D5.1
Architectural Specification of Dynamic Enforcement of Trust-/Network-Aware Path Establishments

Project number:	101167904
Project acronym:	**CASTOR**
Project title:	Continuum of Trust: Increased Path Agility and Trustworthy Device and Service Provisioning
Project Start Date:	1st October, 2024
Duration:	36 months
Programme:	HORIZON-CL3-2023-CS-01
Deliverable Type:	Report
Reference Number:	HORIZON-CL3-2021-CS-01-101167904/ D5.1 / v1.0
Workpackage:	WP5
Due Date:	31st December, 2025
Actual Submission Date:	1st March, 2026
Responsible Organisation:	WINGS
Editor:	Aristi Galani, Sokratis Barmpounakis
Dissemination Level:	Public
Revision:	1.0
Abstract:	This deliverable details the CASTOR fundamentals towards optimizing network- and trust-aware path establishment across the compute continuum. More specifically, CASTOR's Management and Orchestration (MANO) layer is presented encompassing all internal components and building blocks for not only establishing trusted route configurations but also on how all appropriate guarantees are provided for their continuous monitoring and update in case of an (S)SLA violation. This includes mapping of the high-level user needs to the introduced CASTOR policies; appropriate telemetry mechanisms to extract/collect the network- & security- evidence (based on which node- and path-level trust assessment is conducted); and a (network-resources) optimization layer to enforce the CASTOR-calculated policies, on the underlying (routing) devices, based on a novel set of extensions optimizing the synergy between traditional Flex-Algo routing and the targeted use of Path Computation Elements (PCEs). Furthermore, CASTOR's Blockchain-based trust model for inter- and intra-domin path establishment is documented.
Keywords:	Policy Enforcement, Orchestration, Trusted Path Routing

Funded by EU's Horizon Europe programme under Grant Agreement number 101167904 (CASTOR). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

This work has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

Funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee.

Copyright Notice

Project Funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable	R*	
Dissemination Level		
PU	Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)	X
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/ EU-R	EU RESTRICTED under the Commission Decision No2015/ 444.	
Classified C-UE/ EU-C	EU CONFIDENTIAL under the Commission Decision No2015/ 444	
Classified S-UE/ EU-S	EU SECRET under the Commission Decision No2015/ 444	

* R: Document, report (excluding the periodic and final reports)
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
DATA: Data sets, microdata, etc.
DMP: Data management plan
ETHICS: Deliverables related to ethics issues
SECURITY: Deliverables related to security issues
OTHER: Software, technical diagram, algorithms, models, etc.

Editor

Aristi Galani, Sokratis Barmpounakis (WINGS)

Contributors (ordered according to beneficiary numbers)

Nikos Fotos, Sofianna Menesidou, Panagiotis Banavos, Thanassis Giannetsos (UBITECH)
Fabian Schwarz, Meni Orenbach (NVIDIA)
Alexandru Coleș, Ioan Constantin (ORO)
Vangelis Kosmatos, Panagiotis Pantazopoulos (ICCS)
Konstantinos Latanis (SUITE5)
Aristi Galani, Sokratis Barmpounakis (WINGS)
Pablo Martinez, Antonio Skarmeta (UMU)
Alexandros Fakis, Kostas Maliatsos (FERON)
Anuj Pathania, Andy Pimentel (UvA)
Jamie Pont, Budi Arief, Theo Dimitrakos (UKENT)
Christos Dalamagkas, Ioannis Boukas, Evangelos Syrmos (K3Y)

Disclaimer

Executive Summary

Contemporary networking architectures and infrastructure platforms provide mechanisms for exposing network capabilities and resources to applications and service providers; however, these mechanisms generally do not explicitly incorporate the assessment of trustworthiness associated with the underlying infrastructure components, network resources, and end-to-end communication paths. Consequently, trust-related attributes are typically not considered as key parameters in processes related to communication path establishment. Conversely, within the CASTOR architecture, the Trust-/Network-Aware Path Establishment constitutes a key capability, enabling the dynamic selection and provisioning of network paths that jointly satisfy performance-related constraints and trustworthiness requirements.

Within this scope, D5.1 focuses on the management and orchestration framework of the CASTOR architecture, initially introduced at a high level in D2.1. In particular, building upon the functional objectives, motivations, and requirements derived from the engineering stories across the different operational phases that characterize the network entities throughout their operational lifecycle, D5.1 defines the operational workflow among the entities forming the three architectural layers, namely the Orchestration, Facility and Monitoring & Analytics Layer. The Orchestration Layer acts as the brain for consuming trust insights and enabling the enforcement of trust-aware TE policies, while the Facility Layer provides to the other layers a continuously updated view of the network topology, augmented with trust metrics and associated contextual information, and the Monitoring & Analytics Layer transforms data obtained from the Facility Layer and network elements into monitoring intelligence that supports informed decision-making across the other layers.

The CASTOR core architectural components are defined independently of any specific deployment technology, allowing stakeholders to adopt orchestration solutions that best fit their operational, regulatory, and technological constraints. The CASTOR ecosystem encompasses both physical and virtual routers (vRouters) deployed in containerized environment. In this context, CASTOR leverages Kubernetes technology for the instantiation and lifecycle management of virtualized routing functions deployed over container-based infrastructures, as well as for the launching of the Trust Network Device Extensions (TNDE), the main component of CASTOR's on-device trusted computing base (D3.1).

Furthermore, extending the initial work on the Trust Assessment Framework presented in D4.1 and considering the requirements for the realization of the CASTOR DLT, a custom "Phala-like" blockchain architecture is introduced as an optimal architectural choice. The proposed architecture inherits key characteristics of the Phala execution model, including running heavy computational logic in off-chain workers running in isolated environments backed by hardware Root of Trust elements. D5.1 presents the high-level CASTOR DLT architecture and the envisioned smart contracts, which ensure trust, data integrity, and confidentiality across the CASTOR ecosystem, accompanied by high-level interaction flows for each smart contract.

Among the Traffic Engineering (TE) paradigms, Segment Routing-TE was selected as the preferred approach for validating the creation and enforcement of TE policies within the CASTOR framework. SR-TE was selected due to its native programmability and capability to enforce explicit, policy-driven paths. Moreover, its seamless integration with SDN/PCE-based controllers enables intent-driven policy enforcement. These characteristics position SR-TE as a future-proof mechanism for NaaS and related orchestration environments. In this context, D5.1 provides a concise overview of the mechanisms and the

associated protocols supporting source routing, and a running example illustrating trust-aware SR-TE path instantiation.

D5.1 establishes the architectural foundations of the proposed orchestration and management framework. Building upon this baseline, D5.2 and, in a later stage, D5.3 will provide in-depth analysis of the architecture, offering a detailed examination of architectural layers, the involved entities and their interactions, as well as the underlying implementation mechanisms supporting orchestration and management operations across both single-domain and multi-domain deployment environments.

Contents

1 Introduction **3**
1.1 Scope and Purpose . . . 4
1.2 Relation to other WPs and Deliverables . . . 5
1.3 Deliverable Structure . . . 5

2 Orchestrating Dynamic Trust Route Configurations on Next-Generation Networks **7**
2.1 Relevance with SCION Protocol Capabilities . . . 8
2.1.1 SCION as Today's Operating Technology . . . 10
2.1.2 Similarities and Points of Divergence to the CASTOR Vision . . . 10

3 Optimizing Network- & Trust-Aware Path Establishment through CASTOR Orchestration Layer **12**
3.1 Preparedness Phase . . . 12
3.2 Proactive Phase . . . 17
3.3 Reactive Phase . . . 21
3.4 Service Registration . . . 22

4 CASTOR Management and Orchestration Framework towards Flexible & (S)SLA Compliant Traffic Engineering **25**
4.1 Management and Co-enforcement of Trust Insights in CASTOR . . . 28
4.2 Traffic Engineering Policy Engine . . . 28
4.3 Traffic Engineering: Decision and Enforcement Points . . . 29
4.3.1 TE Policy Decision . . . 29
4.3.2 TE Policy Enforcement . . . 30
4.3.3 Network Service Orchestrator . . . 31
4.4 Facility Layer . . . 39
4.4.1 Topology Graph Composition Engine . . . 40
4.4.2 Data Broker . . . 40
4.5 Monitoring & Analytics Layer . . . 41

5 Blockchain-based Trust Model for Inter- and Intra-Domain Path Establishment **46**
5.1 Trustworthy Blockchain Oracles for Securing Zero-Touch Networks . . . 47

5.1.1 CASTOR Design Choice of PHALA Compatible Blockchain Model . . . 49
5.2 PHALA Blockchain Infrastructure . . . 49
5.2.1 Pre-migration PHALA Architecture . . . 50
5.2.2 Post-migration PHALA Architecture and Evolution . . . 50
5.2.3 CASTOR PHALA-alike Blockchain System Model . . . 50
5.3 CASTOR Blockchain-based Trust-Aware Path Establishment . . . 51
5.3.1 Flow 1: CASTOR Preparedness Phase and Trust Policy Secure Storage . . . 54
5.3.2 Flow 2: CASTOR Proactive Phase and Trust Policy acquisition . . . 54
5.3.3 Flow 3: SSLA Secure Storage . . . 55
5.3.4 Flow 4: TNDI Raw Traces Secure Storage . . . 55
5.3.5 Flow 5: Query TNDI Raw Traces . . . 55
5.3.6 Flow 6:TNDI Trustworthiness Claims Secure Storage (Pull Mode - from Local TAF) . 56
5.3.7 Flow 7: TNDI Trustworthiness Claims Secure Storage (Push Mode - from Global TAF) 56
5.3.8 Flow 8: Query Historical Trustworthiness Claims for Trust Evolution Management . 56
5.3.9 Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions . . . 57
5.4 High-level Definition of Smart Contracts . . . 57
5.5 Beynd Zero Trust: Controlled Data Exposure and Query-time Transformation by Harmonzing TCs . . . 59

6 Segment Routing (SR) Policy Construction and Enforcement 61
6.1 Exploring the Path Computation Element Protocol (PCEP) for Routing Configuration and Management . . . 61
6.1.1 Architectural Components . . . 61
6.1.2 Capabilities . . . 62
6.1.3 Integration with Segment Routing . . . 62
6.1.4 Flexible Algorithm . . . 63
6.1.5 Hierarchical PCE . . . 63
6.2 CASTOR Customized Path Selection & Enforcement: Optimizing Traffic Engineering with SRv6 Flex-Algo & PCE Extensions . . . 64

7 Configuring Network Service Topology with SR Flex-Algo Slicing: A Running Example 66
7.1 Routing Protocols and Source Routing in SR Slicing . . . 66
7.2 Updating the Routing table . . . 69
7.3 Instantiating an SR-MPLS TE policy . . . 70
7.4 CASTOR TE Policy Enforcement strategies . . . 77

8 Summary and Conclusions 79

References 82

List of Figures

1.1 Relation of D5.1 with other WPs and Deliverables 5

2.1 SCION high-level architecture 9

4.1 High-level architecture of CASTOR management and orchestration framework 25
4.2 Sequence of basic steps of the CASTOR framework operations 27
4.3 High-level description of the CASTOR normal operation (left) and SSLA violation (right), corresponding to the arrows of Fig 4.2 27
4.4 Implementation elements of a IOS XRd vRouter 33
4.5 Network Service Orchestrator 34
4.6 Kubernetes-based Container Architecture 35
4.7 Enclave-CC architecture and flows. 37
4.8 Architecture overview of the system (source: official website, reproduced as-is) 44

5.1 CASTOR DLT High-Level Architecture 52

6.1 PCE sequence diagram 65

7.1 Single-domain flat IGP networks 67
7.2 Cross-domain IGP networks 67
7.3 Example of overlay Flex-algo topologies on top of a base topology at the infrastructure layer. 69
7.4 How IGP and SR-TE Policy events reach the RP/RSP and trigger FIB updates that propagate to the linecard ASICS/NPUS 70

List of Tables

2.1 Comparative analysis of key SCION and CASTOR characteristics 11

5.1 Comparative analysis of Secure Oracle technologies 49
5.2 Smart Contracts and High-Level Interaction Flows 52
5.3 Smart Contracts and High-Level Interaction Flows 58

Versioning and Contribution History

Version	Date	Author	Notes
v0.1	20.11.2025	Aristi Galani, Sokratis Barmpounakis (SURREY), Nikos Fotos, Sofianna Menesidou (UBITECH)	Create Template and ToC listing the information to be conveyed in this deliverable
v0.2	04.12.2025	ALL	Listing and enumeration of the first version of the Engineering Stories capturing the requirements of the CASTOR Management and Orchestration Layer (Chapter 3)
v0.3	19.12.2025	Konstantinos Latanis (S5), Panagiotis Banavos, Sofianna Menesidou, Thanassis Giannetsos (UBITECH)	Convergence of the PHALA DLT to be adopted in CASTOR. Description of the initial architecture sketch (Chapter 5)
v0.1	07.01.2026	Nikos Fotos, Sofianna Menesidou, Panagiotis Banavos, Thanassis Giannetsos (UBITECH), Fabian Schwarz (NVIDIA), Alexandru Coleș (ORO), Vangelis Kosmatos, Panagiotis Pantazopoulos (ICCS), Aristi Galani, Sokratis Barmpounakis (WINGS), Pablo Martinez, Antonio Skarmeta (UMU), Alexandros Fakis, Kostas Maliatsos (FERON), Anuj Pathania (UvA), Jamie Pont, Budi Arief, Theo Dimitrakos (UKENT)	Architecting the high-level sketch of the CASTOR MANO Framework listing the functionalitis to be provided. First version focused on the functional boundaries between network controller and network service orchestrator (Chapter 4)
v0.4	09.01.2026	Alexandru Coleș, Ioan Constantin (ORO)	Description of a running example of the SRv6 Flex-Algo routing protocol towards configuring a network topology (Chapter 7)
v0.5	16.01.2026	Konstantinos Latanis (S5), Panagiotis Banavos, Sofianna Menesidou, Thanassis Giannetsos (UBITECH)	Final version of the CASTOR's PHALA-alike Blockchain Infrastructure integrating all updates needed for the secure interaction with the CASTOR TNDE established in each routing element (Chapter 5)
v0.6	30.01.2026	ALL	Final version of engineering stories capturing the functional requirements that need to be achieved per CASTOR MANO Framework component. Detailed discussions were held on how to optimize the synergy between traditional flex-algo routing mechanisms with CASTOR's PCE Extensions towards enforcing the CASTOR-calculated (and trust-aware) optimized policies (Chapter 3)

v0.7	13.02.2026	Nikos Fotos, Sofianna Menesidou, Panagiotis Banavos, Thanassis Giannetsos (UBITECH), Fabian Schwarz (NVIDIA), Alexandru Coleș (ORO), Vangelis Kosmatos, Panagiotis Pantazopoulos (ICCS), Aristi Galani, Sokratis Barmpounakis (WINGS), Pablo Martinez, Antonio Skarmeta (UMU), Alexandros Fakis, Kostas Maliatsos (FERON), Anuj Pathania (UvA), Jamie Pont, Budi Arief, Theo Dimitrakos (UKENT)	Final version of the CASTOR MANO Framework architecture capturing the detiled operational workflows of all internal building blocks. Additional time was needed to correctly capture the custom SR policy construction and enforcement operation based on the flex-algo and PCE synergy (Chapter 4)
v0.9	20.02.2026	Konstantinos Maliatsos (FERON), Thanassis Giannetsos (UBITECH)	Review of the entire deliverable prioritizing the MANO Architecture and DLT infrastructure
v0.95	25.02.2026	Aristi Galani, Sokratis Barmpounakis (SURREY), Nikos Fotos, Sofianna Menesidou (UBITECH)	Final refinements and polishing based on the provided comments
v1.0	01.03.2026	Daphne Galani (UBITEH)	Submission of the deliverable

Chapter 1

Introduction

The increasing reliance on digital services across critical infrastructures, cloud environments, and edge deployments has elevated the importance of trust in today's network services. Modern service provisioning no longer depends solely on performance indicators such as latency or bandwidth, but also on the healthiness and security posture of the underlying network infrastructure. In particular, network controllers and orchestration frameworks operating in the routing plane must be capable of evaluating and enforcing trust-aware routing decisions. Within CASTOR, two core aspects are highlighted: (a) the continuous assessment of the healthiness and trust posture of routers and network elements, and (b) the ability of this trust information to assist the Network Controller and Path Computation Element (PCE) in deploying policies that satisfy trust requirements expressed at the service level. Deliverable D5.1 specifies the CASTOR Orchestration Layer that operationalizes both aspects, enabling trust-informed decision-making in the routing and Traffic Engineering (TE) processes.

To address these needs—especially in environments transitioning toward virtualized router appliances (vRouters)—it is essential to securely manage the full lifecycle of routing elements. This includes secure onboarding, continuous attestation, configuration management, monitoring, and controlled decommissioning. Only routers that provide verifiable security assurances should be admitted into the operational topology. Consequently, there is a clear need for a Resource Orchestrator / Network Service Orchestrator capable of securely onboarding vRouters using the protocols and mechanisms defined in WP3 and WP4, ensuring that only trusted and properly attested network elements participate in service provisioning and routing decisions.

CASTOR addresses these challenges holistically. This deliverable presents the CASTOR Orchestration Layer, which integrates lifecycle management of trusted routing elements with trust-aware service fulfilment and assurance. Importantly, CASTOR does not impose a monolithic orchestration model. The Trust Assessment Framework (TAF) and the Optimization Engine are designed in a modular manner and can feed trust and optimization outputs to any compatible orchestration system. In this sense, CASTOR provides both a reference orchestration architecture and reusable trust and optimization components that can enhance existing orchestrators.

As service provisioning extends beyond single administrative domains, additional challenges arise in communicating and sharing trust semantics across domains. Cross-domain service provisioning requires mechanisms for exchanging summarized trust information without compromising confidentiality or operational sovereignty. In this deliverable, we provide an initial analysis and comparison of the CASTOR framework with existing path-aware networking frameworks, with particular emphasis on SCION. From this comparison, two major challenges are identified:

- **Lack of dependency scoping**: In cross-domain environments, trust relationships are context-dependent. A domain A may trust domain B under a specific scope, context, or trust requirement, but not under another. CASTOR addresses this through its Trust Assessment Framework and

subjective-logic-based reasoning, enabling different Actual Trust Levels (ATL) to be calculated per trust property or context.

- **Lack of sovereignty**: In many existing frameworks, trust is implicitly derived from external authorities or a set of trusted domains (e.g., Autonomous Systems). In contrast, CASTOR promotes sovereign trust assessment. Each domain can collect its own evidence, calculate its own trust opinions through the TAF, and avoid blind reliance on third-party assertions, thus avoiding any externally imposed trust relationships.

D5.1 serves as the basis for the implementation aspects of CASTOR Orchestration Layer that will be delivered in D5.2, while the final version of the CASTOR Orchestration Layer extended with cross-domain considerations and enhancements will be addressed in D5.3.

1.1 Scope and Purpose

Deliverable D5.1 focuses on the CASTOR Orchestration Layer and lays out the main challenges and requirements that shape its development. It defines how trust-aware lifecycle management of routers and trust-aware service fulfilment and assurance are integrated within a coherent orchestration framework.

Chapter 4 constitutes the core of this deliverable, presenting the high-level system model of the orchestration layer. It details how the lifecycle management of routers—particularly vRouters—is securely handled, from onboarding to continuous monitoring, and how service fulfilment and service assurance are realized. The chapter provides the complete pipeline covering both the router lifecycle and the service lifecycle, demonstrating how trust evaluation and network telemetry are consumed by orchestration decisions.

Chapter 5 introduces the Distributed Ledger Technology (DLT) layer and explains how auditable storage mechanisms support the secure management of SSLAs, trust policies, and trustworthiness evidence. It further describes how the DLT assists in the controlled exchange of trust information, both within and across domains.

With respect to traffic engineering, D5.1 provides a comprehensive overview of how CASTOR can support and enhance different Segment Routing (SR) and Traffic Engineering (TE) approaches. Chapter 6 discusses the Path Computation Element (PCE) and its role in translating optimization outputs into enforceable routing policies. Chapter 7 demonstrates how CASTOR accommodates different TE strategies and showcases how trust can be embedded into TE provisioning (e.g., through TE policies, Flex-Algo definitions, or PCE-driven paths). Rather than prescribing a single enforcement model, CASTOR offers a gamut of TE solutions and illustrates how trust assessment can systematically enhance these approaches.

Overall, D5.1 provides:

- The requirements that guide the functional specification of the orchestration artifacts.
- The high-level architectural description of the Orchestration Layer, acting as the “brain” that takes decisions for both the lifecycle of a vRouter and the lifecycle of a service.
- The capabilities of the CASTOR framework to incorporate trust in various and well established traffic engineering modalities.
- A structured flow that assists implementation, moving from engineering stories and requirements, to high-level architectural description, and finally to the application of CASTOR across different traffic engineering approaches.

In this sense, D5.1 establishes the architectural and conceptual foundation of the Orchestration Layer, setting up the scene for the implementation and validation activities that follow in subsequent deliverables.

1.2 Relation to other WPs and Deliverables

D5.1 is a core outcome of WP5 and closely interacts with several other CASTOR work packages. In particular, it builds upon the high-level CASTOR architecture and use-case analysis presented in D2.1, the CASTOR TCB Conceptual Architecture and the crypto-primitives for trust-aware orchestration defined in D3.1 enabling the collection of robust insights in WP5 and the types of trust propositions and optimization strategies to co-enforce network and trust requirements that need to be evaluated and investigated at the Orchestration Layer defined in D4.1.

In parallel, D5.1 provides to WP6 the orchestration framework based on which the use case partners can validate and evaluate their applications. In addition, D5.1 provides to D5.2 the requirements and the high-level system model descriptions based on which the 1st release of orchestration layer can be defined. Finally, D5.1 provides to D5.3 the initial set of high-level descriptions and identifies challenges towards cross domain service provisioning. Notably, as explained in the following chapter, these cross-domain challenges constitute areas of synergy with respect to other relevant initiatives such as the SCION project.

Figure 1.1 depicts the relation of D5.1 with other WPs and deliverables.

