybersecurity: A Review of Trends, Techniques and Tools".," in *in the Proceedings of 2nd National Conference on Information Assurance (NCIA). 2013,.*

[222] “Jacobs, Adam. "The Pathologies of Big Data." Queue 7.6 (2009): 10:10,10:19”.

[223] C. Apte., "“The big (data) dig” OR/MS Today. 30.1 (Feb. 2003) pp. 24”.

[224] E. Alpaydın, “Introduction to machine learning. Cambridge, MA: MIT Press, 2014”.

[225] C. Edwards, “ "Growing Pains for Deep Learning."”.*Communications of the ACM, vol.58, no. 7, pp. 14-16.*

[226] C. Italia., “Incident notification form (in Italian).] Available at: https://www.csirt-ita.it/files/Modulo%20Notifica%20Incidente.pdf.”.

[227] M. A. S. K. M. A. Mohammad Almseidin, “Evaluation of machine learning algorithms for intrusion detection system.," In IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), 14-16 Sept. 2017, Subo.

[228] M. Suh, S. H. Park, B. Lee and S. Yang, “Building firewall over the software-defined network controller,” in *16th International Conference on Advanced Communication Technology*, 2014.

[229] J. Collings and J. Liu, “An OpenFlow-Based Prototype of SDN-Oriented Stateful Hardware Firewalls,” in *2014 IEEE 22nd International Conference on Network Protocols*, 2014.

[230] J. G. V. Pena and W. E. Yu, “Development of a distributed firewall using software defined networking technology,” in *2014 4th IEEE International Conference on Information Science and Technology*, 2014.

[231] E. C. F. J. Jon Oberheide, "“CloudAV: N-version antivirus in the network cloud,”," in *in Proceedings of the 17th conference on Security symposium (SS’08), San Jose, CA, 2008, pp. 91-106.*

[232] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236-262, 21 2016.

Annex A Technologies for fast software dataplanes

A.1 Kernel hooks for user-space dataplanes

User-space applications cannot access hardware resource directly. Historically, a number of kernel hooks have been developed to give programmers access to different layers of the kernel networking stack. More recently, specific frameworks have been designed to provide direct access to hardware resources, leveraging DMA and other complementary technologies, to shorten the pipeline and to raise the packet processing speed.

Developing software in user-space is much simpler than in kernel space. In principle, different programming languages could be used, provided that suitable bindings are available for the OS. In practice, the choice is often limited to C and derived languages. Some user-space frameworks available as libraries and are portable to different platforms.

A.1.1 Raw socket interface

The raw socket interface is available in some OSes (e.g., Linux) to send/receive packets at a lowest level than the traditional socket interface. Usually, programmers just need a high-level API to send/receive data without caring about network protocol layers and headers; however, the raw socket interface allows accessing the whole packet structure, including network layers. Indeed, there are two kinds of raw sockets, in the AF_PACKET and AF_INET communication domain. The lowest-level interface is the raw socket in the AF_PACKET domain, which gives access to all headers, including the link layer. The raw socket in the AF_INET domain only gives access to IP and upper headers. Here, we are mainly interested in the first type.

The raw socket interface gets packets directly from the network driver, by registering a handler for all or specific network protocols. Incoming packets are delivered to all handlers managing their protocol type; thus, one should mind that incoming packets are delivered both to raw sockets and the standard protocol stack.

A raw socket only filters incoming packets according to the local interface and link layer protocol type (e.g., IP, ARP); further, raw sockets allow putting the interface into promiscuous mode, to receive all packets transmitted on the local links. All other classifications must be implemented by scanning packet’s headers and content; this is quite easy to implement yet rather long if one needs a full set of rules. To make this operation more efficient, Berkeley packet filters are used (see Section A.3.1).

Normal flow of packets involves copying packets from kernel space to user space and vice versa; however, switching between kernel and user modes can be very expensive in real-time design. To this purpose, some OSes (e.g., Linux) features zero copy through memory mapping, that is network packets are stored in a memory buffer shared by kernel and applications, and synchronization is achieved through some status flags. Memory mapping for network packets is very efficient; on Linux, it provides a circular buffer mapped in user space that can be used to either send or receive packets. This way, reading packets just needs to wait for them, most of the time there is no need to issue system calls. Concerning transmission, multiple packets can be sent through one system call to get the highest bandwidth. This framework is also used by higher-layer libraries, as Pcap libraries (see Section A.1.6).

A.1.2 Data Link Provider Interface

The Data Link Provider Interface (DLPI) provides protocol-independent access to the service implemented by the data link layer [187]. Specifically, the interface is intended to support X.25 LAPB, BX.25 level 2, SDLC, ISDN LAPD, Ethernet, CSMA/CD, FDDI, token ring, token bus, Bisync, Frame Relay, ATM, Fiber Channel and HIPPI. This interface can be used both by layer 3 protocols and user applications, as shown in Figure 13.

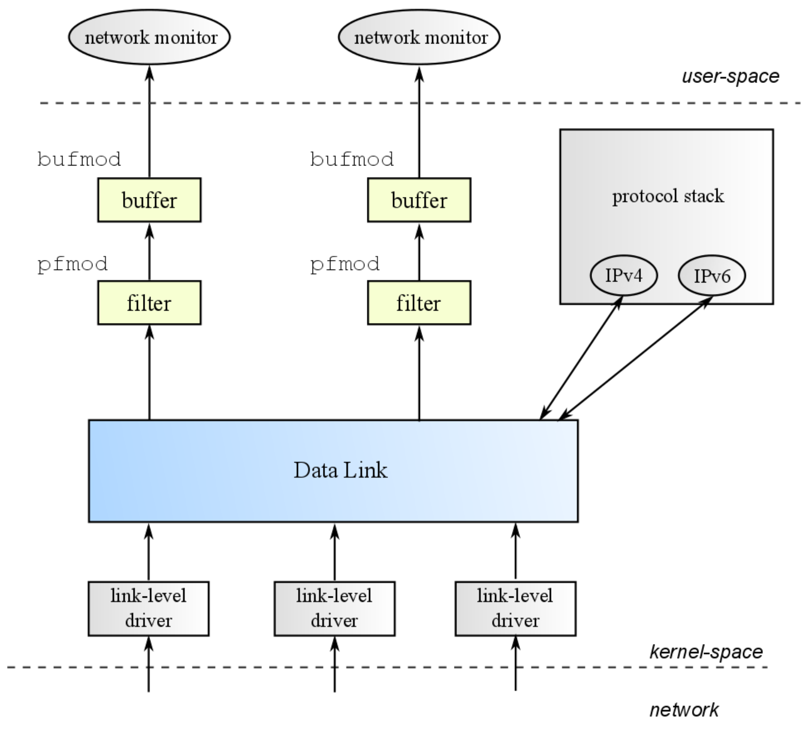


Figure 13. DLPI system architecture.

DLPI was originally designed by AT&T and is available on many SRV4 systems [188]. As shown in Figure 13, there are two main components within DLPI: pfmod which filters the packet stream before it goes to the user space, and bufmod, which buffers filtered packets between the data link layer and the user processes in order to reduce the amount of data and number of system calls. The primary advantage of DLPI over Linux raw sockets is that part of packet filtering can be performed inside the kernel space without the overhead brought by cross-boundary copy of packets. The filter model used by DLPI is straightforward and can be described as a boolean expression tree. Depending on the DLPI implementation, the packet may be copied to give it to pfmod, which may then discard it.

A.1.3 PF_RING

PF_RING is a framework derived from BPF, which has further extended this model by adding new types of filters (perfect and wildcard filters) for fine-grained packet filtering. PF_RING is tailored to the Linux OS and is designed to get the highest performance with heavy traffic load.

The main building blocks of the PF_RING's architecture are (see Figure 14) [189]:

- the accelerated kernel module that provides low-level packet copying into the PF_RING rings;
- the user-space PF_RING SDK that provides transparent PF_RING-support to user-space applications;
- specialized PF_RING-aware drivers (optional) that allow to further enhance packet capture by efficiently copying packets from the driver to PF_RING without passing through the kernel data structures. PF_RING implements a new socket type (named PF_RING) on which user-space applications can communicate with the PF_RING kernel module. In the basic mode of operation PF_RING polls packets from NICs by means of Linux NAPI. This means that NAPI copies packets from the NIC to the PF_RING circular buffer (no per-packet memory allocation/deallocation is performed), and then the userland application reads packets from the ring. In this scenario, there are two polling threads, both the application and NAPI and this results in CPU cycles used for this polling; the advantage is that PF_RING can distribute incoming packets to multiple rings (hence multiple applications) simultaneously.

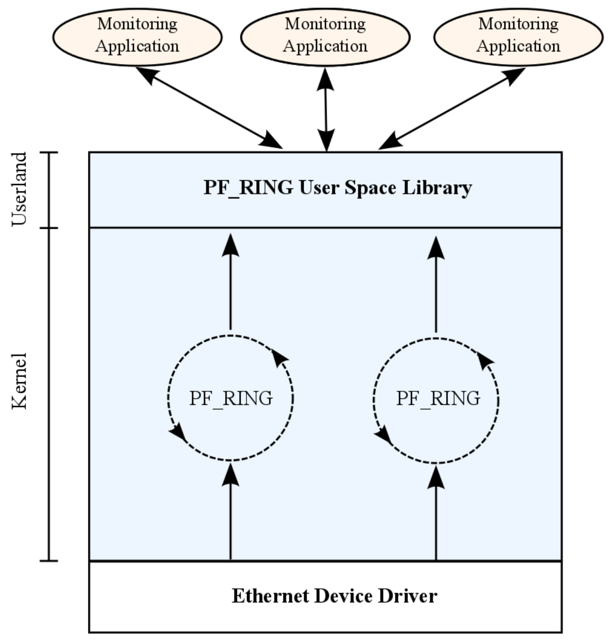


Figure 14. PF_RING architecture.

PF_RING operations can speed up by PF_RING DNA (Direct NIC Access). It is a way to map NIC memory and registers to userland so that packet copy from the NIC to the DMA ring is done by the NIC NPU (Network Process Unit) and not by NAPI. This results in better performance as CPU cycles are used uniquely for consuming packets and not for moving them off the adapter. The drawback is that only one application at time can open the DMA ring (note that modern NICs can have multiple RX/TX queues thus one can start simultaneously one application per queue), or in other words that applications in userland need to talk each other in order to distribute packets. Latest versions have also introduced Zero-Copy drivers (ZC). It is a new generation of DNA that allows packets to be read directly from the network interface by simultaneously bypassing both the Linux kernel and the PF_RING module in a zero-copy fashion. In ZC, both RX and TX operations are supported, but kernel-based functionality (BPF and PF_RING filters) are missing. The main advantage is 0% CPU utilization for copying packets to the host, because the NAPI polling mechanism is not used.

Linux networking drivers have not changed much in the past years and have not taken much advantage of new features such as kernel threads and multi-core. Many modern Ethernet cards allow to partition the incoming RX queue into several RX-queues, one per CPU core (i.e. each RX-queue is mapped to a processor core); the traffic is balanced per-flow (i.e. a flow will be sent always to the same core and not in round-robin mode) across the queues in hardware by the Ethernet cards. This means that the more CPU cores are available the more RX queues are available. PF_RING TNAPI achieves all this by starting one thread per RX queue. Received packets are then pushed to PF_RING (if available) or through the standard Linux stack. Performance evaluation shows that PF_RING+TNAPI is at least 2X faster with respect to plain PF_RING (that's much faster than native Linux NAPI).

Finally, PF_RING also includes the capability for enabling hardware filters. Some NIC are able to filter packets in hardware with ability to specify thousands of filtering rules. Unfortunately, Linux does not allow easily playing with hardware filters, thus often this feature is not used because people do not know how to access it. With PF_RING API, filters provided by the applications are set up in the NIC, if it supports hardware filters.

A.1.4 Netmap

Netmap is a framework that gives applications in user-space very fast access to network packets. Originally developed for FreeBSD, it is now available also for Linux and Windows. Netmap is implemented in the kernel and is designed to remove (or at least mitigate) the main causes that slow down packet processing: per-packet dynamic memory allocations, system call overheads, large buffer structures, memory copies. Netmap puts together several mechanisms and techniques already partially developed and adopted by similar frameworks:

- a lightweight metadata representation, which supports processing of a large number of packets in each system call;
- linear, fixed size packet buffers, which are pre-allocated;
- direct yet protected access to packet buffers, removing the need for data copies;
- large number of packets processed by a single system call.

Netmap partially disconnects the NIC from the standard network protocol stack in the kernel. In particular, the NIC uses Direct Memory Access (DMA) to move packets in packet buffers within a shared memory region allocated by netmap (see bottom of Figure 15); internally, the NIC maintains pointers in circular buffers (*rings*) for transmission and reception (there may be multiple queues/rings, depending on the specific hardware features). NIC rings are kept in sync with netmap rings in the same shared memory (top left part of Figure 15); netmap-aware applications access network packets through these netmap rings. Packet forwarding just requires moving their metadata from one ring to another, hence no copies of packet buffers are needed this results in very low-latency operation.

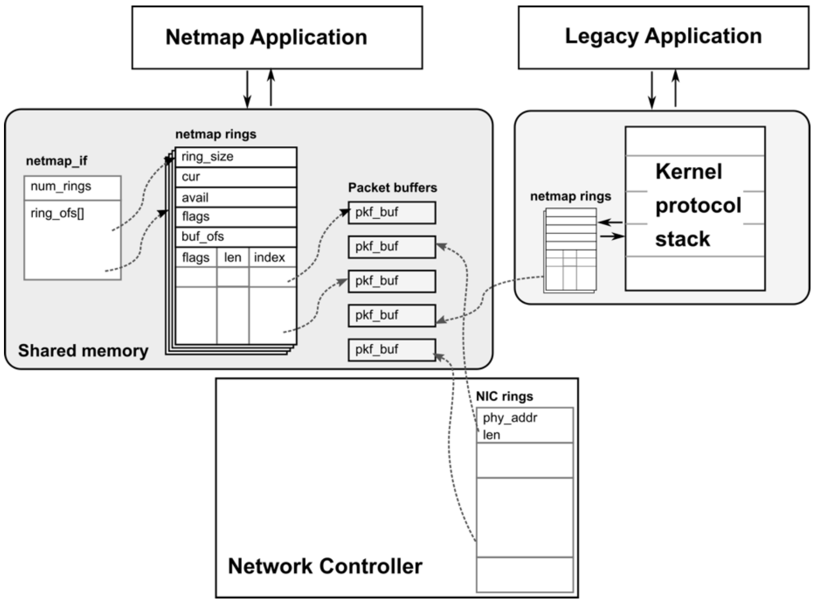


Figure 15. The netmap framework. Applications have direct access to network packets through circular buffers (rings) and packet buffers kept in a shared memory region, also used by NIC with DMA.

An additional couple of netmap rings (one for transmission and one for reception) replace the NIC rings in the standard kernel protocol stack (top right part of Figure 15), so legacy applications continue to send and receive packets on that network interface. Clearly, packets are stored in the netmap packet buffers, and no transmission occurs if not initiated by a netmap application. This setup is useful to implement security middleboxes between the kernel and the physical medium.

Netmap does not include any library. Applications directly access data structures with the help of a few macros, and issues *ioctl* system calls to send and receive packets (i.e., move them between netmap rings and the NIC). Indeed, a typical forwarding application just requires a few lines of code [189].

According to the authors, netmap is safer than other packet I/O systems (UIO-IXGBE, PR_RING-DNA, in-kernel Click) because the shared memory area does not contain critical kernel memory regions; moreover, buffer indexes and lengths are always validated by the kernel before being used. However, netmap's data structures (rings and packet buffers) are shared among all user applications. Hence, a misbehaving process can corrupt someone else's rings or packet buffers.

As regards to performance, netmap achieves full line rate on 10 Gbps interfaces, and nearly 30 Mpps on 40 Gbps NIC (limited by the hardware). Unfortunately, performance degrades when netmap is used within other packet handling frameworks, as happens for netmap-aware libpcap [189].

A.1.5 OpenOnload

OpenOnload is an application acceleration middleware. It is available as open-source for Linux but copyrighted by Solarflare and subject to licenses and patents.

Differently from other acceleration frameworks, OpenOnload is conceived for improving the performance of generic applications, rather than networking operation only. In typical architectures (see Figure 16.a), both the interface with the NIC hardware and networking protocols are implemented in the OS kernel, which gives access to applications through APIs (e.g., BSD sockets). To increase performance and reduce the overhead, by-pass architectures give direct access to the network hardware (see Figure 16.b). This is an effective solution for implementing networking processes (e.g., packet forwarding and routing), which only consider a limited number of headers and metadata; however, creating standard communication channels for generic applications is not trivial, since it requires user-safe hardware and special libraries to re-implement many network protocols (IP, TCP, UDP) in user-space. In addition, handling protocol state in the application context is a challenging task, because network state may persist longer than the application (e.g., when an application exits or crashes), may require asynchronous advancement, may be destroyed or shared (e.g., fork, exec), may need to be synchronized (ARP, IP ROUTE), may be modified by a corrupt/malicious application [190] [191]. OpenOnload uses a hybrid architecture, capable of operating at user-space and kernel-mode for any given network flow and able to choose, dynamically, whichever is appropriate. This hybrid architecture is therefore able to leverage direct access in user-space without the overhead introduced by system calls, while exploiting legacy kernel operation for asynchronous operation and socket APIs. The use of background processing in the kernel context often enables post-protocol processed results to be indicated to the user-space library with lower latency than would otherwise be possible. This feature is important for protocols such as TCP where, for example, the semantics of TCP means it is not sufficient to simply indicate that a packet has received in order to indicate that a file descriptor has data ready. In a similar way, when a thread is scheduled with a set of active sockets, a number of network operations can be performed in short order in user-space during the time-slice available to the thread, even if the stack had been previously operating in kernel mode for some or all of these sockets.

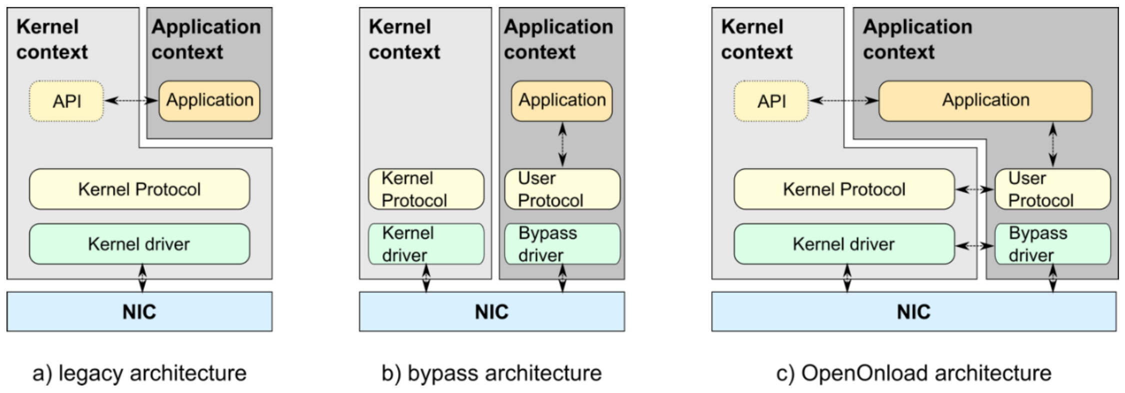


Figure 16. OpenOnload architecture compared with legacy and by-pass architectures.

Hybrid operation is based on a shared protected memory, mapped in the application context. OpenOnload is made of two main components (see Figure 17):

- a driver in kernel space, which holds the protocol state in the protected memory, registers with the Linux kernel control plane and receives full notifications of all network-related state.
- a passive library, where no threading model is imposed, that maps the protected memory in user-space and maintains a copy of the file descriptor table.

The OpenOnload library interposes the file descriptors table, a copy of which is maintained at user-space. OpenOnload forwards to the kernel any operation on pure kernel file descriptors, while handles directly operations on mixed sets of descriptors [191]. As shown in Figure 17, the OpenOnload kernel module links the shared stack state to kernel state (i.e., the kernel socket in the picture). This enables the OpenOnload stack to request resources, such as ports, which are maintained by the native kernel resident networking stack. File descriptors maintained by the OpenOnload library enables to access both the standard kernel stack (by invoking the normal API in libc) and the user-space stack (by accessing the shared memory), with the benefits already described above.

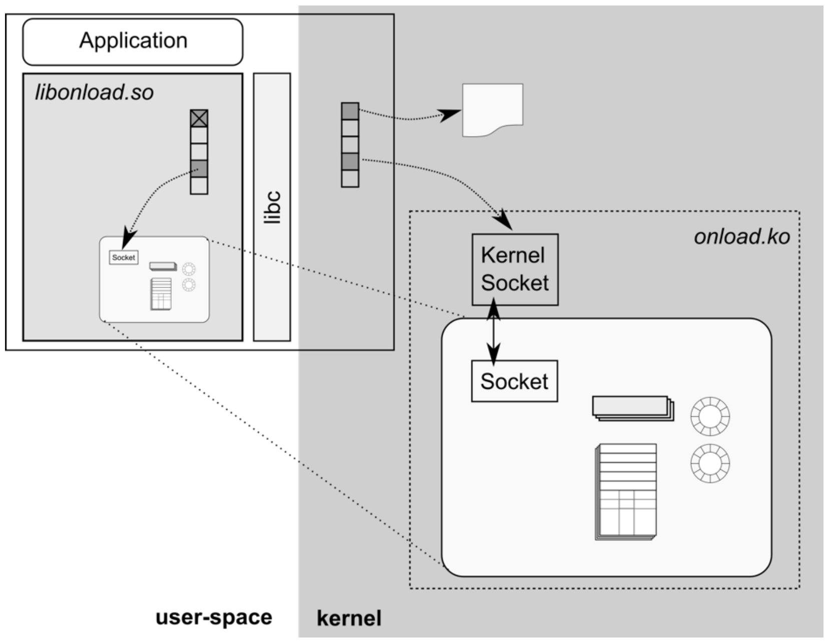


Figure 17. OpenOnload implementation.

Mapping a shared kernel memory region solves a number of practical problems concerned with the presence of multiple processes. As a matter of fact, system calls like *fork()* and *exec()* face the need to manage the process state (which is either wiped or copied). In OpenOnload, every new child process will inherit the environment of the parent and so will dynamically link against and initializes a fresh instance of the OpenOnload library, which in turn will discover and recover a memory mapping onto the shared stack protocol state. This is also useful to support shared operations on sockets: for instance, multiple processes can subscribe to and receive data from the same IP multicast groups without requiring multiple packet deliveries from hardware over the PCI bus.

Management of the physical network interface is performed using the same tools and operations as for regular kernel-mode networking. The OpenOnload kernel module registers itself with the Linux kernel control plane and receives full notifications of all network related state (and changes thereof). Such information is reflected in the shared memory region mapped in the OpenOnload library, enabling user-space access to critical state changes with very low overhead. As a result, OpenOnload is able to correctly determine the correct operation for any API call with complete transparency.

A.1.6 Pcap library

Pcap (which is for Packet Capture) library provides implementation-independent access to the underlying packet capture facility provided by the operating system. They allow a common and uniform abstraction of different mechanisms for packet filtering, thus allowing code to be portable among different OSes.

Pcap is provided as user-space library. Latest versions also allow sending packets, thus behaving just like raw packet filters or BPF. Indeed, Pcap exploits kernel-level BPF implementations where available (e.g., in FreeBSD, Linux and Windows) and provides their own user-mode interpreter to use BPF also on systems without kernel-mode support. The latest approach comes with the drawback that all packets, including those that will be filtered out, are copied from the kernel to user space, thus leading to performance degradation with respects to low level BPF implementations. User-mode BPF can also be used when reading a file previously captured using Pcap.

A.2 Dataplanes in user-space

Beyond kernel hooks, recent interest in Software-Defined Networking (SDN) and Network Function Virtualization (NFV) has fostered the implementation of programmable data-planes in user-space. There are mainly two kinds of solutions available:

- development frameworks, including libraries to access kernel and hardware capabilities that are needed to process packets at nearly wire speed (e.g., sending/receiving packets, longest prefix matching, cryptographic and hashing functions, timers, message queues);
- full data paths already conceived for typical SDN/NFV application (e.g., packet forwarding, service function chaining), which must be programmed to implement specific processing functions or chains.

Both of them usually leverage one or more kernel acceleration frameworks to achieve better performance. Data-planes conceived for SDN/NFV applications typically limit the degree of freedom in packet processing operation to “switch-like” behaviour, but may be enough to collect statistics, filter, classify or redirect packets, and other security-related operations.

A.2.1 Data-Plane Development Kit (DPDK)

DPDK provide a simple yet complete framework for fast packet processing in user-space. It proposes a large set of libraries for common packet-intensive tasks, both giving developers the potential to easily build any number of packet processing solutions with varying forwarding graphs. DPDK provides broad capabilities, while targeting performance comparable to kernel-space processing.

The main concept behind the DPDK framework is rather simple: remove any overhead brought by the intermediate kernel layer, which is designed to work with a broad and heterogeneous set of protocols. Figure 18 compares the legacy networking stack and the DPDK framework. There are two main components in the framework:

- DPDK libraries that provide a number of supporting functions (access to PCI devices, memory allocation, timers, cryptographic functions, hashing, longest prefix matching, fragmentation, etc.) to build network data paths in user applications;
- DPDK drivers, which are mainly used for device initialization and binding PCI interfaces in user-space.

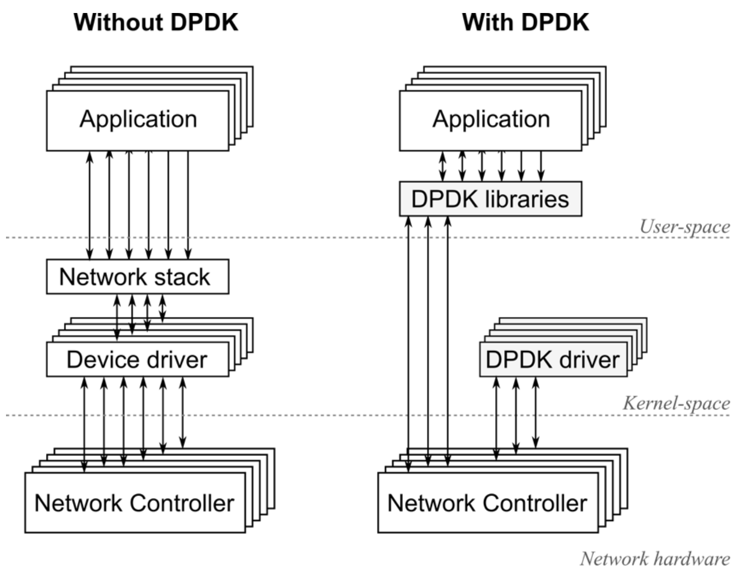


Figure 18. Comparison between standard networking stack in Linux and DPDK framework.

DPDK provides a development environment made of an abstraction layer (Environment Abstraction Layer – EAL) and several core components. The EAL library provides a common interface to applications for accessing low-level resources such as hardware and memory space, hiding the specific hardware and platform implementation. The core components include a set of libraries to manage memory rings, timers, memory allocation, packet manipulation, debug helpers. The whole framework provides a common and uniform abstraction of platform-specific capabilities (hardware and software). DPDK drivers are substantially responsible to initialize the hardware and report their capabilities to the DPDK libraries. Originally developed for Intel cards, additional drivers for kernel virtual interfaces and hardware from other vendors have been developed during the years.

Figure 19 shows the basic mechanism for packet processing in the DPDK framework. Network controllers move incoming packets to a memory pool, where they are stored in message buffers. A lockless ring buffer contains pointers to incoming packets and is used to communicate with the user-space applications. The ring implements a multi-producer/multi-consumer FIFO API in a finite size table. There may be multiple ring queues, both for receiving and transmitting. User-space applications pick packets by polling the ring; latest releases have also introduced the possibility of using interrupts. Packets are then processed without copying them around, hence improving efficiency and speed.

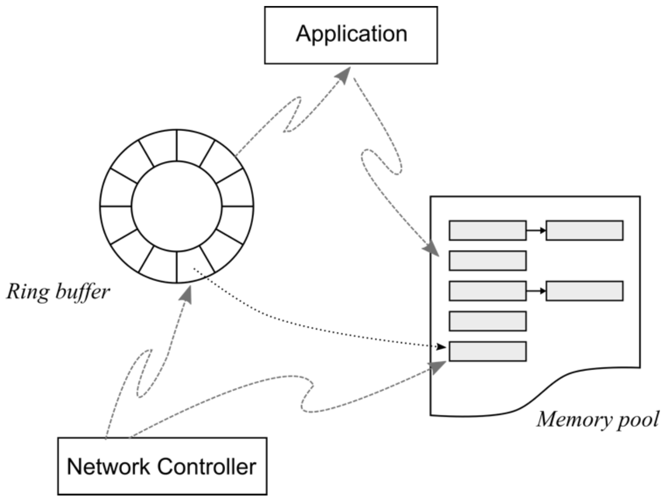


Figure 19. DPDK basic mechanism to process network packets in user-space.

The DPDK implement a run-to-completion model for packet processing. All resources must be allocated prior to calling Data Plane applications, running as execution units on logical processing cores. During initialization, execution units are assigned to threads based on their affinity and a core mask. In addition to the run-to-completion model, a pipeline model may also be used by passing packets or messages between cores via the rings. The model does not support a scheduler and all devices are accessed by polling.

A.2.2 FD.io/Vector Packet Processor

FD.io (Fast data – Input/Output) is a collection of projects and libraries to support fast packet processing in software-defined infrastructures. Its scope covers network I/O (NIC/vNIC), the control plane and the data plane. The main design target is the creation of high-throughput, low-latency and resource-efficient data paths that fit many architectures (x86, ARM, and PowerPC) and deployment environments (bare metal, VM, container).

Table 6. FD.io scope and projects chart.

Layer	Projects
Control plane	Honeycomb, hc2vpp
Data plane	NSH_SFC, ONE, TLDK, CICN, odp4vpp, VPP Sandbox, VPP
Network I/O	deb_dpdk, rpm_dpdk

The core component in the FD.io dataplane is the Vector Packet Processing (VPP) library donated by Cisco. The VPP platform is an extensible framework that provides out-of-the-box production quality switch/router functionality. The underlying concept behind VPP is the collection and processing of large batch of packets, called *vectors*, so to exploit the persistence of common information in the processor caches. With VPP, the processing pipeline is abstracted as a node graph, as shown in Figure 20. All packets in the batch are processed by each node before moving to the next one; this approach mitigates the overhead of context switches in the hardware, hence improving the overall performance (FD.io reports that independent testing shows that, at scale, VPP is two orders of magnitude faster than currently available technologies – i.e., OpenvSwitch). The graph node can be tailored to specific needs by developing additional nodes as plugins. The plugin architecture also allows to push the packet vector to a hardware accelerator.

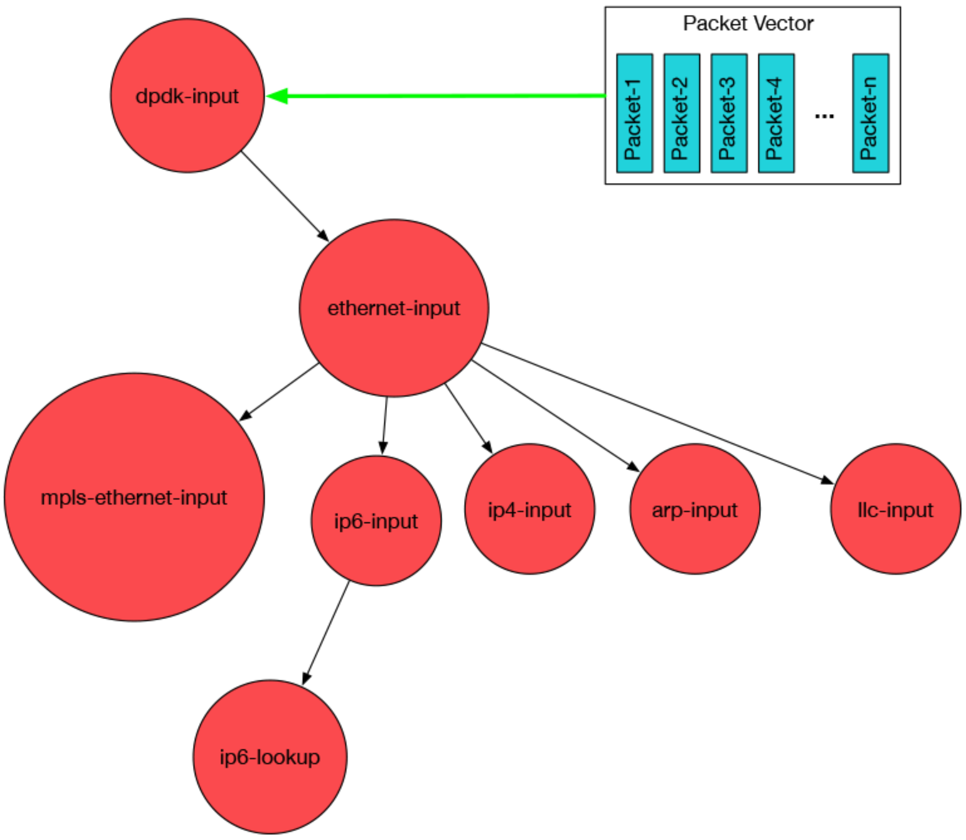


Figure 20. Packet processing graph for Packet Vectors. Source: FD.io project.

The VPP is programmable through APIs based on very efficient shared memory message bus (900k requests/s). FD.io already includes client libraries for C and Java, and a Honeycomb Agent that exposes Southbound APIs (netconf/yang, REST, BGP) for SDN controllers (e.g., OpenDayLight), as shown in Figure 21.

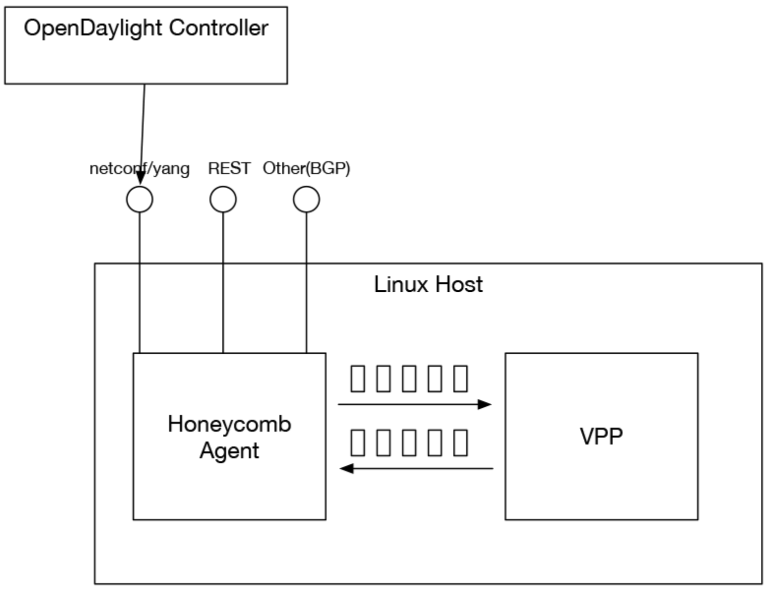


Figure 21. Programmability of VPP through message bus API and integration with SDN controllers. Source: FD.io project.

A.2.3 Open Data Plane

The Open Data Plane (ODP) is an open API for accessing heterogeneous hardware capabilities when building software data planes in user-space. It is conceived to facilitate the portability of data plane applications across a broad range of hardware platforms (including systems-on-chip and servers). Data plane applications include networking software in routers, switches, gateways, set-top boxes, Evolved Node B, as well as data-center applications as OpenvSwitch, TRex, NGiNX that can use acceleration features available in servers.

ODP includes a set of implementations for a wide range of platforms (ARM, MIPS, Power, x86, proprietary SoC architectures). ODP maintains a reference Linux implementation (*odp-linux*), preferring simplicity over performance whenever the choice is necessary, while high-performance implementations come directly from hardware vendors. The Linux implementation is a software-only implementation of ODP provided as reference for developers and to bootstrap ODP applications.

ODP abstracts hardware capabilities for data-plane processing so that applications are more portable. Application developers may make use of important hardware capabilities such as crypto hardware acceleration without needing deep knowledge of the hardware or the vendor-specific SDK associated with it. ODP is only limited to the lowest level abstractions for hardware acceleration and does not add abstractions for the Operating System. The latter is expected to be implemented in software layers above the ODP primitives.

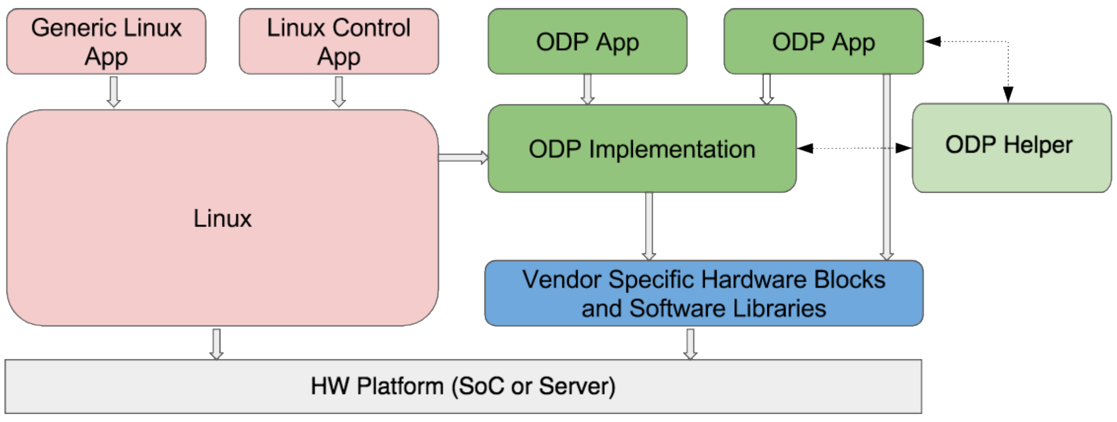

Generic Linux App
Linux Control App
ODP App
ODP App
Linux
ODP Implementation
ODP Helper
Vendor Specific Hardware Blocks and Software Libraries
HW Platform (SoC or Server)

Figure 22. Open Data Plane framework.

ODP applications compliant with the library headers can be ported to any supported system; however, they are compiled against a specific implementation. Such implementation provides an optimal mapping of APIs to the underlying capabilities (including hardware co-processing and acceleration support).

An ODP application is typically structured as shown in Figure 23.

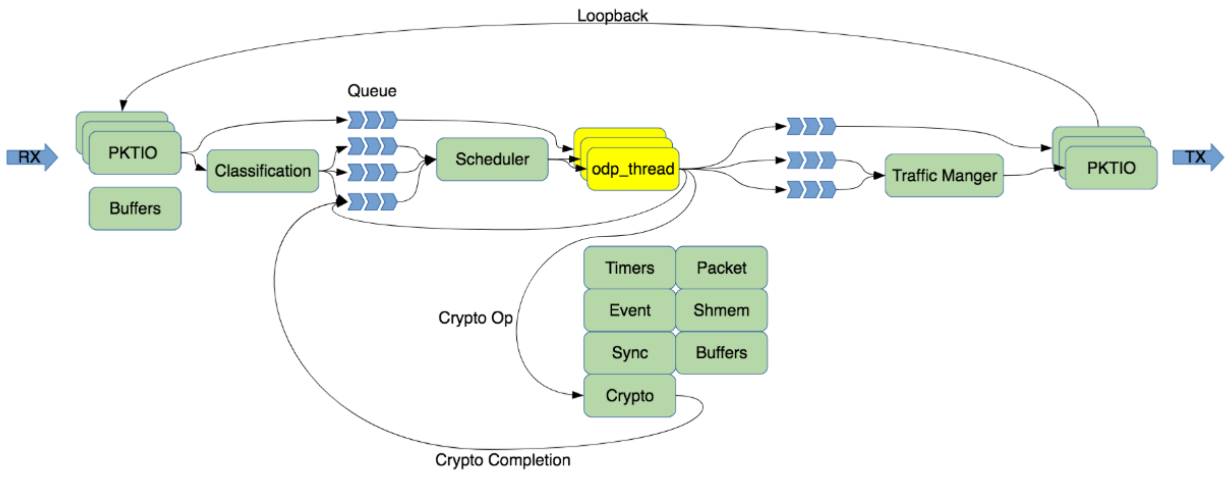


Figure 23. Typical packet flow for an ODP application.

A network data plane receives packets from physical and virtual interfaces, classifies, filters, manipulates, and eventually retransmits them. In some cases (i.e., encrypted packets that cannot be classified and processed), packets may be re-routed back to the input lines for "second pass" processing. All green boxes in Figure 23 correspond to standard operations that are typically offloaded to NIC and other hardware accelerators; instead, the yellow boxes (i.e., ODP threats) implement the application logic for manipulating the packets. Examples of processing include switching and routing decisions, firewalling and deep packet inspection, network address translation. In addition, the application logic should also control the configuration of the offloaded functions and the packet flow within them.

A.2.4 BESS

Berkeley Extensible Software Switch (BESS) is a programmable platform for vSwitch dataplane. It was originally conceived as a software NIC abstraction to hide the heterogeneity of the underlying hardware and provide fast and programmable implementation of common networking functions (e.g., checksumming, reassembly, inspection, switching, parsing) [192], by combining hardware capabilities (where available) with software modules, hence overcoming many issues in hardware implementations: protocol dependence, limited hardware resources, and incomplete/buggy/non-compliant implementation. It has now evolved to a programmable dataplane that can be used to implement high-performance software-switches but also service function chaining [193] (see Figure 24).

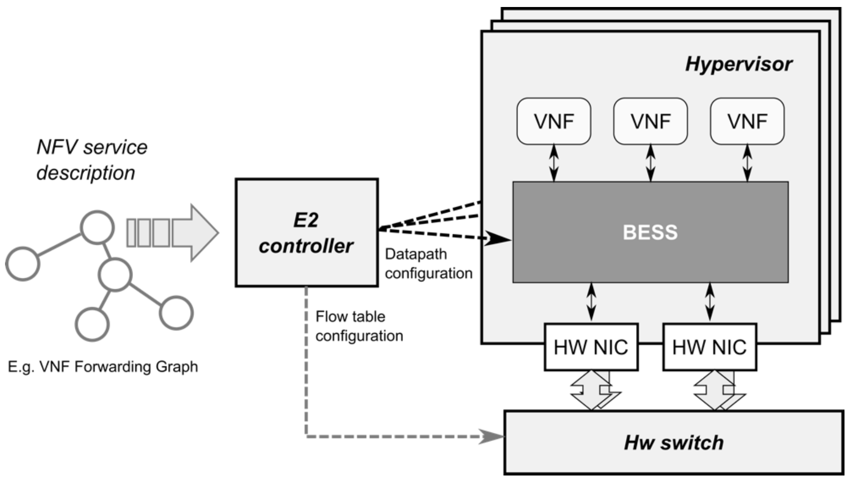


Figure 24. BESS implements a programmable data plane for high-performance switching and service function chaining.

BESS itself is not a virtual switch; it is neither pre-configured nor hardcoded to provide particular functionality, such as Ethernet bridging or OpenFlow-driven switching. Instead, it can be configured (by the CLI or external controller) to compose packet processing pipelines between virtual and physical ports. While the basic concept is similar to Click, BESS does not sacrifice performance for programmability.

Figure 25 shows the overall BESS architecture. Virtual ports (*vports*) are the interfaces between BESS and upper-layer software (both legacy applications and native BESS applications, directly hosted or running in VMs or containers). Physical ports (*pports*) are the interface between BESS and the NIC hardware and expose a set of primitives that are natively implemented in the hardware. A *vport* is an ideal NIC that implements all required features; they can be carried out in hardware, where available, or in software. The modules are combined in processing pipelines between ports (BESS can connect any combination of *vports* and *pports*). Some metadata are added to packet buffers to maintain a state while packets stream along the pipeline. Differently from other technologies, BESS only uses the set of metadata that are required by the modules in the pipeline, which is known when the pipeline is composed. Another important characteristic is resource scheduling for performance guarantees; BESS uses dedicated cores, so there are no context switches among the applications and the BESS software.

BESS runs entirely in user-space. Vports can be directly connected to native applications that bypasses the kernel and implement their own networking stack. A user-level library allows such applications to directly access vport queues, supporting zero-copy. Otherwise, a BESS driver can plug vports as standard NIC into the kernel; in this case, there is little improvement on forwarding performance because zero-copy is not trivial to be implemented in this case.

BESS leverages DPDK for high-performance packet I/O in user-space. This framework is preferred over alternative ones (PF_RING, netmap) that are not able to expose hardware NIC features besides raw packet I/O.

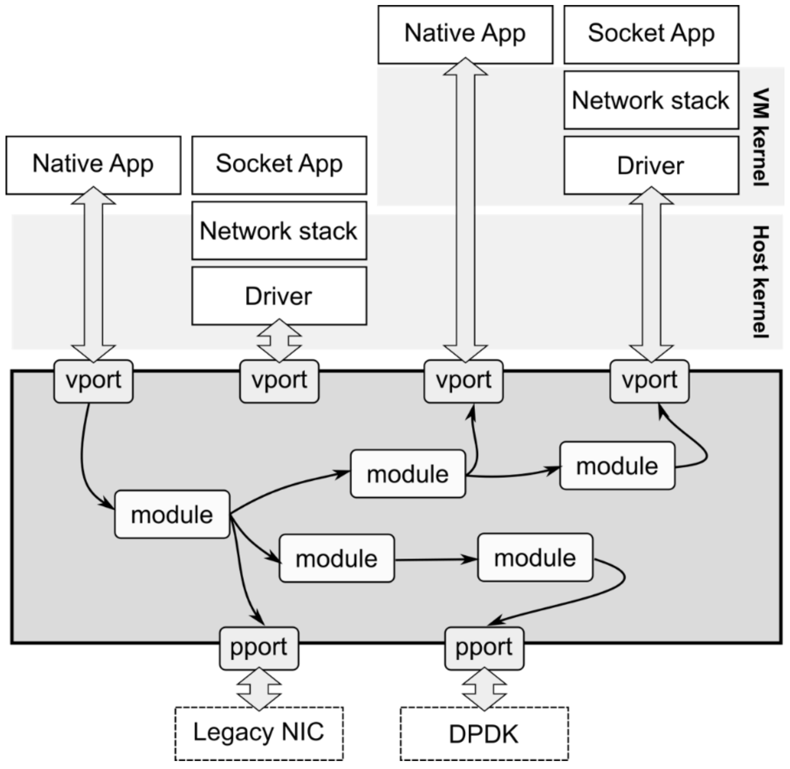


Figure 25. BESS architecture. Packets are processed by pipelines of small modules between virtual and physical ports.

In BESS, everything is programmable, not just flow tables. BESS modules are lightweight tasks that process packets and associated metadata. There are already some basic modules available (checksum offloading, TCP segmentation/reassembly, VXLAN tunnelling, rate limiter, flow steering, stateless/stateful load balancer, time stamping, IP forwarding, link aggregation, and switching), and additional modules can be developed by users in C. JSON-like structured messages are used between BESS and controller; there are 3 ways to control the datapath:

- Python/C APIs;
- Scriptable configuration language;
- Cisco iOS-like CLI.

A.2.5 Snabb

Snabb, originally conceived as an Ethernet networking toolkit (i.e., "Snabb Switch"), has now evolved to an NFV framework to create high-performance packet processing chains in user-space [194]. Snabb is available for Linux only.

The core element is a runtime environment (engine), which executes "designs". A design describes a graph of components and their configuration, corresponding to a service graph (see Figure 26). The elementary components in Snabb are "Apps". An app is an isolated implementation of a specific networking function or device; for example, a switch, a router, a NIC, or a packet filter. Designs consist in Lua scripts, while App may be written in the Lua language or may be native code objects. To support development, Snabb includes software libraries, which are collections of common utilities for Apps and graphs.