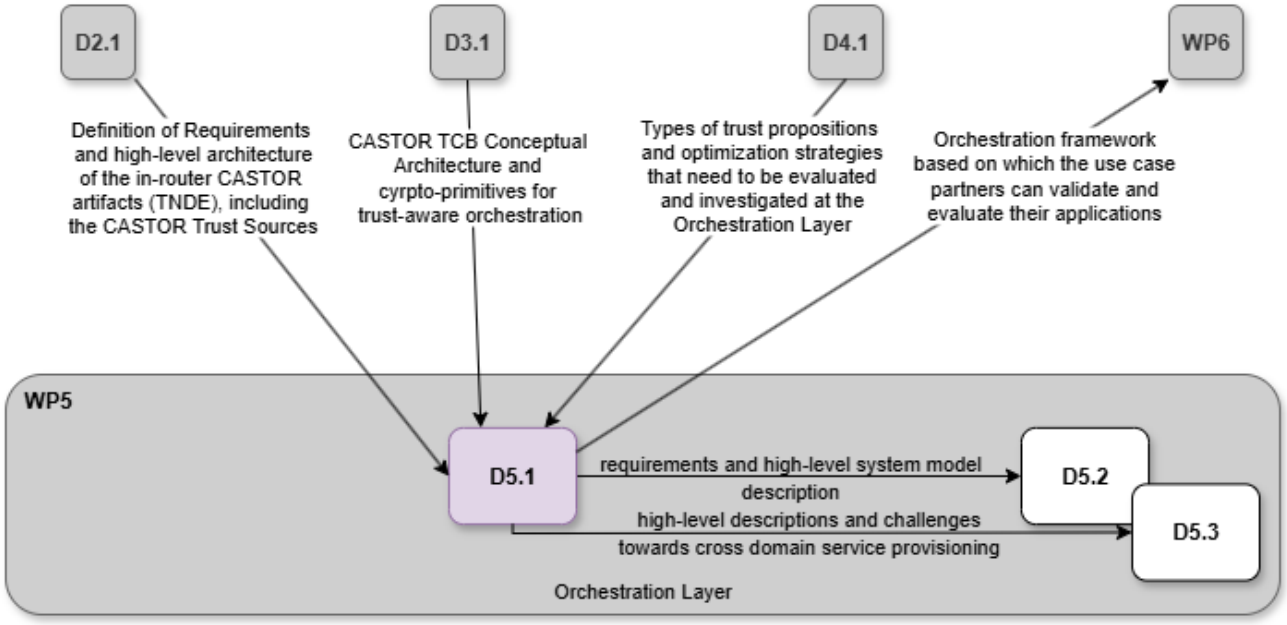

Figure 1.1: Relation of D5.1 with other WPs and Deliverables

1.3 Deliverable Structure

The remainder of this document is structured as follows.

Chapter 2 presents a detailed state-of-the-art analysis of key technologies relevant to WP5, with particular emphasis on path-aware networking and the SCION protocol.

Chapter 3 introduces the WP5 engineering stories, capturing the functional objectives, motivations, and requirements across different operational phases (e.g., preparedness, proactive and reactive).

Chapter 4 describes the CASTOR management and orchestration framework, detailing its layers, components, and operational workflows.

Chapter 5 focuses on the CASTOR Distributed Ledger Technology (DLT), including the state-of-the-art, the Phala-based solution, the operational workflows, the smart contracts and the controlled data

exposure.

Chapter 6 sheds light on the intrinsic functionalities of the Path Computation Element (PCE) and its position in the overarching CASTOR framework

Chapter 7 provides a running example that illustrates trust-aware SR-TE path instantiation and highlights the flexibility of the CASTOR framework to accommodate different TE strategies.

Chapter 8 concludes the deliverable with a summary of key contributions and directions of this deliverable.

Chapter 2

Orchestrating Dynamic Trust Route Configurations on Next-Generation Networks

The necessity of engraining trust directly into the routing plane has become a critical priority, driven by the proliferation of routing attacks, inter-domain hijacking incidents, and the increasing complexity of distributed infrastructures. Building this trust fundamentally requires a bottom-up approach. At the local level of establishing secure router-to-router communication, initiatives such as the IETF's Trusted Path Routing (TPR) and Network Attestation for Secure Routing (NASR) provide essential baseline frameworks. However, as discussed in D3.1 [7], CASTOR does not merely adopt these approaches; it envisions solving critical limitations towards runtime trust assurances, ensuring that trust is continuously monitored allowing the evaluation of dynamic trust properties beyond static ones.

As we level up from intra-domain routing to cross-domain, Autonomous System (AS)-to-AS inter-domain communication, the challenge of trust scales significantly. Addressing trust at this global scale requires broader architectural paradigms. This chapter aims to shed more light on SCION (Scalability, Control, and Isolation on Next-Generation Networks), which currently stands as the main initiative toward distributed trust provisioning in the Internet plane. Originating from over fifteen years of network security research at ETH Zurich [38], SCION's fundamental premise is the redesign of long-standing Internet routing principles to enable higher availability, stronger security guarantees, and explicit path control through the use of cryptographically verified Isolation Domains (ISDs). This naturally opens up a pivotal question: *what is SCION's exact relevance to the CASTOR vision, and how do these macro-level distributed trust paradigms align with the dynamic trust assurances CASTOR seeks to provide?*

SCION promotes path-aware networking by enabling endpoints to select network paths based on explicit trust and network criteria. This is achieved through the use of path-segment construction beacons (PCBs) and cryptographically secured path information, allowing endpoints to choose paths that meet specific policies, including trust and network performance requirements. This allows endpoints to choose paths that meet specific policies, including trust and network performance requirements. The architecture has progressed beyond theoretical design and has been deployed in production environments, while its specifications continue to evolve within the IETF Path Aware Networking Research Group. A key design principle of SCION is the introduction of a logically distinct routing plane that allows traffic to be forwarded over infrastructure meeting predefined trust properties, effectively functioning as a resilient and trust-aware alternative to the traditional Internet routing substrate.

From a technical standpoint, the need for such architectures arises from inherent limitations of the widely deployed Border Gateway Protocol (BGP). BGP operates under implicit trust assumptions: Autonomous Systems (ASes) can announce ownership of IP prefixes without strong built-in validation mechanisms. This model has repeatedly enabled BGP hijacking attacks [9], where traffic intended for legitimate destinations is maliciously redirected through unauthorized or compromised domains. These incidents highlight the structural vulnerability of today's inter-domain routing ecosystem.

To tackle the challenge, SCION introduced the concept of ”Isolation Domains” (ISDs) separating ASes into groups of independent routing sub-planes, called trust domains. These ISDs act as operational clusters of Autonomous Systems (ASes) operating under a shared trust contract (i.e., trust policy shaping the Trust Root Configuration of an ISD), interconnected into manageable zones to form complete and secure routes. The task of path selection is then shifted from a set of network routers to the end-hosts, allowing SCION to achieve increased resilience to link failures but most notably, protection against invalid or unauthorized route advertisements. Nevertheless, the establishment of the novel SCION control plane infrastructure introduces several challenges for the adopting network domains.

First, a fundamental consideration in the context of SCION relates to the fact that trust evaluations are assumed to be securely provided by the core-ASes within an ISD, allowing them to agree on the set of ASes and the Trust Policy characteristics that comprise the ISD-wide Trust Root Configuration (TRC). However, from the perspective of peer ASes within the same ISD, this process provides no verifiable means to continuously measure the trustworthiness of other participants. This opacity could imply either that the initial ISD enrolment process is not rigorous, or a compromised AS could violate its operational guarantees at runtime, regardless of the capabilities it proved during admission. Similarly, even after a Trust Root Configuration (TRC) is established within an ISD, non-core ASes lack control over future updates to the Trust Policy or the roster of accepted ASes. This lack of sovereignty forces these non-core ASes into a purely reactive posture, requiring them to blindly accept and adapt to critical trust revisions. The aforementioned aspects introduce a level of lack of sovereignty in the trust dependencies that are maintained by an arbitrary AS - especially considering its participation in multiple ISDs. To this end, CASTOR capabilities to securely measure, share and assess trustworthiness evidence provide a unique capability for filling this trust characterization gap of the established AS-to-AS trust relationships.

Second, the flexibility in the expression of TRCs that can accommodate arbitrary trust policies introduces an non-trivial challenge for managing end-to-end service needs. For instance, this would refer to the case where two ASes are reachable under a given TRC, say TRC_1, which focuses on high integrity assurances, whereas they cannot establish a connection under TRC, say TRC_2, that is related to availability. In this context, the ingress AS would have to clearly define the per-traffic policies to steer the appropriate services through the TRC_1 compliant end-to-end path. This example clearly illustrates the need for a transparent mechanism for managing the trust dependency scoping. Through CASTOR’s Trust Assessment Framework, trust evaluations are realized in a context-dependent manner enabling the inter-AS trust characterization under different trust properties and scope.

Consequently, these challenges outline a clear path for conceptual synergy between the CASTOR and SCION projects in the context of inter-domain traffic engineering. While the first release of the CASTOR project focuses on evaluating trust within single-AS environments - thus enabling the elevation of node-level trust insights to the intra-AS path and domain level — our progression to the second release envisions the evaluation of the overarching framework in broader inter-domain scenarios. Ultimately, evaluating CASTOR in these environments aims to demonstrate how it can effectively address sovereignty and dependency-scoping gaps within cross-domain trust relationships, especially in scenarios with potentially different trust semantics.

2.1 Relevance with SCION Protocol Capabilities

The SCION architecture enables a hierarchical routing structure that distinguishes between intra-domain (within an ISD) and inter-domain routing (between ISDs) and allows for path-aware packet-steering. An ISD i.e., Isolation domain (see Figure 2.1) is a collection of Autonomous Systems (AS) that are grouped under a common trust notion.

The process begins with the discovery of paths, where the core ASes (marked with bold circles in Fig. 2.1) of an Isolation Domain periodically initiate a Path Beaconing process. These Path Construction Beacons

(PCBs) propagate through the network, accommodating cryptographic and metadata information from each participating AS. Each hop in the network results in the addition of a 12-byte component within the packet header, the ”Hop Field”, protected by a Message Authentication Code (MAC) generated by a secret key only known to that specific AS. Thus, generated paths cannot be forged or maliciously altered. When beacons reach the leaf ASes having traversed the domain, the discovered path segments are stored in a dedicated Path Server (the Control Service in Fig. 2.1 realizes the functionality of the Path Server).

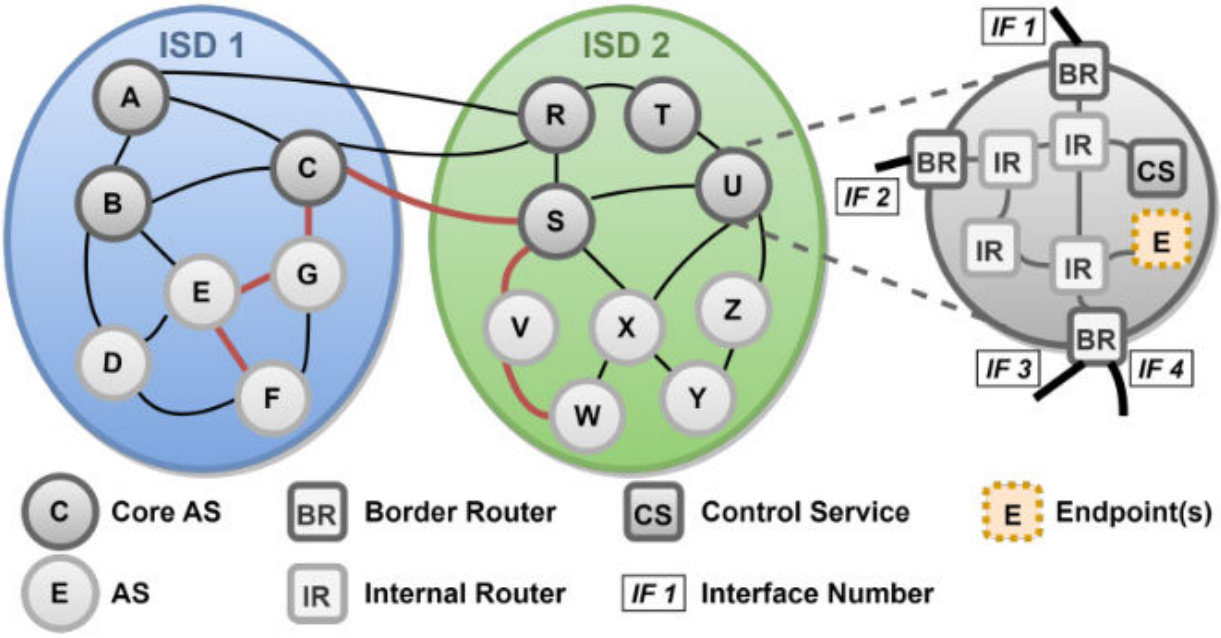

Figure 2.1: SCION high-level architecture

When a source host needs to establish a connection (to a desired destination, perhaps in another ISD), it queries its local Path Server to retrieve the available segments. The host (Ingress Router) selects a path based on performance, cost, or sovereignty (e.g., ”Avoid ISD 2”). Then, puts together the set of segments (Up-segment, Core-segment, and Down-segment [1]) creating a complete and authenticated end-to-end path which is encapsulated directly into the SCION Packet Header.

When the packet reaches the Border Router (BR) (see BR in the zoom-in rightmost part of Fig. 2.1), the BR validates the segment for its own AS and sends it across the ”Inter-ISD” link. In contrast to BGP, where routers look up destinations in a massive Routing Information Base (RIB), a SCION router (along the path) performs a simple parsing of the packet header to verify the MAC of its own Hop Field and thus, ensure the path is authorized. Then, it forwards the packet to the specified egress interface. The process, when considered scaled to the whole path, achieves a drastic reduction of the state information required on the involved routers, since such information is carried by the packet itself.

Further research of the SCION team has introduced advanced extensions to the baseline protocol to address security and resource management challenges.

The DRKey (Dynamically Reachable Key) derives (at line speed) symmetric session keys from a local master key. This enables routers to perform packet authentication avoiding the overhead induced by traditional Public Key Infrastructure (PKI) processes, performed for every single packet.

[1] The Up-segment connects a non-core AS (like a university network) to a Core AS within its own Isolation Domain (ISD). It gets packets out of a local network and forwards them into the ”backbone” of the SCION network. The Core-segment connects two Core ASes while the Down-segment connects a Core AS to the specific non-core AS destination; essentially, the latter handles the ”last mile” of the inter-domain routing delivering the packet from the backbone to the target network.

The COLIBRI (COntroLling Inter-domain Bandwidth Resource Allocation) extension addresses the ”noisy neighbor” problem [2] in shared networks by allowing ASes to negotiate and enforce inter-domain quality-of-service (QoS) guarantees. An ISP can therefore, reserve a specific ”virtual circuit” across multiple ISDs, ensuring that critical network traffic (like the one of finance applications) receives dedicated bandwidth avoiding any public internet congestion.

2.1.1 SCION as Today's Operating Technology

The SCION technology has considerably matured, moving away from a mere laboratory experiment and shifted to a production-grade operating technology supporting highly sensitive networked environments. Its most significant real-world deployment is the Secure Swiss Finance Network (SSFN), a dedicated infrastructure developed in collaboration with SIX Group and the Swiss National Bank [24]. The SSFN connects hundreds of financial institutions over a SCION-based network topology creating a private, high-availability structure that is physically and logically isolated from the public web vulnerabilities and secured against any BGP hijacking attempt.

Furthermore, the SCION technology has been commercialized relying on Anapaya Systems hardware, which provides the gateways and edge routers used by enterprises seeking to adopt a path-selection capability. Current implementations have also expanded to the Health Info Net (HIN), addressing security challenges of medical data of Swiss citizens. At the same time, SCION is being evaluated by European telecommunications providers as a ”Sovereign Cloud” transport layer. By offering sub-100ms path failover and guaranteed latency (see the COLIBRI functionality in the previous paragraph), SCION is combining secure networking with increased performance to meet the needs of future Internet.

2.1.2 Similarities and Points of Divergence to the CASTOR Vision

CASTOR, being a certain lifetime EU project aims to research a broad set of challenges related to the ‘trusted path routing’ concept. As such, it exhibits considerable overlap with the SCION protocol even if the latter has been a mature innovation outcome of more than a decade of work.

Both initiatives advocate for a networking technology of transparency and path control, where the user (or the service provider) retains full visibility and selection capability of the employed (routing) path. On the other hand, they diverge in a number of technical points; SCION is essentially focusing on a redesign of the Network Layer (Layer 3) while CASTOR operates combining an innovative trust assessment approach to drive the trusted network path selection, coupled with Management and Orchestration (MANO) primitives.

Further to that, CASTOR invests more effort in the exploration of relevant innovation threads rather than aiming towards a highly mature product; beyond the SCION COLIBRI resource allocation, CASTOR looks into the efficient key management, the routers' state-machine modelling, the network elements attestation techniques, the utilization of a blockchain layer and the cost-effective solution to the path-optimization problem. Finally, both CASTOR and SCION try to solve the dependency scoping and the sovereign operation challenges, but in a different manner. SCION, handles it during the setup of a network's ”back-up” (secondary) network plane and CASTOR during runtime operation of the primary network plane.

A synopsis of the SCION-CASTOR comparison along a basic set of parameters is depicted in the following table.

[2] The ”Noisy Neighbor” problem emerges when a network user or application monopolizes shared resources, causing performance degradation

Parameter	SCION	CASTOR
Primary Goal	Secure path-aware routing innovation	Trust and optimization driven path routing
Architectural Layer	Layer 3	Layer 3 and orchestration layer
Trust model	PKI-based	Subjective logic based
Routing mechanism	Source routing principle	Source routing principle (starting from intents)
Path Optimization	Enforcing QoS guarantees	Solving online complex optimization
Maturity	Commercial system	Proof of concept with strong standardization footprint
Path Computation	End-host selects from beaconed segments	PCE computes SR-TE policies from intents
Dependency scoping and sovereign operation challenges	during network setup	during runtime operation

Table 2.1: Comparative analysis of key SCION and CASTOR characteristics

Chapter 3

Optimizing Network- & Trust-Aware Path Establishment through CASTOR Orchestration Layer

This chapter specifies important required behaviours of the CASTOR Orchestration Layer in the form of engineering stories. Engineering stories are structured descriptions of system interactions that capture well-defined usage scenarios from the perspective of specific roles or groups (e.g., Domain Operator, Administrator, Network Service Orchestrator etc.). Each engineering story describes the associated requirements that the orchestration-related components or actors must satisfy in order to enable trusted path establishment within the CASTOR architecture.

The purpose of this chapter is to provide a high-level yet precise specification of the Orchestration Layer requirements that drive the CASTOR's architectural and implementation choices. Thus, in this deliverable, we capture the core orchestration capabilities, while all engineering stories covering the rest of the layers of the CASTOR architecture are presented in D3.1 [7] and D4.1 [5].

The engineering stories are organized according to the operational phases defined in D2.1 [4]. Starting from the general assumptions, the chapter delves into the engineering stories that are relevant for the proactive phase, focusing primarily on the secure onboarding of a new network element. Subsequently, the chapter addresses engineering stories relevant to the preparedness and reactive phases, including service registration. These scenarios collectively define how the Orchestration Layer acts as the decision-making "brain" of CASTOR, ensuring that trust evaluation, resource management, and traffic engineering policies are coherently enforced throughout a service lifecycle. Through this structured set of engineering stories, Chapter 3 establishes the functional blueprint that guides the realization of a trust-aware Orchestration Layer.

3.1 Preparedness Phase

Engineering Story-I

As a Domain Operator, I want to be able to specify varying levels of network and trust offerings, so that I can provision a service catalogue that accommodates trust-aware Traffic Engineering

Objective The Domain Operator wants to have the ability to define and deploy tiered network service capabilities/profiles with varying levels of security, reliability, and performance, allowing the creation and provision of a service catalogue, where the provided options correspond to trust-aware Traffic Engineering policies deployment in the underlying infrastructure, ensuring demanded security posture, and SLA/SSLA

compliance.

Motivation Current networking architectures rely on attaining performance (network) goals. There are initiatives that aim to reshape Internet (e.g. SCION), trying to incorporate trust as a negotiating term towards the establishment of end to end paths. However, they do not provide the mechanisms to systematically quantify and model trust. Recent advances to IETF Trusted Path Routing, incorporate trust as a prerequisite towards shaping trusted topologies, but do not elaborate on the runtime evaluation of trust relationships.

The collection of trust insights at node, link, path, and domain level throughout the lifespan of a domain is a fundamental requirement for enabling trust-aware service provisioning, assurance, and service orchestration. Network domains are not static entities; they evolve over time due to changes in topology, software upgrades, policy updates, workload placement decisions, and interactions with external domains. As a result, the trustworthiness of individual components may vary during operation. Capturing trust-related evidence only at isolated points in time is therefore insufficient to accurately reflect the evolving security posture of the domain. At the node level, trust insights provide visibility into the integrity and reliability of individual network elements, including their execution environment, configuration state, and operational behaviour, while at the link level, trust insights allow the assessment of the reliability and policy compliance of individual connections between nodes. Since network services typically traverse complex paths, the overall trustworthiness of a service cannot be inferred from isolated elements alone. At the path level, trust insights support the evaluation of end-to-end trust properties across multiple nodes and links. Continuous path-level trust assessment enables informed routing, traffic engineering, and service placement decisions. Finally, at the domain level, aggregated trust insights provide a holistic view of the domain's security posture and operational reliability. **All these trust insights can be turned into attainable objectives that can be expressed as part of an extended SSLA.** This provides the ability to Service Providers to select network solutions that are suitable for their demands, based on both performance and trust criteria, as also captured in the Use Cases that were presented in D2.1 [4] (e.g., UC1's radar application data require high integrity in the transmission challenge to the Airport zone Area Operator, while UC3's notification messages require high availability and redundancy in the communication channel towards the Traffic Operator Backend services).

While the CASTOR framework is capable of accommodating heterogeneous and composable network- and trust-requirements, exposing them as well-defined tiers enables deterministic resource modelling, and demonstrable SSLA enforcement. So, in the context of CASTOR, the Domain Operator will be able to offer to Service Providers a service catalogue that exposes abstracted network and trust capabilities. These capabilities will encompass guaranteed performance and QoS/QoE KPIs (e.g. maximum end-to-end latency, jitter, throughput, application/service response time, throughput stability), as well as trust-related capabilities relating to integrity, confidentiality and availability, reflecting that application and service workloads will be routed through trustworthy network nodes and paths. The availability of such a catalogue, forming CASTOR's Path Profile Catalogue for a specific domain, will constitute the basis of the negotiation process between the Service Provider and the Domain Orchestrator for the definition of the Service Level Objectives of the SLA/SSLA to be established between the involved parties.

Requirements To achieve this objective:

- the **Path Profile Catalogue Engine** should map the aforementioned path profiles to predefined quantitative network and trust technical terms, so that they clearly represent the alternative offerings available through the service catalogue,
- the **Network Service Orchestrator** should monitor the availability of the required compute and network capabilities (e.g.vRouter onboarding) to ensure compliance with the provided SLA/SSLA/path profiles. Furthermore, the Network Service Orchestrator should receive telemetry data (metrics, events) and alarms, such as, but not limited to, vRouter Resource & Compute metrics (correspond-

ing to pod/container metrics), Pod/container crash, failed VNF onboarding, attestation failure, performance metrics, KPI deviations, contributing to Topology Graph. Resource Manager of Network Service Orchestrator should process this data and, if needed, invoke Network Service Orchestrator to trigger the appropriate adaptive actions in order to ensure that the SLA/SSLA requirements are fulfilled,

- the **Optimization Engine** uses a fresh snapshot of the Topology Graph information (as explained in D4.1 [5]), in order to determine the different paths that correspond to different network and trust capabilities (new recommendations).

Engineering Story-II

As the Network Service Orchestrator, I want to be able to interact with the routing plane and shape the “appropriate” (routing) paths through a physical topology in line with the agreed SSLA.