Apps must implement a specific interface, which allows the Snabb core engine to create the pipeline and manage them. The interface includes input and output, connection to hardware (NIC), reconfiguration, monitoring and management hooks (e.g., start, stop, report, reconfigure). Apps receive packets on input ports, perform some processing, and transmit packets on output ports. Each app has zero or more input and output ports. For example, a packet filter may have one input and one output port, while a packet recorder may have only an input port. Apps may also represent the I/O interface towards physical hardware (i.e., NIC) and virtual function (e.g., VMs).

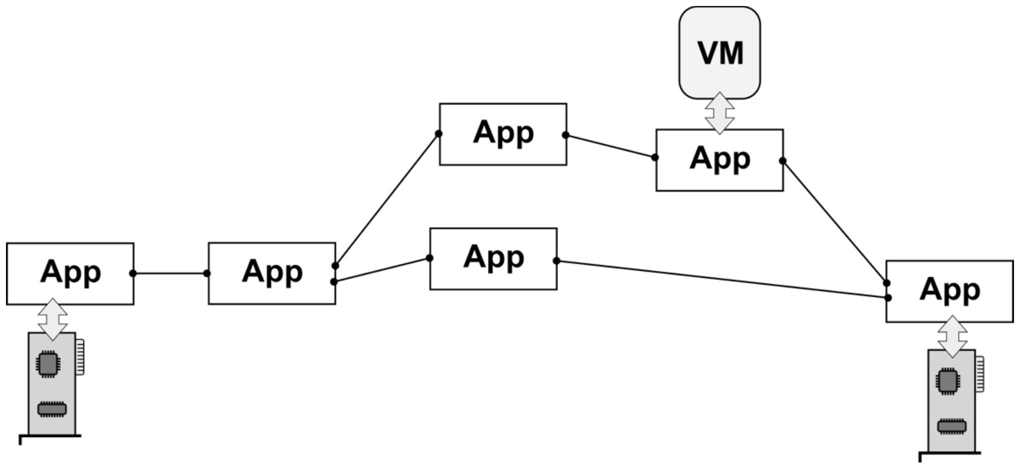


Figure 26. Layout of a typical Snabb service graph.

Snabb is entirely implemented in user-space and adopts a typical kernel bypass approach. PCI devices are configured to directly access reserved memory regions via DMA in Memory-Mapped IO (MMIO). That region hosts two ring-based FIFO structures, for transmission and reception descriptors, together with actual memory addresses where packets are stored. A peculiar characteristic of Snabb is that it does not leverage hardware features in NICs for offloading specific tasks. The motivation is the large heterogeneity in NICs and their features. Instead, Snabb makes use of Single Instruction Multiple Data (SIMD) technology, available on all modern processors, to perform the same instruction on multiple functional units in parallel. This largely mitigates the impact of memory transfers; with technologies like Intel's Data Direct I/0 (DDIO), the data can be directly loaded into the L3 cache of the CPU from which they can easily be accessed and worked with.

Snabb enables high-speed inter-VM communication, by improving the base virtio framework. As shown in Figure 27.a, the virtio framework for paravirtualization defines an interface between the frontend driver in the Guest OS and backend driver in the Hypervisor. The interface is based on a transport module and a shared memory area, where rings and queues structures are hosted. When used in the KVM/QEMU combination, device emulation is performed in user-space, hence introducing latency due to copy operations and context switches. Snabb replaces the backend driver and device emulator with a *vhost-user* App, which directly access the *vring* structures in the shared memory through a *tuntap* device (Figure 27.b). In the Snabb framework, the *vhost-user* App provides an input interface which delivers packets to the connected VM, and an output interface which receives packets sent from the VM. When the *vhost-user* App is plugged in a service graph, communication among VMs, NICs and other Snabb modules happens at very high-speed.

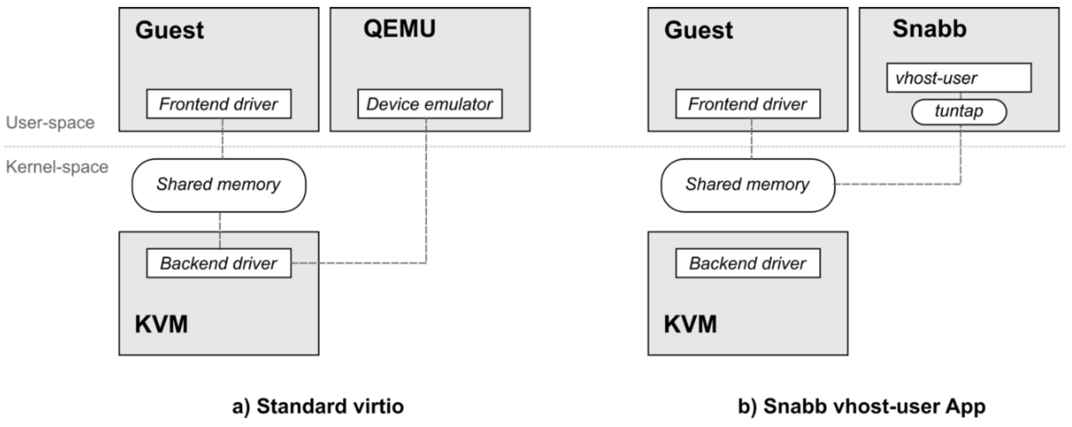


Figure 27. Comparison between standard virtio and Snabb enhancement in user-space.

Finally, Lua not only provides a programmable high-level interface, but it also improves performance. Indeed, Snabb uses the just-in-time LuaJT compilers, which is a trace based compiler that gathers profiling information during runtime to compile parts of the application in an optimized way.

A.3 Dataplanes in kernel-space

Processing packets in kernel-space shortens the pipeline and gets priority over userland processes. In addition, this leverages the huge base of networking software (e.g., routing, bridging, natting, firewalling) that has been integrated in the Linux kernel over the years, avoiding the necessity to re-implement everything from scratch such as in other approaches. However, developing software in kernel-space is more difficult because of the specific environment and the risk that any error may result in faults of the entire system. For this reason, these frameworks are usually developed by a few skilled people (often involved in the same kernel development) and consist of virtual machines that execute user programs in a controlled environment.

This approach generally leads to OS-specific solutions that are not portable to other systems.

A.3.1 Berkeley Packet Filter (BPF)

The Berkeley Packet Filter (BPF) provides on some Unix-like OSes a raw interface to data link layers in a protocol-independent fashion, and the potential to operate with custom code on the intercepted packets. All packets on the network, even those intended for other hosts, are accessible through this mechanism, provided that the network driver support promiscuous mode. BPF roughly offers a service similar to raw sockets, but it provides packet access through a file interface rather than a network interface (the packet filter appears as a character special device, like /dev/bpf0, /dev/bpf1, etc.).

The BPF was designed as a common agent to allow network monitoring by multiple applications running in user space, with the specific purpose of minimizing packets getting copied between user and kernel space, which is known to lead to large performance degradation [188]. Thus, BPF is specifically designed with efficient and effective packet filtering in mind, in order to discards unwanted packets as early as possible within the OS's stack.

Associated with each open instance of a BPF file is a user-settable packet filter. Whenever a packet is received by an interface, all file descriptors listening on that interface apply their filter. Each descriptor that accepts the packet receives its own copy. Reads from these files return the next group of packets that have matched the filter[3]. Consequently, any modification performed on captured packets do not influence the actual data, as in BPF packets are always a copy of the original traffic.

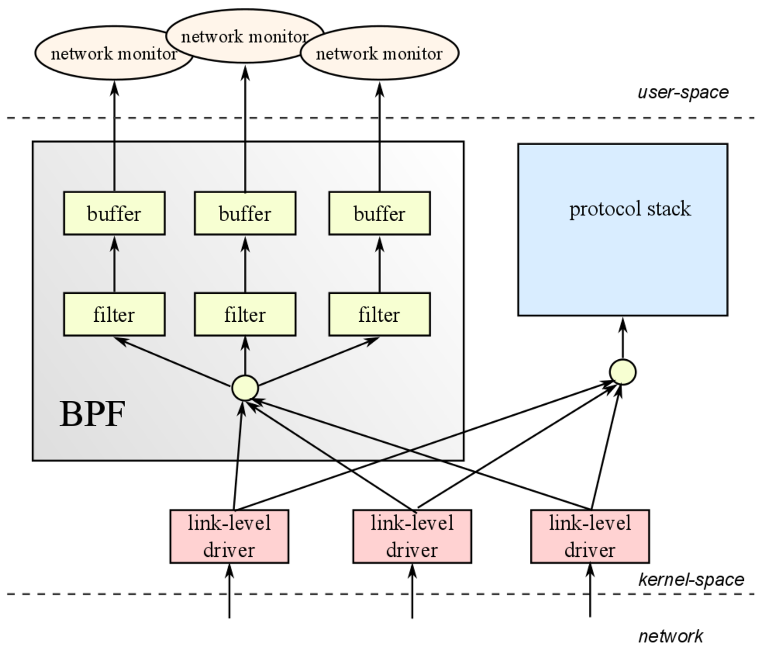


Figure 28. Overview of BPF architecture.

BPF has two main components: the network tap and the packet filter. The network tap collects copies of packets from the network device drivers and delivers them to listening applications. The filter decides if a packet should be accepted and, if so, how much of it to copy to the listening application. Figure 28 illustrates BPF's interface with the rest of the system.

Because network monitors often only want a small subset of network traffic, a dramatic performance gain is realized by filtering out unwanted packets in interrupt context. To minimize memory traffic, the major bottleneck in most modern workstations, the packet should be filtered 'in place' (e.g., where the network interface DMA engine put it) rather than copied to some other kernel buffer before filtering. Thus, if the packet is not accepted, only those bytes that were needed by the filtering process are

[3] Since a huge number of packets can be collected that are separated by few milliseconds, calling a system call to read every packet is not usually possible. Every read called on the file interface returns a group of packets; each packet is encapsulated in a header including time stamp, length and offsets for data alignment.

referenced by the host. BPF uses a re-designed, register-based 'filter machine' that can be implemented efficiently on today's register-based CPUs. Further, BPF uses a simple, non-shared buffer model made possible by today's larger address spaces. The model is very efficient for the 'usual cases' of packet capture. The design of the BPF was guided by the following constraints:

1. It must be protocol independent. The kernel should not have to be modified to add new protocol support.
2. It must be general. The instruction set should be rich enough to handle unforeseen uses.
3. Packet data references should be minimized.
4. Decoding an instruction should consist of a single C switch statement.
5. The abstract machine registers should reside in physical registers.

The BPF abstract machine consists of load, store, ALU, branch, return and miscellaneous instructions that are used to define low-level comparison operations on packet headers; on some platforms, these instructions are converted with just-in-time compilation into native code to further avoid overhead. The FreeBSD implementation of BPF supports any link level protocol that has fixed length headers; however, currently only Ethernet, SLIP, and PPP drivers have been modified to interact with BPF and only writes to Ethernet and SLIP links are supported.

BPF often refers to the filtering mechanism, rather than the entire interface. With this meaning, it is sometimes implemented in OS's kernels for raw data link layer socket filters (e.g., in Linux).

A.3.2 eBPF

Initially proposed by Alexei Staravoitov in 2013, eBPF is the next version of BPF, which includes both modifications to the underlying virtual CPU (64-bit registers, additional instructions) and to the possible usages of BPF in software products. Furthermore, packets are no longer a *copy* of the original data, but eBPF program can operate and modify the packet content, hence enabling a new breed of applications such as bridging, routing, NATting, and more. The "Classic" BPF is not used anymore, and legacy applications are adapted from the BPF bytecode to the eBPF.

An overview of the runtime architecture of eBPF is shown in Figure 29. The following subsections will explain some of the relevant parts of the architecture and point out some of the main improvements in eBPF.

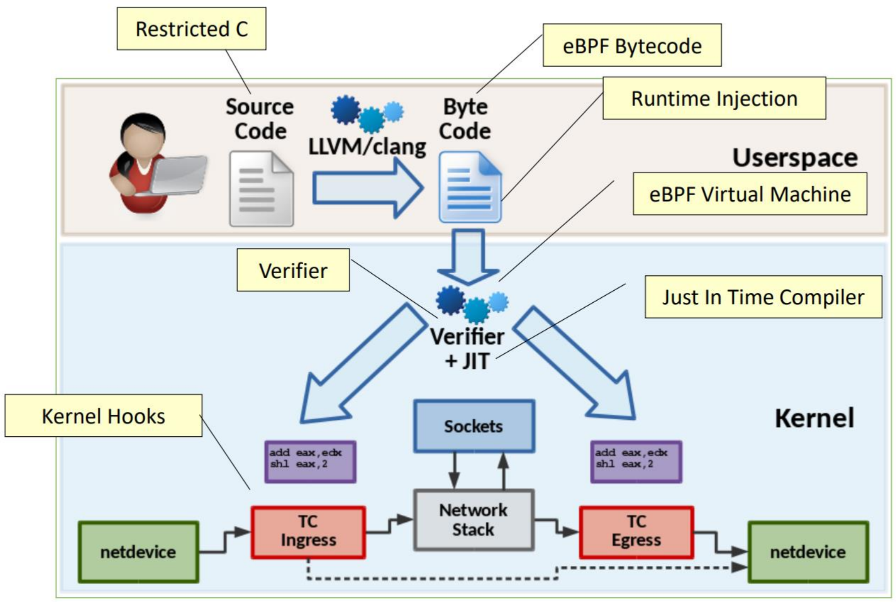


Figure 29. eBPF architecture.

C-based programming

eBPF code can be written in (a restricted version of) C, which allows for easier program development and more powerful functionalities with respect to bare assembly.

Maps

An eBPF program is triggered by a packet received by the virtual CPU. To store the packet in order to process it, eBPF defines a volatile "packet memory", which is valid only for the current packet: this means there is no way to store information needed across subsequent packets.

eBPF defines the concept of state with a set of memory areas, which are called maps. Maps are data structures where the user can store arbitrary data with a key-value approach: data can be inserted in a map by providing the value and a key that will be used to reference it.

An important feature of maps is that they can be shared between eBPF programs, and between eBPF and user-space programs. This is especially important for all those applications that need to perform operations that exceed the complexity allowed by the eBPF bytecode. In fact, maps allow to split complex processing in two layers (fast eBPF datapaths and slow user space control paths), keeping the state information shared and synced. Another important advantage of using maps is that their content is preserved across program executions.

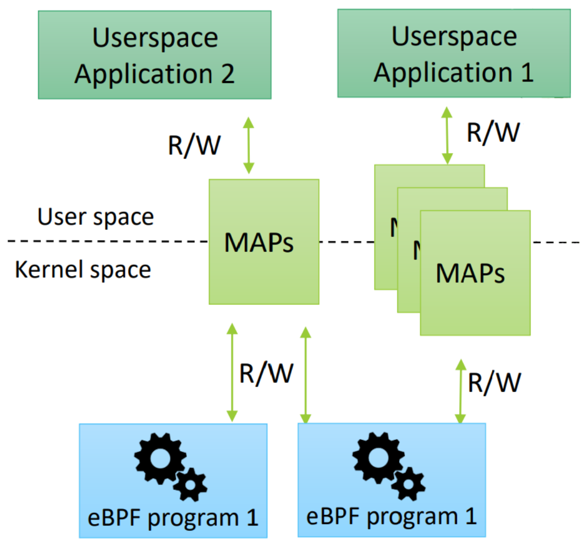


Figure 30. eBPF maps.

Technically, maps are not buffers. If they were, there would be a certain number of issues under the responsibility of the developer, such as concurrent access. This means maps are never accessed directly: we read and write maps with predefined system calls. An important side effect of using maps is that the state of the program is decoupled from the code. Instructions are in the program, the data used by such instructions are in the maps.

We report here the most common map types [195] available in the Linux kernel that can be used in user-defined data paths:

- array: data is stored sequentially and can be accessed with a numeric index from 0 to size - 1. This map type is ideal when the key corresponds to the position of the value in the array.
- hash: data is stored in a hash table. This map type is very efficient when it comes to direct lookups: a hash of the provided key is computed and used as an index to access the corresponding value.

- LRU hash: sharing its internals with the hash type, it provides the ability to have a hash-table that is smaller than the total elements that will be added to it, because when it runs out of space it purges elements that have not recently been used. This map type is useful to keep track of caching entries, which will eventually expire forcing their own refresh.
- LPM trie: data is stored as an ordered search tree. As the name suggests, this map type allows performing lookups based on the Longest Prefix Match algorithm. This is extremely handy when developers want to manage the granularity of a rule match field, granting that the most fitting rule is applied to a packet when other matching rules are also present.

Hooks

eBPF programs can react to generic kernel events, not only packet reception: they can react to any system call that exposes a hook.

Considering a network packet and recalling how the netfilter hooks work, with eBPF we can listen to any of the predefined hooks to trigger programs only at certain steps during packet processing. Netfilter is a set of linked modules but it has no filtering concept: attaching to a hook means receiving all the packets. eBPF can attach to hooks and filter packets.

Speaking of hooks, Figure 31 shows the difference in number and position of the networking hooks in Netfilter and eBPF.

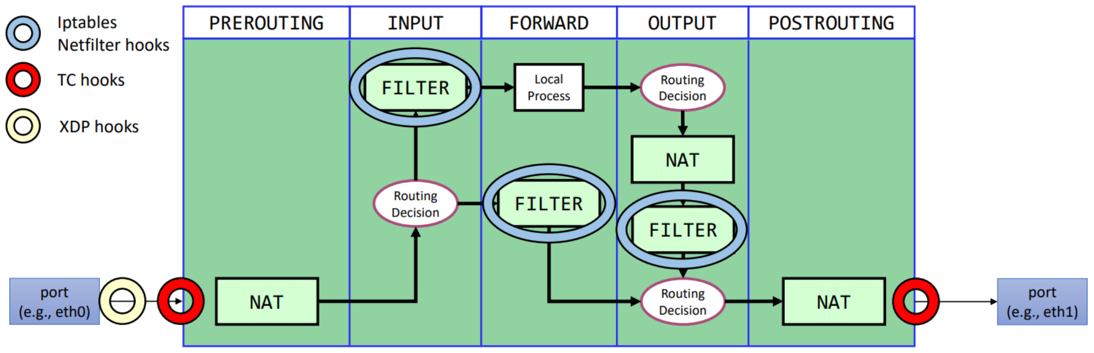


Figure 31. Networking hooks in eBPF and Netfilter.

eBPF hooks are colored in red: the one on the left is called the TC Ingress hook and intercepts all the packets that reach the network adapter from the outside, while the one on the right – the TC Egress hook – deals with the outgoing packets immediately before sending them to the network adapter.

It is clear that Netfilter can provide a much more fine-grained control over a flow of packets inside the Linux kernel, whereas the intermediate hooks must be emulated in eBPF.

Service chains

BPF did not quite have the concept of multiple cooperating programs: each parallel program receives a copy of the packet and processes it. eBPF can link multiple programs to build service chains, such as in Figure 32. Service chains can be created exploiting direct virtual links between two eBPF programs or tail calls. Tail calls can be thought as function calls: the eBPF programs are separated, but the first one triggers the execution of the second by calling it. This allows developers to overcome the program size limitation in the JIT compiler: starting from one big program, this can be split in multiple modules, perhaps functionally distinct such as header parsing, ingress filtering, forwarding, and analytics.

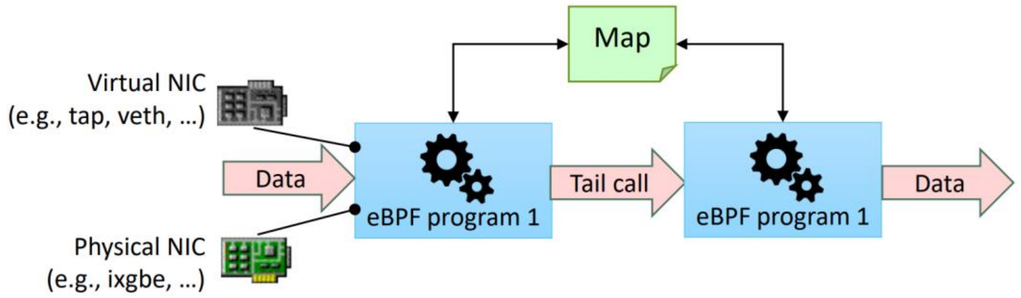


Figure 32. Example of a possible eBPF service chain.

Helpers

Coding in C is fun, libraries are better. Helpers are sets of functions precompiled and ready to be used inside the Linux kernel. eBPF programs can call such functions, which are outside the virtual CPU (e.g. function to get the current timestamp). Helpers delegate complex tasks to the operating system, overcoming the complexity restrictions in the eBPF validator and allowing developers to exploit advanced OS functionalities.

One drawback of helpers is that they must be available in the Linux kernel, which means that the kernel must be recompiled every time a new helper is added, which is not very fast considering the Linux kernel release schedule.

XDP

XDP – eXpress Data Path – is a programmable, high performance packet processor in the Linux networking data path [196, 197] and its main architecture is shown in Figure 33. It provides an additional hook to be used with eBPF programs to intercept packets in the driver space of the network adapter, before they are manipulated by the Linux kernel.

The main advantage of this early processing is that it avoids the overhead and the memory consumption added by the kernel to create the socket buffer – namely the *skb* data structure – which wraps the packet for standard Linux processing in TC mode.

XDP runs in the lowest layer of the packet processing stack, as soon as the NIC driver realizes a packet has arrived, similarly to other frameworks such as netmap and DPDK. However, packets here are not delivered to userspace, but to the injected eBPF program executed in kernel space. One of the main use cases is pre-stack processing for filtering or DDOS mitigation.

It is important to mark the difference between XDP and kernel bypass: XDP does not exclude the TCP/IP stack in the Linux kernel as it is designed to perform a preliminary packet processing to achieve better performance. Kernel bypass (such as netmap and DPDK) ignores the TCP/IP stack and performs all the packet processing on its own.

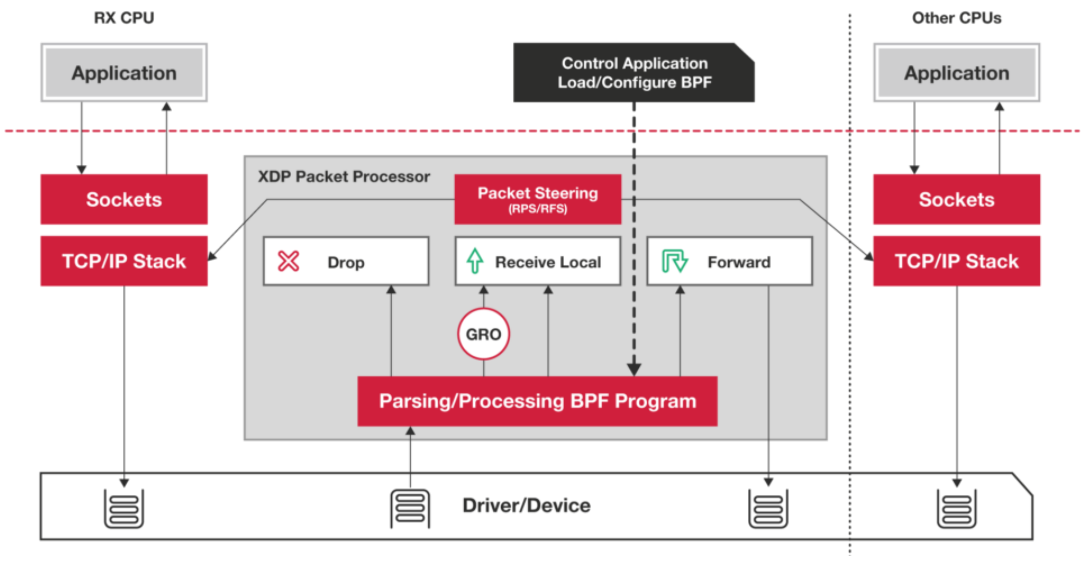


Figure 33. XDP architecture.

System-wide monitoring capabilities

In addition to filtering packets, eBPF programs can be attached and run anywhere in the Linux Kernel through kprobes [198]. This allows inspecting the execution of the kernel and userspace programs and opens the door to high-performance Linux monitoring without running extra kernel modules.

Potential kernel crashes caused by eBPF are prevented through static analysis. The kernel runs a verifier on every eBPF program before allowing its execution.

Without eBPF, kprobes must be injected in kernel modules, which is unsafe, dependent on the architecture, and exceptionally fragile. Though still sensitive to kernel-API changes, after using eBPF, the kprobes are injected from the userspace, safe, and architecturally independent.

Using eBPF for dynamic tracing is very powerful and easier, as eBPF programs can now be coded in C thanks to an LLVM backend. The BPF Compiler Collection (BCC) simplifies development even further, although it comes with additional runtime dependencies (Python, kernel headers and LLVM).

Thanks to this technology, any system call in the kernel can be monitored, and its calling/return parameters can be inspected, paving the way for a holistic view of any kernel activity in real-time.

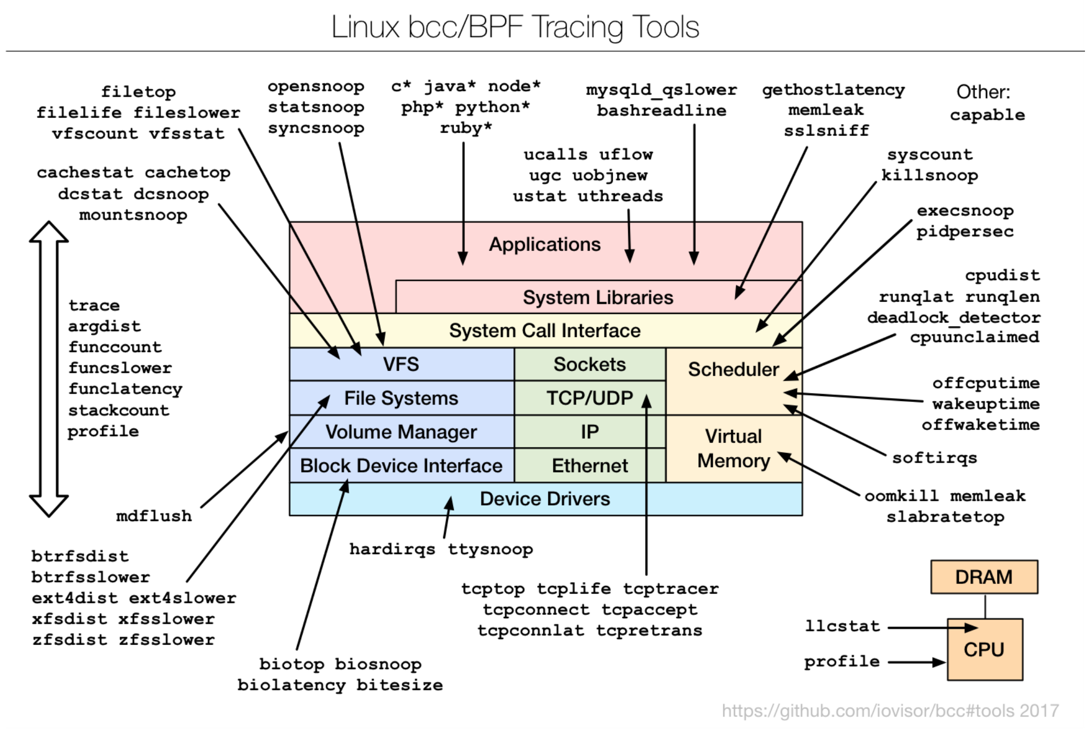


Figure 34. Available tracing tools.

A.3.3 Open vSwitch (OVS)

Open vSwitch is a multi-layer, open-source virtual switch for all major hypervisor platforms, also widely used in physical OpenFlow switches. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g., NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).

OVS is made of three main components, as shown in Figure 31. The *ovs-vswitchd* is a daemon that manages and controls any number of virtual switches on the local machine. It implements the OpenFlow protocol and can receive flow definitions from external SDN controllers. Beyond Openflow, ovs-vswitchd can be configured with a number of features: L2 switching with MAC learning, NIC bonding with automatic failover, 802.1Q VLAN support, port mirroring, Netflow flow logging, sFlow Monitoring. The *ovsdb-server* component extends OpenFlow control functions with a broader set of management capabilities, encompassing the switch layout (switch instances, ports), configuration of QoS queues, association to OpenFlow controllers, enabling/disabling STP, and so on. The ovsdb-server can be controlled remotely by the ovsdb protocol; between ovsdb-server and ovs-vswitchd multiple sockets types can be used (unix, tcp, ssl). The last component is the *datapath*, which is a dedicated pipeline for fast packet processing. One common ovs-vswitchd daemon is available for multiple platforms, while different datapaths are developed to leverage specific software hooks or hardware capabilities. A separated datapath enables to easily port the OVS architecture of different hardware targets, each with its own packet processing pipeline. Implementations for Linux include a user-space datapath (netdev), a DPDK datapath, and a kernel datapath. The kernel datapath is the most common solution.

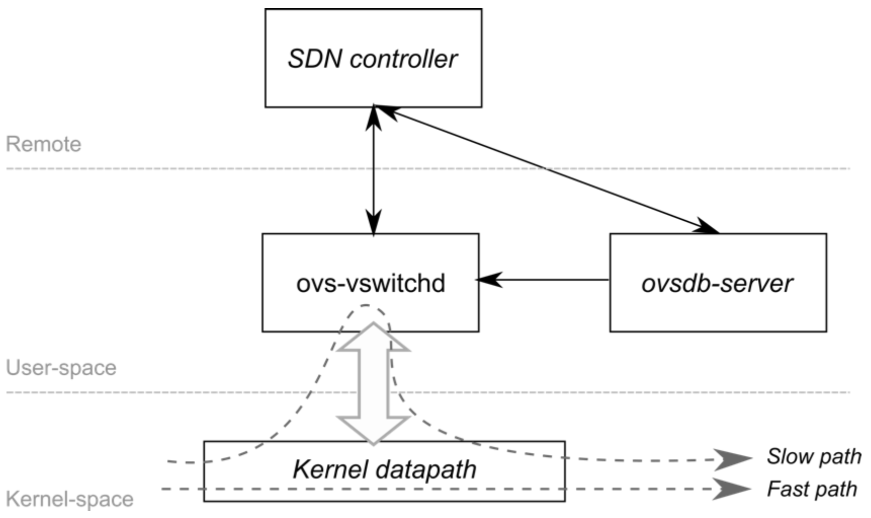


Figure 35. Open vSwitch components.

Figure 31 also depicts how the two main OVS components work together to forward packets. The datapath module in the kernel receives the packets first. It is nothing more than a table with matching entries and associated actions. If a match is found, the corresponding action is invoked (*fast path*). However, for performance and scalability reasons, the table size is rather limited; in case of miss, the packet is forwarded to the ovs-vswitchd component, which holds all OpenFlow rules (*slow path*). In user-space, ovs-vswitchd determines how the packet should be handled, then it passes the packet back to the datapath with the desired handling; usually, it also tells the datapath to cache the actions, for handling similar future packets.

It is rather intuitive that the hit rate in the kernel datapath is a key performance factor. In Open vSwitch, flow caching has greatly evolved over time. The initial datapath was a *microflow cache*, essentially matching all set of OpenFlow packet headers and metatada. Microflow caching collapses all matching fields in a single lookup per packet, leading to tremendous speedup for most workloads. However, the need to match all possible fields means that no wildcard is possible, and this is problematic with specific traffic patterns: malicious port scans, peer-to-peer rendezvous applications, network testing. The existence of short-lived microflows ultimately leads to low hit rate, and an excessive recourse to the user-space (slow) daemon. In later versions, the datapath has two layers of caching: a microflow cache a secondary layer, called *megaflow cache*, which caches forwarding decisions for traffic aggregates beyond individual connections (see Figure 36). The magaflow cache is a single flow liikup table that supports generic matching, i.e., it supports cacheing forwarding decisions for larger aggregates of traffic than individual connections. While it more closely resembles a generic OpenFlow table than the microflow cache does, it is still strictly simpler and lighter in runtime for two primary reasons: a) it does not have priorities, so the search terminates as soon as it finds any match; b) there is only one megaflow classifier, instead of a pipeline of them, so userspace installs megaflow entries that collapse together the behaviour of all relevant OpenFlow tables.

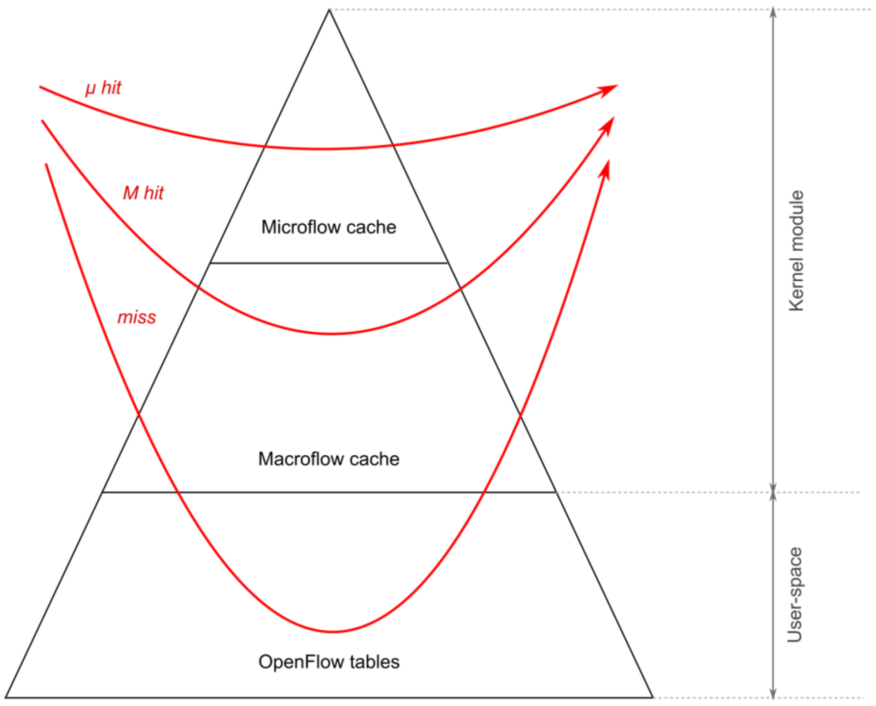


Figure 36. OVS two-tier cache model.

Recently, an eBPF-based datapath has been proposed for OVS. The goal is not to pursue better performance, but to make the datapath more extensible. The main design challenges are to overcome the limitations in writing eBPF programs (number of bytecode instructions, number of instructions traversed by eBPF verification, isolated environment, dynamic looping disallowed).

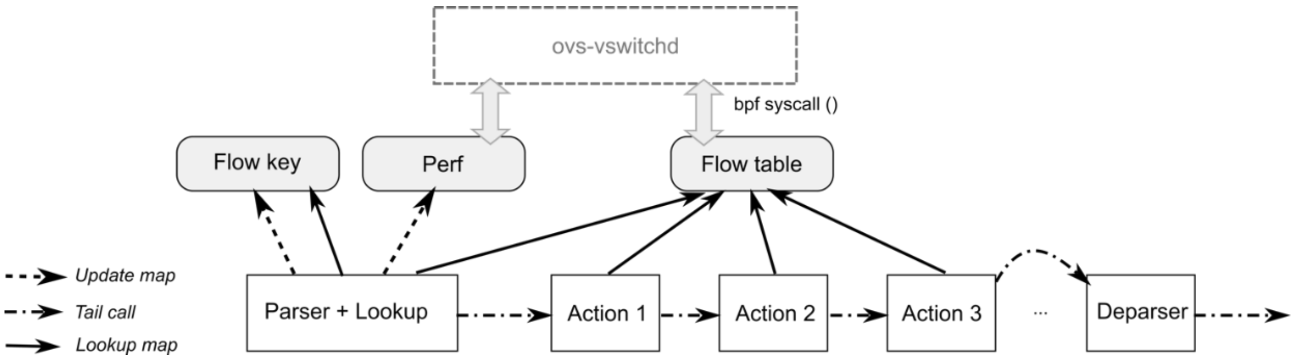


Figure 37. Reference architecture for the eBPF datapath.

Figure 33 shows the overall structure of an eBPF datapath. It is composed of multiple eBPF programs and *maps*. Maps are used for configuration and integration with ovs-vswitchd:

- Flow key: this is the internal representation of protocol headers and metadata;
- Flow table: it associates the flow key to an array to actions;
- Per Ring Buffer: it is used to pass data to the ovs-vswitchd in user-space.

The first eBPF stage (Parser + Lookup) is triggered by the TC ingress hook associated with a particular network device. The parser includes two components: standard protocol parsing and Linux-specific metadata parsing. Protocol headers are described in P4 and then compiled in eBPF bytecode, which is then extended to include metadata for TC context (e.g., incoming device index). Protocol headers and metadata are assembled as flow key, which is then used to lookup the flow table map and get an array of actions. Since eBPF have a maximum size, different actions are encoded as separate eBPF programs; in addition, since dynamic looping is disabled, each program tails call other programs, as showed in Figure 33. The list of actions is pushed in the flow table by ovs-vswitchd. In case of table miss, the slow path is used, by sending the packets and metadata (together with the flow key) to ovs-vswitchd through the Perf buffer. After processing in ovs-vswitchd, the packet is reinserted into the datapath through a TAP interface; an additional eBPF program is attached to such interface for further processing. Finally, similar to typical P4 structure, a deparser program ends the chain and applies all changes and actions that have been accumulated in the metadata during the previous phases.

A.3.4 Programmable dataplanes

High-performance networking devices are traditionally implemented in hardware, envisioning a fixed pipeline of operations on configurable tables (e.g., ARP and forwarding information) that are filled in by the control plane. These implementations are usually limited to a fixed and static set of network headers, and the introduction of a new protocol or just a new version of existing ones might require years to re-design the chip.

The need for more flexibility in design and upgrade of high-speed network devices has leveraged the usage of more flexible technologies like ASICs, FPGAs, network packet processors. These devices can effectively support data processing up to terabit speed, but programming them is far from easy. Ultimately, this has resulted in the effort for high-level programming languages, as P4. It is mainly conceived for hardware targets, but it can also be compiled for software switches and data planes. In this respect, we believe worth including this technology in our analysis.

A.3.5 P4

The Programming Protocol-independent Packet Processors (P4) is a high-level language for programming protocol-independent packet processors. The explicit target for P4 is to tell the switch how to operate, without be constrained by a rigid switch design. The challenging design requirement for P4 is protocol and target independence. The former means that the language must be agnostic of network protocols and packet headers, whereas the latter requires a compiler to translate the high-level description into target-dependent instructions.

Though P4 was sometimes indicated as a ground-breaking evolution of OpenFlow, it has been clarified that P4 is not “OpenFlow 2.0” [199]. Indeed, OpenFlow is a standard, open interface to populate the forwarding tables (i.e., the hash tables for Ethernet address lookup, the longest-prefix match tables for IPv4 and the wildcard lookups for ACLs) in data planes that typically implement IEEE and IETF standard protocols in silicon. In other terms, OpenFlow does not really control the switch behaviour; it gives us a way to populate a set of well-known tables for a number of protocol headers that have largely blown up in the years (i.e., from the original Ethernet, VLAN, IPv4, ACLs, to currently about 50 different header types).

Figure 27 shows the main concept behind the P4 language. In a traditional switch architecture, the data plane provides a given set of functions and capabilities. For example, legacy Ethernet switches implement the IEEE 802.1D protocol, whereas OpenFlow switches filter and forward packets according to the rules inserted in specific tables. Even with the separation between the data and control plane, programmability is only meant to push control information into the data plane (i.e., table management), subject to a fixed set of parsing and forwarding operations. A P4-programmable switch enables the definition of custom headers for parsing, and control flows for packet management. This removes the need for a-priori knowledge of the network protocol, since the data plane provides generic capabilities for receiving, buffering, manipulating, forwarding packets. The control plane is still responsible for filling internal tables (e.g., forwarding routes, ARP tables), but the set of tables and other objects in the data plane are no longer fixed, and can be easily adapted to the evolving control and management needs. While OpenFlow is designed for SDN networks in which we separate the control plane from the forwarding plane, P4 is designed to program the behaviour of any switch or router, whether it is controlled locally from a switch operating system, or remotely by an SDN controller.

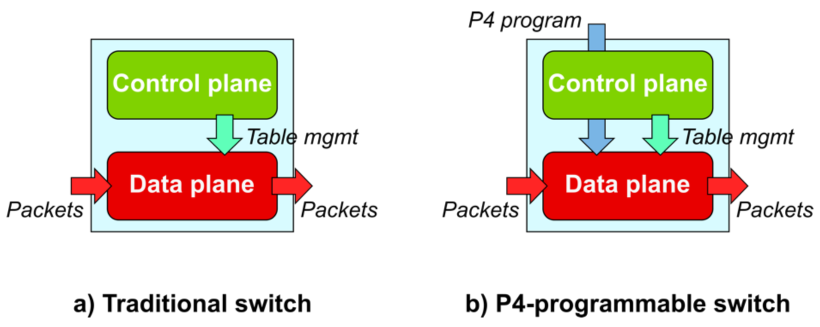


Figure 38. Comparison between traditional switch architecture and P4.

P4 [200] is a domain-specific language, which means it is not conceived to run on general purpose processors; expected target platforms include network interface cards, FPGAs, software switches, and hardware ASICs. As such, the language is restricted to constructs that can be efficiently implemented on all of these platforms; though this may appear as a limitation in the expressiveness, it is a necessary condition for enabling fast packet processing.

The implementation of a P4 program depends on the hardware capabilities. To this aim, manufacturers of targets must provide both an architecture model and a compiler for their platforms. Figure 39 depicts a typical workflow when programming with P4. P4 programmers write programs for a specific architecture model, which defines a set of P4-programmable components and their data plane interfaces. On the other hand, target manufacturers provide the architecture model and a P4 compiler for their target. P4 follows a typical approach of many modern programming languages. The target hardware is described in terms of architecture descriptions (*packages*) that must be instantiated by the user to develop and build a program. This declaration is somehow similar to Java interfaces and C++ virtual classes; user-defined types that are unknown at the declaration time are described using type variables, which must be used parametrically in the program – i.e., similar to Java generics, not C++ templates. The architecture description may include pre-defined data types, constants, helper package implementations, and errors.

The output from compilation consists of:

- a data plane configuration that implements the forwarding logic described in the input program;
- an API for managing the state of the data plane objects from the control plane.

To foster portability and interoperability, the P4 community is working towards the definition of common control APIs that can be adopted by the programmers (P4Runtime).

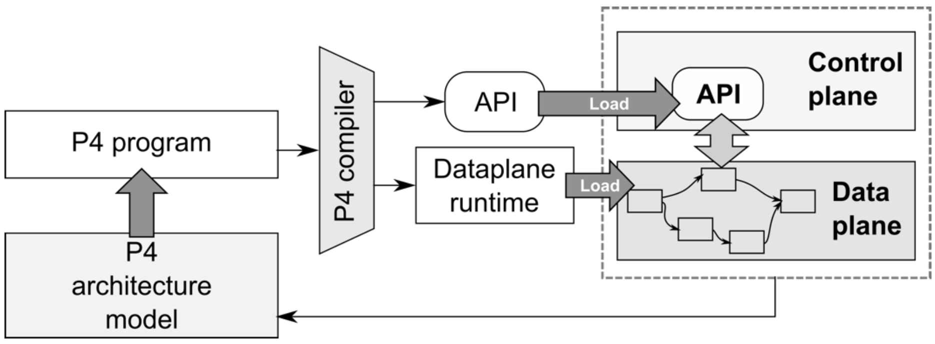


Figure 39. Workflow for programming with P4.

The P4 architecture identifies P4-programmable blocks (e.g., parser, ingress control flow, egress control flow, deparser, etc.), which are declared in the target architecture model. A P4 program consists in providing code fragments for each block. P4 blocks interface with the hardware and between them through metadata (see Figure 40):

- *intrinsic metadata* are control registers and signals which are exposed by the target;
- *user-defined metadata* are data associated to each packet and defined by the programmer.

In addition to control registers, P4 programs can invoke services provided by external objects and functions provided by the architecture (e.g., checksumming).

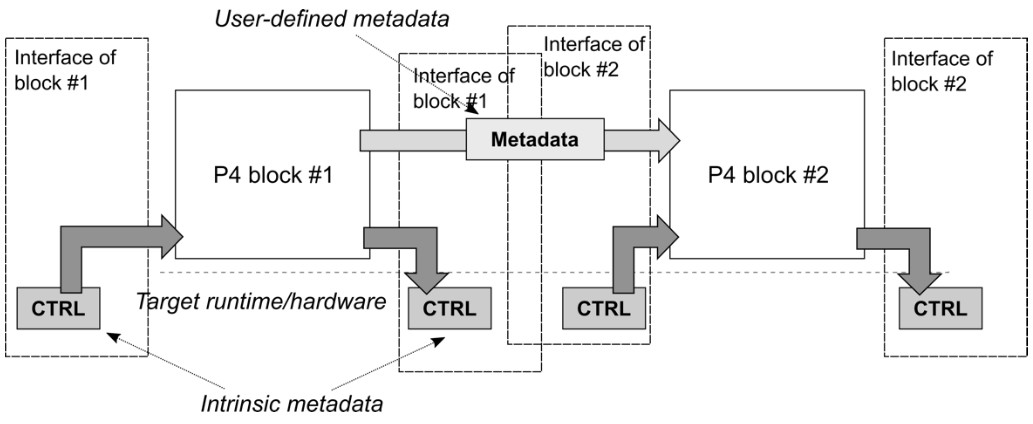


Figure 40. P4 interfaces between blocks and target hardware.

There are two types of P4 blocks: *parsers* and *controls*. Parsers are used to extract user-defined headers from the packets; controls build processing pipelines. They can be composed in parallel, sequential or mixed chains; the conditional traversing of these chains is settled by the program logic.

A *parser* describes a state machine with one start state (start) and two final states (accept/reject). In-between there may be multiple user-defined states, corresponding to different headers. The ability to define header structure in software is one of the main benefits brought by P4, which removes the need for fixed and static header definitions in hardware. A parser starts execution in the start state and ends execution when one of the reject or accept states has been reached. Transitions between the states are driven by specific header fields. For instance, the *etherType* field in the Ethernet header can be used to select the next state (i.e., parsing an IPv4 header or a VLAN tag). In addition, the P4 syntax also allows to verify the field values (e.g., protocol versions) and check the correctness of the header (i.e., comparing checksum values). The result of a parser block is the set of recognized headers and metadata about possible errors that occurred (e.g., unsupported protocols or versions, invalid options, wrong checksums).

A *control* block describes processing pipelines following the typical structure of a traditional imperative program. Control blocks are made of match-action units, which are invoked to perform data transformation. A match-action unit is logically represented by a {key, value} table, where keys are used for look-ups and values correspond to actions to be performed. Key/action pairs are usually pushed by the control plane (e.g., ARP information, forwarding rules, etc.), but there is also the possibility to insert static information from the P4 program. Figure 41 shows the logical structure of a match-action unit: starting from packet headers and metadata, a key is constructed (1) and used for lookup (2); the corresponding action is invoked on input data (3) that may update the headers and metadata (4). The programmer defines as many tables as he needs for processing packets; he also defines the sequence of tables to be applied to packet headers and metadata according to typical imperative constructs (even including conditional evaluation).

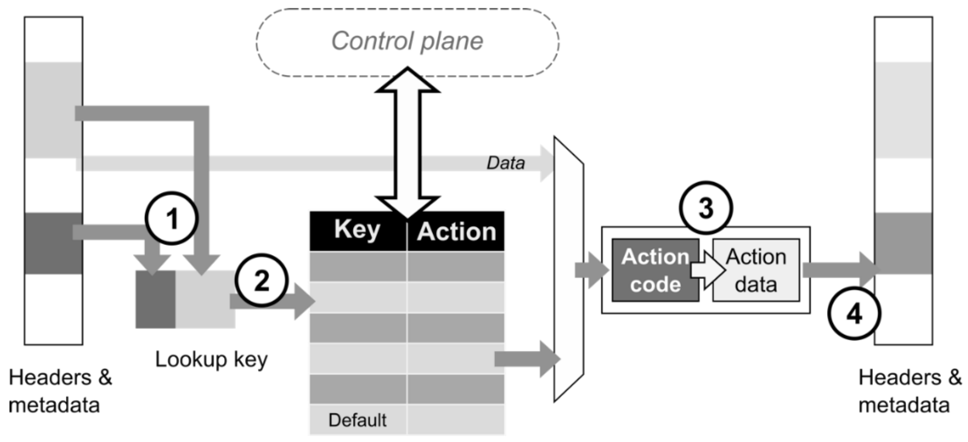


Figure 41. Logical structure of a match-action unit in P4.