Objective To facilitate the selection of a path that optimizes both network metrics and trust values reflected in the corresponding SSLA, the coordination of the process will be driven by an application-level orchestrator software. Relevant tasks, beyond managing the intents translation which lies out of the CASTOR scope, include the lifecycle management of CASTOR applications (deployed across the network devices), the management and control of the path establishment process over the network topology as well as the (telemetry-based) performance monitoring. What is then the objective, is to develop the capability of the orchestration software to establish “hooks” to the forwarding plane and control the relevant decisions and functionality.

Motivation Our main driver remains the CASTOR concept which suggests the capability of shaping trusted and resource-optimized paths over (topologies making up) the network continuum. Examples of mechanisms for shaping routing paths include the enforcement of SR policies through the Network Controller, the provisioning of resources to manage the lifecycle of the TNDE, the refinement of Trust Policies when needed (see Engineering Story-IV and D4.1 [5]), and the extraction of network and trust-related data (see Engineering Story-IV). Further, the considered engineering story which points to the shaping of the routing paths accounts for the (ever-increasing) virtualized nature of network resources. While elements for computing paths are relying on rigorous optimization techniques, they typically lack the broader context of service lifecycles and virtual resource availability; the latter can become even more demanding when routing decisions must take into account trust calculations that potentially call for increased (virtualised) resources.

Requirements This story requires the availability of all network and trust information (see OSS.R.3 in D2.1) as well as the operation of the orchestration software that can handle policies (see OSS.R.2). More importantly, the story requires the introduction of the ”hooks” to the forwarding plane, the design and operation of the path calculation element and primarily the network controller that acts as a bridge from the orchestrator software to the router-level topology. Towards that end, the controller utilizes both southbound interfaces (such as OpenFlow or NETCONF) that instruct switches and routers on the way to handle the involved traffic as well as northbound ones (usually RESTful). The latter is used to receive orchestration requests abstracting the underlying hardware details.

Engineering Story-III

As the Orchestration Layer or an Administrator, I want to store (S)SLAs and global/local Trust policies per path profile in a secure and reliable manner in the CASTOR DLT through the Secure Oracle, so that they are used for the (re-)establishment of trustworthiness in the CASTOR framework.

Objective The SSLA Monitor or the Central Control Service shall be able to store (S)SLAs events and (S)SLAs IDs in the CASTOR DLT. In addition, the Risk Assessment or the CASTOR Administrator shall

be able to store Trust Policies, as soon as their data veracity is verified by the Computation Worker, a dedicated enclave to support this operation (see Chapter 5 for more details). Both (S)SLA and Trust Policies facilitate the (re-)establishment of trustworthiness in the network elements of the CASTOR Framework.

Motivation During the service registration phase within the CASTOR Orchestration Layer, the (S)SLAs of a router, established as service intents by the Application Service Provider and converted into secure requirements, should be recorded in the CASTOR DLT (as described also in Section 5.3.3). Keeping these parameters on-chain supports both real-time auditability, as historical agreements are validated with transparency, and high level of security and veracity when managing the (S)SLAs. Apart from the initial recording, the Orchestration Layer performs continuous monitoring of these (S)SLA parameters. If a violation is detected, a corresponding event is issued by the SSLA Monitor. This event is stored on-chain to generate a traceable audit log and initiate remedial actions like path reconfiguration, policy adjustment, or the renegotiation of agreements. Moreover, storing (S)SLAs on-chain allows authorized external parties to sign up for violation alerts. Depending on the needs of the integration, these alerts are channelled through either the Trust Exposure API (see Section 5.3.9). Access to these notifications is managed by the multi-layered CASTOR strategy (see Engineering Story-VII). This entire workflow establishes the groundwork for reactive, trust-centric network management and real-time compliance oversight, ensuring the (S)SLA guarantees are consistently verifiable and enforceable.

Meanwhile, the operational trust requirements of compute continuum network components are governed by Trust Policies. These policies outline the behavioural expectations, guarantees, and compliance benchmarks that must undergo evaluation during runtime. They specify which trust propositions the framework must assess and which raw traces need to be collected. Integrating into the broader CASTOR trust model, these policies include Required Trust Levels (RTLs) paired with specific path profiles (e.g., high availability, high integrity), which function as the reference criteria for deciding if nodes and links qualify for a specific routing path. This mechanism enables CASTOR to efficiently and securely ingest runtime trust evidence from edge routing components and verify it prior to DLT anchoring, maintaining integrity, confidentiality, and low operational overhead while providing continuous runtime trust evaluation. Thus, Trust Policies are also stored to the CASTOR DLT (see Section 5.3.1).

Requirements

Auditable sharing and a high level of security and veracity of (S)SLAs and Trust Policies is of paramount importance. Such a requirement can be acheieved by storing (S)SLAs and Trust Policies to CASTOR DLT. The interaction between the CASTOR layers and the DLT should follow a structured sequence of authorization, validation, and secure communication (see also Section 5.3.1 and Section 5.3.3). The process begins when the SSLA Monitor or the Central Control Service initiates a request to Computation Worker and the provided TNDI-SP Validation, which serves as a trusted mediator with the DLT. If validation is successful, the (S)SLA is recorded within the CASTOR DLT, where it remains available for updates throughout the service lifecycle. In the same sense, during the preparedness phase, the Risk Assessment or CASTOR Administrator alternatively submits initial Trust Policies, derived from risk analysis and Required Trust Levels (RTL), to the DLT through the Computation Worker or Whitelisting access control respectively.

Engineering Story-IV

As the Domain Operator, I want to have access to granular events related to the trust evaluation of my infrastructure layer, so that I can evaluate and (post)analyze any incident that may occur and apply the respective mitigation actions.

Objective The Domain Operator wants to have access to real-time telemetry data enriched with trust-related information. This will enable Domain Operator to perform post-incident analysis in order to support proper mitigation and remediation actions, to ensure the establishment of a Trusted Network Topology.

Motivation Traditional network telemetry — covering performance, availability, and resource utilization metrics — is no longer sufficient to support advanced, automated, and trust-aware operational decisions. Trust-related telemetry enables the Domain Operator to obtain a comprehensive and real-time view of the security posture of the underlying infrastructure components and network paths (as highlighted by the analysis of UC1.US.1c "Trust Degradation Alert" in D2.1 [4]). The enriched information will include attestation evidence, trust scores, confidence levels, integrity status of network and compute nodes, provenance of measurements, and trust-related events or anomalies. Without access to such fine-grained trust metrics, indicators and events, the Domain Operator lacks visibility into trust degradation, ability to correlate trust incidents with network behaviour and specific network components, and sufficient evidence to justify and apply mitigation actions. Provisioning of telemetry data with trust-related information enhances situational awareness, operational resilience and trust assurance, while supporting post-incident forensics, auditability and accountability. In CASTOR, access to trust events encompasses all layers, ranging from the infrastructure layer, via the Orchestration Layer, to the service provider. The Network Service Orchestrator should be informed by receiving either per-TNDI trust reports via the TNDI-SP data channels or trust assessment evaluation results from Global Trust Assessment Framework, in order to proceed to the proper actions to resolve the problematic condition.

The trust reports (Local TAF assessments) that result from the trust evaluations and refer to low-level trust propositions on the state of a router (e.g., state changes, deviations and anomalies within the underlying computing host, anomalous route announcements) are taking place within the TNDE of vRouters. These reports are exposed through defined telemetry framework and APIs (e.g. Prometheus), by extended their mechanisms, and are jointly evaluated together with network metrics in order to guide the NSO towards required reconfigurations of existing worker nodes or the addition of new ones. The Global TAF assessments (e.g., trust reports), concerning high level trust propositions are shared to the Data Broker, and are used to populate the Topology Graph, which offers rich trust insights of the entire topology to the Domain Operator (as also stated in Engineering Story-III). So, provided telemetry data extended with the aforementioned trust-related information enable the Domain Operator to address issues that jointly consider performance, reliability, and trustworthiness across the infrastructure continuum.

Requirements To accomplish this objective, CASTOR telemetry entity should support the continuous and near real-time reception of telemetry data from multiple heterogeneous telemetry sources, including network devices, virtualized and cloud-native components. Furthermore, telemetry data should be received through well-defined and standardized interfaces and data models, in order to enable the seamless deployment and integration of monitoring and orchestration functions.

Regarding the acquisition of trust-related data, the first step involves the routers being securely on boarded in the topology. Then, the mechanisms that form the Trusted Computing Base (TCB) of each network element have to be deployed and operate properly, enabling the collection of evidence related to state changes, anomalies, and quantifiable deviation across the underlying infrastructure, which may indicate a successful breach of a system or the identification of malicious activity, as well as, anomalous behaviour relative to the expected operation of network protocols and traffic. Within this context, it is necessary to define a detailed list of the information and events to be monitored, along with the mapping of these events to specific threat categories. Furthermore, for evidence collection, the deployment of suitable Trace Units in the infrastructure layer is required. These requirements are addressed within the scope of deliverable D3.1 [7].

The fulfillment of all requirements ensures that CASTOR orchestrator deploys and manages services, remaining compliant with their SLA/SSLA-defined objectives, while also contributing to the implementation of the OSS.R.3 Functional Requirement as documented in D2.1 [4] (i.e., Accurate and fresh synchronization of the network topology attributes and trust assurance reports).

3.2 Proactive Phase

Engineering Story-V

As a Network Service Orchestrator, I want to be able to verify trustworthiness evidence from the topology, so that I can ensure the secure lifecycle management of the network elements and the deployed network services

Objective The Network Service Orchestrator needs to provide the verification mechanisms in order to securely on-board new network elements in the topology and manage them throughout their operational lifecycle. In addition, it shall expose the necessary service assurance mechanisms in an efficient manner.

Motivation Advancements in establishing trust within a network topology go beyond the mere transmission of trustworthiness evidence vertically, i.e., from the (Prover) network element to its control service at the orchestration layer. Specifically, in alignment with the IETF Trusted Path Routing paradigm [3], CASTOR adopts - and envisions to extend - the concepts of exchange of trust-related information across the forwarding plane. In IETF's TPR system model, this refers to the exchange of trust-related information between two neighbouring network elements as part of link layer authentication. A key element of this interaction is the Stamped Passport, an Attestation Results (AR)-augmented piece of evidence (as per the building blocks defined in IETF's Attestation Results for Secure Interactions [35]). The Stamped Passports conceal important information for the decision making of an adjacent network element - acting as the Relying Party: (i) fresh evidence about the trustworthiness of the attester network element, and (ii) attestation results from the verifier entity at the Orchestration Layer after the secure on-boarding process.

Overall, within the context of CASTOR, the **Network Service Orchestrator plays a pivotal role in enrolling network elements into the topology** and enabling them to establish a Trusted Topology in accordance with the IETF Trusted Path Routing (TPR) framework. To support this process, CASTOR defines a secure onboarding protocol in D3.1 [7], which allows the Network Service Orchestrator—via its Network Function Manager component—to attest the TNDE platform, with a primary focus on the components comprising the router's Trusted Computing Base (TCB). As a core step of the onboarding phase, the Network Function Manager provisions the attester network element with the necessary cryptographic material, enabling it to exchange Stamped Passports with candidate adjacent routers. These Stamped Passports convey evidence of the verifiable launch of the CASTOR TNDE, for example through attestation quotes demonstrating the correctness of the deployed TNDE artifacts.

Following the onboarding protocol, the Network Service Orchestrator can configure the TNDE artifacts of the attester router, enabling it to introspect critical network functions of the vRouter software stack through the available Trace Units and to collect runtime traces. These traces can subsequently be used for the continuous evaluation of the vRouter's trustworthiness throughout its operation. **As a result, CASTOR enables the Network Service Orchestrator to extend the Stamped Passport mechanism by incorporating evidence related to the runtime trustworthiness of network elements**, as detailed in D3.1 [7].

Finally, the verification capabilities of the Network Service Orchestrator extend beyond the assessment of the trustworthiness of individual (internal) router capabilities. As described in D3.1 [7], the CASTOR composite attestation protocol aims to provide aggregated attestation evidence that enables the **Network Service Orchestrator to validate the correctness of an enforced routing path**. This includes verifying that each element along the path is in a valid state, as demonstrated by its ability to produce partial attestation evidence, as well as ensuring the correct ordering of routers within the provisioned path. By acting as the lead verifier in this composite attestation protocol, the Network Service Orchestrator derives meaningful trust assurances about the end-to-end path, which are subsequently consumed by the Global TAF for its evidence-based trust quantification process, as outlined in D4.1 [5].

Requirements It is worth noting that, within such attestation schemes, the Network Service Orches-

trator may not have full access to the reference values required to verify the collected evidence. In the context of CASTOR, router vendors may be unwilling to disclose information describing the "correct" internal state of a router to verifiers operated by (external) Network Service Orchestrators. As a result, the verification process at the Network Service Orchestrator may need to rely on external verifiers operated by the corresponding router vendors/manufacturers in order to reach an attestation decision regarding the trustworthiness of a router element. Consequently, CASTOR designs its attestation schemes with explicit consideration of multi-verifier environments, in which verification decisions may be derived under reduced trust assumptions on individual verifiers.

Engineering Story-VI

As a Service Provider, I want to receive trust capabilities of the domain in an abstracted manner through the Trust Exposure API, so that I can evaluate the trustworthiness in cross-domain scenarios.

Objective A Service Provider (as well as external authorized entities) shall be able to receive abstracted trust capabilities from the Trust Exposure API offered by the Security Context Broker towards monitoring the trustworthiness of the network infrastructure in cross-domain scenarios without revealing sensitive internal data of the network.

Motivation The CASTOR framework allows external entities to evaluate network trustworthiness through regulated access to trust-centric data, ensuring that internal sensitive data remain protected (see also Engineering Story-X). Rather than relying on traditional static abstraction, where simplified data is pre-recorded on the blockchain, CASTOR implements a dynamic approach where information is filtered at the moment of request (see also Section 5.5). This allows the Trust Exposure Layer to tailor the visibility of data, revealing only the essential details dictated by the specific credentials and authorization level of the requesting external entity (see Section 5.3.9). These trust-based revelations might encompass summarized benchmarks, the outcomes of compliance audits, or alerts regarding (S)SLA breaches. Such insights are accessible only by verified entities, including certification bodies, cross-domain orchestrators, and service providers.

Requirements Built upon the foundational CASTOR DLT, the Trust Exposure Layer (i.e., the Security Context Broker acting as Trust Exposure Layer) functions by augmenting standard network exposure features with specialized trust-related data. The process begins when external CASTOR entities, such as service orchestration platforms managing external administrative domains, initiate a request through the Trust Exposure API. Access is strictly conditional, requiring external entities to supply specific attributes through ABAC mechanisms offered by the SCB. Once a query event is authorized, the Trust Exposure Layer extracts the necessary information from the DLT and subjects it to internal processing to satisfy privacy constraints and abstraction requirements. Only the resulting permitted trust insights are then delivered to the requesting external entity. By following this sequence, the system ensures that sensitive internal network configurations remain hidden while still supporting compliance verification and cross-domain partnerships. This workflow serves as a vital addition to runtime trust evidence flows and (S)SLA monitoring, as it allows external entities to obtain policy-governed trust conditions. Consequently, such external entities are able to respond effectively to detected non-conformant trust states or violations, integrating this trust data into broader routing decisions and traffic engineering without jeopardizing infrastructure confidentiality.

Engineering Story-VII

As an administrator of a CASTOR-enabled domain, I want to define and enforce fine-grained access and modification control policies, so that each actor can access, modify, or record only the trust-related data it is authorized to use on the DLT.

Objective Access to trust-related data managed via the Distributed Ledger Technology (DLT) infrastruc-

ture must adhere to different policies, depending on the sensitivity, origin and intended use of this information. Since these data include trustworthiness claims, trust policies, runtime attestation submissions and SLA/SSLA agreements, their management must respect auditability requirements while ensuring operational efficiency for time-critical processes such as trust assessment and traffic engineering.

Motivation To enable selective access over highly granular trust-related content, CASTOR enforces **three tier-access control layers**, each mapped to a specific category of system actors. Following a top to bottom approach, and considering also the overall CC-wide CASTOR architecture in Figure 5.1, we distinguish the following access control layers:

- **Layer 1: Attribute-based Access Control per interaction** CASTOR components or external actors such as service providers, certification authorities (CAs) or cross-domain orchestrators are granted access on a per-request basis and must present proof of attribute ownership using Verifiable Credentials (VCs) aligned with CASTOR-defined roles and policies. Such an access control mechanism is offered by the Security Context Broker. In the context of an external entity, the Security Context Broker acts as a Trust Exposure Layer offering the Trust Exposure API based on predefined rules and access policies. If approved, the appropriate abstraction function gets invoked so that the necessary obfuscation mechanisms are applied and the appropriate trust-related insights are returned (e.g. minimum trust level, SLA/SSLA compliance flag), preserving confidentiality.
- **Layer 2: Whitelisted domains**. As already mentioned in the introduction of this section, the CASTOR-enabled managed domains are able to interact with the CASTOR DLT for auditability and trust-related data exchange with the necessary level of abstraction. In this context, the main services - such as the Global TAF and the Risk Assessment - that are running with the same locality as the Service Orchestrator are considered trusted components. In addition, even the rest of the CASTOR components at the orchestration layer that can exhibit acceptable level of isolation guarantees do not require updatable access policies. Hence, the concept of whitelisting in this layer to encompass the static policies that are associated with the overarching CASTOR framework at the orchestration layer. As shown in Chapter 5, these components interact with DLT-stored data using on-chain whitelisted blockchain addresses, eliminating the need for continuous authentication or presentation of Verifiable Credentials (VCs). This approach supports low-latency access to time-sensitive data required for seamless multi-path control and overall lifecycle management of the underlying infrastructure layer.
- **Layer 3: TNDI-SP Validation**. CASTOR deployed elements contributing runtime trust evidence, such as the Local TAF and other CASTOR components such as Risk Assessment, Global TAF and Central Control Service must authenticate when establishing communication session(s) with CASTOR DLT. CASTOR uses session-based authentication, where identity validation occurs once per session — typically during secure onboarding or session renewal. Authentication leverages CASTOR Verifiable Credentials where applicable, and secure channel establishment uses Diffie–Hellman-like authenticated key exchange mechanisms directly between the TNDE enclave and the Secure Oracle running within a Trusted Execution Environment (TEE). This reduces handshake frequency and minimizes protocol overhead. In case of session updates, cryptographic key rotation is used instead of full re-authentication. The TN-DSM supports key generation, management, and refresh procedures. Details on the overarching concepts around the internal TNDE architecture and the requirements that guide the designs on the confidential TNDI-SP data channels is provided in Deliverable D3.1 [7].

Requirements To optimize resource and bandwidth usage while balancing resource consumption with system security, CASTOR shall implement a tiered access-control model. For external components and for interactions with the Security Context Broker, CASTOR shall provide an attribute-based access control

(ABAC) mechanism. In addition, a whitelisting-based access control model, implemented through static policies, shall be used to support access to the CASTOR DLT by trusted CASTOR components, as well as the TNDI-SP validation services provided by the Secure Oracle Layer for operations involving sensitive information, such as SSLAs and trustworthiness claims. A gatekeeper shall be responsible for the secure distribution of secret keys, the onboarding of workers into the infrastructure, and, more generally, the coordination of trust relationships across the network, enabling secure interaction with the DLT.

Engineering Story-VIII

As the Network Service Orchestrator, I want to continuously evaluate service-level SSLA compliance by comparing real-time trust and network telemetry against the registered SSLA requirements, so that I can detect an SSLA violation and notify relevant entities

Objective As a service provider, I want to be notified when the established (S)SLAs are no longer satisfied, while CASTOR Network Service Orchestrator takes the appropriate mitigation actions to minimize service disruptions.

Motivation Service Intent Translation & Decomposition Service (SITDS), as defined also within the context of Engineering Story-XI, decomposes high-level intent/SSLA into measurable Service Level Objectives (SLOs), covering network (e.g., latency, throughput, availability) and security/trust (e.g., integrity level, attestation status, trust score thresholds) requirements, a significant subset of which is mapped to specific service components, resources, and network paths by Path Profile Engine. CASTOR Network Service Orchestrator receives telemetry data, time-aligned and contextualized per service instance. These data are enriched with trust-related information (e.g. performance metrics, reliability indicators, trust events, attestation evidence, security posture updates) from the Global TAF. The Network Service Orchestrator evaluates the metrics against the defined SLOs. Compliance checks will be performed periodically and also triggered by events and alarms (e.g., trust degradation, failed attestation, abnormal behaviour). Metrics are evaluated in conjunction with trust indicators, allowing for example the Network Service Orchestrator to distinguish between performance degradations caused by resource constraints and those caused by trust or security violations. The outcome of the evaluation determines whether the service remains SLA/SSLA-compliant. When deviations from SLOs are detected, the Network Service Orchestrator identifies the scope (e.g. specific components, paths, or the end-to-end service), severity, and root cause of the violation, and proceeds with the appropriate actions depending on the outcome. If the outcome is attributed to the faulty operation of a worker node/server, the corresponding components of the Network Service Orchestrator undertake the resolution of the issue. Otherwise, the Network Service Orchestrator notifies the Traffic Engineering Policy Engine of the type of violation (network- or trust-based), the involved elements, and a simplified assessment of the severity of the violation. Based on this information, the Traffic Engineering Policy Engine decides whether a transition to a Fallback SSLA is required. In such case, the Traffic Engineering Policy Engine informs the PCE to enforce the corresponding paths. These actions aim to restore SLA/SSLA compliance while minimizing service disruption. The procedure supports post-incident analysis, enabling continuous improvement of service assurance process. Trust Awareness API enables relevant information transfer in order to ensure consistency in the abstracted capabilities of the domain/path that are exposed to Service Providers.

Requirements CASTOR Network Service Orchestrator should receive real- time network and trust telemetry data and assess this data in relation to the defined service objectives of the (S)SLA. This operation, regarding trust telemetry data, implies the need to have an established Trust Assessment Framework in order to process low-level trust-related evidence and assessments and derive SSLA-level trust characterizations that can be used for the evaluation of trust-related SLOs. A prerequisite for this, is a comprehensive risk analysis of the underlying network topology, which eventually will culminate in the specification of Trust Policies that can dictate the necessary parameters for the derivation of (Actual Trust Levels) ATL values and the specification of the Required Trust Levels (RTLs), the comparison of which enables the

aforementioned trust characterization (see D4.1 [5]). After processing based on defined abstraction and privacy constraints, proper information is transformed to machine-readable messages that are exposed to authorized Service Providers, in order to have insight into updated network and trust capabilities in the domain of their established services.

3.3 Reactive Phase

Engineering Story-IX

As a Traffic Engineering Policy Engine, I want to be able to dynamically revise traffic engineering policies depending on the status of the topology, so that I can ensure the (S)SLA compliance in a resilient manner.

Objective As the Facility Layer shall maintain continuous awareness of SR policy violations by integrating runtime Actual Trust Level (ATL) measurements from the ingress node with Required Trust Level (RTL) constraints, and shall trigger the Optimization Engine to recompute and enforce updated routing decisions, ensuring minimal service disruption during trust-aware path transitions.

Motivation Policy violations can arise from performance degradation, security compromises, or deliberate cyberattacks on network infrastructure. In CASTOR's zero-trust model, static trust assumptions are insufficient; trust must be continuously validated at runtime. Detecting violations enables the network to distinguish between temporary fluctuations and persistent threats, triggering proportionate responses. This requires three complementary capabilities: (1) active monitoring of network devices and continuous telemetry ingestion into an enriched Topology Graph maintaining up-to-date trust attributes for all visible segments; (2) real-time trust assessment via the Global Trust Assessment Framework (TAF) that fuses evidence from multiple trust sources and contextualizes trust decisions within the operational lifecycle; and (3) rapid re-optimization and enforcement of alternative paths that satisfy both network performance and trustworthiness constraints. CASTOR achieves this through a federated trust architecture where Local TAF agents on routers evaluate integrity via TNDE-enabled mechanisms, while the Global TAF orchestrates federated trust decisions across domains, and the Optimization Engine computes multi-path solutions that balance conflicting network and trust objectives.

Requirements The system shall integrate two complementary evidence sources: the Facility Layer's Telemetry API delivering network-driven events (link failures, congestion, latency) from TNDE-enabled routers via PCE, and TNDI-SP data channels delivering trust-related information (attestations, FSM violations, integrity checks) to the Global TAF. Furthermore, the Global TAF shall perform periodic or event-triggered re-evaluation of the Topology Graph; upon detecting significant trust or topology changes, the Optimization Engine shall asynchronously recompute alternative paths without blocking active traffic. The system shall maintain continuous comparison of ATL against Required Trust Levels (RTL) derived from risk analysis and service-level objectives. When ATL $<$ RTL for any path element, the Facility Layer shall immediately notify the Traffic Engineering Policy Engine.

Engineering Story-X

As the CASTOR service orchestrator residing in a certain domain, I need to avail a mechanism that facilitates the exchange of trust-related information with other domains as well as a way to seamlessly determine a path that may traverse numerous domains to reach its final destination.

Objective The aim of this story is to have the CASTOR service orchestrator availing all the information and tools that will enable it to control and orchestrate the establishment of paths made-up by optimised trust metrics and network resources, beyond the scope of a certain domain.

Motivation The driving need behind the considered story is the CASTOR capability to establish optimised paths across multiple domains (beyond the one where the considered orchestrator is deployed).

Requirements

The CASTOR capability to establish paths across multiple domains suggests the need to meet a number of demanding requirements:

- A commonly-agreed way to summarize trust (levels) information in each (AS). Detailed trust information is neither practical nor in line with the networking ISP practice to be exchanged; importantly, strict confidentiality and privacy requirements [1] are posed by ISPs regarding the data of their networks (domains).
- A way to exchange the aforementioned trust summary in a privacy-preserving manner (see Engineering Story-XV in D3.1 [7]). Approaches such as the appropriate extension of the current reachability information dissemination mechanisms (e.g., BGP) or the introduction of dedicated protocols may be needed.
- A mechanism, potentially based on a hierarchical structure (see Section 6.1.5) that enables the capability of domain path selection, identifying entry/exit border nodes between the involved domains. The involved domain path selection needs to be determined utilised link state information announced between domains (e.g., BGP-LS) as well as TE and trust (summary) characteristics (see above).

3.4 Service Registration

Engineering Story-XI

As a Domain Operator, I want to translate high-level service requirements into enforceable network and trust policies through a structured multi-phase process, so that I can ensure accurate and trust-aware service fulfilment.

Objective Enable the seamless translation of operator requirements across three distinct phases: i) from high-level user intents to established SSLAs, ii) from SSLAs to domain-specific Path Profiles, and iii) from Path Profiles to enforceable Segment Routing Traffic Engineering (SR TE) policies and Trust Policies. This structured approach ensures that abstract service requirements are progressively refined into concrete, actionable configurations tailored to the network topology and trust posture.

Motivation Service fulfilment in trust-aware networks requires a clear decomposition of requirements across multiple abstraction layers. By distinguishing these three phases, the architecture can handle each translation step with appropriate mechanisms—validating feasibility, mapping to available capabilities, and enforcing policies—while maintaining traceability from high-level intents down to infrastructure-level configurations.

- **Phase 1: High-Level User Intents to SSLA**: High-level user intents represent abstract service requirements expressed by application providers, encompassing both network objectives like latency and bandwidth alongside trust needs such as integrity or confidentiality guarantees. In this initial phase, these intents undergo negotiation between service providers and domain operators to

[1] The trade-off between the need for an optimal end-to-end path and the requirement to keep the secrecy of the involved TE database has been broadly identified. For instance, the IETF RFC 5298 clarifies that exposing topology information leads to ”scalability and confidentiality issues.”

formalize them into Security Service Level Agreements (SSLAs), which encode measurable commitments including trust-related Security Service Level Objectives (SSLOs). This step remains largely orthogonal to CASTOR's core mechanisms, assuming an SSLA already exists upon entry into subsequent phases; however, CASTOR explores extensions to standards like GSMA or TM Forum schemas to natively incorporate trust attributes, enabling machine-readable SSLA representations that align with intent-based networking paradigms. This phase is outside the scope of CASTOR.

- **Phase 2: SSLA to Path Profile**: Once SSLAs are established, the Service Intent Translation & Decomposition Service (SITDS) maps their network and trust requirements to domain-specific Path Profiles from a predefined catalogue prepared during the CASTOR Preparedness phase. Path Profiles encapsulate feasible combinations of network attributes (e.g., low-latency paths) and trust characteristics (e.g., required attestation levels or telemetry behaviors), serving as intermediate abstractions that bridge high-level agreements with infrastructure capabilities. In CASTOR , this translation is a direct association rather than complex decomposition—consulting the Topology Graph and Path Profile Catalogue to verify feasibility—though future extensions could leverage domain-specific intent languages like MSPL for more nuanced mappings.
- **Phase 3: Path Profile to Enforceable Policies**: The final phase refines Path Profiles into actionable configurations via the Traffic Engineering Policy Engine, generating Segment Routing Traffic Engineering (SR TE) policies for network enforcement (e.g., Flex-Algo definitions or PCE-driven SID lists) and Trust Policies for the Global and Local TAF agents. Trust Policies specify Required Trust Levels (RTLs), evidence fusion rules, and monitoring behaviors tailored to profile demands, distributed via the CASTOR DLT for immutable access by TNDE-enabled routers. CASTOR treats this as a straightforward mapping informed by Optimization Engine recommendations and real-time Topology Graph updates, ensuring paths dynamically adhere to SSLA terms while enabling reactive adaptations to topology changes.

The three phases are: Phase 1, translation of high-level user intents into Security Service Level Agreements (SSLAs) that capture both network and trust requirements; Phase 2, mapping of SSLA requirements to domain-specific Path Profiles that capture the network and trust characteristics required to satisfy the agreed service terms; and Phase 3, translation of Path Profiles into concrete, enforceable policies: SR TE policies for network requirements and Trust Policies for trust-related requirements.

Requirements To perform these translations accurately and ensure requirement feasibility, the Service Intent Translation & Decomposition Service (SITDS) must consult: i) the current status of the network topology via the Topology Graph, ii) the complete catalogue of available Path Profiles, and iii) critical infrastructure information provided by the Service Orchestrator. By continuously monitoring these inputs, SITDS can determine whether operator requirements align with actual network and trust capabilities, enabling dynamic adaptation when topology changes occur or new network resources become available.

Engineering Story-XII

As the Traffic Engineering (TE) Policy Engine, I want to be able to translate paths into low level configurations, so that CASTOR can apply reinforcement in the internet equipment.

Objective The TE Policy Engine has a complete overview on the network status (both in terms of network performance and trust posture) and constitutes the main control service at the orchestration layer. When necessary, the Traffic Engineering Policy Engine receives recommended paths from the Optimization Engine and is responsible for converting these abstract, high-level path specifications into concrete, device-ready configurations that can be deployed across the network infrastructure. This translation process encodes the optimal paths determined by the Optimization Engine, into enforcement mechanisms: Flex Algo Definitions (FAD) that are advertised across the network segments via routing protocols, and

Segment Routing (SR) policies that are instantiated at the ingress nodes of the requested workloads, enabling direct network control without reliance on distributed routing protocol convergence, among others.

Motivation To execute this translation accurately, the Traffic Engineering Policy Engine must obtain comprehensive information from multiple sources: the Topology Graph maintained by the Facility Layer to understand the current network topology and trust attributes, the Path Profile Catalogue to ensure that the generated configurations satisfy the network and trust requirements of each offered service profile, and the specific recommended paths from the Optimization Engine along with the path profile requirements that guide the configuration generation. By consulting these inputs, the Traffic Engineering Policy Engine can generate routing configurations that not only reflect the optimal paths but also enforce the necessary trust constraints and network policies established in the SSLAs. The reason for considering the use of different enforcement mechanisms is to be able to contemplate different scenarios: FADs, to provide the network with dynamic algorithms so that it always has a routing level, CASTOR PCE, to specify explicit and static routes that always follow the results of independent optimization engines; and vanilla PCE, which provides a higher level of dynamic abstraction in the network if CASTOR PCE is not available.

Requirements Once the low-level configurations are generated, the Traffic Engineering Policy Engine communicates these enforceable policies back to the Facility Layer, which then coordinates their deployment either through the Network Controller—using management interfaces such as RESTCONF or other legacy management protocols for control-plane orchestration/communication—or through the Path Computation Element (PCE) using, for example, the PCEP protocol for data-plane enforcement on ingress routers. This mechanism ensures that the computed paths are consistently applied across the network infrastructure, bridging the gap between optimized path recommendations and actual forwarding behavior on router elements.

Chapter 4

CASTOR Management and Orchestration Framework towards Flexible & (S)SLA Compliant Traffic Engineering

The high-level architecture of the CASTOR framework is illustrated in Figure 4.1. In detail, during the operation of the framework, a Service Provider can request the establishment of a path to accommodate traffic of a new service with specific network and trust requirements. The involved requirements are provided by the Service Provider through intents, which are typically described in a domain-specific language or lately expressed (even) in natural language. These intents are subsequently translated by the Service Intent Translation component into High-Level (HL) SSLAs using a set of predefined HL SSLA templates (or blueprints). Note that the intent modelling and translation remains out of the CASTOR scope and therefore, the CASTOR framework is designed to operate in line with the IBN principles [23] but has abstracted much of the technical details.

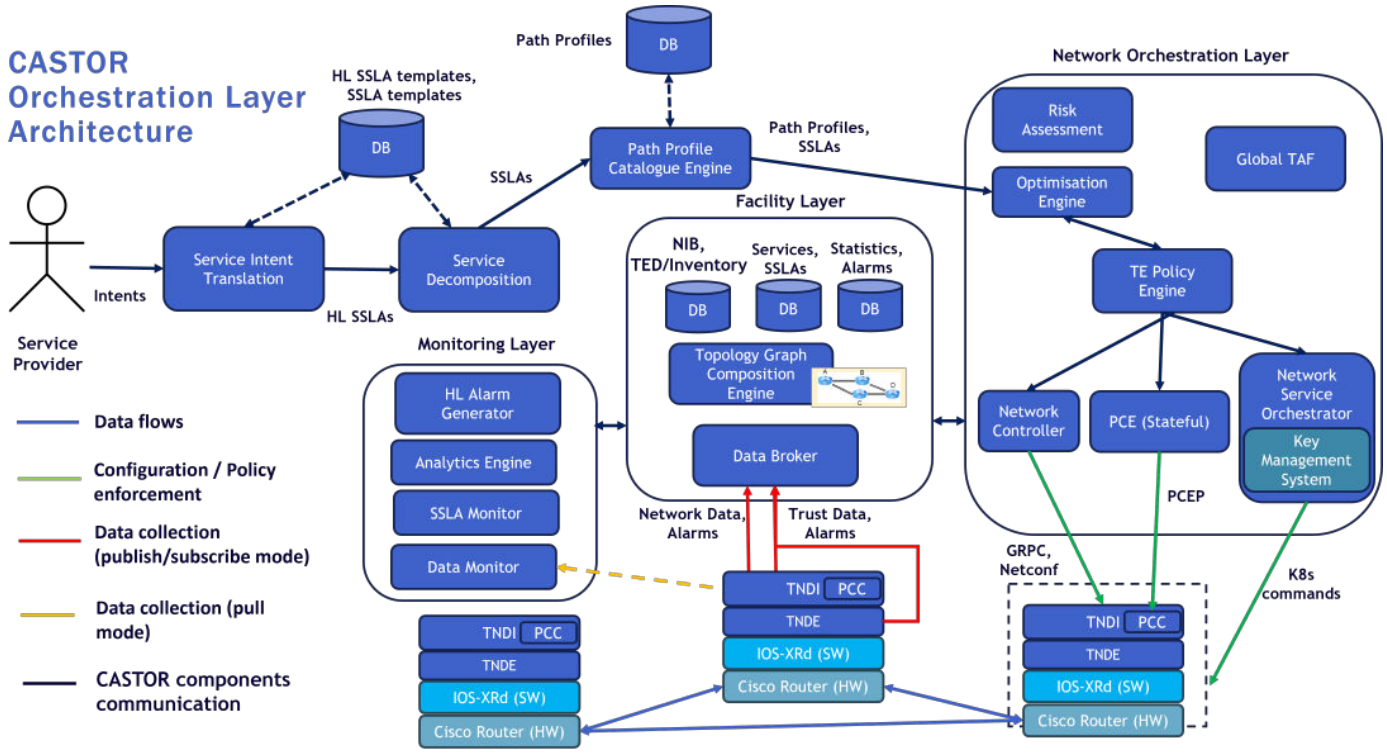

Figure 4.1: High-level architecture of CASTOR management and orchestration framework

An HL SSLA is expected to include a set of basic functionalities required for the fulfilment of the Service

Provider requirements. Then each HL SSLA is further translated to a set of agreements that can be offered by the CASTOR framework. This translation is realised by the Service Decomposition component (see Figure 4.1) and the outcome is forwarded to the Path Profile engine. The latter, utilising a set of Path Profiles templates (which are stored in the relevant database), identifies those profiles (i.e., network and trust characteristics of network paths) that can support the original (routing) needs expressed in the intent.

The generated Path Profiles and SSLAs are forwarded to the Network Orchestration layer for the actual establishment of the required network paths with the appropriate network and trust characteristics. In the Network Orchestration layer, the TE Policy Engine is responsible to generate the policies that need to be enforced on the network elements in order to establish the path for the traffic of the new service(s). The network controller and PCE constitute (see Section 4.3.2.1 paragraph) the main tools towards that end. As explained later, SR policies associated with FlegAlgo-derived ”best” paths are realised by the Network Controller. On the other hand explicit path determined by the CASTOR Optimization Engine are enforced by the PCE.

Four are the main CASTOR modules that are implementing the required functionality. The three layers (shown in bubbles in Figure 4.1) i.e., the network orchestration, facility, and monitoring & analytics layer, which are coupled with the software stack residing on the router devices (i.e., TNDE, TNDI etc).

In the network orchestration layer, the TE Policy Engine sends the path decisions to the Network Service Orchestrator which is responsible to manage the VM/Docker resources. It typically examines if further resources are required to be allocated.

Alongside the Network Orchestration layer, the Facility layer provides to the other layers an up-to-date view of the network topology enriched with trust, statistics and alarms related information. The Topology Graph Composition Engine is responsible to retain and share among components this dynamic enriched topology graph. In addition, it includes all the required databases, as well as a data broker for the real time exchange of messages among components and with the TNDIs established on the network elements.

Finally, a dedicated monitoring and analytics layer continuously monitors the involved data and service levels (both network and trust aspects). It verifies that at any time SSLAs are fulfilled for all existing services. It includes the required modules for SSLAs monitoring as well as data collection and analysis. The Data Monitor component is responsible to collect selected data from the Data Broker (located in the Facility Layer) and applies data pre-processing and storing to several DBs located in Facility layer. The SSLA Monitor continuously verify that the SSLA related statistics collected from the network elements are in line with the agreed SSLAs. The established services/SSLAs parameters are retrieved from the services/SSLA database located in the Facility layer. In case of service degradations (essentially an SSLA violation) the SSLA Monitor generates SSLA Alarms which are forwarded to the TE Policy Engine which drives the subsequent CASTOR operations (see the text below and the flow in Figure 4.2). The Monitoring & Analytics layer, finally, includes an analytics functionality which uses post-processing algorithms over stored data and the corresponding findings feed an HL Alarm Generator.

CASTOR high-level operational flow: In Figure 4.2 we move beyond the static description of individual modules; we provide a *dynamic* yet *abstracted* view of the CASTOR framework, highlighting a sequence of steps for two cases of operation: the (so-called) normal one and the case of an SSLA violation.

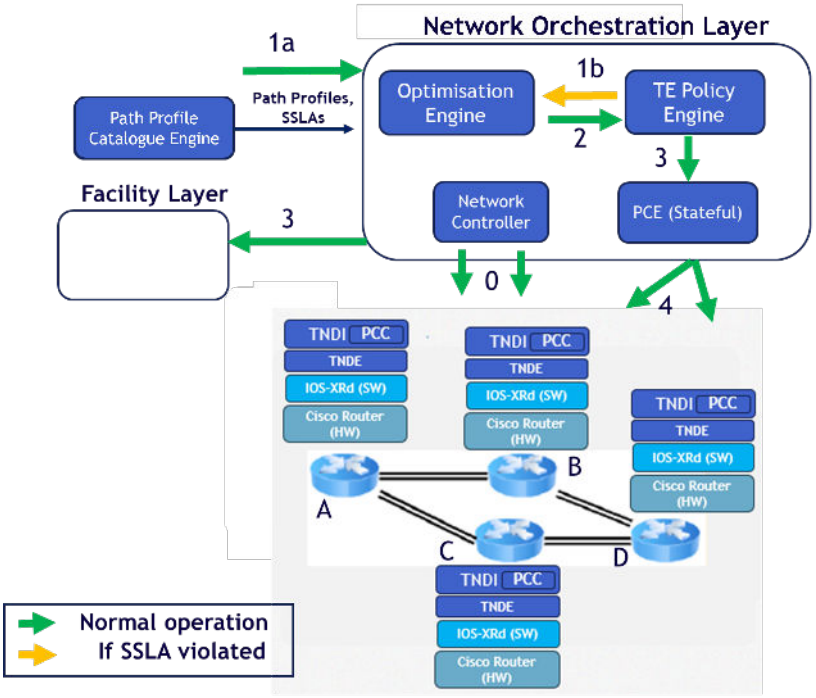

Figure 4.2: Sequence of basic steps of the CASTOR framework operations

0. Initial network elements configuration and background/previous states Flex-Algo config
1a. New SSLA signed and put into force
2. Optimized path response
3. Selection of policy (optimal path) and communication to the PCE and facility layer [FL needs to be informed to allocate relevant resources]
4. PCE enforces the policy [routers A → B → D]

0. Initial configuration network elements and background/previous states Flex-Algo config
1b. An SLLA violation detected
1c Get me a new path [due to a change]
While waiting for the optimization outcome (2), use network controller (0) to establish a Flex-Algo path
2 Optimized path response
3 Selection of policy (optimal path) and communication to the PCE and facility layer
4 PCE enforces the new policy overwriting the previous one.

Figure 4.3: High-level description of the CASTOR normal operation (left) and SSLA violation (right), corresponding to the arrows of Fig 4.2

In both considered cases, we assume an initial configuration phase (see step '0' in Figure 4.2) and comments in Figure 4.3) that includes the bootstrapping and configuration of all network elements (e.g., onboarding of 'new' devices). This step also includes the identification of Flex-Algo paths determined by optimizing typical IGP metrics enhanced with a basic trust dimension (e.g., low/medium/high trust).

The CASTOR normal operation considers a newly-introduced SSLA (see step '1a' in Figure 4.2) made known to the orchestration layer. SSLAs may call for evolved requirements pertaining to combinations of network resources and trust guarantees. To meet them, an optimised path (step '2a') is being computed by the relevant engine. The outcome is communicated to the PCE and the Facility Layer so as to (re)allocate any required (virtualised) resources. PCE then communicates (step 4) with the network routers to realise the network path, over which the relevant traffic will be steered.

When a change in the underlying network topology or trust values takes place causing SSLA degradation, an SSLA violation is detected (step 1b). A new request for an optimised path is directed to the optimization engine (step 1c). While the result being expected, CASTOR resorts to the Flex-Algo paths, estimated

earlier (in step '0'), to support the new SSLA needs. Those paths are expected to be shaped by a less fine-grained trust representation compared to the rigorous optimization computations, but are expected to serve as a fast CASTOR response to violation events. It is to be explored how efficient the approach would be and what footprint on network resources may pose. By the time the optimization engine result becomes available, the steps that follow are similar to the normal operation (step 3 and 4) updating the previously identified path.

The following section provides an initial description of the components that comprise the CASTOR Orchestration Layer, establishing the context for their functional specification in the upcoming deliverables.

4.1 Management and Co-enforcement of Trust Insights in CASTOR

One of CASTOR's primary objectives, as defined in D4.1 [5], is the systematic measurement of trust. The goal is to transform trust into meaningful insight for traffic engineering and, ultimately, to co-enforce it alongside standard network requirements. In D4.1 [5], the exploration of these objectives culminates in the realization of two key components: the Risk and Trust Assessment Framework and the Optimization Engine. To function effectively, these components require secure and robust evidence collection mechanisms capable of reporting information from the routing plane (infrastructure layer) back to the orchestration layer. This requirement is highlighted in the CASTOR device-side Trusted Computing Base (TCB) specifications found in D3.1 [7].

These capabilities position CASTOR as a key element for incorporating trust into any network orchestration framework—whether it acts as a central control service within a single domain or across multiple autonomous systems. A guiding requirement for all critical CASTOR artifacts is the decoupling of design from technology lock-ins. For instance, D3.1 [7] highlights that the in-router TCB—part of the CASTOR Trust Network Device Extension (TNDE)—is designed to be agnostic to specific technologies. This "crypto agility" allows the CASTOR TNDE to be instantiated on various router models with different Root of Trust capabilities, provided they meet the minimum security requirements elaborated in D3.1 [7].

Similarly, the Trust Assessment Framework and the Optimization Engine are designed with a high degree of integration flexibility. They do not rely on a specific orchestration solution, allowing CASTOR to support diverse environments. This includes network controllers managing hardware lifecycles, SDN controllers focused on control-plane programmability, or—as examined below—comprehensive orchestration solutions that manage the secure lifecycle of virtualized network elements from fulfilment to continuous assurance.

4.2 Traffic Engineering Policy Engine

The *Traffic Engineering (TE) Policy Engine* acts as the decision-making component within the Orchestration layer. It consumes topology, telemetry, and optimization outputs, and maps them into concrete configuration and policy enforcement actions on the network. Depending on the context, the TE Policy Engine may revise local router configuration, enforce Segment Routing Traffic Engineering (SR TE) policies, or coordinate with external control elements such as a PCE.

At a high level, the actions triggered by the TE Policy Engine can be grouped into baseline network configuration and overlay traffic engineering policies.