A particular case for a control block is *deparsing*. It assembles the manipulated headers into a new packet and send the result to the correct output port.

Though there are many benefits brought by P4 compared to state-of-the-art packet-processing systems (e.g., based on writing microcode on top of custom hardware) in terms of flexibility, expressiveness, software engineering, libraries, and debugging, the is also a major limitation that P4 programs are not expected to be portable across different architectures, due to the heterogeneity of the underlying hardware and the expected different architecture models from vendors. This goes against all trends to re-use software and services; indeed, the P4 consortium is working on the definition of common architecture model that may be adopted by vendors (i.e., the Portable Switch Architecture – PSA).

Notably, the eBPF can be used as target by P4 programs. This allows the same P4 program to be portable on hardware and virtual devices, with very similar performance. The current compilation method is a two-stage process that translated P4 programs into C code, which must then be compiled with the standard eBPF Compiler Collection. This poses some limitations on P4 programs, which will be hopefully reduced in the future [201].

A.4 Programming abstractions

Software-Defined Networking (SDN) is a mean for more pragmatic and effective control of network operations, offering network administrators the freedom to build complex and stateful logics (i.e., ‘programs’) behind the mere configuration of predefined and rigid operational patters. The basic SDN concept is the separation between the data plane and the control plane, mediated by a network controller that provides the programmatic interface to the network. The network is thus abstracted as a single system, and control applications can be written in a more intuitive, portable and scalable way.

Roughly speaking, the main target for SDN is to write control applications that run in the SDN controller and programmatically configure the local data plane of network devices for fast packet processing. Control applications represent a complementary part of data planes, since they implement complex logics that is not possible (or convenient) to hardcode in a fast packet pipeline.

The logical separation of the control and data plane requires some forms of standard interfaces for interworking of products from different vendors. The main purpose for these interfaces is the definition of the logical operations to be performed on packets, either taken singularly or as aggregated flows. Several efforts have been undertaken in the last decade to design abstractions and models for data plane operations, which differ in terms of granularity, efficiency, flexibility. As a result, some protocols are mostly suited for Northbound interfaces (i.e., between applications and the controller), whereas others are specifically conceived for the Southbound interface (i.e., between the controller and network devices).

A.4.1 OpenFlow

OpenFlow is perhaps the most known protocols for SDN. In a typical SDN architecture, OpenFlow is used as Southbound interface between the controller and the network switches [202]. Indeed, OpenFlow plays a double role:

1. it is a protocol between SDN controllers and network devices, used to configure local data planes from a centralized control application;
2. it is a specification of the logical structure of the network switch functions, which are abstracted at high level as a set of configurable tables, leveraging the consideration that most vendors implement somewhat similar flow tables for a broad range of networking functionalities (L2/L3 forwarding, firewall, NAT, etc.).

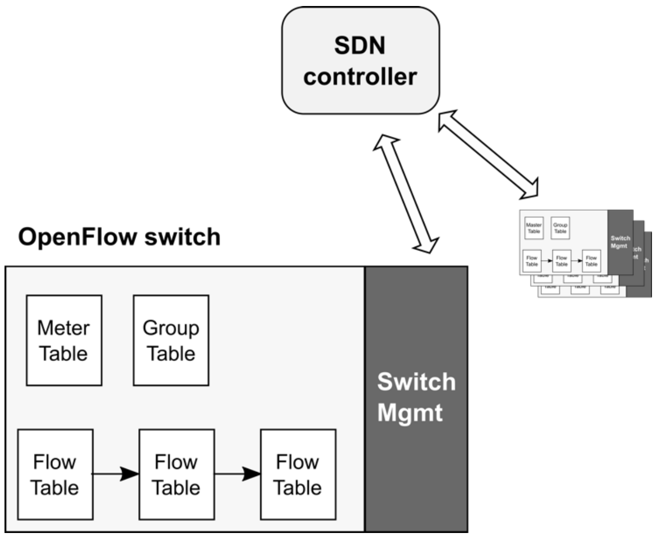


Figure 42. OpenFlow environment.

Figure 42 depicts the basic structure of the OpenFlow environment. The OpenFlow protocol is used between the SDN controller and the internal management of the software/hardware switch.

The OpenFlow switch architecture [42] is seen as a set of configurable tables, which may be implemented in hardware, firmware, or even software. The internal architecture can be different for every vendor and does not need to be public; the processing pipeline is fixed and cannot be modified. The Flow table associates *flows* with specific processing instructions; the Group table describes common actions that affect one or more flows; the Meter table triggers a variety of monitoring actions on a flow. In the OpenFlow terminology, a flow is a set of packets that share common values in the header fields.

A Flow table is composed by entries with the following fields:

- *Match fields*: used to select packets according to the content of their headers;
- *Priority*: relative priority of table entries;
- *Counters*: meters on the number of packets and amount of bytes for different events, flow duration;
- *Instructions*: actions to be taken in case of match;
- *Timeouts*: lifetime of the flow;
- *Cookie*: opaque data value chosed by the controller to filter flow statistics, flow modification, and flow deletion.

Each flow traverse a table until a highest-priority match is found; there may be a *table-miss* entry which provides a default treatment for non-matching flows. Each packet may traverse a single table or may be pipelined across multiple tables. The match fields cover both hardware properties (i.e., incoming port), protocol headers (including Ethernet, IP, TCP/UDP/SCTP, MPLS, ICMP), and metadata that may have been added by a previous table.

If a packet matches an entry, the corresponding instruction is performed. There are several kinds of instructions that:

- forward packets through the pipeline to another table;
- perform specific actions on the packet;
- update the action set for the packet;
- update metadata associated to the packet.

OpenFlow actions include packet forwarding (e.g., output queue), packet modification (e.g., push/pop VLAN tags, rewrite source or destination addresses, change TTL), and group table processing operations. While traversing multiple tables, actions may be accumulated in metadata and indicated as "action set"; the full list of actions is then performed when the packet exits the processing pipeline.

Typical OpenFlow programs usually do not install all possible flows from the beginning, to avoid overwhelming the memory of network devices. There is a specific action included in the specification, usually associated to the *table-miss* entry, which sends the packet to the controller for further processing. The controller runs control applications that can install additional flows in the network devices, or drop the packet. Clearly, this behaviour introduces large latency in packet processing, but the delay is only experienced by the first packet of the flow, whereas all subsequent packets will be processed by the newly installed flow.

A.4.2 OpenState/Open Packet Processor

One commonly-criticized aspect of OpenFlow is the limited "match/action" abstraction, which allows the programmer to broadly specify a flow via an header *matching* rule, associate forwarding/processing *actions* to the matching packets, and access bytes/packet *statistics* associated to the specified flow. Though simple and elegant, the original abstraction cannot fully fit the real-word needs. The initial abstraction has therefore been enriched during the standardization process with supplementary actions, more flexible header matching, action bundles, multiple pipelined tables, synchronized tables, and so on. Despite the many extensions, OpenFlow devices still remain "dumb", with all the smartness placed at the controller side. The main implication is that many (even simple) stateful network tasks (as MAC learning operation, stateful packet filtering, outbound NAT) must be implemented on the centralized controller. The involvement of the controller for any stateful processing and for any update of the match/action rule leads to extra signalling load and processing delay [14].

OpenState is an OpenFlow extension that explicitly targets stateful processing in network devices. It exposes devices' hardware capabilities (packet processing and forwarding actions, matching facilities, TCAMs, arithmetic and logic operations on registry values, etc.) to programmers as a sort of machine-level "configuration" interface, hence without any intermediary compilation and adaptation to the target (i.e., unlike the case of P4). While events and related actions are fully identified by the packet header in OpenFlow, they also depend on the state and influence the state transitions in OpenState.

Formally, OpenState is modelled as a simplified type of eXtended Finite State Machine (XFSM), known as *Mealy Machine*. A further extension, indicated as Open Packet Processor (OPP), overcomes this limitation and implements a full XFSM model [15]. Table 7 compares the different abstraction models of OpenFlow, OpenState, and OPP.

Table 7. Comparison among abstraction models of different OpenFlow-related data planes.

Data plane	Abstraction model
OpenFlow	$T: I \rightarrow O$
OpenState [14]	$T: S \times I \rightarrow S \times O$
OpenPacketProcessor [15]	$T: S \times F \times I \rightarrow S \times U \times O$

Symbol	XFSM formal notation	Meaning
I	Input symbols	All possible matches on packet header fields.
O	Output symbols	OpenFlow actions.
S	Custom states	Application-specific states, defined by the programmer.
D	n-dimensional linear space $D_1 \times \ldots D_n$	All possible settings of *n* memory registers; include both custom per-flow and global switch registers.
F	Set of enabling functions $f_i: D \rightarrow \{0,1\}$	Conditions (Boolean predicates) on registers
U	Set of update functions $u_i: D \rightarrow D$	Operations for updating registers' content.
T	Transition relation	Target, state, actions and register update commands associated to each transition.

Figure 43 depicts the conceptual OpenState architecture. Selected fields from packet headers (which identify the "flow") are used as key to query the State table (1). States are arbitrarily defined by the programmers, and are used as metadata in addition to the packet header (2). The combination of matching fields in the packet headers (3) and the current state is used to lookup the XFSM table (4), which gives the processing actions (e.g., drop the packet, forward the packet, re-write header fields) and the next state. The next state is then used to update the State table, so that the next packet may be properly processed (5). Beyond the presence of two separate tables (one for storing the state and the other for storing processing actions and state transition), the other fundamental difference with plain OpenFlow is the sharp separation between *events* and *flows*, which is reflected in two different lookup scopes (1) and (3).

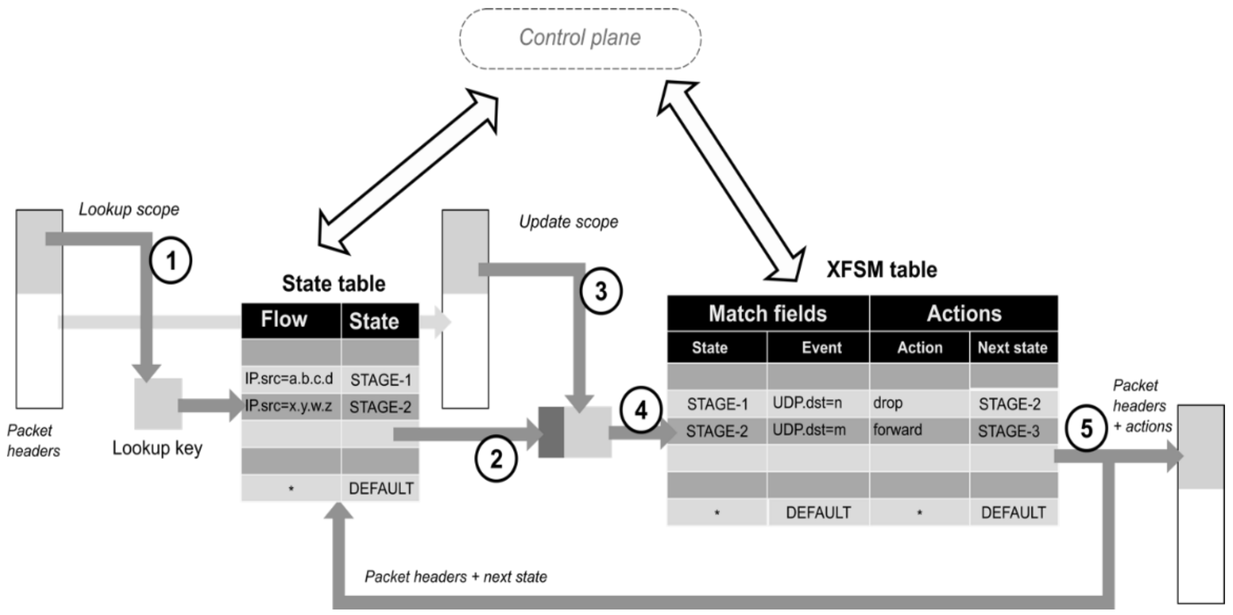


Figure 43. OpenState architecture.

The flow identity represents the set of packets that are processed by a single OpenState program; it is therefore associated to the current state. The flow identity is accessed by the "lookup scope" (1), which consists of a set of header fields and matching rules. As an example, we can identify different flows based on their source IP address. Events are associated to packet header as well, but they use a different set of header fields and matching rules, which represent the "update scope" (3). For instance, we can look at the current destination port. This distinction is a fundamental abstraction for implementing a broad set of networking tasks. The MAC learning algorithm is the most straightforward example: in fact, packets are forwarded based on their *destination* MAC address, whereas the forwarding database is updated using the *source* MAC address.

An OpenState program consists of a set of logical states, lookup and update scopes, and transition rules. They are pushed by the control plane in the XFSM table, while the State table is initialized with the flow identification and the initial state. Typical network state machines also include timeout values, to avoid stale conditions. Handling timer expiration in OpenState is trivial when it is managed by a state lookup which retrieves an expired state and returns a DEFAULT value, but it is more complex in case of unsolicited transitions.

Even if the Mealy Machine abstraction is already powerful (e.g., examples of applications are proposed in OpenState SDN project), the implementation of full XFSMs requires the ability to verify run-time conditions. Indeed, the sheer majority of traffic control applications rely on comparisons, which permit to determine whether a counter exceeds some threshold, or whether some amount of time has elapsed since the last seen packet of a flow. Example of possible conditions are flow statistics, queue levels, port number, etc. Conditions can be associated to memory registers; there may be both "global" registers, which keep aggregate information for all flows, and per-flow registers, which keep separate counters for each flow.

Figure 44 shows the improved architecture of Open Packet Processor (OPP), which includes both control registers and specific functions to update them. Selected packet headers are used to derive a lookup key as for the OpenState, but the State table is now replaced by an augmented Flow context table (1). The flow context is composed of a state and *k* per-flow registers (R_1, … R_k) defined by the programmer. The state and per-flow registers represent metadata that are propagated alongside the packet header in the pipeline. A Conditional block is added to compare per-flow registers and global variables (G_1, …, G_h) to specific values, which may also be contained in packet headers (e.g., transport ports, time-to-live, priority) (2). Formally, the Condition block corresponds to the set of enabling functions in Table 7. The output is a boolean vector $\underline{C}=(c_1,\ldots, c_m)$, which indicates whether the *i*-th condition is true or false. The packet header, current state, and Boolean vector are then used in the update scope for lookup in the XFSM table (3). The conditional bits can be masked depending on the specific state (i.e., evaluation of a given condition may not be of interest for that state). The output of the whole pipeline is a set of actions on the packet (4), including header manipulation (e.g., change addresses or ports, set time to live or type of service) and forwarding behaviour (drop, forward, duplicate, etc.). In addition, the XFSM table provide the information needed to update the per-flow and global registers (5).

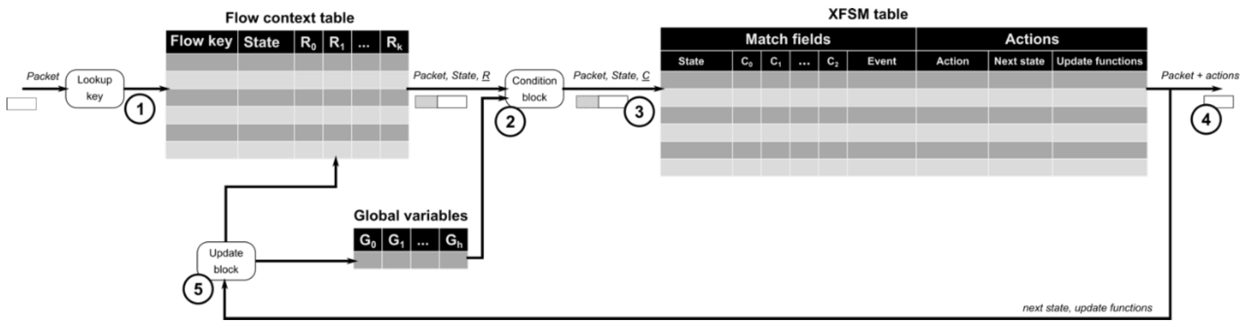


Figure 44. Open Packet Processor conceptual architecture.

While the original OpenFlow proposal builds on the recognition that several different network devices implement somewhat similar flow tables in hardware for a broad range of networking functionalities, the OpenState and OPP architecture require additional hardware to be present: TCAMs (for State/Context and XFSM tables), boolean circuitry (for Conditional block), ALUs (for updating registry), and memory registers.

Interestingly, the OpenState architecture has already been used for DDoS protection, as an alternative approach to OpenFlow and sFlow [16]. Indeed, the main limitation of OpenFlow is the impossibility to de-couple forwarding and monitoring (flows must be installed to monitor network counters), while the main drawback of sFlow is the need for packet sampling, which may miss relevant events.

OpenState and OPP are specifically conceived to enhance the programmability of the hardware, hence they do not fit well virtual machines. Compared to OpenFlow, they allow the creation of "state machines", which are far more powerful than stateless configuration. However, the processing pipeline is fixed (i.e., the conceptual schemes shown in Figure 43 and Figure 44), unlikely P4 where programmers have the possibility to declare and use own memory or registries. However, the flexibility of P4 leverages more advanced hardware technology, namely dedicated processing architectures or reconfigurable match tables as an extension of TCAMs, even if P4 programs may also be compiled for eBPF in the Linux kernel (see Section A.3.5).

A.4.3 NETCONF/RESTCONF/Yang

NETCONF [43] and RESTCONF [12] are two alternative protocols to configure network devices using tree-like data models described in the YANG language [203]. They both use a client/server approach, where the client plays the role of "manager" and the server the role of "agent". The conceptual framework is shown in Figure 45. Network elements are seen as a sort of distributed configuration database, which are managed by the NETCONF/RESTCONF protocols. There may be multiple databases; for instance, NETCONF usually envisages three configuration sets: running, startup, and candidate (this latter used to edit the configuration without interfering with device operation). Configurations are described by the YANG modelling language.

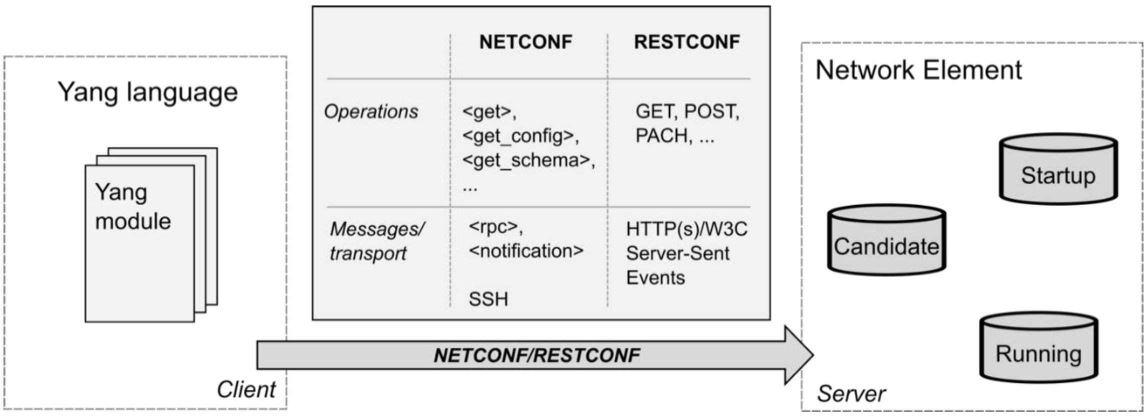


Figure 45. NETCONF/RESTCONF framework.

NETCONF and RESTCONF are alternative protocols to perform the same task; they define the logical operation and the messages to exchange between the client and the server to retrieve and modify the configurations. They build on the experience from SNMP, by using a very similar framework for configurations in addition to monitoring; Table 8 shows a quick comparison among the three protocols. RESTCONF provides a simplified interface, leveraging a well-known and consolidated transport protocol (HTTP), whereas NECONF defines its own set of RPC primitives over a SSH connection. As shown in Figure 45, they provide similar CRUD operations. In addition, both protocols can retrieve supported capabilities and their descriptive schemas (see below).

Table 8. Comparison of SNMP/NETCONF/RESTCONF.

	SNMP	NETCONF	RESTCONF
Descriptive language	SMIv2	Yang	Yang
Data model	MIBs	Yang modules	Yang modules
Encoding	ASN.1 BER	XML/JSON	XML/JSON
Transport	UDP	SSH	HTTP(s)
Operations	GetRequest, SetRequest, GetNextRequest, GetBulkRequest	<get-config>, <edit-config>, <copy-config>, <delete-config>, <lock>, <unlock>, <get>, <close-session>, <kill-session>	OPTIONS, GET, PUT, POST, DELETE, PATCH
Notification	Traps/Inform requests	<notification> rpc	W3C Server-Sent Events
Security	Authentication: HMAC-MD5, HMAC-SHA Encryption: CBC_DES and CFB_AES_128	SSH mechanisms	SSL mechanisms

The description of the configuration uses the YANG modelling language and can be encoded in XML or JSON. YANG is a descriptive language, which defines a syntax tailored to network environments. The language includes built-in data types, basic structures, and the possibility to add user-define data types. It uses a tree-like abstraction of the network, starting from the whole device and flowing down until the most elementary configurable elements. YANG models are based on descriptive *modules*. Each module schema describes configurable parameters in terms of key/values, which must be only readable or also writable by the client. Configurations are modified by changing values in the modules and/or adding/removing code snippets. YANG models (or, better, their schemas) can be retrieved from three main sources:

- vendors, which provide tailored Yang modules for their devices;
- standardization bodies (IETF/IEEE), which has only produced drafts for base functions (i.e., interfaces and routing);
- OpenConfig, a consortium created by largest telco's and service providers, which modules only covers high-end equipment.

Annex B Detection and enforcement

B.1 Software threats modelling

There are several threat modeling techniques:

- **STRIDE:** The STRIDE technique was developed by Loren Kohnfelder and Praerit Garg in 1999, in a Microsoft technical document as part of the Microsoft Trustworthy Computing Security Development Lifecycle. [204] The name provides a mnemonic for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege, which are the cornerstones of this technique. The STRIDE process consists of decomposing an application into constituent components on data flow diagrams and analyzing each component against possible threats from the aforementioned categories. STRIDE has two different applications, namely on per-element basis [205] [206] and on per-interaction basis [196]. More specifically, the per-element method, focuses on each element that the model has, producing a more in-depth result, while the per-interaction focuses on the interactions between elements. Both methods can be used in conjunction to produce more complete models [207].
- **Attack Tree:** Attack trees use tree-like diagrams to represent the possible attack paths that are available in the system. More specifically, the root of each tree is a vulnerability identified to be a threat to the system, the leaves represent the attack surface and the rest of the nodes are subtasks that lead up to the realization of the attack. The modeling process with attack trees has multiple steps [208]:
  - Identify the threats that the system might have and set them as attack tree roots.
  - Decompose each threat to subtasks iteratively until the subtasks consist of basic functionalities which belong to the attack surface. Some tasks might be composed of all of its subtasks, and some only of one, depending on this setting, there are logical AND OR nodes to be used.
  - Each task is assigned a cost number. If the final attack has a smaller damage than its summed subtasks costs, then the attack is deemed not likely to occur. On the other hand, attacks with a low-cost implementation and highly damaging results, indicate that the attack must be mitigated as it is likely to occur.
- **Attack Library:** Attack libraries are collections of known attacks compiled into searchable databases. They are general purpose checklists that aim to provide detailed information for attacks, in order to help threat modelers to understand each threat from the perspective of the attacker. Some major libraries are:
  - OWASP
  - CWE
  - CAPEC
  - WASC Threat Classification
- **T-MAP** [209]**:** It is a quantitative threat modelling technique for assessing security risks by calculating the total severity weights of relevant attack paths. This method begins by identifying attack paths on a four-layer model. The layers consist of: the firewall, the commercial of the shelf (COTS) systems, the IT infrastructure and the organization's core values. These layers are described by 22 attributes which are derived from the Common Vulnerability Scoring System (CVSS). After that, weights are applied to each of those attributes depending on their severity, and each attack path can be evaluated based on those weights. The overall threat of the system can be quantified by summing the weights of each attack path, in order to provide a security evaluation of the whole system. Finally, the threat modeler, evaluates all the countermeasures for each threat, depending on their efficiency and cost. This way, optimal countermeasures can be chosen for the system and minimize both costs and possible damages.