- **Initial Network Configuration**
  The TE Policy Engine may trigger changes to baseline router configurations to support traffic engineering provisioning, including initial interface bring-up, routing protocol instantiation (e.g., IS-IS

or OSPF), control-plane tuning such as BGP policy and parameters, and the definition of Flex-Algo Definitions (FADs). These actions can be loaded either as configuration during the launch of the network element or enforced via management interfaces (e.g., NETCONF) by the Network Controller. As detailed in Chapter 7, this configuration may also describe the technical details in order for a network element to be able to communicate with a PCE functionality, allowing the enforcement of policies through it as well (e.g., using PCEP sessions as illustrated in Chapter 6).

- **Enforcement of New SR TE Policies with Explicit Paths**
  Based on the output of the Optimization Engine, the TE Policy Engine may compute and install new SR TE policies with explicitly defined active and backup paths. These policies are pushed to the network to steer traffic along optimized paths that satisfy performance, resilience, or capacity objectives.

- **Augmentation of Existing SR TE Policies via PCE-Computed Paths**
  The TE Policy Engine may incorporate explicit paths provided by a Path Computation Element (PCE) into existing SR TE policies. This model allows routers to request paths from a PCE, while the TE Policy Engine remains responsible for validating, selecting, and enforcing the resulting policy through the Network Controller.

- **Derivation of Dynamic SR TE Policies**
  The TE Policy Engine may derive dynamic SR TE policies from Optimization Engine outputs. For example, it may construct policies that avoid links with specific colors or administrative constraints. While such dynamic policies may not be fully explored in *CASTOR*, they represent an important class of the TE Policy Engine actions that adapt forwarding behaviour to evolving network conditions.

- **Topology Annotation and Attribute Management**
  The TE Policy Engine is responsible for enforcing topology annotations such as link coloring and Flex-Algo attributes. These actions are applied via the Network Controller and directly influence both optimization outcomes and the behavior of SR TE policies.

4.3 Traffic Engineering: Decision and Enforcement Points

4.3.1 TE Policy Decision

As mentioned in Section 4.2, there are multiple types of network configuration that allows the overall management of Traffic Engineering (TE) policies. In this section, we examine two distinct approaches to Traffic Engineering (TE) enforcement in the network. The first approach represents the conventional method, where the Network Controller is responsible for applying all types of network configuration in a deterministic manner. The second approach leverages the Path Computation Element (PCE) workflow (as further analyzed in Chapter 6), which is instantiated with a specific purpose to enforce explicit Segment Routing paths and provide a higher level of automation for dynamic and efficient network optimization.

1. **Enforcement through Network Controller** The Network Controller serves as the base source of enforcement for all network configurations across routers. It covers all types of configuration, including IGP and BGP settings (e.g., OSPF, BGP configurations), affinity bits for path computation based on trust evaluation, and Segment Routing (SR) policy provisioning for head-end routers. Whenever a new router is onboarded (i.e., once it has been admitted by the Network Service Orchestrator), the TE Policy Engine triggers the Network Controller to push the required configuration. Similarly, upon a new service request, the associated SSLA gets mapped to one of the supporting path profiles.

This allows the TE Policy Engine to provision the appropriate network configuration to be enforced via the Network Controller. As illustrated in the example of Chapter 7, the Network Controller is able to interact with the ingress router and apply the appropriate SR policy that fulfils the new service request, including any preliminary network configuration as highlighted in Section 4.2.

2. **Enforcement through PCE** The PCE is responsible for enforcing explicit Segment Routing paths using label stacks. PCE-based decisions are applied either in response to service requests or as a reaction to topology changes. CASTOR extensions on the PCE enable the TE Policy Engine to enforce network- and trust-aware optimization results to the ingress router of the associated workload traffic. As detailed in Chapter 6, depending on the applied policy, the PCE can perform standard optimization requests (i.e., vanilla PCE functionality) until the results of the CASTOR Optimization Engine results are available for enforcement. As a fallback, SR policies can rely on candidate paths with lowest priority, potentially leveraging Flex-Algo optimization, either executed at the PCE or directly on the ingress router (a relevant example is provided in Chapter 7).

4.3.2 TE Policy Enforcement

4.3.2.1 Network Controller

Typically, the term Network Controller is used as the core building block of a Software-defined networking (SDN) architecture [39], which is a network management method that supports dynamic programmable network configuration.

In traditional networking, network devices can be divided into the management plane, control plane, and forwarding plane. Management plane is responsible to orchestrate services and formulate service policies. The control plane, using various algorithms, calculates and maintains forwarding decisions. The forwarding plane handles the movement of data packets to the appropriate interfaces. The SDN concept suggests the decoupling of the control and forwarding functions of network devices so that the control plane of network devices can be directly programmed and network services can be abstracted, practically removing any dependency from the underlying hardware. In this perspective, the Network Controller is responsible for network device management, network service orchestration and service traffic scheduling, which features low costs, centralized management, and flexible scheduling. SDN Network Controllers are provided by several vendors (e.g. Cisco, HP, Juniper, NEC) as well as open source solutions: NOX [25], POX [29], OpenDaylight [27] and recently Open Network Operating System (ONOS) [26].

In the CASTOR architecture, the Network Controller is responsible for a subset of the aforementioned functionalities. This design choice is intentional: CASTOR aims to specifically highlight the challenges associated with the virtualization of network elements and their dynamic resource management, which include not only performance and efficiency concerns, but also additional security considerations. By focusing on these aspects, CASTOR enables a more targeted investigation of these critical issues, leaving other functionalities to complementary components. In detail, the Network Controller is mainly responsible for network device management (explicitly) and service traffic scheduling (implicitly). The network service orchestration is realised by the CASTOR Network Service Orchestrator.

Regarding network device management, the Network Controller is responsible to enforce toward the network elements (e.g. routers) the appropriate configuration actions that will enable routing decisions in line with the policies generated by the TE Policy Engine. The communication of the involved parameters can be realized through several protocols including Netconf, Restconf or GRPC. In detail, when a new network element (e.g., a router) is onboarded into the network, the TE Policy Engine generates the initial network policies along with the required static configuration and forwards them to the Network Controller, which enforces them on the network elements (e.g., routers).

It is important to stress that in the CASTOR system, two components are responsible to enforce policies to

the network elements: network controller and PCE. Network controller enforces static policies to realise the initial configuration (on the onboarding case), as well as policies which are related with FlegAlgo configuration and use (more details are presented in Chapter 7), while PCE is responsible to enforce policies in a dynamic manner to realise explicit paths. In the latter case the decisions are made on the Optimisation Engine, the policies are generated in the TE Policy Engine and PCE is responsible to enforce the policies to the network elements.

Regarding service traffic scheduling, the Network Controller does not decide on the traffic scheduling algorithms/policies to be adopted or applied, but it enforces to the network elements, the traffic scheduling policies decided by the TE Policy Engine. Similarly to the previous paragraph, the Network Controller is responsible for applying the TE decisions to the routing plane.

4.3.2.2 PCE for Explicit Paths

A path computation element (PCE), explained in Section 6.1 represents a specialized network component or dedicated server capable of performing constraint-based path computations for traffic engineering systems operating within multiprotocol label switching (MPLS), Generalized MPLS (GMPLS), and segment routing (SR) environments. Rather than distributing path computation responsibilities across individual network routers — which typically possess limited topology knowledge and computational capacity — the PCE consolidates path computation into a dedicated, resource-aware platform that maintains comprehensive network state information through a centralized Traffic Engineering Database (TED). The PCE receives path computation requests from Path Computation Clients (PCCs) such as head-end routers or management systems, processes these requests by applying constraint-satisfaction algorithms considering criteria including bandwidth availability, latency bounds, policy constraints, and shared risk link groups (SRLGs), then returns optimized explicit paths that satisfy all specified requirements. This centralized computation model significantly reduces computational strain on individual routers while enabling more globally optimal path selection than would be possible with distributed routing protocols, particularly valuable in large, complex multi-domain networks where optimal path determination requires consideration of extensive topology information.

The PCE architecture has evolved substantially beyond simple stateless path computation to encompass hierarchical, stateful, and domain-coordinated deployment models suitable for contemporary network complexity. Hierarchical PCE (H-PCE), explained in Section 6.1.5, architectures organize multiple PCE instances hierarchically, with parent PCEs maintaining inter-domain topology abstractions and coordinating computations across administrative boundaries, while child PCEs manage intra-domain path computations and implement parent-level decisions. Stateful PCE implementations maintain synchronized databases tracking not only available network resources (the TED) but also existing label-switched paths (LSPs) and reservations, enabling the PCE to compute optimized new paths that account for real-time network state and existing service commitments rather than merely considering theoretical resource availability. The PCE communicates with other network components through standardized protocols, particularly the Path Computation Element Communication Protocol (PCEP), which defines request/response semantics and enables integration with orchestration and control plane systems in Software-Defined Networking (SDN) environments. Through these evolution, PCE architecture has emerged as a foundational component enabling sophisticated traffic engineering, service orchestration, and dynamic network optimization in multi-domain, multi-technology environments.

4.3.3 Network Service Orchestrator

A network service orchestrator (NSO) functions as the central coordination and execution component within management and orchestration (MANO) frameworks, automating the lifecycle of virtualized network services. This centralized orchestration approach fundamentally transforms network service de-

ployment from a manual, operator-intensive process requiring days of configuration work into a highly automated workflow executable in minutes. The orchestrator maintains real-time visibility into infrastructure capacity across multiple Points of Presence (PoPs) or administrative domains and leverages this knowledge to make dynamic placement decisions that optimize service instantiation while respecting specified constraints around latency, cost, or resource efficiency.

The Network Service Orchestrator (NSO) of CASTOR Orchestration Layer is responsible for network node onboarding, monitors the availability of the required compute and network capabilities to ensure compliance with SLA/SSLA/path profiles, and triggers the appropriate adaptive actions when needed based on information from Monitoring and Analytics Layer and TE Policy Engine.

The CASTOR architecture is inherently orchestration-agnostic, with core architectural components defined independently of any specific deployment technology. This designing approach ensures extensibility and long-term sustainability, allowing different stakeholders to adopt orchestration solutions that best fit their operational, regulatory, or technological constraints.

The NSO is responsible for the instantiation, configuration, and operational management of virtual routers (vRouters). The NSO leverages Kubernetes [21] for the instantiation and lifecycle management of virtualized routing functions, deployed over container-based infrastructures. It provides an effective control framework for managing distributed, containerized network components. Kubernetes natively supports automated lifecycle management, scalability, and resilience, making it well-suited for implementing orchestration logic that must dynamically deploy, configure, and adapt services across heterogeneous infrastructures. Its extensible control plane, rich API ecosystem, and strong integration with networking, monitoring, and policy-enforcement mechanisms enable the realization of a flexible orchestration environment, fully aligned with the objectives of CASTOR. Nevertheless, Kubernetes does not constitute a mandatory architectural dependency, but a realization choice aligned with current industry practices.

In this context, IOS XRd [10], a containerized virtual router offered by Cisco, is leveraged as a representative and practical realization of a router platform that aligns well with CASTOR's architectural requirements. The use of IOS XRd does not constitute a design constraint, but rather an implementation choice that facilitates prototyping, experimentation, and integration with Kubernetes-based infrastructures. IOS XRd can operate as a core router, edge/border router, WAN gateway, or inter-domain router. Moreover, IOS XRd operates as a fully containerized instance of Cisco IOS XR software, packaged as a Docker container that runs within Kubernetes pods (see Figure 4.4).

Managing high-performance vRouters as typical Kubernetes Pods presents inherent challenges, for instance, regarding the requirements for direct PCIe device access and CPU pinning. Therefore, we aim to explore a DaemonSet-based deployment model where a dedicated data-plane instance is instantiated on each worker node. Under this model, Kubernetes guarantees that exactly one Pod instance is deployed on each eligible worker node, without requiring explicit replica management. As a result, exactly one IOS XRd instance is instantiated per Kubernetes worker node (VM), and each selected node hosts a single Pod encapsulating the IOS XRd container that implements the required routing functionality. The Pod represents the schedulable and lifecycle-managed entity, while the container(s) realize the actual execution logic. The NSO ensures that each IOS XRd instance is provisioned with the required compute capacity and network connectivity to operate as a router. The NSO does not participate in routing protocols realization, it delegates all protocol execution to IOS XRd. From the orchestrator's perspective, IOS XRd is treated as a service endpoint that provides routing capabilities.

Based on the domain operator requirements and the incoming service requests, the NSO determines where IOS XRd instances should be deployed, how many instances are required, and which capabilities (interfaces, performance profiles) must be allocated. Beyond initial deployment, the NSO manages the full lifecycle of IOS XRd vRouters, including configuration, updates, healing, and decommissioning. In the event of failures at the Pod or node level, Kubernetes ensures re-instantiation, while the NSO maintains consistency of the routing and service state by rapid fallback activation of worker nodes or capacity boosting of other worker nodes.

IOS XRd supports multiple configuration management paradigms (e.g., CLI, NETCONF/YANG, gNMI and RESTCONF) and exposes routing state, interface statistics, and performance metrics via telemetry mechanisms. The NSO's Resource Manager consumes this information that receives from the Monitoring and Analytics Layer, to verify SLA/SSLA compliance, detect failures or performance degradation, and trigger remediation or fallback actions at the pod level (e.g. redeployment, reconfiguration). This enables closed-loop orchestration without compromising routing autonomy.

The deployment of IOS XRd requires a container runtime, which includes the complete IOS XR operating system, routing protocols and management interface, and kubelet for Pod execution and lifecycle management (acts as the execution and supervision component that ensures the correct operation of IOS XRd at the node level). The Kube Proxy is not used for traffic forwarding, but can be leveraged for management functionality (e.g., secure onboarding flow presented in D3.1 [7], the establishment of TNDI-SP communication channels between the orchestration layer and a vRouter's TNDE). Kubernetes invokes the appropriate CNI during Pod instantiation, enabling the creation of multiple interfaces via mechanisms (e.g. Multus, SR-IOV). These interfaces are attached to the vRouter Pod, allowing the vRouter to participate in routing, without awareness of the underlying Kubernetes environment. IOS XRd natively implements the Path Computation Client (PCC) functionality and no additional components are required beyond configuration and connectivity to an external PCE. Figure 4.4 depicts the aforementioned implementation elements of an IOS XRd vRouter in Kubernetes environment, namely the implementation of the TNDI.

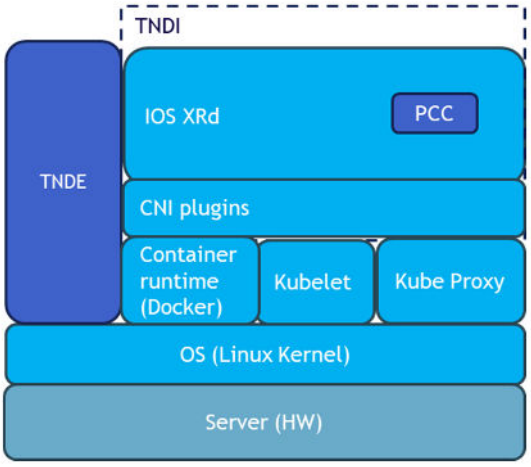

Figure 4.4: Implementation elements of a IOS XRd vRouter

The Control Plane Node of NSO provides a logically centralized but physically distributed control framework that governs how workloads and network functions are instantiated and managed across worker nodes. When IOS XRd vRouters are deployed as DaemonSet Pods, the Kubernetes Control Plane Node includes the API server, the etcd, and the Controller Manager (which includes Daemonset Controller). The DaemonSet Controller ensures that exactly one IOS XRd instance is instantiated on each eligible worker node (VM). No routing, forwarding, or SRv6 control-plane functionality is executed on the Control Plane Node. At the core of the Control Plane Node lies the Kubernetes API Server, which acts as the single entry point for all cluster-level operations. All operations related to cluster resources (Pods, Nodes, etc.) are performed exclusively via the API Server. For network node onboarding, the API Server stores and exposes the DaemonSet specification, the DaemonSet Controller ensures that one IOS XRd Pod is instantiated on every eligible node, and the worker-node kubelet performs the actual Pod creation and execution. Subsequently, NSO components interact with the API Server to trigger IOS XRd instantiation and decommissioning, apply configuration updates, enforce placement constraints (node selection), and monitor cluster and workload status. The etcd is the datastore that persists cluster state. It is the single

source of truth for Kubernetes control-plane decisions, storing the DaemonSet-based desired deployment topology, bootstrap templates and credentials, and node lifecycle state.

SLA/SSLAs and path profiles are translated into specific network and resource requirements by the Resource Manager, and then are mapped onto concrete resources (worker nodes, NICs) by Resource Abstraction, while K8s Lifecycle Manager monitors the deployment and the lifecycle operation of the required Pods (instantiation, scaling and placement control, upgrade and decommissioning). The Network Function Manager (NFM) is responsible for initial bootstrap process of IOS XRd instances and the management of configuration templates. These templates define diverse configuration patterns for different vRouter's demands and roles (core, edge, border, inter-domain). During instantiation or reconfiguration, the NFM selects the appropriate template and alters the runtime parameters based on node/server capabilities and service requirements.

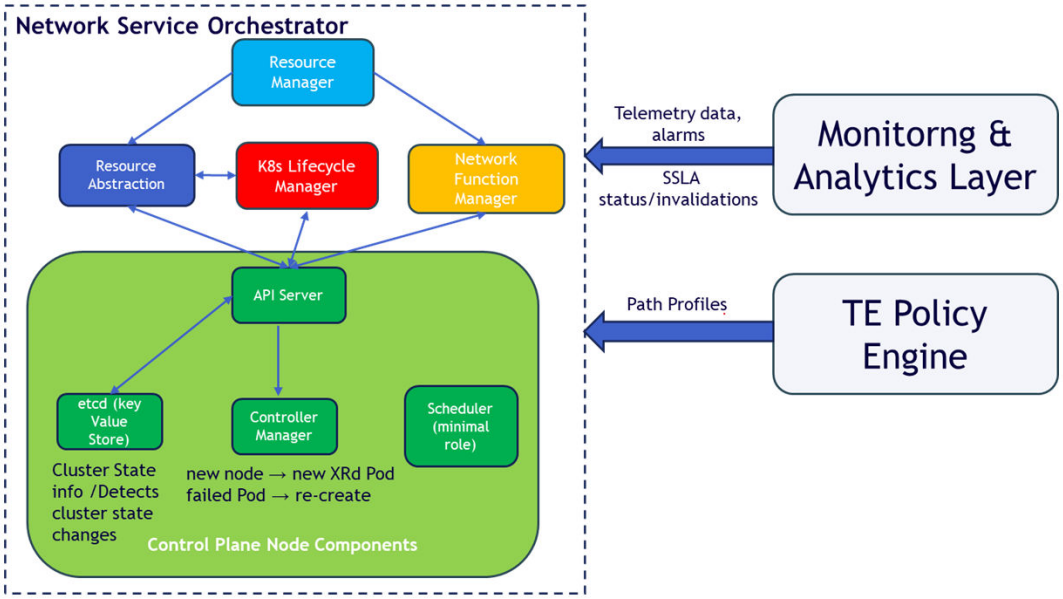

Figure 4.5: Network Service Orchestrator

4.3.3.1 Launching Securely Verifiable Confidential Containers

First and foremost, in order for the CASTOR Trust Assessment Framework to function properly, the TNDE artifact in each vRouter should be securely launched (see Figure 4.6, which introduces a zoomed-in illustration of the Kubernetes-based security architecture). To achieve this, CASTOR is directing its investigation towards Kubernetes technology, with K3s currently considered one of the primary candidates. K3s is a fully conformant, production-ready Kubernetes distribution tailored for operational efficiency in unattended, resource-constrained, and remote environments. It ensures the reliable deployment and management of containers, supporting the needs of production workloads. In the present deliverable, the Kine, the Supervisor and the Tunnel Proxy are explained, that facilitate the launching of the TNDE-based containers. More information regarding the Kubernetes internal blocks and their functionalities are available in D5.2 [8].

The Kine database system stores cluster state data, Kubernetes resources, worker node status, user roles, roles bindings, and container scheduling and orchestration information, enabling Kubernetes components to perform their functions and storing Kubernetes secrets for authentication. The Tunnel Proxy in K3s ensures secure and efficient communication between control and worker nodes, encrypting data transmission and protecting the cluster from external threats. Lastly, the Supervisor in K3s enhances communication between worker nodes and the server's control plane functions by acting as an intermediary firewall. It establishes a web-socket connection for registration, connects to the supervisor and

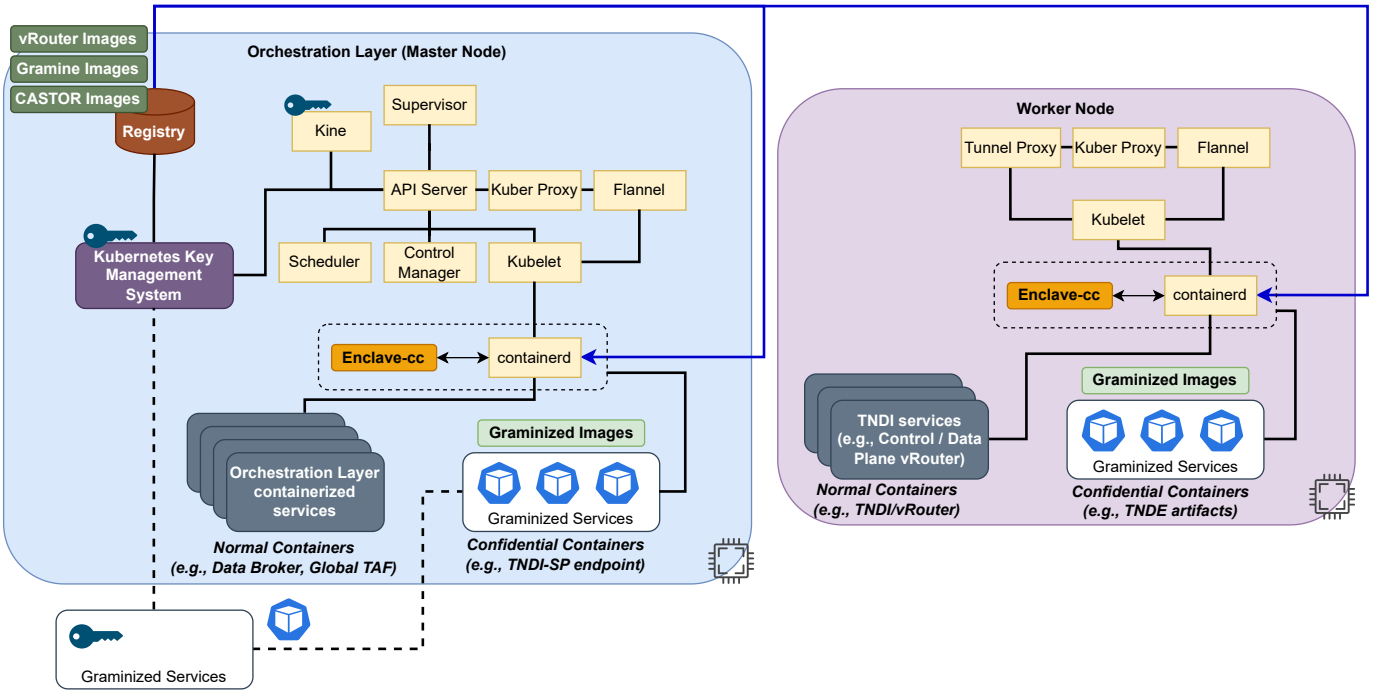

Figure 4.6: Kubernetes-based Container Architecture

kube-apiserver via a load-balancer, and ensures resilient connections during server outages. These components are crucial for launching kubernetes containers.

As previously explained, CASTOR aims to follow the Confidential Computing (CoCo) paradigm. Enclave-cc (or Edgeless Systems' MarbleRun project) is leveraged to launch confidential containers. All containers are initially deployed through a legacy container. The security added by leveraging enclave-cc and CoCo depends on the trust requirements. **These trust requirements are dictated by the equilibrium between safety (i.e., which may be affected by the overhead of such operations) and security; hence are dependant on the offered service and its operational profile. Thus, not all containers conform to identical principles.** *It should be mentioned that for the purposes of CASTOR demonstration activities, Kubernetes and enclave-cc is leveraged to launch confidential containers.*

Towards this direction, the Registry residing at the cloud-level comprises two types of docker-based image files (i.e., including their Manifest files), that introduce the: i) the *service manifest* files (i.e., vRouter service docker image) and ii) the *Confidential Computing (CC)-enabled service manifest* files (i.e., the TNDE extensions that will run within a TEE). The first type comprises the standardised docker image information used for the deployment and orchestration of the service, specifying the size, bandwidth, network dependencies of the container. The second type, the CC-enabled image defines the trusted files and what runs isolated within the TEE. *Note that for CASTOR the employed TEE is Gramine, which is based on Intel SGX, hence, for the remainder of this document we will refer to the secure conversion of a legacy container with the term "*Gramine-enabled image*".*

The Gramine-enabled image is built leveraging enclave-cc, which constructs the .sgx variant of the service manifest file including information on the security requirements of specific files, processes, system calls, etc. of the service and the level of isolation they need for their secure execution. **In essence, the enclave-cc builds an overlay of the Gramine-enabled image on top of a regular image file**. This Gramine-enabled image leverages the Intel SGX hardware in order to launch confidential containers, hence the manifest includes relevant information for the deployment over this trusted hardware.

The Gramine-based Manifest file further includes the MR Enclave Reference Value Measurement, that is in essence, a reference value, based on the bootup measurement of the enclave. Furthermore, in addition to the Gramine-based Manifest file, enclave-cc supplies the digest of the application intended for deployment within the container, referred to as the MR Enclave. This digest is signed to generate the .sig

file, allowing anyone to verify its authenticity. Kubernetes acts the underlying technology that enables the deployment and orchestration of these manifests at the Infrastructure Layer.

To ensure that the correct application is being launched, **verification through recalculation of the MR Enclave Reference Value Measurement** also takes place. This verification is performed by the Kubernetes Key Management System, residing at the cloud, which accesses the available information within the Registry. To perform this task, the Kubernetes Key Management System recreates the Manifest file for a given Gramine-based docker image, based on the MR Enclave measurement, and compares the two values; the one that it calculated with the one received by the registry. After the successful verification, the Kubernetes Key Management Service releases the Kubernetes secret key (i.e., used for the secure communication with other containers or with the Master Compute Node) and the certificate. This certificate includes the public part of the Kubernetes public key, to enable the authentication of the container's workload. Whenever a confidential container is launched, the Kubernetes secret key and the certificate need to be retrieved as they dictate the expected state of the container, which is needed for its verification.

4.3.3.2 Verification of CoCo Workload & Container Binding

Up until now, the secure deployment/launching of a container has been covered. When leveraging the term secure deployment, we also capture the final operation of the integrity check of the launched (confidential) container. This is done by extracting and sharing the MR Enclave Reference Value Measurement that holds the hash of the whitelist of binaries been instantiated as part of the Gramine-enabled container. The verification is performed by the Kubernetes Key Management Service as part of verifying the correctness of the container prior to releasing the kubernetes encryption key further enabling the Establishment of secure and authentication channels with other containerized services and/or the Master Compute Node.

However, while this operation ensures the integrity of the launched containerised service, it does not provide any evidence on the provenance of the container itself. This is required in complex and multi-tenant environments where multiple instances of the same application and/or security services might be instantiated and, thus, enhanced authentication capabilities are required for guaranteeing the communication with the intended service.

To achieve this, in CASTOR we proceed with binding the public part of the container's Kubernetes (identity) key, as part of its certificate issued by the Kubernetes Key Management Service, with the attestation secret/key based on which an attestation attribute is monitored and signed. This allows any external verifier (e.g., the Network Service Orchestrator) to not only attest to the correct state of the TNDE container (based on the attestation signature received) but also have the necessary guarantees of the authenticity of the container: The presented certificate is bound to the container's unique Kubernetes key, thus, no one else is able to use it or show this certificate without proof of possession of this unique identifier.

More specifically, CASTOR suggests associating the TNDI-bound attestation key, utilised by the TNDE's Attestation Source for signing evidence, with the certificate. This association signifies that the TNDE can construct the Trustworthiness Claims (TCs) only when such a certificate is present, facilitating verification through a key restriction usage policy. This, in turn, ensures that the enclave has been launched correctly. Consequently, both the certificate and the attestation key act as evidence that the confidential container has been launched correctly (i.e., boot-up integrity); hence, the TNDE logic running in the enclave is the correct one. More details on this are described in D3.1 [7].

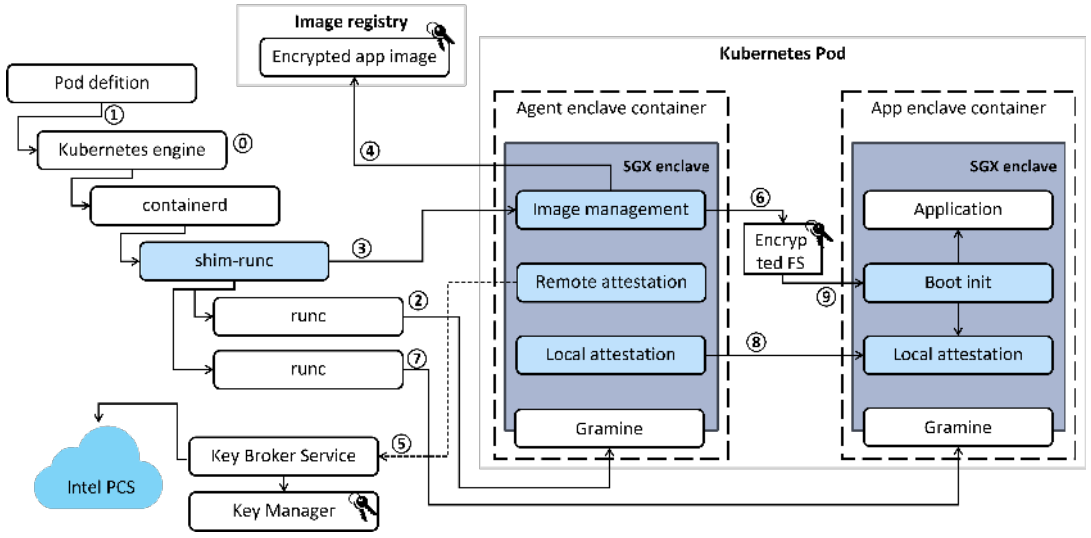

Figure 4.7: Enclave-CC architecture and flows.

4.3.3.3 The Architecture of Enclave-CC

Figure 4.7 depicts the architecture and the general flows of Enclave-CC coupled with Gramine and Intel SGX. In the figure, the blue-painted boxes are the components that are developed in the scope of the Enclave-CC project. Note that integration of Enclave-CC and Gramine is depicted in the figure, but it is currently an *ongoing effort*.

The main components in the Enclave-CC architecture are:

- **Agent enclave** – an Enclave-CC process that runs inside Gramine-SGX. It accepts requests from `shim-runc` and handles image management, SGX local and remote attestation, and encrypted file system management.
- **App enclave** – a user-application process that runs inside Gramine-SGX. It starts as a generic empty enclave and waits for instructions from the Agent enclave. When the Agent enclave has prepared all app-enclave data, the App enclave obtains the encrypted file system blob, decrypts it, and starts the execution of the client application.
- `shim-runc` – a standard shim component that sits between `containerd` and `runc`. It accepts requests from `containerd`, starts and pauses the Agent enclave container and asks the Agent enclave to perform image management actions. All containers started by `shim-runc` are instantiated by the traditional `runc` tool.
- **Image registry** – The Agent enclave container image and the encrypted App enclave container images are put in the image registry. Upon request from the Agent enclave container, the image registry releases the encrypted App enclave container image.
- **Key Broker Service (KBS)** – a service that verifies the trustworthiness of the Agent enclave, by performing SGX remote attestation with it and consulting the Intel PCS service to make sure the Agent enclave is a genuine SGX enclave executing the expected code. Key Broker Service also serves as a front-end to the Key Manager database that holds the encryption keys for encrypted application images.

The typical flow depicted in Figure 4.7 proceeds as follows. **(step 0)** The operator of the cluster installs the Enclave-CC run-time in the cluster. In particular, this installation copies `shim-runc` and `containerd` binaries into Kubernetes master nodes, adds image bundles of the Agent enclave container and the Boot

(empty generic) enclave container to the image registry, and re-configures `containerd` to enable Enclave-CC run-time. After the Enclave-CC run-time is successfully installed by the operator, the application workloads can be deployed.

To deploy an application in the Kubernetes cluster, the remote user defines a Pod configuration to describe the workload and run-time requirements **(step 1)**. The user deploys this Pod definition into Kubernetes and the request is propagated to `containerd`. The `shim-runc` tool receives the container creation request from `containerd` and creates the Agent enclave container using `runc` **(step 2)**. Next the `shim-runc` tool requests the Agent enclave to pull an encrypted application container image **(step 3)**. The Agent enclave receives this image pull request and downloads the encrypted application image from the image registry **(step 4)**.

Then the Agent enclave must make sure that the encrypted image is known to the system; to this end the Agent enclave performs an SGX remote attestation with the trusted Key Broker Service, which verifies the identity and trustworthiness of the Agent enclave and confirms the correctness of the downloaded image, as well as sends the encryption key for this image **(step 5)**. After remote attestation, the Agent enclave decrypts the downloaded application image, re-encrypts the application image with a randomly-generated ephemeral key and constructs a new encrypted file system out of this image **(step 6)**. In parallel to this process, `shim-runc` creates the Application enclave container and starts the Gramine instance, which performs the “empty generic enclave” boot-up **(step 7)**.

After the Application enclave is fully booted and initialised, it waits for the SGX local attestation request from the Agent enclave. During local attestation, the Agent enclave sends the application workload configuration, the path to the encrypted file system blob and the encryption key for this file system **(step 8)**. The Application enclave locates the encrypted file system, moves it inside the SGX enclave, decrypts it and installs it as the Gramine file system **(step 9)**. After this flow completes, the application can finally be run inside Gramine.

4.3.3.4 Life-Cycle Management of TEE-Enhanced Containers

In Section 4.3.3.1, we outlined how to create a TEE-enhanced container that is ready for deployment in a Kubernetes cloud infrastructure. We now describe considerations for the life-cycle management of a TEE-enhanced container (see Engineering Story-V). The core requirement is that the container implements features to support the lifecycle of a Kubernetes Pod [21]. If we assume that the original container (without TEE-enhancements) implemented these requirements, then our goal is that the recently added TEE does not break this implementation. In general, the container life-cycle is structured into these phases:

Pending: The Pod has been accepted by the Kubernetes cluster, but one or more of the containers has not been set up and made ready to run. This includes time a Pod spends waiting to be scheduled as well as the time spent downloading container images over the network.

Running: The Pod has been bound to a node, and all of the containers have been created. At least one container is still running, or is in the process of starting or restarting.

Succeeded: All containers in the Pod have terminated in success, and will not be restarted.

Failed: All containers in the Pod have terminated, and at least one container has terminated in failure. That is, the container either exited with non-zero status or was terminated by the system.

Unknown: For some reason the state of the Pod could not be obtained. This phase typically occurs due to an error in communicating with the node where the Pod should be running.”

By default, most containers are stateless while state is kept either outside the system (on a database server) or in a few well-defined containers that are part of a `StateFulSet`. The goal of this approach is

that it supports scaling (any number of containers can be started for a given container image to scale the workload. Similarly, without state, a `failed` container can be destroyed and restarted to try to recover the workload. Since a TEE only adds protection (and not state), the same holds for TEE-enhanced containers. They can be killed and restarted and - whatever signals the container received - they will automatically also affect the TEE that is part of this application.

4.4 Facility Layer

The facility layer is one of the central entities of the CASTOR architecture and facilitates the involved data handling along three main roles:

- It retains central and enhanced data storage capability with which information can be collected from CASTOR components, stored and shared among CASTOR components on demand.
- It includes a component named Topology Graph Composition Engine which is responsible for synthesizing and retaining an enriched dynamic graph of the network. In this direction, the networking graph is further enhanced with trust data, statistics, alarms and Path Profiles/SSLA related information.
- It acts as a data broker among the castor components enabling fast and reliable exchange of data between CASTOR components.

In this direction, the Facility Layer includes i) a set of data storage components capable of storing either static or dynamic data, ii) a dynamic component which synthesize these data into one dynamic enriched graph, and iii) a data broker component responsible to facilitate the communication among CASTOR components. In detail, Facility Layer includes the following distinct functionalities:

- a Traffic Engineering Database (TED) that stores detailed topology, link state, and constraint information (like bandwidth, latency) for path computation.
- a Management Information Base (MIB) that stores information about the capabilities and configuration (e.g., type of interfaces, cards etc) of network devices
- a Path Profiles/SSLA Database for storing information related with the path profiles established on the network and the related SSLA
- a Statistics/Alarms Database in which statistics and alarms are stored for future use. In the case of alarms, different types of alarms can be stores including: network alarms (e.g. a link is down), trust related alarms (e.g. ATL of a link is below the predefined RTL threshold), SSLA related alarms (e.g. an SSLA is violated due to low trust).
- a Topology Graph Composition Engine which combines the data from the aforementioned databases and enriches them with trust related information. In the generated Topology Graph the network topology is stored, together with all the related trust information. In addition, for each element of the topology references are retained toward the information stored in the aforementioned data structures.
- a Data Broker which enables the exchange of data between the CASTOR components through a publish/subscribe approach.

The Topology Graph Composition Engine and Data Broker components of Facility layer are explained in more detail below.

4.4.1 Topology Graph Composition Engine

The Topology Graph Composition Engine is instantiated per domain, therefore, each domain has its own topology graph. The topology graph is a dynamic and enriched data structure generated and retained in the Facility layer. It is synthesized with the actual network topology (physical and virtual) as the backbone, enhanced with trust related information generated by the CASTOR TAF trust evaluations (primarily from the Global TAF evaluations) and information collected from the monitoring layer. The data is reflected in the Topology Graph in the form of network- and trust-related attributes that may characterize any element in the topology: a node, a link, or an entire path. The data stored in the Topology Graph is dynamically updated and retained enabling all the CASTOR components to have access to fresh and rich data representation of the whole infrastructure including a) network elements and their capabilities/ configuration, status and statistics; b) trust status and statistics.

The Topology Graph Composition Engine is responsible to synthesize an enriched dynamic Topology Graph using data from different CASTOR components:

- Network related data is collected from the telemetry approach of CASTOR either directly through the Data broker (raw data) located in the Facility layer or from the Monitoring & Analytics layer (post-processing data collected as part of Section 4.5)
- Trust related data (ATLs) for the network elements are retrieved from the Global TAF or from the local TAFs through the Data broker.
- Path related information from the Traffic Engineering Policy Engine

The Topology Graph, since generated, is further exposed to other CASTOR components through an API provided by Topology Graph Composition Engine to facilitate their functionalities. In detail:

- Optimisation Engine requests information from the Topology Graph Creator of Facility layer related with network configuration, network status, TED information, established services and agreed SSLAs.
- PCE and Network Controller requests information from the Topology Graph Creator related with network configuration, network status and TED information.
- Network Service Orchestrator requests information from the Topology Graph Creator related with network topology (virtual topologies), established services and agreed SSLAs.

4.4.2 Data Broker

The Data Broker acts as an intermediate between the different CASTOR components enabling efficient and scalable event-driven communication between them. It serves as the central interconnect within the Orchestration Layer, linking its distinct layers of the architecture.

The Data Broker adopts a publish-subscribe (pub/sub) approach [1] for collecting and providing CASTOR information, organized under a schema of topics (of information). In detail, producers (publishers) generate data and publish to specific topics (channels), while consumers (subscribers) are registered to selected topics in order to receive related data. This approach is essential because single data sources are often required by multiple entities simultaneously. Furthermore, these communication flows are typically heterogeneous and asynchronous. To facilitate this, CASTOR explores event-based streaming mechanisms (e.g., Apache Kafka) to realize these high-volume information flows.

For example, the TE Policy Engine of the Orchestration layer could register to a dedicated topic, namely “castor.alarms”, in order to receive any SSLA violations alarms. In this case, the alarms are generated by the SSLA Monitor of the Monitoring and Analytics layer and published in the same topic.

The main information flows that are identified are the following:

- Telemetry data from various streams. In detail, NSO’s telemetry framework is collecting network and trust data on Pod level. The Network Controller is collecting data on vRouter level;
- Data models (e.g., NIB/Inventory/TED) generated in the Data Monitor;
- Network topology graph generated in NSO and Network Controller;
- Information and alarms generated in different Monitoring & Analytics services. In detail, SSLA Monitor is generating SSLA Alarms; HL Alarm Generator is generating network and trust alarms as an outcome of the Analytics processes;
- Coordination between the TE Policy Engine and the network enforcement components: Network Controller, PCE, NSO .

The exact details on the integration of the relevant artifacts of the Orchestration layer and the envisioned topics for achieving event-driven capabilities are presented in D6.1 [6]. An indicative list of the main actors interacting with the Data Broker are the following:

- Topology Graph Composition Engine
- Network Controller
- NSO
- SSLA Monitor
- HL Alarm Generator
- TE Policy Engine
- Global TAF
- PCE

4.5 Monitoring & Analytics Layer

Monitoring and analytics capabilities in modern networks often rely on large scale heterogeneous infrastructure in multi-tenant environment. Its role is to provide continuous and accurate visibility into the operational state, performance characteristics, and trust posture of the underlying infrastructure, network functions, and deployed services. The Monitoring & Analytics Layer ensures that the monitoring capabilities remain scalable and resilient to dynamic changes in the underlying infrastructure, enabling associated entities to make informed, data-driven decisions. The monitoring data continuously feeds control logic at the Orchestration Layer and enables detection of SLA/SSLA deviations, triggering of remediation or fallback actions, support for reconfiguration and traffic re-routing decisions, and continuous validation of service intent fulfilment.

Such monitoring and analytics capabilities may involve a combination of pull-based (allowing systems to periodically retrieve metrics from agents deployed on network elements), push-based (enabling network

elements to proactively stream measurements or alerts when predefined conditions are met), and event-driven (capturing discrete events such as failures, threshold violations, or state changes, enabling rapid detection of anomalies) telemetry mechanisms to ensure scalability. These mechanisms may coexist to support diverse operational requirements, from long-term trend analysis to near-real-time reaction.

The telemetry data collected from heterogeneous sources may be highly diverse in format, granularity, and semantics. The Monitoring & Analytics Layer therefore should include mechanisms for normalization of data into common schemas and data models, aggregation across time windows, network segments, or service instances, correlation of metrics originating from different layers (infrastructure, network, service), and contextualization of measurements with respect to services and service providers. This operation targets at transforming data into actionable monitoring intelligence consumable by the other layers.

A significant part of the operation of the Monitoring & Analytics Layer is to expose monitoring data through defined APIs to external authorized entities. Through these APIs, the authorized entities can retrieve real-time and historical metrics, subscribe to events and threshold violations, access service-specific monitoring views and correlate performance data with network topology and policies.

The CASTOR ecosystem encompasses both physical and virtual routers (vRouters) deployed in containerized environment. So, the Monitoring & Analytics Layer should support continuous observation of network, compute, and service resources, collection of performance, availability, and reliability metrics, exposure of telemetry and event data to Orchestration Layer, support for SLA and SSLA compliance monitoring, provision of monitoring evidence to necessary remediation actions and enablement of closed-loop control and zero-touch network management. The CASTOR Monitoring & Analytics Layer consists of the following entities: Data Monitor, SSLA Monitor, High Level (HL) Alarm Generator. Each entity fulfils a well-defined role within the monitoring pipeline.

The main responsibility of the Data Monitor is collecting telemetry data and event-based information from the Data broker and metrics via pull-based mechanisms from network elements, compute nodes, and deployed services, and normalizing raw data into specific schemas and formats. The telemetry data (metrics, events) and low-level alarms, include information concerning:

- network operation (from network entities as virtual routers, gateways, and service function chain (SFC) elements), such as latency, jitter, packet loss, throughput, interface counters (packets/bytes in–out, drops, errors), error rates, congestion indicators, routing and forwarding state (e.g., adjacency status, path changes);
- infrastructure resources, such as CPU, memory and storage utilization of physical routers, vRouter resource and compute metrics (corresponding to pod/container metrics);
- end-to-end service provision, such as end-to-end latency, service availability, throughput stability, application response times and error rate;
- trust, such as the evaluations of the overall CASTOR Trust Assessment Framework or even Trust Source information such as attestation and integrity status of nodes, security posture indicators, provenance and authenticity metadata;
- alarms, such as failed VNF onboarding, attestation failure.

Telemetry data is enriched with contextual metadata to enable correlation across abstraction levels, including service and service-instance identifiers, tenant identifiers, topology and placement information, and precise timestamps for time alignment.

The SSLA Monitor is responsible for evaluating monitoring data from the Data Monitor against predefined SLA and SSLA parameters regarding performance, availability, and trust-related objectives retrieved from Services/SSLA database of the Facility Layer. When SLA/SSLA compliance indicators exhibit deviations

outside the predefined evaluation window, these indicators are propagated to the Traffic Engineering Policy Engine (see Section 4.3.1), in order to trigger remediation or fallback actions.

The enhancement of the available fresh domain-level information with granular trust insights enables advanced data processing and intelligence capabilities on top of the collected monitoring data from Data Monitor and the SLA/SSLA compliance data from the SSLA Monitor. For example, the extraction of patterns, trends, correlations, and predictions that go beyond simple threshold-based monitoring could potentially provide proactive capabilities to the (network and trust) policies that can be shaped by the TE Policy Engine (see Section 4.3.1 on TE Policy decision). Such enhanced information could provide correlation of metrics across multiple abstraction levels, identification of performance degradation patterns, root-cause analysis of detected anomalies, result analysis and early warning detection, and forwarding of enriched monitoring data to Orchestration Layer. Essentially, this enables the Monitoring & Analytics Layer to leverage the core CASTOR framework in order to transition from reactive observation to proactive and predictive assurance.