B.2 Intrusion detection

The vast majority of detection system can be described in terms of the general architecture shown in Figure 46.

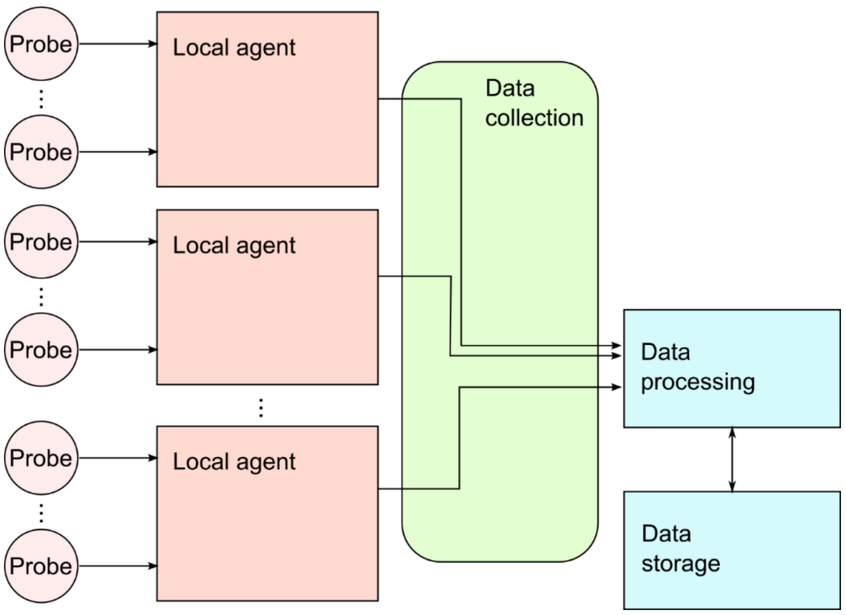


Figure 46. General architecture for Intrusion Detection Systems.

Probes and local agents are used for monitoring and inspection tasks. Though there is not a common and unambiguous definition, we use the two terms to denote different capabilities. On the one hand, probes are dumb sensors that gather raw data in the execution environment. For instance, probes are typically used in network devices to count the number of transmitted/received packets, to filter packets according to specific values in the protocol headers or bit patterns in the packet body; probes may also be used in the operating systems to report system calls or access to I/O resources. On the other hand, agents are smarter elements that include processing capabilities. Depending on the specific architecture, agents may carry out simple filtering/aggregation operations (e.g., select specific records in the system logs, cross-match logs from different application logs, implement packet processing pipelines).

The data collection process gathers events and data for processing. It is unavoidable in distributed architectures, whereas not strictly necessary as a separate component in standalone systems (see later). Finally, centralized data processing and storage are the intelligent elements of the detection process.

The great heterogeneity in the tools, architectures, and methodology for intrusion detection hinders a simple classification. The list of complementary aspects to consider for a thorough taxonomy is quite large [210], but we think that the most significant for the Project can be restricted to the following:

- **Data sources**, which considers the type of information collected by local probes and agents. Relevant sources for cloud services include:
    - *logs* collected by applications and system daemons, as part of their normal operation, which may include accounting information;
    - *traces and binary dumps* that can be collected by the operating system, often limited to system calls, I/O data, system commands;
    - *network traces*, which may include coarse information of traffic statistics (input/output rates, cumulative transmitted/received packets, SNMP info), logs of established connections, dumps of network packets.

- **Frequency of data**, according to how often data are collected:
    - *continuous*, in case of seamless flow of information;
    - *periodic/bath*, when data are aggregated and sent at fixed periodic time intervals;
    - *on-demand*, if data is explicitely requested by the processing/storage entity when needed.
- **Timeliness of detection**, depending on the temporal lag between detection and processing of data:
    - *real-time*, when data are processed immediately after detection;
    - *offline*, when historical data are processed from the data repository.
- **Architecture**, relative to the participation of multiple entities in the overall detection process:
    - *standalone*, when all different systems operate independently and do no share any information (e.g., a separate IDS for each virtual service);
    - *centralized*, typical of cloud-based detection services, where multiple systems rely on the processing power and the strong correlation capability of a common processing engine (this is a common approach for DDoS detection and mitigation);
    - *distributed*, when every system has its own detection process, but shares information and collaborates with peer system to improve the likelihood of detection and accuracy. In this case, parallel, collaborative, cloud or grid-based computing paradigms can be used by exchanging relevant events and high-level information among the local agents.
- **Processing**, the presence of both local agents and centralized processing and storage entities leaves great flexibility in where locating aggregation, fusion, analysis, and correlation operation of the data:
    - *remote processing*, when all operations are performed by the central entity and no local agents are used;
    - *local processing*, when all processing tasks are performed locally and no central entity is present;
    - *distributed processing*, when some preliminary tasks are performed locally (for instance, to reduce bandwidth usage) and main analysis and correlation are kept centrally.
- **Scope**, based on where the agent is deployed and what event types it can recognize: host, network or both, as discussed in more details in Section B.2.1.
- **Reactivity**, depending on reaction and mitigation capability:
    - *passive notification* is the main purpose for pure IDS;
    - *reactive or proactive enforcement* is the typical behaviour of IPS or enhanced IDS tools.
- **Detection methodology**, which is the mechanism to identify the attack by looking and the data, as discussed in Section B.2.2.
- **Analysis techniques** are the mathematical methods used to detect the attacks, by using one or more detection methodology. From simple event monitoring these methods have progressively evolved to more complex statistical analysis and artificial intelligence, with the objective to automate as much as possible the learning process.

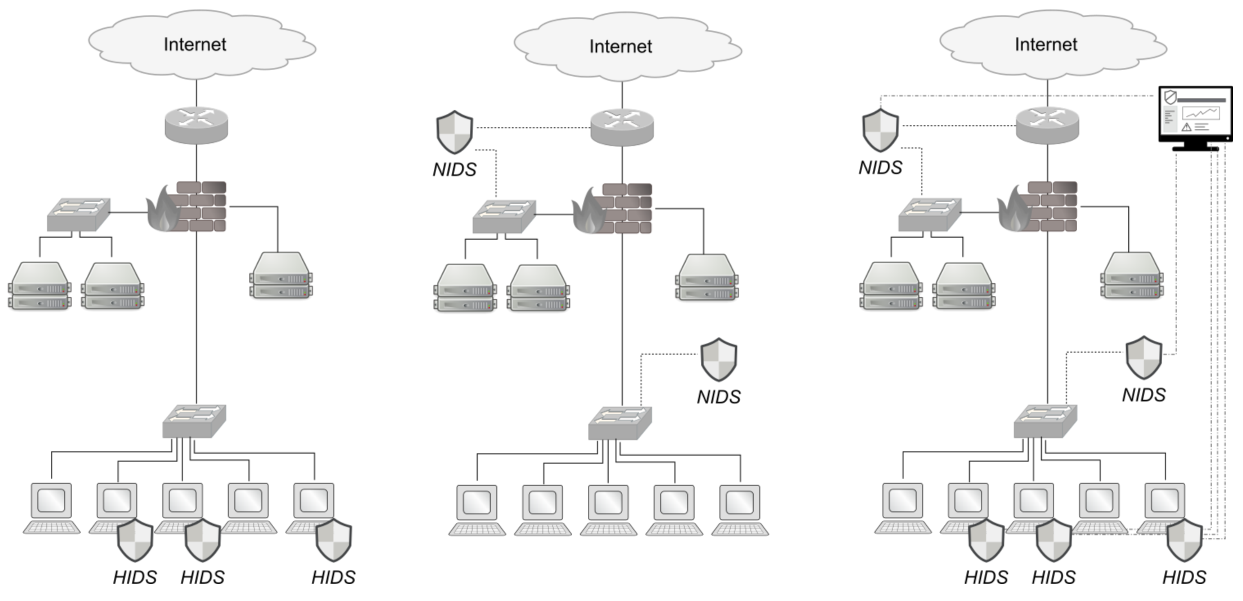
B.2.1 Technologies and scope

Traditionally, most IDS implementations have focused on specific domain scopes, i.e., computing or networking. More recently, the growing complexity of attacks have boosted more interest in more general solutions, which can inspect and correlate data from both domains (see Figure 47).

Host-based Intrusion Detection Systems (HIDS) monitor and analyze events and data from host machines (see Figure 47.a). HIDS usually detect access to the file system (distinguishing between users' folders and system folders), report I/O events (especially network activity), monitor system calls, trace modifications in the system kernel, libraries, and applications, check logs, observe the execution of a program, and may also acquire binary dumps of the memory and processor's registers. Both signature and anomaly-based detection are commonly used in these systems. From an implementation perspective, the most common approach is a standalone architecture with local processing, though latest commercial products are even more integrated through SIEM systems.

Network-based Intrusion Detection Systems (NIDS) monitor network segments and analyze packets (see Figure 47.b). Differently from HIDS, they collect data from probes on network devices, hence gaining visibility on whole network segments rather than single host. In this respect, they typically adopt standalone architectures with remote processing. NIDS collect traffic statistics (i.e., number of packets, size of packets, intermittence of packets), which can be used to detect anomalies that are symptomatic of DoS attacks. NIDS also inspect packet headers (IP and transport layer) of individual packets to detect intrusion attempts: connection to closed ports, port scanning, spoofing, man-in-the-middle, etc. NIDS protect entire networks rather than individual hosts, but they are largely ineffective with encrypted traffic. In fact, they can only monitor communication statistics of encrypted packets, but cannot inspect their headers.

Compound Intrusion Detection Systems (CIDS) adopt multiple technologies to improve their scope and likelihood of detection. Differently from HIDS and NIDS, there is not a common notation for this kind of systems, which are also indicated as Mixed IDS [44] or Distributed IDS [45]. Despite of the notation, CIDS consists of several IDS (e.g., HIDS, NIDS) over a large network, which collaborate together through centralized or distributed architectures. In a centralized architecture (see Figure 47.c), distilled events, alerts, and data are converted into common formats and delivered to a central analyzer that aggregates information and analyses it. This is the typical commercial solution, where a centralized dashboard collects data through SIEM tools. In a distributed architecture, every H/NIDS works independently, but reports alerts to the other systems, which collectively judge the severity of the threat and proactively add rules to their block list. Distributed architectures have been expensively investigated in the literature, especially for the cloud [211], but they are of limited interest in commercial implementations.



a) Host-based IDS **b) Network-based IDS** **c) Compound IDS**

Figure 47. Different types of IDS.

B.2.2 Detection methodologies

Techniques for intrusion detection can be coarsely classified into three main methodologies: signature-based, anomaly-based, state-based [45].

Signature-based detection aims at identifying known patterns for attacks. The signature is a set of rules for identifying behavioural patterns in the network traffic (i.e., sequence of packets), applications (i.e., sequence of requests and/or operations), system usage (i.e., system calls). For example, in Snort, a popular network intrusion detection software, the parts of a signature are the header (e.g., source address, destination address, ports) and its options (e.g. payload, metadata) which are used to determine whether or not the network traffic corresponds to a known signature. The main challenge for signature-based algorithms is the identification of known patterns even when they are interleaved with other (legal) events. Such mechanisms are able to attain high levels of accuracy (i.e., low number of false positive), yet the detection is only possible for known attacks. Therefore, there main limitation is the impossibility to detect zero-day attacks and variation of known attacks, hence detection systems based on this approach can be easily defeated by the polymorphic nature of attacks [212] [213]. The ease in creating, maintaining and updating the signature is one of the main motivation for the popularity of these mechanisms, which are largely used in firewalls, intrusion detection systems, and virus scanner. On the other hand, due to an exponential increase in the variety of attacks, signature based IDS also face the challenge of a rapid increase in signature rules. Identification of the signature for new attacks or variants of existing attacks (also known as a mimicry attack) is often a manual process, based on human's skills; keeping the signature database up to date is therefore a time-consuming task. For instance, Snort has more than 2500 signature rules [214].

Anomaly-based detection aims at identifying deviations from the normal system behaviour. The normal system behaviour is defined statistically by usage profiles, which consider the legitimate behaviour of users in terms of regular operations, network connections, hosts, and peers. The anomaly is a meaningful difference in the statistics of reference attributes (failed login attempts, file system access, applications used, e-mails sent, received/transmitted network packets) over a period of time. There are two main difficulties with this mechanism. The first is the initial identification of legitimate profiles, which likely differ for different users (e.g., administrative staff, commercial staff, development staff, system administrators) and in time (new applications installed, new procedures adopted). To this purpose, a broad range of techniques can be used, including data mining, statistical modelling, machine learning, and hidden Markov models. The second is the application of statistical tests: anything abnormal is considered an attack, therefore false positives can likely occur when minimal deviations are reported [215] [216]. On the other hand, this methodology is explicitly designed to overlook the knowledge of attack patterns, so it can detect zero-day and mimicry attacks. However, this also means that such approaches lack the semantics that are necessary to understand the contextual nature of an attack and its consequences.

Stateful-protocol analysis is based on precise modelling of the system state evolution. For example, requests can be paired with replies. The fundamental difference with anomaly detection is that the legitimate behaviour is build by the knowledge of the protocol/application, rather than inferred by statistical usage. Therefore, it may be a long and cumbersome process, especially for complex protocols and applications.

Table 9 reports a short summary evaluation of the alternative detection methodologies considered so far.

Table 9. Comparison among alternative detection methodologies [44]**.**

	Signature-based	Anomaly-based	State-based	Hybrid
Pros	• Contextual analysis • Easiness in creating rules • Effective detection • High true positives for known attacks	• Less dependent on protocols, applications, operating systems • Effective against zero-day attacks • Facilitate detection of privilege abuse • Easy to automate the whole process	• Know and trace the legitimate behaviour from certain knowledge • Effective against zero-day attacks	• High detection rate, low false positives
Cons	• Ineffective to detect unknown attacks, evasion attacks and variants of known attacks • Difficulty to keep signatures/patterns up to date • Manual identification of attacks • High false negatives for unknown attacks	• Need to periodically adapt the profiles • Need time to build the profiles • No protection while building the profiles • Need time to compute statistics and trigger alarms	• Consume resources to trace and check the states • Unable to detect attacks hidden behind legitimate behaviour • Complex and manual procedures to describe correct state sequences	• High computation costs • High resource usage

To improve the likelihood and accuracy of the detection, most IDS combine multiple methodologies. For example, signature-based and anomaly-based methods are largely complementary, because the former cover known attacks (with high accuracy) whereas the latter can identify new and mimicry attacks (improving the effectiveness).

A common hindrance to all techniques is the uncertainty and inaccuracy of true and reliable data to build pattern rules or to learn usage profiles. The application of soft-computing techniques such as Artificial Neural Networks, Fuzzy logic, Support Vector Machines, Genetic Algorithms, etc. can be used to improve the accuracy and efficiency of the detection.

B.2.3 Distributed detection

Centralized architectures that mix both local and remote processing are the main target for the ASTRID project. In this respect, understanding the main requirements in terms of scaling, data collection, computation offloading is important especially in the design phase. Since the elaboration of new detection algorithms is not listed among the main objectives of the Project, the analysis of the scientific literature is deliberately restricted to a few papers among the most recent that point out different approaches and methodology.

Oberheide et al. [112] were among the first researchers to develop a distributed antivirus (CloudAV). Their architecture is based on three main elements: *i)* a host agent, which detects new files and sends them for analysis to the network service (monitoring both process execution and file system changes); *ii)* a network service, which analyses files with N detection engines in parallel (which are common antivirus software deployed in VMs); and *iii)* archival and forensics service, which store information about what files were analysed and provides a query and alerting interface for operators. The usage of multiple detection engines in parallel increases the likelihood of detection and enable correlation between analysis from different engines, but also raises false positives compared to a 1-version engine. Numerically, they measured 35% better coverage than single antivirus and 98% detection coverage of an entire dataset. The local agent only checks whether each new file has already been analysed previously; no real local computation is performed, but many files might be transmitted on the network.

Therdphapiyanak and Piromsopa [217] proposed a K-Means clustering on Hadoop for analysis of security logs. They collected logs from Apache by Flume, and store them in HDFS. They used Mahout/Hadoop for clustering logs based on the distance from the center of each cluster. The main contribution of their work is the definition of a method to convert textual logs into sequence of numbers. Their proof-of-concept worked, but there is no clear indication about the usage of such system as IDS.

Zargar et al. [114] designed a collaborative framework among IDPS deployed in different domains of a cloud environment (host, network, VMs). Their framework share information at both the local (in clusters of similar hosts) and global level (among clusters), so that attacks and security events can be correlated to identify complex patterns. The shared information includes attack models (in the form of attack trees), policies (about collection of information, attack detection, mitigation actions), rules (to control information flows among the different domains), and audit logs. The duplication at the local and global level introduces redundancy and also communication overhead. Their work identifies the motivations and benefits behind their approach, but does not specify how security data are correlated.

Dastjerdi et al. [116] designed an agent-based framework for virtualised services. Their approach is based on Static and Mobile Agents (SA/MA). The former are inspection tools statically deployed in each VMs, which are responsible to detect suspicious conditions and to report to a central IDS Control Centre (IDS CC). The latter are detection agents for specific attacks, which are sent to VMs to investigate in detail suspected conditions. The system also envisions the possibility of cooperation between multiple domains, by a neighborhood watching mechanisms. The authors claim the implementation of a prototype based on the TRACY toolkit, but evaluation is only limited to comparing network load with a client/server case, without clear explanation about test conditions and the software used for comparison.

Vieria et al. [115] proposed a distributed IDS for the Grid. Their approach is based on standalone IDSes that share the security context for detection. They consider both messages exchanged by the grid nodes and logs from the grid middleware. Detection is based on behavioural anomalies (through neural networks) and knowledge (rules-based). Evaluation is very limited: false negatives and false positives get by the neural network under different learning periods, and detection times for rules matching.

Siddiqui et al. compared detection results for APTs for k-Nearest Neighbours (k-NN) and fractal correlation dimension, observing that the latter performs better than simple Euclidean dimension based algorithm primarily because of its capability to extract multiscale measurement with is better than ono scale Euclidean measure.

Friedberg et al. [218] developed Automatic Event Correlation for Incident Detection (AECID), a tool that collects log files from multiple sources and creates binary fingerprints that can be used more practically in mathematical models than text strings. Their tool correlates events, by creating hypotheses about causes and effects. If a hypothesis can be confirmed as valid, it becomes a part of a model of the system. These rules are continuously evaluated, extended and updated to build a dynamic notion of normality. Statistical analysis of correlation results allows the system to infer the degree of deviation and determine a degree of anomalous behaviour.

B.3 Enforcement of network policies

The process of enforcing network policies consists in comparing network packets with a set of rules. The rules are typically in the form ‘if <condition> then <action>’, so the enforcing process is usually based on two operations:

- *packet filtering and classification*, which inspects protocol headers and/or the packet body to match specific values or bit patterns in the enforcing rules;
- *packet manipulation*, which involves a number of actions to be applied to the packet, based on the result of the classification.

The result of the classification is usually a binary indication <match/do not match> relative to the condition, which determines whether the action is applied or not. In the simpler systems, the action is typically limited to “accept” or “drop”[4]. In more complex systems, actions may also include packet manipulation (e.g., by natting), redirection (e.g., pushing a VLAN tag to quarantine or forward the packet to specific inspection tools), state update (e.g., to apply final actions based on the result of multiple rules), logging, etc.

While considering network packets alone is the main issue in many simple contexts, taking into consideration also users and resources is a common need in more complex enterprise environments. In this case, internal security policies are mainly based on the identification of users and the resources they are allowed to use, and packet inspection falls short to achieve this purpose. For this reason, solutions like Network Access Control (NAC) and Identity-Based Networking Services (IBNS) integrate firewalls, authentication servers, access control mechanisms.

In the following, we limit the discussion to firewalls, which at this stage appears the most relevant aspect for ASTRID.

B.3.1 Firewalls

Though the general concept of firewalling is known to everyone, there are different types of firewalls, with different technical implications.

Stateless packet filters are the simplest form of firewalling. They take the decision whether accept or reject each packet based exclusively on the protocol headers and incoming/outgoing interface. There is no state kept, so there is no way to identify logical flows (e.g., a TCP connection) or to match responses with requests (e.g., in case of simple UDP transactions). They are very simple to implement and result in extremely fast packet processing. The lack of state hinders the dynamic identification of logical flows, so rules often result in coarse policy definition (e.g., accept all packets with TCP’s ACK flag set) or frequent manual intervention.

Stateful packet filters process packets by considering both their protocol headers (usually limited to IP and TCP/UDP/ICMP) and an internal state. The state is used to follow the protocol behaviour (e.g., the different phases of a TCP connection), and to selectively accept/discard packets based on its correct evolution. For example, a stateful packet filter may be configured to allow outbound connection to HTTP servers; incoming packets from HTTP servers are only accepted after the initial outbound request. The richer semantics and the dynamic behaviour turn to more complexity in their implementation, and the need for additional resources to keep the connection states. This may turn into a limitation (max numbers of connection) or vulnerabilities (resource exhaustion in case of DoS attacks). Application of stateful packet filters is natural for connection-oriented protocols (i.e., TCP), but trickier for connectionless protocols (e.g., UDP). Anyway, simple state machines can also be devised for the latter, based on the assumption that they are often used in request/response schemes with short timeouts.

[4] Some tools distinguish between “reject” and “filter”, based on the fact that the packet is either silently discarded or a notification message is sent back to the origin (usually by ICMP).

Circuit-level gateways operate at the transport layer of the OSI or internet reference models and, as the name implies, implement circuit level filtering rather than packet level filtering. It checks the validity of connections (i.e. circuits) at the transport layer (typically TCP connections) against a table of allowed connections, before a session can be opened and data exchanged. The rules defining a valid session prescribe, for example, the destination and source addresses and ports, the time of day, the protocol being used, the user and the password. Once a session is allowed, no further checks, for example at the level of individual packets, are performed. There are two operating modes for these devices: transparent and circuit-breaking. In transparent mode, the hosts are not aware of the firewall, which is traversed by the traffic and checks that every packet belongs to an allowed connection. In the other mode, the host that originates the connection must first connect to the firewall and declare the intended endpoint. After verification and (optionally) authentication, the firewall sets stateful packet rules for the specific connection; in addition, it may hide the identity of the origin, hence implementing privacy services. The control channel between the host and the firewall uses the main connection (e.g., SOCKS protocol) or a dedicated connection (e.g., MEGACO and IETF MidCom protocols). Circuit-level gateways are very useful for multimedia protocols as SIP and H.323, where the media are transmitted with RTP/UDP over dynamic ports.

Application-Level Gateways (ALG) improves the accuracy of the detection by deep-packet inspection, including innermost applications. This solves many limitations of the other approaches, that implicitly assume a tight correspondence between transport ports and applications. ALG enables to check the correctness of the transactions at the application layer, validating the remote applications and the data exchanged over the connection. They can operate both in transparent and non-transparent mode. In transparent mode, the ALG is basically a deep-packet inspection firewall. In non-transparent mode, the client connects to the ALG and indicates the destination endpoint, then all messages are directly sent to the ALG, that operates in a man-in-the-middle fashion. Authentication may be required by the ALG. Non-transparent mode is typical of web proxies. Given the specific design with each application, ALG are able to understand what ports should be dynamically open, e.g., in case of multimedia protocols. Alternatively, specific protocols can be used as for circuit-level gateways.

B.4 Security Analytics

The Cyber Research Alliance (CRA) identified the application of Big Data Analytics to cyber security as one of the top six priorities for future cyber security research and development [204]. Big Data Analytics (BDA) is the aggregating and correlating of a broad range of heterogeneous data from multiple sources, and has the potential to detect cyber threats within actionable time frames with minimal or no human intervention [204].