The High-Level Alarm Generator is responsible for transforming monitoring and analytics results into actionable, service-aware alarms suitable for consumption by Orchestration Layer. Unlike low-level alerts, high-level alarms provide contextualized and aggregated notifications that reflect the actual impact on services. Its responsibilities include aggregation of multiple low-level events into single high-level alarms, contextualization of alarms with service, tenant, and domain information, classification of alarms based on severity and impact, filtering and prioritization of alarm notifications. The HL alarms may indicate that fallback choices and remediation actions failed to satisfy the requirements, necessitating the renegotiation of the SLA/SSLA terms between Service and Domain Provider.

CASTOR architecture leverages established protocols/mechanisms and interfaces such as SNMP, RESTCONF, or vendor-specific and domain-specific collectors for collection of information related to network flow characteristics, traffic engineering (TE) and advanced routing behaviour. Furthermore, Prometheus is adopted as as the primary mechanism for pod-level observability [32]. Prometheus is an open-source, cloud-native monitoring and alerting system, widely adopted for the collection, storage, and processing of telemetry data. It is designed to operate in highly dynamic and distributed environments characterized by frequent changes in topology and workload placement, and is suitable for Kubernetes-based infrastructures. Prometheus provides a unified mechanism for collecting time-series telemetry data, where each metric is uniquely identified by a metric name describing the measured quantity and a set of key–value labels, that capture the context under which the measurement was taken, such as the monitored entity, the service instance, the network interface, or the SLA profile. So, this model enables fine-grained per-service, per-instance, per-interface, or per-path observability, efficient aggregation and filtering operations, and correlation of telemetry data with orchestration constructs (e.g., pods, worker nodes/servers, network services). Telemetry endpoints exposed by monitored entities are dynamically discovered and scraped, allowing the Monitoring & Analytics Layer to automatically adapt to the instantiation, scaling, and decommissioning of resources. Specifically, Prometheus follows a pull-based telemetry collection model, in which it periodically scrapes metrics exposed by monitored entities via HTTP(S)-based endpoints, using the Prometheus exposition format. Collected metrics are stored as time-series data and can be queried using a query language, enabling real-time monitoring, historical analysis, and alert generation. In CASTOR, Prometheus is further extended beyond conventional resource monitoring to incorporate trust-related telemetry, such as attestation evidence, and security posture indicators exposed by the TNDE. In this way, Prometheus acts as a common and extensible anchor for operational and trust-aware telemetry, while co-existing with complementary mechanisms that address network-specific and TE-related monitoring requirements

The Prometheus monitoring ecosystem consists of multiple components (many of which are optional), as there are depicted in Figure 4.8 from the official site of Prometheus. To facilitate a clearer understanding of the fundamental architectural components in the figure, we clarify that in Prometheus terms [32], an endpoint you can scrape is called an instance, usually corresponding to a single process. A collection of instances with the same purpose, for example a process replicated for scalability or reliability, is called a

job.

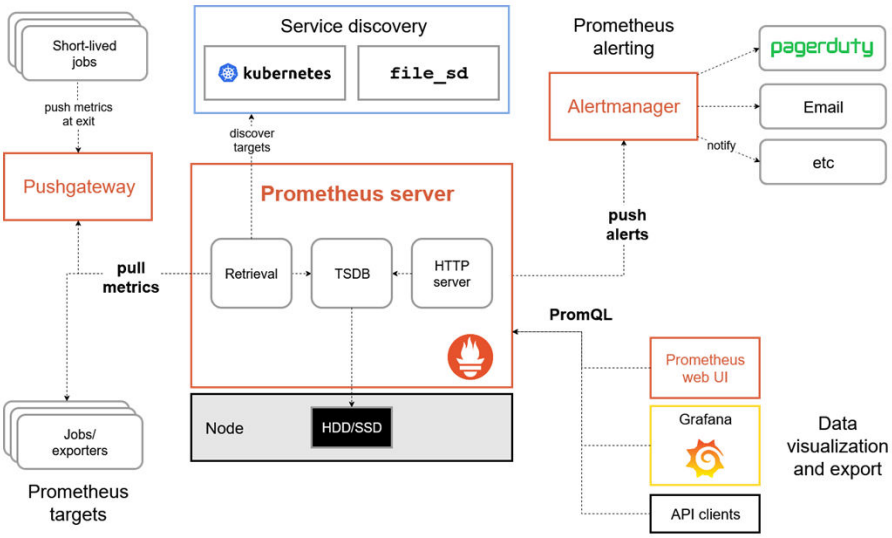

Figure 4.8: Architecture overview of the system (source: official website, reproduced as-is)

The core components of Prometheus include:

- Prometheus Server, which is responsible for scraping telemetry data (Retrieval), storing time-series metrics (Time-Series Database -TSDB), and evaluating alerting rules
- Exporters, which expose metrics from monitored systems in a Prometheus-compatible format (act as adapters between monitored components and Prometheus).
- Service Discovery mechanisms, enabling Prometheus to dynamically discover monitoring targets in cloud-native environments.
- Alerting and Event Generation mechanisms, which trigger notifications/alerts based on predefined rules (thresholds or conditions). When a rule is triggered, an alert is generated and is forwarded to the Alertmanager component. These mechanisms allow Prometheus to act as a real-time event source for fault detection, performance degradation, and policy or SLA violations.
- Query and visualization interfaces, allowing operators and higher-layer systems to access telemetry data. Grafana or other API consumers can be used to visualize the collected data [18] . Prometheus provides PromQL, a query language that supports aggregations, temporal analysis (rate, increase, over time windows), and label-based filtering.

This modular architecture allows Prometheus to act as a centralized telemetry aggregation point while maintaining loose coupling with monitored entities.

In Kubernetes-based infrastructures, the Prometheus Server runs as a Pod (or a set of Pods for high availability) and leverages Kubernetes service discovery to automatically identify and monitor workloads, nodes, and infrastructure components. Kubernetes annotations and labels are used to expose telemetry endpoints, enabling Prometheus to dynamically adapt to changes in the cluster topology. As new Pods or worker nodes are instantiated, Prometheus automatically discovers them and begins scraping their metrics, while removed resources are seamlessly de-registered.

In CASTOR architecture, virtual routers will be implemented using IOS XRd and deployed as containerized workloads within a Kubernetes cluster. Each IOS XRd instance operates as a vRouter, providing

advanced routing and forwarding capabilities while being fully managed by Kubernetes. Prometheus integrates with IOS XRd–based vRouters through telemetry exporters. These exporters are used to expose pod-level execution-environment information, providing visibility into the operational status and resource utilization of IOS XRd instances. They translate pod-related telemetry into Prometheus-compliant metrics and make them available through an HTTP(S) endpoint, while detailed network flow and traffic engineering telemetry, rely on existing data collection mechanisms. The exporters can be deployed either as sidecar containers co-located with the IOS XRd Pods or as external components interfacing with IOS XRd through standardized telemetry interfaces (e.g., gRPC, NETCONF).

Prometheus, complementary to existing telemetry mechanisms, can collect a wide range of metrics from IOS XRd vRouters, including but not limited to Control Plane Metrics (e.g. number of active routing adjacencies) data plane and interface metrics (e.g. interface throughput (bytes/packets in/out)), system and resource metrics (e.g. CPU and memory utilization of the IOS XRd instance). As previously stated, the Prometheus monitoring framework can be, also, extended to support the collection and evaluation of trust-related telemetry. Within this context, IOS XRd is treated also as a trust-observable network function, whose operational state and security-relevant attributes can be assessed and correlated with service-level objectives. These metrics obtained from TNDE (Local TAF) (e.g. image integrity status, configuration consistency indicators, authentication status of routing adjacencies) can be exposed through telemetry exporters in a Prometheus-compatible format, allowing them to be collected and evaluated by applying rule-based expressions over trust-related metrics. Thresholds, temporal patterns, and composite conditions can be defined to identify deviations from expected trust postures. This allows the monitoring framework to detect early signs of misconfiguration, degradation, or potential security incidents.

Telemetry data collected from vRouters can be correlated with higher-level service instances, allowing the monitoring framework to assess the operational state of network services instantiated by the Network Service Orchestrator. Furthermore, alerting rules in Prometheus can be defined to detect deviations from expected performance or operational conditions, triggering notifications or automated remediation workflows (functionalities covered by the Data Monitor and HL Alarm Generator as elaborated in D5.2 [8]). Prometheus-generated alerts and metrics can be consumed by the Network Service Orchestrator to initiate corrective actions, such as reconfiguration, scaling, or re-instantiation of vRouter instances.

Chapter 5

Blockchain-based Trust Model for Inter- and Intra-Domain Path Establishment

An additional core capability of CASTOR lies in the auditability of all information upon which trust-aware path establishment is achieved in both inter- and intra-domain topologies. This essentially translates to the secure management of the evidence, extracted from all routing elements, based on which the trust characterization of both **node- and link/path-level trust** is calculated and dynamically updated - reacting to any changes in the properties defined in the overall trust model such as *router integrity, communication resilience and availability, etc.* As detailed in D3.1 [7] and D4.1 [5], two are the core dimensions that need to be accounted for such a continuous trust assessment process: the **verifiability** of all trust sources and trustworthiness evidence, based on which the evaluation of the Actual Trust Level (ATL) of each node and path is calculated and reasoned against the already defined Required Trust Level (RTL) per trust profile, and the **privacy-preserving sharing of trustworthiness claims** between domains that may be managed by different network operators.

The former is manifested by the integration of the CASTOR Trusted Network Device Extensions (TNDE) to each onboarded routing element providing the necessary tracing capabilities and attestation services for the secure monitoring and reporting of the router's state - defined as **raw traces** that are monitored by the employed eBPF monitoring hooks and are then consumed by the CASTOR Attestation and FSM agents producing trustworthiness evidence. Such evidence include rich information on both static (e.g., configuration integrity of the launched SW stack) and volatile (e.g., integrity of the update process of routing structures such as *nftables, link-state tables, etc.*) properties of router's behaviour and their verifiability during extraction is guaranteed by the underlying CASTOR Trusted Computing Base. This unlocks (both the local and global) Trust Assessment Framework (TAF) agents producing **Trustworthiness Claims (TCs)** that can further allow the resolution of any topology-specific trust dependencies in an attempt to guarantee the compliance of the agreed (S)SLAs. The data structure of the TCs will be based on the (IETF-adopted) Yet Another Next Generation (YANG) data model as part of the exchange of trust-related information between adjacent routing nodes. They will be extended to also capture they more dynamic information needed for depicting the runtime state of all routers.

The latter, on the other hand, translates to the requirement of preserving the privacy of the trust route configurations already established within a domain. In the case of intra-domain path establishment, while trust evaluation necessitates the exchange of TCs between BGP nodes, this must not leak any sensitive information of the comprising routers and their SW/HW co-design. Hence, CASTOR employs advanced crypto protocols including order-revealing encryption [7] to allow for the advertisement of the minimum trust level guarantees that can be achieved by a domain but without sharing any more information. However, since trust-aware traffic engineering decisions rely on the continuous management of tracing/attestation data, trustworthiness claims, policies and (S)SLAs, it is essential that they remain tamper-resistant, traceable, and verifiable over time. To this end, **CASTOR leverages a Distributed Ledger Technology (DLT) for auditability for provenance across administrative domains.**

However, auditability alone is not sufficient. While the DLT guarantees immutability and provenance of stored data, it does not inherently guarantee the correctness or veracity of the information being recorded. In other words, a ledger can prove that data has not been altered after its (on-chain) storage, but it cannot determine whether the data was truthful or valid at the time of its submission to the Blockchain node (i.e., bridge for orchestrating the execution of the data management transactions). This distinction introduces a fundamental challenge: *how to reliably exchange and consume trust-related evidence without blindly assuming its validity*? This is where the concept of a secure oracle becomes essential.

In the context of CASTOR, a secure oracle acts as a trusted mediation and verification mechanism that enables the secure ingestion and exchange of trust-relevant information between domains. It serves as a controlled "rendezvous" point where evidence—such as remote attestation measurements, trustworthiness claims, or (S)SLAs—is validated before being anchored on the DLT or consumed by CASTOR components. In what follows, we elaborate on the state-of-the-art in secure oracle mechanisms, and then converge on the technology adopted in CASTOR. We also capture all detailed operational through which trust artifacts are validated, recorded, and exchanged across domains. Innovation of CASTOR lies on the establishment of a **trust exposure layer**, that can enable transferability of trust relationships between multiple domains (over which service graph chains will be established), while accounting for the necessary level of abstraction/harmonization so as to obfuscate any sensitive information between the communicating domains.

5.1 Trustworthy Blockchain Oracles for Securing Zero-Touch Networks

The evolution of decentralized systems has reached a point where the utility of a Blockchain is no longer defined solely by its internal ledger, but also by its ability to interact with the world around it. At the heart of this interaction lies the Oracle, a specialized middleware layer designed to bridge the gap between deterministic Blockchain environments and the non-deterministic, external world [2]. In general, an oracle acts as a bridge between on-chain logic and external data sources. The mass adoption of Blockchains and smart contracts has raised a great need for fetching real-world data to reside on-chain to carry out all the required computations. By their nature, Blockchains are "closed-loop" systems, as they cannot natively access external APIs, web servers, or IoT devices because doing so would break the consensus that requires every node to reach the same result. The Oracles, being an off-chain technique, allow a wide degree of cross-communication across Blockchains and enterprise systems and have succeeded in bringing external information to the Blockchain for smart contract execution.

However, this bridging function introduces a fundamental trust challenge: Blockchains provide integrity and immutability guarantees only for data that has already been committed to the ledger, but they cannot inherently verify the correctness or authenticity of externally sourced information. More precisely, traditional Oracle solutions often introduce a "single point of failure" where the data provider or the transmission layer could be compromised. This gave rise to the concept of Secure Oracles. **A Secure Oracle goes beyond mere data transmission by providing verifiable veracity** [36] [14]. By utilizing advanced technologies, Secure Oracles ensure: (a) Data Integrity: The data is not tampered with during its journey from the source to the ledger, (b) Confidentiality: Sensitive data (like private API keys or personal identifiers) remains encrypted even while being processed by the oracle, and (c) Attestation: The Blockchain can cryptographically verify that the oracle logic was executed exactly as programmed, within a secure hardware enclave. By integrating a Secure Oracle layer, a Blockchain infrastructure can be transformed from a simple transaction log into a **reactive engine that can execute complex business logic** based on high-integrity data.

To address this limitation, various secure oracle architectures have been proposed. At this moment, the following top-notch Secure Oracle technologies have been considered for integration into the CASTOR

Blockchain Infrastructure:

Chainlink:[1] It leverages decentralized oracle networks with hundreds of independent nodes, making it nearly impossible to corrupt. It can verify private data without revealing the data itself using Zero-Knowledge Proofs, while it enables secure cross-chain data movement. However, the ”Push” model and decentralized consensus make it the most expensive and slowest to update during high volatility, as well as it is difficult to run complex custom computations inside a standard Chainlink node.

PHALA:[2] It is a TEE-based oracle and confidential computation framework that initially relied on Intel SGX enclaves to execute off-chain logic while anchoring results on-chain. Through Phat contracts, the platform enabled general-purpose off-chain applications rather than simple data fetching. Over time, the architecture has evolved toward more modular confidential computing models, expanding beyond enclave-based execution to include virtual machine–level isolation (e.g., Intel TDX), emerging GPU-based TEEs, and so-called microprocesses in microprocessors—a term used in recent PHALA literature to describe fine-grained, isolated execution components that decouple coordination, execution, and networking responsibilities. These changes improve flexibility, scalability, and resilience against certain hardware-level attacks while broadening the range of supported workloads. However, deploying and developing such applications still requires specialized hardware support and familiarity with Rust and TEE programming environments, making integration more complex compared to lightweight oracle solutions.

Pyth:[3] Data comes from first-party publishers, eliminating the risk of ”middleman” manipulation. However, it cannot process private enterprise data within a secure enclave and while decentralized, it is highly based on trusting the accuracy of the large financial firms that publish data.

RedStone:[4] It uses a ”Pull” model where data is only put on-chain when a user transaction triggers it. On the contrary, the fact that it relies on Arweave for data storage adds another platform dependency.

Chronicle:[5] It uses Schnorr signatures to reduce the cost of on-chain verification, while every data point can be traced back to its specific voter through a public dashboard. However, it is primarily designed for price feeds, lacking general-purpose computation capabilities.

Town Crier:[6] By using Intel SGX, Town Crier provides cryptographic proof (attestation) that the data has been witnessed by a secure CPU from a specific TLS-certified source. Nevertheless, over the years, SGX became the target of numerous side-channel attacks, effectively breaking the promise of total confidentiality. Also, it has been designed as a centralized proxy, meaning that while it proved data veracity, it lacked network veracity. Table 5.1 summarizes the capabilities of these Secure Oracle technologies is presented below.

Feature	Town Crier	Chainlink	PHALA	Pyth	RedStone	Chronicle
Trust Root	Hardware (Intel SGX)	Math (ZKP/MPC)	Hardware (TDX)	First-Party Source	Modular Storage	Reputation/ Quorum
Veracity Layer	Hardware Proof	Consensus-based	Hardware Proof	Source-based	Signature-based	Reputation-based
Compute Complexity	Minimal (Scraping)	Low (Scripts)	High (Full Apps)	Minimal	Medium	Low
Privacy (In-Use)	High (Enclave)	Via ZKP (Slow)	Max (VM-level)	None	Minimal	None
Latency	Medium	Medium	Low	Lowest	Low	Medium

[1] https://chain.link/
[2] https://PHALA.com/
[3] https://www.pyth.network/
[4] https://www.redstone.finance/
[5] https://chroniclelabs.org/
[6] https://github.com/bl4ck5un/Town-Crier

Feature	Town Crier	Chainlink	PHALA	Pyth	RedStone	Chronicle
Attack Resilience	Low (Side-channels)	Network effects	TDX Isolation	Redundancy	Storage history	Transparency

Table 5.1: Comparative analysis of Secure Oracle technologies

5.1.1 CASTOR Design Choice of PHALA Compatible Blockchain Model

The secure oracle solutions presented above share a common set of fundamental capabilities, including verifiable execution, privacy-in-use, protection of sensitive credentials such as API keys, and in some cases chain-agnostic operation. The main differentiating factor between these systems is not the existence of these guarantees, but rather their agility and native compatibility — that is, how easily the oracle logic, cryptographic configuration, and operational policies can evolve over time without requiring architectural redesign.

For instance, traditional secure oracle designs like Town Crier are very flexible regarding attestation evidence but are not very adaptable when it comes to introducing new cryptographic primitives or new data sources. In environments such as CASTOR, where trust policies, telemetry sources, and verification mechanisms may evolve dynamically across administrative domains, the ability to modify configuration and execution logic without redesigning the infrastructure becomes a primary requirement.

PHALA was selected as the conceptual foundation because its architecture is designed around general-purpose trusted execution rather than fixed data delivery semantics. It supports confidential computation, programmable off-chain logic, and chain-agnostic integration, enabling flexible evolution of oracle behaviour. However, PHALA itself is currently undergoing a technological transition: earlier implementations relied on SGX-based workers and Phat contracts, while newer specifications are evolving toward broader confidential computing environments (e.g., TDX and heterogeneous trusted execution accelerators). As a result, the official PHALA system is in a transitional stage where legacy implementations are deprecated and the new execution specifications are still under active development.

For this reason, CASTOR adopts a PHALA-compatible design rather than directly depending on a specific PHALA deployment. CASTOR implements its own PHALA-like Blockchain ecosystem that preserves the core properties of the model — verifiable execution, confidential processing, programmable oracle logic, and chain-agnostic interaction — while remaining independent of a specific underlying TEE technology. This approach provides native compatibility and architectural flexibility without fragmentation or dependence on hardware-specific assumptions.

The detailed mapping between PHALA's architectural concepts and the CASTOR implementation is described in the following sections, including both the evolution of PHALA's architecture and the design of the custom CASTOR system. When the official PHALA specifications stabilize, the CASTOR system is designed to align with them and interoperate with minimal adaptation, as it already follows the same execution and trust principles.

5.2 PHALA Blockchain Infrastructure

As aforementioned, PHALA is the best option to adopt in CASTOR. However, PHALA Network has witnessed a remarkable shift in its architecture between its pre-migration form and its post-migration form. Although both systems share a similar foundational principle in which sensitive computations are executed inside Trusted Execution Environments (TEEs) and anchored on-chain, there have been significant shifts in the role of core components, assumptions, and developer interfaces between the two architectures. This has a direct impact on the replication, extension, or analysis of similar systems as of today.

The pre-migration implementation is now deprecated and no longer supported as a buildable reference system, while the post-migration architecture does not yet provide a publicly available open-source implementation, motivating the **need for a custom PHALA-like solution**.

5.2.1 Pre-migration PHALA Architecture

In its pre-migration architecture, PHALA had employed a set of tightly coupled and specialized roles that, in combination, defined the backbone of its confidential computation model. At the heart of the design was the presence of Gatekeepers, which functioned as privileged TEEs that were critical to the management of keys, workers, and the coordination of secure computation. Gatekeepers were responsible for the sharing of secrets, the onboarding of workers, and the coordination of trust within the network.

In addition to Gatekeepers, the network of workers functioned to execute computation within SGX environments and communicate directly with the on-chain ledger. In the original architecture, the Blockchain functioned to manage the registration of workers, accounting, routing of messages, and the persistence of computation results. Although the original architecture of PHALA was instrumental in the provision of robust confidentiality and verifiability, it was associated with considerable operational complexity, which, in effect, meant that the extension of the architecture beyond its original scope proved to be challenging.

5.2.2 Post-migration PHALA Architecture and Evolution

After the migration, PHALA's architecture has continued to develop towards a more streamlined and modular form. Specifically, PHALA's post-migration architecture has moved away from privileged coordination roles such as Gatekeepers and has placed more emphasis on achieving a clearer distinction between on-chain coordination and off-chain execution. In addition, critical management and orchestration aspects of PHALA's architecture have been streamlined, bringing PHALA's architecture even closer to a generalized form of "TEE-backed off-chain execution with on-chain settlement."

This evolution in PHALA's architecture has been necessary to reduce operational complexity, to make PHALA more accessible to developers, and to allow for more flexibility in how PHALA's TEE-based computation can be used in conjunction with different runtime environments. Although PHALA's post-migration architecture represents the current state of PHALA's development trajectory, PHALA's pre-migration architecture is currently considered deprecated and is no longer maintained as a buildable or extensible reference architecture. PHALA's post-migration architecture has yet to deliver a publicly available experimental framework/reference implementation.

5.2.3 CASTOR PHALA-alike Blockchain System Model

In order to overcome the disadvantages of the deprecated pre-migration architecture and the currently unavailable post-migration reference implementation, we envision to develop and implement **a custom "PHALA-like" Blockchain architecture** that meets the needs to support the trusted path routing. The purpose of this architecture is to inherit the key characteristics of the PHALA execution model, which include trusted execution in SGX enclaves, coordination on the Blockchain, and verifiable commitments, but with a simpler architecture, as inspired by the evolution of the PHALA post-migration architecture.