Security Analytics is the application of Big Data Analytics to cyber security. Security Analytics is a new trend in the industry, and interest is expected to gain momentum quickly. Finding appropriate algorithms required to locate hidden patterns in huge amounts of data is just one of the several challenges that must be overcome. Incomplete and noisy data are additional factors that must be considered. Finally, the massive scale of enterprise security data available poses the greatest challenge to a successful Security Analytics implementation [204].

Security Analytics differs from traditional approaches by separating what is normal from what is abnormal. In other words, the focus is on the action or user activity instead of the payload content or signature [205].

B.4.1 Background technologies

Big Data

Most computer systems record events in log files [219]. Operating systems, firewalls, and Intrusion Detection Systems (IDS) record event information in log files. Applications also record user activities in log files. The type and structure of log files vary widely by system and platform. For example, weblogs are produced by web servers running Apache or Internet Information Server (IIS) among others. Any activities performed during a security breach will most likely result in log entries being recorded in one or more log files. These attacks cannot be identified by a single log entry occurrence; instead, they can be identified through a series of entries spanning several minutes

The amount of data logged per system can be more than several thousand events per minute. Additionally, these files are typically distributed across the network.

In order to process and analyze the log data, they must be integrated and stored in a central location. Integrating highly heterogeneous data from multiple sources requires a massive centralized data repository [220]. Such a data repository should meet the complexity requirements as defined by Big Data.

Big Data is defined by three characteristics: volume, velocity, and variety. Volume is the size of the data stored and is measured in terabytes, petabytes, or Exabytes. Velocity is the rate at which data is generated. Variety refers to the types of data, such as structured, semi-structured, or non-structured [221].Structured data is data that typically reside in a database or data warehouse. Examples of unstructured data are documents, images, text messages, and tweets. Log data is considered semi-structured. In some cases, log data contains key-value pairs or is stored in CSV format. Adam Jacobs, in "The Pathologies of Big Data," defines Big Data as "data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time" [222].

Big Data presents new challenges to searching and processing of data. These new challenges require new techniques and methods, such as data mining or Big Data analytics.

Big data analytics employs data mining techniques for extracting actionable insights from data to make intelligent business decisions [223].

Commonly, the first step in Big Data analytics is Extract Transform Load (ETL) [221].This is a pre-processing step that transforms data into a format that is compatible with data mining algorithms [221].

The processing or analysis step applies an algorithm, such as clustering, to the transformed data. Finally, the results are displayed on a dashboard or in a report. Data mining is defined as the application of machine learning methods to large datasets [223].

Machine Learning

Machine learning (Figure 48) is a subfield of artificial intelligence that allows a computer to learn using sample data without being programmed to anticipate every possible situation [224]. The two most common types of machine learning are supervised and unsupervised learning. Supervised learning is used when a dataset of labelled instances is available. Supervised learning is used to solve classification problems. The goal of supervised learning is to train the computer to learn to predict a value or classify an input instance accurately. Unsupervised learning is used when a labelled dataset is not available. Clustering is an unsupervised learning technique which results in grouping similar instances in clusters. Clustering is used to discover patterns in data. In some cases, clustering is performed to classify an unlabelled dataset and using the resulting classified dataset for supervised learning [224].

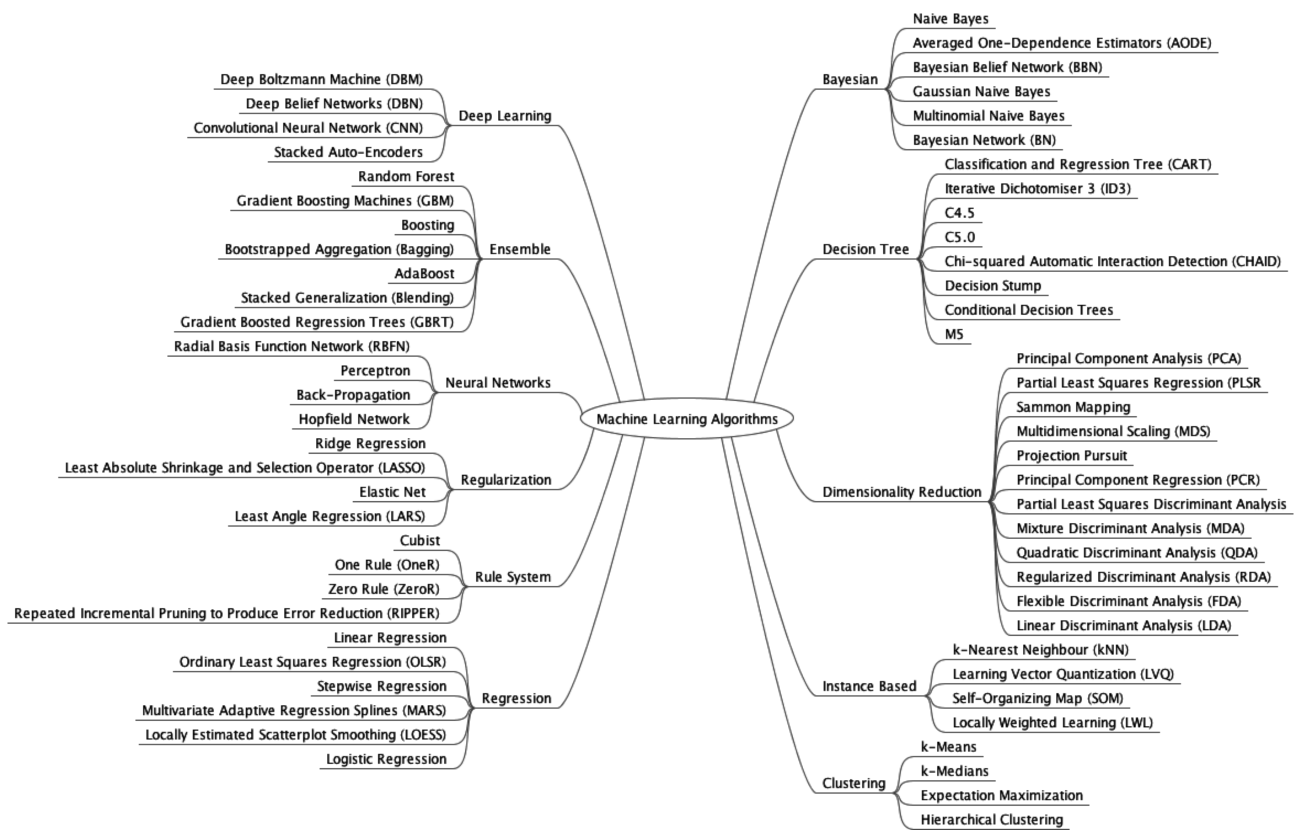

Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Figure 48. Machine Learning Algorithms.

Artificial Neural Network

Artificial Neural Network (ANN), proposed fifty years ago, is a collection of supervised learning models inspired by the human brain. A simple neural network or multi-layer perceptron is composed of three layers; an input layer, a hidden layer, and an output layer. Each layer is composed of neurons, which are interconnected to all the neurons in the next layer. The network is trained by adjusting the weights of the neurons to minimize the error between the output neuron and the desired result [225]. A neural network (Figure 49) using a large number of hidden layers is referred to as a deep neural network and training is referred to as deep learning.

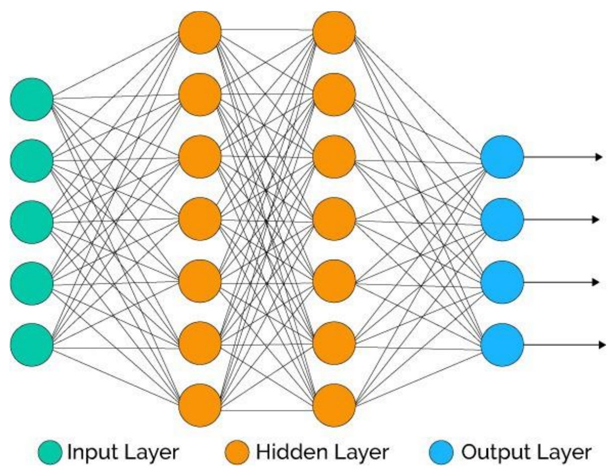


Figure 49. Neural Network Diagram.

In 2006, Geoffrey Hinton and Ruslan Salakhutdinov developed techniques using multiple hidden layers. Pre-training was one such technique where the upper layers extract features with a higher level of abstraction which is used by the lower layers for more efficient classification. Unfortunately, since this technique requires billions of floating point operations, it was not computationally feasible until recently. The recent advent of technological advances in hardware caused a resurgence of interest due to the resulting improvements to performance. For example, a researcher at the Switzerland-based Dalle Molle Institute for Artificial Intelligence claims in one instance the training phase took only three days using graphic processing units (GPUs) where using CPU's would have taken five months [225]. Deep learning works well with large datasets of labelled data.

Annex C Communication interceptions and Digital Forensic

A non-exhaustive list of some of the most popular free tools and toolkits (i.e. collections of tools) exploitable for communication interceptions and for digital forensic investigations.

1. TOOLS FOR TRAFFIC INTERCEPTION AND ANALYSIS

1.1. Aircrack-ng (https://www.aircrack-ng.org/)

Aircrack-ng is a complete suite of tools to assess Wi-Fi network security; it also allows to crack WEP and WPA PSK Wi-Fi passwords.

1.2. CoWPAtty (https://tools.kali.org/wireless-attacks/cowpatty)

CoWPAtty allows to perform offline dictionary attacks against WPA-PSK Wi-Fi networks

1.3. Wireshark

Wireshark is a widely-used IP network protocol analyzer; it allows to capture and display network traffic message structure.

1.4. TCPDump (https://www.tcpdump.org/manpages/tcpdump.1.html)

TCPDump prints out a description of the contents of packets on target network interfaces; packet data may be saved into a file for later analysis

1.5. DPDK (https://www.dpdk.org/)

DPDK is a set of libraries to be utilized to accelerate packet processing workloads (for, e.g., filtering and capturing packets) running on a wide variety of CPU architectures.

2. TOOLKITS FOR FORENSIC INVESTIGATIONS

2.1. CAINE (https://www.caine-live.net/)

CAINE (Computer Aided INvestigative Environment) is a GNU/Linux live distribution created as a Digital Forensics project. CAINE offers a complete forensic environment that is organized to integrate existing software tools as software modules and to provide a friendly graphical interface.

2.2. DEFT

DEFT is a Linux Live CD which includes some of the most popular free and open source computer forensic tools available; among others, it contains tools for Mobile Forensics, Network Forensics, Data Recovery, and Hashing

2.3. HELIX3 Free (http://www.e-fense.com/products.php)

HELIX3 Free is toolset, which includes some open-source software programs for computer forensic, ranging from hex editors to data carving software to password cracking utilities, and more; it is a Live CD based on Linux that was built to be used in Incident Response, Computer Forensics and E-Discovery scenarios

2.4. Paladin Forensic Suite (https://sumuri.com/software/paladin/)

Paladin Forensic Suite is a Live CD based on Ubuntu that includes a large set (80+) of open source forensic tools, which are organized into categories, including Imaging Tools, Malware Analysis, Social Media Analysis, Hashing Tools, etc.

2.5. **SANS SIFT** (https://digital-forensics.sans.org/community/downloads)

The SANS Investigative Forensic Toolkit (SIFT) is an Ubuntu based Live CD which includes tools to conduct a forensic or incident response investigation. SIFT includes tools such as log2timeline for generating a timeline from system logs, Scalpel for data file carving, Rifiuti for examining the recycle bin,

3. TOOLS FOR DIGITAL FORENSIC INVESTIGATIONS

3.1. **Bulk Extractor** (http://www.forensicswiki.org/wiki/Bulk_extractor)

Bulk_Extractor is a computer forensics tool that scans a disk image, file, or directory of files and extracts contents, matching specific patterns, which may consist of credit card numbers, web domains, e-mail addresses, URLs; the extracted information is output to a series of text files which can be reviewed manually or analysed using other forensics tools or scripts.

3.2. **Cain and Abel** (http://www.oxid.it/cain.html)

Cain and Abel are a password recovery tool for Microsoft Windows, using methods such as network packet sniffing, dictionary attacks, brute force and cryptanalysis attacks.

3.3. **CrowdStrike CrowdResponse** (https://www.crowdstrike.com/resources/community-tools/crowdresponse/)

CrowdResponse is a lightweight console application that can be used as part of an incident response scenario to gather contextual information such as a process list, scheduled tasks, or Shim Cache. Using embedded YARA signatures, it also allows to scan hosts for malware and report if there are any indicators of compromise.

3.4. **ExifTool** (https://www.sno.phy.queensu.ca/~phil/exiftool/)

ExifTool is a command-line application used to read, write or edit file metadata information; it can be used for analyzing the static properties of suspicious files in a host-based forensic investigation.

3.5. **FireEye RedLine** (https://www.fireeye.com/services/freeware/redline.html)

RedLine offers the ability to perform memory and file analysis of a specific Windows host. It collects information about running processes and drivers from memory, and gathers file system metadata, registry data, event logs, network information, services, tasks, and Internet history to help build an overall threat assessment profile.

3.6. **Free Hex Editor Neo** (https://www.hhdsoftware.com/free-hex-editor)

Free Hex Editor Neo is a basic hex file editor (i.e. able to handle files down to single byte level); it is useful for loading large files (e.g. database files or forensic images) and performing actions such as manual data carving, low-level file editing, information gathering, or searching for hidden data.

3.7. **FTK Imager** (https://accessdata.com/product-download)

FTK allows to examine files and folders on local hard drives, network drives, CDs/DVDs, and review the content of forensic images or memory dumps. FTK Imager can also be used to create SHA1 or MD5 hashes of files, export files and folders from forensic images to disk, review and recover files that were deleted from the Recycle Bin (providing that their data blocks haven't been overwritten) and mount a forensic image to view its contents in Windows Explorer.

3.8. <u>**HxD**</u> (https://mh-nexus.de/en/hxd/)

HxD is a further hex editor that allows to perform low-level editing and modifying of a raw disk or main memory (RAM).

3.9. <u>**LastActivityView**</u> (http://www.nirsoft.net/utils/computer_activity_view.html)

LastActivityView is a tool for Windows operating system that collects information from various sources on a running system and displays a log of actions made by the user and events occurred on this computer. The activity displayed by LastActivityView includes: Running .exe file, Opening open/save dialog-box, Opening file/folder from Explorer or other software, software installation, system shutdown/start, application or system crash, network connection/disconnection and more

3.10. <u>**Linux 'dd'**</u>

'dd' comes by default on the majority of Linux distributions (e.g. Ubuntu, Fedora). The tool can be used for various digital forensic tasks such as forensically wiping a drive (zero-ing out a drive) and creating a raw image of a drive. A patch to the GNU dd program, named 'dc3dd' is available online (http://sourceforge.net/projects/dc3dd/); this version has several features intended for forensic acquisition of data, including hashing on-the-fly, split output files, and file verification.

3.11. <u>**PlainSight**</u> (http://www.plainsight.info/download.html)

PlainSight is a Live CD based on Knoppix (a Linux distribution) that allows you to perform digital forensic tasks on Windows targets, such as viewing internet histories, data carving, USB device usage information gathering, examining physical memory dumps, extracting password hashes, and more.

3.12. <u>**ShellBagger**</u> (http://www.4discovery.com/our-tools/)

ShellBagger analyze Microsoft Windows "ShellBags" information; Windows tracks user window viewing, by storing the size, view, icon, and position of a folder from Windows Explorer; this information is referred to as "ShellBags", and are stored in several locations within the Registry; these keys can be extremely useful to a forensic investigator since the ShellBags are persistent and remain behind even if the directory is removed.

3.13. <u>**The Sleuth Kit and Autopsy**</u> (http://www.sleuthkit.org/)

The Sleuth Kit is an open source digital forensics toolkit that can be used to perform in-depth analysis of various file systems. Autopsy is a GUI that sits on top of The Sleuth Kit and provides with features like Timeline Analysis, Hash Filtering, File System Analysis and Keyword Searching out of the box, with the ability to add other modules for extended functionality.

3.14. <u>**USB Historian**</u> (http://www.4discovery.com/our-tools/)

USB Historian parses USB information, primarily from a Windows platform's registry, to provide a list of all USB drives that were plugged into the machine; it displays information such as the name of the USB drive, the serial number, when it was mounted and by which user account.

3.15. <u>**Xplico**</u> (http://www.xplico.org/download)

Xplico is an open source Network Forensic Analysis Tool (NFAT) that extracts applications data from internet traffic (e.g. Xplico can extract an e-mail message from POP, IMAP or SMTP traffic).

4. TOOLS FOR SECURITY INFORMATION AND EVENT MANAGEMENT (SIEM)

Security Information and Event Management (SIEM) solutions combine services of Security Information Management (SIM) and Security Event Management (SEM) solutions, providing real-time analysis of security alerts but also forensic analysis by searching across information for different time periods.

4.1. **SELKS** is both Live and installable ISO based on Debian implementing a ready to use Suricata IDS/IPS. The name comes from its major components: Suricata Elasticsearch Logstash Kibana Scirius. After starting or installing SELKS, you get a running Suricata with IDPS and NSM capabilities, Kibana to analyze alert and events and Scirius to configure the Suricata ruleset. SELKS is released under GPLv3 license. Sources, issues tracker and wiki are hosted on GitHub.

4.2. **Security Onion** is a Linux distro for IDS (Intrusion Detection System) and NSM (Network Security Monitoring). It's based on Xubuntu 12.04 and contains Snort, Suricata, Sguil, Squert, Xplico, tcpreplay, scapy, hping, and many other security tools.

4.3. **Simplewall** is a software appliance SMB firewall. It delivers simplest and most user friendly UTM for small & medium sized businesses globally to protect their business networks, maximize bandwidth usage and define personalize content filtering policies for employee productivity. Most importantly, Simplewall is easy to setup and manage for the system administrator as well as to the business owner.

4.4. **OSSIM** is a software appliance that contains a Host Intrusion Detection System (HIDS), a Network Intrusion Detection System (NIDS), and Security Information and Event Management (SIEM).

Annex D Description of requirements

ID	Priority	Usage Scenario
R001-CS-DSG	Top	SAW, MON
Title		
Data correlation over the whole graph		
Description		
One of the main pillars behind the ASTRID concept is the ability to detect multi-vector attacks and advanced persistent threats. This implies the capability to analyze and correlate the security context in both space and time dimensions. In this respect, there must be a common and logically centralized detection logic, which may include multiple analysis algorithms for specific threats and attacks.		
Notes		
The detection logic may be deployed as part of the service graph and share the same lifecycle. Alternatively, the detection logic may be an external component which exists independently of the specific service graph (see also R002-CS-DSG).		

ID	Priority	Usage Scenario
R002-CS-DSG	Medium	SAW
Title		
Data correlation over multiple service graphs		
Description		
As the level of automation grows, many similar services may be instantiated from a common template, hence sharing vulnerabilities. In addition, the increasing adoption of new business models, where services and processes from multiple organization are pipelined to create new value chains, is creating stronger security dependencies among organization and their business. A compromised service from a supplier becomes a privileged attack vectors towards its customers. Similarly, advanced persistent attacks might carry out different intrusion attempts on interconnected services. The ability to correlate security-related data and events from multiple services is expected to improve the detection capability, while propagating faster alerts along the business chain. Similar to R001-CS-DSG, a common detection logic is necessary to effectively correlate data from multiple sources.		
Notes		
A detection logic common to multiple services must be deployed outside the single graphs. Sharing security data among different service providers brings complex concerns about privacy and access control.		

ID	Priority	Usage Scenario
R003-CS-DSG	Top	SAW, DFW, MON, SAF
Title		
Local processing and programmability		
Description		
The need for deep correlation and centralized processing (see R001-CS-DSGR001-CS-DSG) poses efficiency concerns on the retrieval of security data. The two main issues are bandwidth usage to transfer that information and latency. It is obvious that many inspection operations cannot be performed in a centralized way but must be implemented locally: the analysis of network packets is the most obvious example. Therefore, local agents must be deployed in all service functions, which are responsible for data collection, fusion, and access. To improve the overall efficiency, it is also necessary to collect the right amount of information; for example, when early signs of attacks are detected or new threats are discovered, it may be necessary to analyze finer-grained information than during normal operation. Fine-grained information may consist in more frequent sampling values or additional inspection data (for example, in case of network packets the innermost protocol bodies may be checked in addition to standard TCP/IP headers). "Programmability" is therefore required to change the monitoring processes at run-time.		
Notes		
The minimal requirement on programmability is the ability to run pre-defined inspection tasks. The generation of run-time code according to high-level instructions or policies (i.e., a sort of "compiler") represents a great improvement, but it falls outside the Project's objectives.		

ID	Priority	Usage Scenario
R004-CS-DSG	Top	All
Title		
Secure communication		
Description		
According to R001-CS-DSG and R003-CS-DSG, the ASTRID monitoring and detection framework will have a distributed architecture, made of local processing tasks and centralized analysis algorithms. Secure communication is needed to retrieve security-related data and to steer local inspection and enforcement tasks. The specific target is to avoid increasing the attack surface, by giving the attacker the possibility to alter collected data or inject malicious operations. Keeping into account this target, the following security services are expected: • confidentiality; • integrity; • authentication.		
Notes		
Secure communication mechanisms are not restricted to pure encryption but must also include integrity check and digital signature (especially for configurations and programs). Authentication may either leverage a PKI infrastructure or use some password-based mechanism (e.g., LDAP, Kerberos).		

ID	Priority	Usage Scenario
R005-CS-DSG	Medium	SAW, MON, RAF
Title		
Historical data		
Description		
Both detection and investigation of attacks often require the availability of historical data to identify possible patterns and correlations.		
Notes		
This requirement implies other implementation-specific requirements. First, the storage capacity that should be sized according to the maximum time window for extracting relevant patterns and time series. Second, confidentiality, integrity, and certificability of the data (see also R006-CS-DSG and R040-LAW-FUN).		

ID	Priority	Usage Scenario
R006-CS-DSG	Low	All
Title		
Privacy and sensitive data		
Description		
Inspection of applications' logs and network packets may disclose private or sensitive information. The most straightforward example is the logs from a web proxy, which records the web sites and frequency access. To address this issue, monitored data should be classified and labelled according to the presence of private or sensitive data, and there should be some technical or procedural mean to limit the processing or extrapolation of such kind of information.		
Notes		

ID	Priority	Usage Scenario
R007-CS-DSG	Top	All
Title		
Access control		
Description		
Monitoring data and security events are very critical information for privacy issues and for safe operation of the related service. It is therefore important to establish different roles in the systems, with different privileges in accessing and modifying the security context. For example, there may be an administrator role, who can change any configuration anytime; there may be a sink role, which can access data and logs, but cannot configure the monitoring/inspection processes; there may be a collector role, which can change the monitoring/inspection tasks. The access control is based on fine-grained policies, like in ABAC (Attribute Based Access Control) and ABEC (Attribute Based Encryption Control).		
Notes		

ID	Priority	Usage Scenario
R008-CS-DSG	Top	All
Title		
Repository of trusted security programs		
Description		
Based on R003-CS-DSG, there must be a collection of monitoring, inspection, and enforcement tasks that can be dynamically loaded at run-time by the security agents deployed in virtual functions. The repository must ensure the origin, integrity, and trustworthiness of such programs.		
Notes		
The programs are executable (either as byte-code instructions or by an interpreter). There is no need for access control to the repository, since the programs do not contain any private information.		

ID	Priority	Usage Scenario
R009-CS-DSG	Top	SAW, MON, SAF, RAF
Title		
Collection of detection algorithms		
Description		
Based on R001-CS-DSG, detection is performed by multiple algorithms. A collection of available algorithms is required, with proper metadata to describe their function, control interface, and execution requirements.		
Notes		
The programs are executable (either as byte-code instructions or by an interpreter). They access security data through a common abstraction interface (see R013-CS-FUN).		

ID	Priority	Usage Scenario
R010-SD-DSG	Top	All
Title		
Integration with orchestration tools		
Description		
The on-going evolution towards more automation in IT operations, needed to keep the pace with ever shorter software life-cycles, also encompasses security management. The deployment, configuration, and operation of security appliances is expected to adapt to these emerging trends, based on software orchestration. The ASTRID framework must therefore be designed as “orchestratable”: that means is should be conceived as a set of virtual functions, configuration hooks, and management policies that can be easily adapted into different orchestration paradigms.		
Notes		
The lack of a common standard for software orchestration hinders the definition of a specific implementation requirements. The requirement on orchestrability is therefore kept at the design level, meaning that the whole structure should be decomposable in multiple components and implemented as “virtual functions” or “micro-services”. The Project plans to use two different orchestrators to verify the achievement of this requirement.		

ID	Priority	Usage Scenario
R011-CS-FUN	Top	SAF
Title		
Hybrid software vulnerability analysis covering both pre- and after-deployment of software services		
Description		
The ASTRID framework will combine static and dynamic software analysis tools so as to design a hybrid VA tool which leverages the properties of both approaches in a complementary manner, to find deeper bugs not only during the creation of a micro-service but also after its deployment. The goal is to investigate the application of traditional source code analysis techniques coupled with more advanced control flow attestation mechanisms for capturing diverse instantiations of software exploits that hijack a program's control flow. Such attacks tamper with state information in the program's data memory area, e.g., the stack and the heap. Software bugs allow an attacker to arbitrarily alter state information and hijack the program flow of applications to induce malicious operations.		
Notes		

ID	Priority	Usage Scenario
R012-CS-FUN	Top	SAF
Title		
Support Vulnerability Analysis Protocol and Algorithms Agility		
Description		
The ASTRID framework will combine static and dynamic software analysis tools so as to design a hybrid VA tool. This novel component must allow cyber-security staff to choose among various software threat analysis primitives and protocols (static and/or dynamic) that provide a specific service.		
Notes		

ID	Priority	Usage Scenario
R013-CS-FUN	Top	SAW
Title		
Access to heterogeneous security context		
Description		
The detection of complex Advanced Persistent Threats is largely based on the availability of data and events from multiple sources (packet headers, application logs, system calls). These data are usually collected by different subsystems, and different interfaces are currently used to export this information. In addition, different detection algorithms may require accessing the same data, and it is important to avoid wasting bandwidth and resources by duplicating the retrieved information. It is therefore necessary to implement a common framework for the collection of data, which he also responsible for data abstraction and fusion. The framework must abstract the underlying graph topology and available security context, also mediating conflicting requests by the detection algorithms (e.g., in terms of frequency of collection or header fields to be inspected).		
Notes		
The same framework is also expected store historical data and to integrate with the certification process envisioned by R040-LAW-FUN.		

ID	Priority	Usage Scenario
R014-CS-FUN	Top	SAW, DFW, MON
Title		
Packet inspection		
Description		
One of the most valuable monitoring targets for NFV is network traffic. The ASTRID framework must support deep packet inspection, without limiting the match to standard protocol headers (i.e., IP, TCP, UDP, ICMP). Taking into account R008-CS-DSG, that means a proper number of inspection programs must be implemented.		
Notes		
Inspection programs as defined in R008-CS-DSG are expected to be programmable, so their number depends on the number of different protocols to be supported. This requirement will be translated into a more specific implementation requirement after the definition of the Use Cases (D4.1). At this stage, the minimal inspection capabilities must cover the following headers: • Ethernet • IP • ICMP • ARP • TCP • UDP		

ID	Priority	Usage Scenario
R015-CS-FUN	Medium	DFW
Title		
Distributed firewalling		
Description		
Packet filtering is the most straightforward enforcement action in NFV environments. The ASTRID framework should provide the ability to install and configure a packet filtering program, so to implement a distributed firewalling function. The framework should provide a filtering program, according to R008-CS-DSG. Configuration will enable to set the matching fields, either with exact values or wildcards.		
Notes		
This feature is already present in many open-source and commercial CMS, so the Project should demonstrate the advantages of the ASTRID framework. The target protocols are the same identified for R014-CS-FUN.		

ID	Priority	Usage Scenario
R016-CS-FUN	Top	SAF
Title		
Behavioural analysis through attestation techniques		
Description		
The ASTRID framework checks the software execution during its lifetime, to detect any anomaly that may indicate an attack. The target is protection from both software vulnerabilities and network threats, hence involving a mix of run-time code analysis, log parsing, network analytics, and packet filtering techniques. The combination of multiple heterogeneous sources allows to combine and correlate information from both. For example, logs may reveal unauthorized operations that are not visible in encrypted traffic, whereas port scanning may indicate an attempt to find backdoors or DoS attacks that cannot be inferred by logs.		
Notes		

ID	Priority	Usage Scenario
R017-SD-FUN	Medium	DFW, MON
Title		
Automatic firewall configuration		
Description		
There are different models that can be used to design service graphs, either proposed by standardization bodies, research projects, or vendors (e.g., TOSCA, NFV, Juju, UNIFY). They share the common idea of describing the service topology as graphs of nodes and links, but the semantics of these elements' changes for each paradigm. Nevertheless, a common trend is to deploy individual functions in separate virtualization environments (virtual machine, logical container, or whatever else provided by the virtualization infrastructure). That implies a lot of firewalling rules to be configured at the network level. However, the logical structure of all orchestration models enables to infer the communication requirements and to automatically configure the necessary firewalling rules.		
Notes		
Firewalling rules for internal communication can be easily inferred by logical dependencies in the service graph (e.g., database client/server). External connections might require explicit configuration from the service developer, who knows which are the service access points and any usage restriction. This configuration is expected to happen through high-level policies (see R024-CS-USB).		

ID	Priority	Usage Scenario
R018-SD-FUN	Medium	RAF, LAW
Title		
Dynamic modification of the service topology		
Description		
The programmability of the virtualization environment is not enough for implementing any security functionality devised by the ASTRID concept. For instance, for lawful interception and on-line investigation of attacks may require additional security functions. In the first case, a "duplication" function is required to decrypt and replicate traffic to an authorized agency. In the second case, honeypots are created dynamically, and suspicious network traffic is redirected there.		
Notes		
All required security functions might be included in the initial deployment of the service graph. However, this solution wastes resources and leads to higher operational costs. The modification of the service graph at run-time is not trivial in existing orchestration paradigms (the typical solution is re-deployment from scratch of a new instance). Indeed, all required security functions should be integrated in the enhanced version of the service graph created according to selected security services (see R021-SD-FUN), and then deployed only when they are going to be used. The feasibility of this approach will be verified at the design/implementation stages.		

ID	Priority	Usage Scenario
R019-CS-FUN	Low	DFW, MON, RAF
Title		
Autonomous operation		
Description		
According to the description of the usage scenarios, local programmability (R003-CS-DSG) is expected to change the scope and deep of inspection. This is useful to balance the amount of collected information to the actual detection needs, but it also helps investigate new threats and unknown attacks by defining new monitoring metrics and inspection rules. However, in case of well-known attack patterns, part of these operations could be easily automated, hence contributing to reduce both the burden of repetitive tasks on humans and the latency to detect attacks. Autonomous operation needs a reasoning engine that takes decisions based on notifications and alerting from the detection algorithms. For instance, when the network traffic intended to a specific port grows above a given threshold, it may start dropping new connections (to avoid DoS) or may install a finer-grained packet classifier that reports statistics for source address or other relevant parameters.		
Notes		
As the above description suggests, autonomous operation may leverage a policy framework (i.e., a set of rules in the form 'if even, then action'), which is the simplest form of intelligence to switch between pre-defined behaviours. A more sophisticated framework may be based on machine learning or other artificial intelligence techniques. From an implementation perspective, such intelligence may be a standalone component or may be integrated in each detection algorithm.		

ID	Priority	Usage Scenario
R020-SD-FUN	Medium	RAF
Title		
Recovery from compromised graphs		
Description		
Even with the better detection algorithm in place, it is possible that an unknown threat or a very unlikely condition be exploited to perform a successful attack. In case of critical services, it is vital to quickly recover and return to a safe state. The ASTRID framework notifies both security staff and the orchestration tool of any compromised system (detected by the run-time software analysis or reports from CERTs/CSIRTs). Under this condition, the orchestration tool replace the compromised components with clean instances to return to a safe and trusted condition.		
Notes		
Replacement of a compromised function is only possible under specific conditions. It is quite simple for stateless functions (as in case of many network functions), but it is more difficult when a state must be managed.		

ID	Priority	Usage Scenario
R021-SD-FUN	Medium	SAW, DFW, MON
Title		
Automatic enhancement of the service graph		
Description		
One of the fundamental pillars behind the ASTRID proposal is the separation of the main business logic from security management (see Objective I in the proposal). The design of suitable tools and interfaces (see R024-CS-USB) for security staff is probably the most obvious effort in this direction, but the definition of suitable interfaces between the two roles is as much important. The Service Developer is the actual designer of the service, so he does know the operation requirements and the critical aspects that should be protected. For instance, he knows what communication flows are required (both from/to the outside), what the software and the users are supposed to do, how many resources and workload the service is expected to manage. However, the Service Developers should not be a security expert and should not know the technical details of the ASTRID framework. In this respect, the GUI that he uses for service design and orchestration should include a set of "security policies" that enhance his plain design with ASTRID components. Such policies describe security features at a very high-level, easily understandable by any software designer. Examples of security policies may include: • Firewalling • DoS mitigation • Intrusion detection • Malware detection • Spoofing identification • ... The list of ASTRID components that should be embedded in the service graph includes: • local agents for monitoring, inspection, and enforcement; • detection algorithms; • certification components; • hooks for lawful interception.		
Notes		
Most common (and important) options may be enabled by default (i.e., firewalling). Additional sub-options may be included to enable specific functions (e.g., firewall type: packet filtering, deep-packet inspection, proxy).		

ID	Priority	Usage Scenario
R022-SD-IFA	Top	All
Title		
REST API		
Description		
The need to implement new paradigms for virtualized environment cannot break existing security practice and processes. In this respect, it is important to facilitate interoperability and, most of all, integrability with existing tools. The REST API will provide full control of the ASTRID framework, including the ability to: i) upload, remove, start, stop, and configure detection algorithm; ii) upload and remove inspection programs; iii) manage the certification process; iv) enforce mitigation and response actions; v) configure monitoring tools, so to directly investigate attacks that cannot be automatically detected by the algorithms; vi) configure authentication, authorization, and access control. The REST API can be used to develop user interfaces or to integrate the ASTRID framework in existing tools for monitoring, reaction, mitigation, risk assessment.		
Notes		
The definition of REST-based API is a common practice to develop alternative front-ends for a common back-end. This approach is largely used in virtualization environments; for instance, OpenStack API are implemented by both CLIs and web GUIs.		

ID	Priority	Usage Scenario
R023-CS-SUP	Low	RAF
Title		
Cyber-Threat Intelligence (CTI)		
Description		
The large number of threats and the speed at which their knowledge propagates within the hackers communities require effective means to build, conserve, and share cyber-threat intelligence. CERTs/CSIRTs networks are usually in charge of keeping vulnerabilities databases and publish security bulletins at the national level; despite of massive digitalization in all economical and business sectors, sharing of information about cyber-threats and cyber-security incidents is still largely based on paperwork and emails [226]. Indeed, national CSIRTs provide support at the procedural level, foster cooperation and sharing of best practice, but do not cover technical issues [70]. This ultimately delay the knowledge of new threats and attacks, as well as the elaboration of remediation actions and countermeasures for every different context. The increased automation in the investigation process, effectively supported by local programmability (R003-CS-DSG) and the user interface (R024-CS-USB), should also be reflected in an increasing automation in importing and exporting threat intelligence.		
Notes		
Partial automation of exporting the collected information after investigation is feasible in ASTRID. However, importing CTI is not realistic right now, because it largely depends on the complete digitalization and the interoperability of CERTs/CSIRTs networks.		

ID	Priority	Usage Scenario
R024-CS-USB	Medium	All
Title		
Graphical User Interface		
Description		
The ASTRID GUI provides visual representation of the service graph and main security concerns. By clicking on the graph nodes, it is possible to show the current monitored information. Alerts from the detection algorithms are shown prominently in the interface, capturing the attention of the operator. The combination of a general menu and a context menu provides quick access to the main functions, including configuration, management, and response. Thanks to the ASTRID GUI, security staff can focus on high-level elaboration, rather than wasting their time to learn commands and instructions to interact with the framework.		
Notes		
It is a common understanding today that graphical interfaces facilitate the representation of complex information to users. Nevertheless, some skilled users still prefer CLIs, because GUIs often do not implement all available APIs and cannot automate tasks through scripts. This basically motivates the classification as "medium" priority.		

ID	Priority	Usage Scenario
R025-CS-USB	Low	SAW, DFW, MON, SAF, RAF
Title		
Notification of security events		
Description		
The ASTRID GUI builds situational awareness for security graphs, but requires physical presence and attention from security staff. To improve the likelihood that specific events are not missed or overlooked, it is also useful to notify high-level security staff and other relevant roles, as the management and legal staff. Notifications should be sent by different media (emails, SMS, socials) and should be limited to important events that have not been managed for a certain period of time.		
Notes		

ID	Priority	Usage Scenario
R026-CS-PRF	Top	SAF
Title		
Average time to detect compromised software		
Description		
Safe and trustworthy operation relies on timely detection of attacks. Real-time detection is unfeasible for many threats, since the attack might not produce any visible effect until specific events (e.g., a software routine is executed, a backdoor is used, private data are being downloaded to external locations). According to the impact described in the proposal, the average time to detect compromised software should be less than 1h. The detection time also depends on the complexity of the attack and the service graph; the target value is set for the Use Cases that will be implemented by the Project in WP4.		
Notes		
The target value was set according to existing literature and strategic roadmap from ECSO. The value might be revised during the Project according to the evolving SoA and the indications from the Advisory Board.		

ID	Priority	Usage Scenario
R027-CS-PRF	Top	MON
Title		
Average time to detect network anomalies		
Description		
Safe and trustworthy operation relies on timely detection of attacks. Real-time detection is only possible for many threats (i.e., DoS, port scanning), whereas the detection of advanced persistent threats may require days or weeks. The detection on anomalies in the network traffic may represent an early sign of a network-based attack. At the proposal stage, a time window of 8h was set as target value for the analysis of network streams.		
Notes		
The target value was set according to existing literature and current technologies. The value might be revised during the Project according to the evolving SoA and the indications from the Advisory Board.		

ID	Priority	Usage Scenario
R028-CS-PRF	Top	SAW, MON, SAF
Title		
Average time to respond to attack		
Description		
Detecting an attack is just the preparatory phase to an effective mitigation and/or response. The effectiveness of the response is largely measured by the timeliness of the action, in order to limit the impact of the attack. A clear understanding of the current situation and automation of most mechanical processes can greatly reduce the time from detection to response. By leveraging these two aspects, ASTRID expects to keep the response time within a few minutes for known attacks.		
Notes		
The target value includes the time to notify the attack to the operator, to choose the proper remediation action (either automatically for simplest threats or manually for most complex ones), and to initiate the response process (typically through the graphical interface).		

ID	Priority	Usage Scenario
R029-CS-PRF	Low	MON, SAF
Title		
Detection rate		
Description		
The typical performance index for intrusion detection software is the detection rate, i.e., the number of successfully detected attacks over the total. The target value should be above 85%, considering the application of machine learning and artificial intelligence, which has been estimated according to reported performance of Patternex and the analysis of several artificial intelligence tools [227].		
Notes		
The detection rate is among the most important evaluation parameter for any IDS. This performance index is not mandatory for ASTRID because the Project is not explicitly committed to create new detection algorithms.		

ID	Priority	Usage Scenario
R030-CS-PRF	Low	MON, SAF
Title		
Precision of detection		
Description		
Beyond the detection rate, it is a common belief that false positives overwhelm the system and reduce the attention of security operators. It is therefore important to achieve high rates of precision in the detection of attacks, possibly in the order of 99%. The estimation is based on the same sources as the detection rate (R029-CS-PRF).		
Notes		

ID	Priority	Usage Scenario
R031-CS-PRF	Medium	SAF
Title		
Performance of attestation services		
Description		
The use of ASTRID Control Flow Attestation services to attest (during run-time) code snippets running in a virtualized environment should be similar or better than the current existing ones (also benchmarked against the use of Trusted Execution Environments). For the static analysis that will be performed prior to the deployment of the microservices to the trusted repository, this doesn't affect the performance of the overall system as this can also be done offline. In this case, the attestation overhead can actually be defined as "Low" - we cannot have an exact numeric value as this will depend on the actual code size of the microservice to be attested and measured. On the other hands, for the run-time control flow attestation, we have two performance indicators of interest: the attestation overhead and the tracing overhead as these are interdependent with each other. Attestation will take place based on the information collected through the tracing. Again, this depends on the size of particular execution paths that we want to measure but due to the real-time nature of this functionality, it has to be as fast as possible. In short: For static analysis: • Attestation Overhead: Low - based on the code size of the microservice. For dynamic analysis: • Tracing Overhead: Low -in the order of seconds, ideally less of 10 seconds. • Attestation Overhead: Medium - it depends on the size of the execution path to be measured. Thus, we need to identify only the core properties of interest that need to be attested per microservice/application. Ideally this would be less of 5 seconds per specific execution path.		
Notes		

ID	Priority	Usage Scenario
R032-SD-PRF	Top	RAF
Title		
Average time to replace a single function		
Description		
When recent development and orchestration paradigms are used, compromised services can be partially or totally re-deployed in a matter of minutes instead of hours, so drastically increasing the resilience to attacks. According to current technology status, the average time to replace a single compromised function is expected to be below a few minutes.		
Notes		
The target value entails re-deployment of a compromised virtual function. For critical applications, where service continuity must be guaranteed, the process could be reduced to a few seconds, by pre-instantiation of backup functions (see R034-SD-PRF).		

ID	Priority	Usage Scenario
R033-SD-PRF	Top	All
Title		
Average time to re-deploy the service graph		
Description		
Replacement of a compromised function is the most effective remediation action when a single component is affected, but it might not be possible for stateful functions. In addition, multiple functions may be compromised before the attack is detected, so there are cases where re-deployment of the whole graph is the most convenient (if not the only possible) option. By using orchestration tools, the whole process is expected to take less than 5 minutes, for average applications made of no more than 6 software components.		
Notes		
While the replacement of a single function implicitly suggests to guarantee service continuity, the re-deployment of the service graph does not. Therefore, re-deployment of the service graph does not need to save and restore the system state; persistent data (e.g., those in databases) are any expected to be maintained across multiple re-initializations.		

ID	Priority	Usage Scenario
R034-SD-PRF	Top	RAF
Title		
Average time to switch between redundant deployments		
Description		
When recent development and orchestration paradigms are used, compromised services can be partially or totally re-deployed in a matter of minutes instead of hours, so drastically increasing the resilience to attacks. To further reduce the latency and increase service continuity for critical applications, redundant deployments may be used. In case of attacks or compromised functions, all workload is moved to the redundant installation. Redirection of the workload to the backup instance is expected to happen within a few dozens of seconds.		
Notes		
The usage of redundant deployments decrease service unavailability but increases the operational cost. Service continuity is also affected by the possibility to keep the state synchronized between the redundant deployments.		

ID	Priority	Usage Scenario
R035-SD-PRF	Top	DFW, MON
Title		
Average time to change the forwarding rules in a service chain		
Description		
Upon detection of a network attack, malicious or even suspicious traffic may be dropped or redirected to specialized appliances for analysis and investigation. Dropping packets is mainly useful for DoS attacks, while redirection towards honeypots is a common technique for stealth investigation. The quicker the reaction, the more effective the action (shorter unavailability for DoS, lower probability to alert the attacker in the other cases). Relying on software-defined networking and Network-as-a-Service paradigms should make possible to perform this process in less than 30 seconds.		
Notes		
The target value includes both the time to re-configure network processes (mainly software switches) and the time to recover from the topological changes; the latter may require the exchange of ICMP redirects or ARP announcements.		

ID	Priority	Usage Scenario
R036-SD-DSG	Medium	All
Title		
Lightweight operation and small impact on the service graph		
Description		
Lightweight operation and small impact on system performance is a common requirement for any security appliance, ranging from antiviruses to more complex intrusion detection systems. Differently from typical large applications, virtual services are often conceived just to provide better isolation among multiple users, by replicating the same graph into independent execution environments. If large security appliances have to be installed for each virtual function, the total amount of computing and storage resources might grow exponentially, making the small-services paradigm much less effective that current practice. In addition, monitoring and inspection operation must not slow down the execution of the main business logic; this is especially difficult to achieve for network functions that process packets at nearly line speed.		
Notes		
The lightweight operation requirement mostly applies to local processing. The logical separation between local inspection and centralized detection helps fulfil this requirement (see R001-CS-DSG/R003-CS-DSG). In this respect, it is worth pointing out that the same requirement is not binding for detection, since it is expected to lie outside the original graph and may be shared among multiple instances (R002-CS-DSG).		

ID	Priority	Usage Scenario
R037-SD-RLB	Top	All

Title

The attack surface does not increase

Description

According to the NIST vulnerabilities database, the number of security appliances' reported vulnerabilities per year is not decreasing. This is not reassuring, because the same software that should protect the service is itself a potential security threat. It is therefore necessary to reduce the impact of security software on the attack surface, by keeping the following aspects into account since the early design stage of security appliances:

a) reduce the amount of software that is potentially exposed to attacks;
b) perform deep software analysis before deployment;
c) run security software in controlled and safe environments;
d) create dedicated sandboxes for execution of the main detection logic.

Notes

It is objectively difficult to develop fully-safe software. It is also difficult to attest the trustworthiness of security appliances developed by third parties. Therefore, this design requirement is mostly oriented to find architectural solutions that help reduce the impact of (potentially) vulnerable software rather than pursuing safe implementation practices.

ID	Priority	Usage Scenario
R038-SW-DSG	Low	All

Title

Digital right management

Description

The ASTRID concept envisions a number of logical components that are suitable to create a modular architecture. This opens the possibility for multiple business opportunities, starting from the development of complementary software components to system integration. In particular, the development of monitoring/inspection programs and detection algorithms is expected to be one of the most relevant business cases for the Consortium partners and external stakeholders. While detection algorithms may consist of legacy applications with embedded validation of the license, monitoring and inspection programs are expected to follow simpler paradigms, which are unlikely to integrate such kind of validation. In this case, local agent may include some mechanisms to check licenses before execution, something conceptually similar to the verification made by DVD/CD players.

Notes

Ensuring license verification on software is a good incentive to foster software developers to create software for the ASTRID framework.

ID	Priority	Usage Scenario
R039-SW-DSG	Medium	SAW, SAF, RAF
Title		
Integration with existing logging facilities		
Description		
Software developers can write application logs to files, but they can also use specific frameworks that provide logging utilities. Usually, they enable to easily re-direct logs to files, network, or sockets without changing the application; this allows unified management of logs from different applications (this is often used by SIEM for collecting logs). The ASTRID framework maintains compatibility with existing tools (e.g., systlogd), to avoid software developers to change their applications.		
Notes		

ID	Priority	Usage Scenario
R040-LAW-FUN	Top	RAF, LAW
Title		
Certification and legal validity		
Description		
Data collected by monitoring and inspection processes must be usable as evidence in case of attacks or illegal activity. The certification process is responsible to ensure the verifiability of origin, time, and integrity of the data. It is also responsible to capture enough information to trace security attacks in a reliable manner and to interpret the data post-factum, and to conserve the data for enough time. Implementation should provide key features as trustworthiness, availability, integrity, resilience, resistance to attacks, and scalability, in order to prevent alteration or losses of data in case of attack.		
Notes		
The repository should be based on storage solutions specifically designed and implemented to comply with requirements for lawful applications.		

ID	Priority	Usage Scenario
R041-LAW-FUN	Low	LAW
Title		
Automatic encryption of interception channels		
Description		
Confidentiality of the user's sessions must be guaranteed also in case of lawful interception. Public security agents will operate remotely, by either redirecting the traffic or by using an interface provided by the SP.		
Notes		
Automatic set-up of encrypted communication channels can be easily integrated in the dynamic modification of the service topology (R018-SD-FUN).		