The CASTOR PHALA Blockchain system is based on the Besu ledger, which is used for distributed coordination and persistence, and makes use of a small set of well-defined roles for the enclaves, which are dedicated to different types of operations. In CASTOR the following three enclaves are used.

1. **Oracle Worker (Enclave A):** The Oracle Worker follows an event-driven execution paradigm. It listens to particular on-chain events triggered by oracle trigger contracts and responds to them in-

dependently. Upon the reception of a trigger, the Oracle Worker fetches the assessment information from trusted external sources, which in this case is modeled by the Local Trust Assessment Framework (TAF). The fetched information is validated for authenticity and integrity, processed based on application-defined logic, and finally stored back to the ledger as signed output. This worker is mainly accountable for initializing trust-related entries on-chain, namely local trust assessments.

2. **Computation Worker (Enclave B):** The Computation Worker has an invocation-driven execution paradigm. It is invoked via the Security Context Broker (SCB) and not through on-chain events. The Computation Worker accepts invocation inputs, validates them, securely computes or aggregates them, and writes the signed results to the ledger. This role is normally used for enriching or updating existing on-chain data, such as storing global trust evaluations from Global TAF, Trust Policies or writing SSLA compliance outcomes. The decoupling of this role from the Oracle Worker enables the system to have flexible client-driven workflows that are not event-driven.

3. **(Optional) Data Exposure / Query Worker (Enclave C):** In addition to the execution-oriented workers, CASTOR's architecture optionally includes a specialized Data Exposure or Query Worker. The purpose of this worker is to manage sensitive read operations and apply privacy-preserving transformations to data before returning it to consumers. While the current version of this system offers query-time transformation and access control at the SCB level, this worker offers a future direction for performing sensitive query processing within a trusted execution environment. This will enable raw on-chain data to be retrieved and transformed within the enclave, improving the overall security of query response data.

Aside from the enclave-based workflow, the system also has a direct evidence anchoring path, whereby the SCB directly commits raw trace evidence to the ledger without the use of any enclaves. This is done when no confidential computation is necessary, and the primary goal is to achieve auditability, timestamping, and immutability of raw data.

The CASTOR PHALA Blockchain system is therefore a system that encapsulates the lessons learned from the architectural evolution of PHALA. It overcomes the need for privileged coordination in the baseline execution path for gatekeepers while offering strong integrity properties via enclave-signed commitment schemes. In this architecture, the role of Gatekeeper is limited to administrative governance, such as populating and enforcing policies in the Blockchain's Enclave Registry. It is separate from orchestration, trusted execution, governance, and persistence concerns. This architecture is feasible to implement and deploy in the present day while offering a strong basis for experimentation and research in PHALA-like TEE-based computation without depending on deprecated implementations or unavailable post-migration reference code.

5.3 CASTOR Blockchain-based Trust-Aware Path Establishment

This section presents the high-level CASTOR DLT architecture (see Figure 5.1) and highlights the nine supported workflows, comprising five storage flows and four query flows. At the top of the figure, the Off-chain Storage component is depicted, while the bottom part illustrates the three secure enclaves, namely the Oracle Worker, the Computation Worker, and the Data Exposure/Query Worker. The CASTOR DLT is positioned centrally, acting as the core trust anchor that interconnects all architectural components.

CASTOR DLT components are encapsulated by the Security Context Broker, which enforces secure interaction and controlled data exposure across the system. The gray-shaded boxes denote the three supported layers of access control, each applied at different stages of data storage and retrieval. The entities interacting with the CASTOR DLT include components of the Router Element and the Orchestration Layer, as well as administrative and external actors, such as an Administrator and External Entities (e.g.,

a Certification Authority or a device manufacturer). For External Entities, a dedicated Trust Exposure API is provided, as highlighted by the red box on the left of Figure 5.1. Last but not least, as already mentioned the Gatekeeper is responsible for administrative governance interacting with the Blockchain's Enclave Registry.

Table 5.2 provides a consolidated overview of the nine CASTOR flows, serving as an introductory reference before the detailed per-flow descriptions presented in the subsequent subsections. Numeric identifiers denote query flows, while Latin numerals and alphabetic labels denote storage flows. Color coding is used in both the Figure 5.1 and the Table 5.2 to distinguish flow categories: red indicates Trust Policy flows, purple corresponds to SSLA flows, dark blue represents Raw Trace flows, light blue denotes Trustworthiness Claim flows, and black indicates queries issued by External Entities (e.g., queries related to trust characteristics, raw traces, and SSLAs).

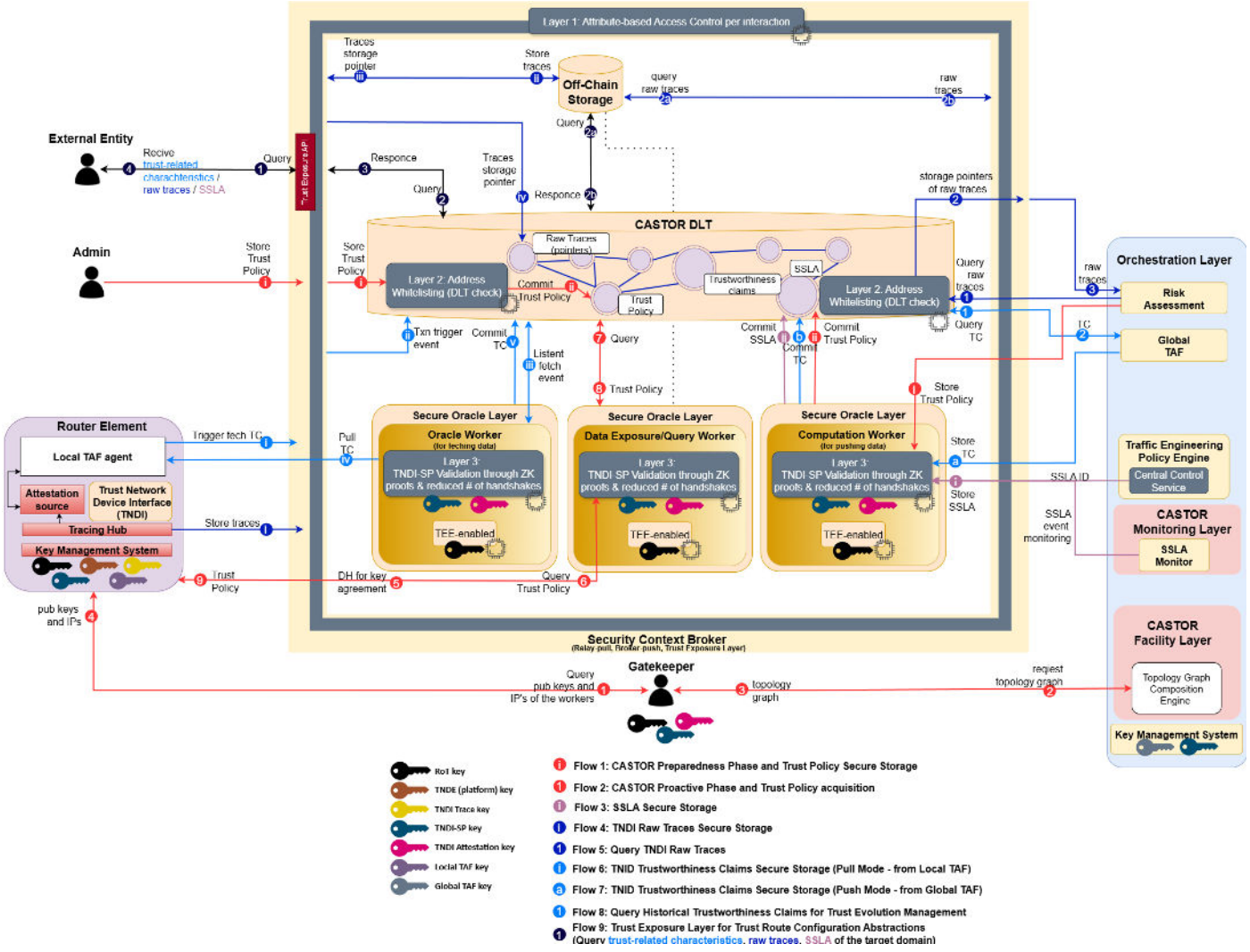

Figure 5.1: CASTOR DLT High-Level Architecture

Table 5.2: Smart Contracts and High-Level Interaction Flows

Flow	Description	Related SC flow
Flow 1: CASTOR Preparedness Phase and Trust Policy Secure Storage	During this flow either the **Admin** or the **Risk Assessment** component push and dynamically update the Trust Policy to the **CASTOR DLT**.	WorkerRegistry; TrustPolicyRegistry

Flow	Description	Related SC
Flow 2: CASTOR Proactive Phase and Trust Policy acquisition	During this flow the **Router Element** is in the process to be registered to the **CASTOR DLT**. The flow depicts all the steps from the acquisition of the registered worker public keys and IPs, to the verification of the router inclusion in the topology graph, to the establishment of the TNDI-SP session keys and finally the reception of the Trust Policy for its instantiation and enforcement in the **Router Element**. Trust policy querying will eventually be accommodated through a separate enclave (**Data Exposure/Query Worker**); however, in the first version of the CASTOR framework this process is performed through the **SCB**. **TNDI-SP keys** are session keys created during this flow and used for the secure interaction between the **Routing Element** and the active enclave/worker of the **CASTOR DLT**.	TrustPolicyRegistry
Flow 3: SSLA Secure Storage	The **SSLA Monitor** and the **Central Control Service** of the **Traffic Engineering Policy Engine** push the SSLA–related content to the **CASTOR DLT** directly. More specifically, the **SSLA Monitor** pushes information regarding the SSLA event monitoring and the Central Control Service pushes the SSLA ID field for supporting fallback mode.	WorkerRegistry; SSLACommit
Flow 4: TNDI Raw Traces Secure Storage	Raw traces generated by the **Tracing Hub** of the **Routing Element** are stored directly in CASTOR DLT. To ensure scalability, the trace data itself is stored off-chain, while only storage pointers are recorded on-chain. Each raw trace is digitally signed using the Tracing Hub's TNDI trace key, which is part of the TCB, thereby ensuring the integrity and authenticity of the traces.	RawEvidenceCommit; AccessControlManager (whitelisting)
Flow 5: Query TNDI Raw Traces	The raw evidence stored on the CASTOR DLT can be queried by an authorised entity. In particular, (a) the **Risk Assessment** component can query raw traces through Layer 2 (address whitelisting) to calculate a new risk index and RTL that may lead to a Trust Policy update, and (b) an authorised **External Entity** (e.g., Routing Element manufacturer or a CA) can query raw traces through Layer 1 (attribute-based access control). In both cases, the access flow is mediated by the SCB, which manages and coordinates the interaction with the off-chain storage.	RawEvidenceQuery; AccessControlManager (whitelisting)
Flow 6:TNDI Trustworthiness Claims Secure Storage (Pull Mode - from Local TAF)	The **Local TAF** deployed in the **Routing Element** produces new trustworthiness claims and notifies the **CASTOR DLT** to fetch them. The **Oracle Worker** is subscribed to these events and, upon notification, initiates the fetching process.	WorkerRegistry; OracleJobRequest; LocalTrustAssessment-Commit
Flow 7: TNDI Trustworthiness Claims Secure Storage (Push Mode - from Global TAF)	After the initialisation of the trustworthiness claims by the Local TAF, the **Global TAF** pushes its trustworthiness claims to the same object through the **Computation Worker** enclave.	WorkerRegistry; GlobalTrustAssessment-Commit
Flow 8: Query Historical Trustworthiness Claims for Trust Evolution Management	The **Global TAF** can query the historically stored trustworthiness claims from the **CASTOR DLT** to improve its computations.	TrustAssessmentQuery
Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions	An authorised **External Entity**, through the **Trust Exposure API**, can query trust-related characteristics in a privacy-aware manner using transformation functions. It can also query SSLAs and raw traces through the same interface.	RawEvidenceQuery; SSLAQuery; TrustAssessmentQuery; AccessControlManager (whitelisting)

5.3.1 Flow 1: CASTOR Preparedness Phase and Trust Policy Secure Storage

This flow is two-fold. It depicts the secure storage of a Trust Policy to the **CASTOR DLT** by the **Admin** and by the **Risk Assessment** component, since both can push and dynamically update the Trust Policy to the **CASTOR DLT**. The two parallel flows are described below:

i. The **Admin** requests to store or update the Trust Policy through the **Security Context Broker** and the **Layer 1: Attribute-based Access Control**. If the **Admin** is authorized, then the request is forwarded to the second layer of authentication, the **Layer 2: Address Whitelisting**.

ii. If the second authorisation is successful, the Trust Policy is committed to the **CASTOR DLT**.

i. The **Risk Assessment** requests to store or update the Trust Policy through the **Layer 3: TNDI-SP Validation** authorization of the **Computation Worker**.

ii. If the authorisation is successful, the **Computation Worker** commits the Trust Policy to the **CASTOR DLT**.

5.3.2 Flow 2: CASTOR Proactive Phase and Trust Policy acquisition

As already documented in D2.1 [4], the CASTOR proactive phase addresses all the aspects of the secure enrolment of a new network element (e.g., vRouter/Routing Element) in the topology: from the initial attestation of the underlying platform up to the provisioning of the necessary cryptographic material in order to interact with the existing network topology and establish communication links. The final step of the proactive phase requires the acquisition of all the necessary aspects in the form of Trust Policies that need to be enforced prior to its enrollment in the topology. In the following steps we will focus on the steps of the querying the necessary public keys and IPs of the workers and the Trust Policies from the **CASTOR DLT**, assuming that the TNDI attestation key bound to TNDE (platform) key is successfully created, the Orchestration Layer has already interacted with the vRouter requesting the necessary guarantees (e.g., attestation evidence) and the vRouter have successfully provided evidence that its functionality has been securely launched and all the CASTOR artifacts are deployed. The steps are the following:

1. The **Routing Element** requests from the **Gatekeeper** the public keys and IPs from all the active workers (e.g., the Oracle Worker, the Computation Worker and the Data Exposure/Query Worker), in order to be able later to mutual-authenticate with the CASTOR workers and establish a session key per worker.
2. The Gatekeeper requests the topology graph from the **Topology Graph Composition Engine**, in order to check if the specific router ID has already been onb-oarded on the specific domain.
3. The **Gatekeeper** receives the requested topology graph and checks if the router is already on-boarded on the topology graph or not.
4. If the router is successfully on-boarded on the topology graph, the **Gatekeeper** provides back the public keys (certificates) and the IPs. At this state, the Routing Element has received the validated, registered and non-revoked public keys of the active workers.
5. The **Routing Element** establishes a secure communication channel with all the workers including the Data Exposure/Query Worker using the Diffie Hellman protocol, resulting in a different TNDI-SP (session) key per worker.
6. The **Routing Element** queries the Trust Policy via the **Layer 3 TNDI-SP Validation** to the **CASTOR DLT**.

7. After the successful Layer 3 authentication, the worker forwards the query to the **CASTOR DLT**.
8. The **CASTOR DLT** provides the requested Trust Policy back to the worker.
9. The worker responds to the **Routing Element** with the Trust Policy and then instantiates the trust models in **Local TAF** and extracts the RTL. Then generates the trustworthiness claims that will be shared to the **Orchestrator Layer** as the next step towards joining the topology.

5.3.3 Flow 3: SSLA Secure Storage

This flow showcases the secure storage of the SSLA, which is divided into two parts. The **SSLA Monitor** stores the information regarding the SSLA event monitoring and the **Central Control Service** stores the SSLA ID field in order to support the fallback mode. The steps are the following:

i. The **SSLA Monitor** requests to store the information regarding the SSLA event monitoring and the **Central Control Service** requests to store the SSLA ID field in order to support the fallback mode via the **Layer 3 TNDI-SP Validation** of the **Computation Worker** to the **CASTOR DLT**.

ii. If the authorization is successful, the Computation Worker commits the SSLA to the **CASTOR DLT**.

5.3.4 Flow 4: TNDI Raw Traces Secure Storage

This flow depicts the secure storage of the raw evidence created by the Tracing Hub of the **Router Element** to the **CASTOR DLT**. CASTOR's architectural choice is to store these types of data to the off-chain storage and keep only the storage pointers on the chain. Such a decision is for performance reasons since raw evidence data are created frequently and are of high volume. In tandem, raw evidence is already signed with the TNDI Trace key for integrity. The flow is the following:

i. The **Tracing Hub** of the **Routing Element** generates the raw evidence, sings it with the **TNDI Trace key** and initiates the process to store them on the **CASTOR DLT** through the **Security Context Broker**, where a first layer of access control is performed.

ii. If the authorization is successful, the **Security Context Broker** forwards the raw traces to the **Off-Chain Storage**.

iii. The **Off-Chain Storage** returns back to the **Security Context Broker** the storage pointers in order to be stored on the chain.

iv. Finally, the **Security Context Broker** stores the storage pointers to the **CASTOR DLT**.

5.3.5 Flow 5: Query TNDI Raw Traces

This flow depicts the query of the stored raw traces.The raw traces can be queried by both the **Risk Assessment** component through the **Layer 2: Address Whitelisting** in order to calculate a new risk index and RTL that could lead to a Trust Policy update and by an external authorised entity (e.g., the vRouter Manufacturer or a CA) through the **Layer 1: Attribute-based Access Control**. In the following steps we will describe the former case, while the latter case will be described in the Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions along with the other queries form an external entity.

1. The **Risk Assessment** queries to receive the raw traces from the **CASTOR DLT**, through the **Layer 2: Address Whitelisting** access control.

2. If authorization is successful, the storage pointers are returned to the **Security Context Broker**.
3. Then the **Security Context Broker** handles the interaction with the **Off-Chain Storage** by providing the storage pointers and receiving the raw traces.
4. Finally, the **Security Context Broker** provides the raw traces to the **Risk Assessment**.

5.3.6 Flow 6:TNDI Trustworthiness Claims Secure Storage (Pull Mode - from Local TAF)

This flow shows the storage of Trustworthiness Claims by the **Local TAF**. In this case the pull mode is used (e.g., the Security Context Broker is acting as a Relay) and the claims are stored via the **Oracle Worker**. Keep in mind that the **Oracle Worker** is already subscribed to listen to these notification events and when the **Local TAF** notifies that new Trustworthiness Claims exists, initiates the fetching process. The steps are the following:

i. The **Local TAF** requests to store the Trustworthiness Claims through the **Layer 1: Attribute-based Access Control** of the **Security Context Broker**.
ii. If the authorization is successful, the **Security Context Broker** emits a transaction trigger event to the **CASTOR DTL**.
iii. The already subscribed **Oracle Worker** listens to this event and initiates the process to fetch the Trustworthiness Claims.
iv. The **Oracle Worker** pulls the Trustworthiness Claims by the **Local TAF**.
v. Finally, the **Oracle Worker** commits the Trustworthiness Claims to the **CASTOR DLT**.

5.3.7 Flow 7: TNDI Trustworthiness Claims Secure Storage (Push Mode - from Global TAF)

This flow shows the storage of Trustworthiness Claims by the **Global TAF**. In this case the push mode is used and the claims are stored via the **Computation Worker**. The steps are the following:

a. The **Global TAF** requests to store the Trustworthiness Claims through the **Layer 3: TNDI-SP Validation** authorization of the **Computation Worker**.
b. If the authorisation is successful, the Computation Worker commits the Trustworthiness Claims to the **CASTOR DLT**.

5.3.8 Flow 8: Query Historical Trustworthiness Claims for Trust Evolution Management

This flow depicts the query of the stored Trustworthiness Claims. The Trustworthiness Claims can be queried by both the **Global TAF** component through the **Layer 2: Address Whitelisting** as historical data and by an external authorised entity through the **Layer 1: Attribute-based Access Control**. In the following steps we will describe the former case, while the latter case will be described in the Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions along with the other queries form an external entity.

1. The **Global TAF** queries to receive the Trustworthiness Claims from the **CASTOR DLT**, through the **Layer 2: Address Whitelisting** access control.

2. If authorization is successful, the Trustworthiness Claims are returned to the **Global TAF** via the **Security Context Broker**. In the case of high volume data where the Trustworthiness Claims are stored off the chain the **Security Context Broker** is responsible for the acquisition of the Trustworthiness Claims by the **Off-Chain Storage** and the forwarding to the **Global TAF**. These steps are not depicted in Figure 5.1 for simplicity but are the same as steps 2, 2a, 2b and 3 in Flow 5: Query TNDI Raw Traces.

5.3.9 Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions

This flow focuses on how an external entity can query either trust-related information, raw traces and SSLAs through the offered **Trust Exposure API** by the **Security Context Broker**. In a nutshell, the Trust Exposure API facilitates the transformation functions provided to the **External Entity**.

1. The first step is an external entity to query specific trust-related characteristics (e.g., trust-related information, raw traces and SSLAs) providing its attributes for authorisation through the **Layer 1: Attribute-based Access Control**.

2. If the entity is authorized by the ABAC mechanism, the Security Context Broker acting as a **Trust Exposure Layer** forwards the query to the **CASTOR DLT**.

3. The DLT provides the response with either the requested trust-related information back to the **Security Context Broker** or the storage pointers of the requested information. In the latter case, the **Security Context Broker** (2a) queries data from the **Off-chain Storage** providing these pointers and (2b) the requested trust-related information is acquired by the **Security Context Broker**.

4. Upon the successful acquisition of the requested data, the **Security Context Broker**, forwards the CASTOR's DLT response to the **External Entity**.

5.4 High-level Definition of Smart Contracts

This section details the smart contracts of CASTOR DLT. More specifically, the use of smart contracts in CASTOR facilitates the managing of Trustworthiness Claims, raw traces, SSLA and Trust Policies ensuring trust, data integrity and confidentiality across the CASTOR ecosystem. The key envisioned smart contracts are summarised in Table 5.3 including a short description and a high-level flow per smart contract.

Table 5.3: Smart Contracts and High-Level Interaction Flows

Contract Name	Description	High-level flow
TrustPolicyRegistry	Stores Trust Policy objects {`uuid, vendor, rtl, trustModel, threatProfile`}. Acts as the on-chain source of truth for Trust Policies.	**Admin** → **TrustPolicyRegistry**: The Admin populates and enforces Trust Policies (see Flow 1: CASTOR Preparedness Phase and Trust Policy Secure Storage for more details). **Router Element** → **Data Exposure \Query Worker** →**TrustPolicyRegistry**: Router Elements query Trust Policies using `(vendor, threatProfile)` to retrieve applicable Trust Policies (see Flow 2: CASTOR Proactive Phase and Trust Policy acquisition for more details).
SSLACommit	Commit contract for SSLA results in the form {`serviceID, vrouterIDs[], sslaID, sslaCompliance, timestamp`}. Enforces enclave authorization (`COMPUTE_B`) and anti-replay.	(a) **Traffic Engineering Policy Engine** → **SSLACommit** and (b) **SSLA Monitor** → **SSLACommit**: These components submit SSLA compliance results to be anchored on-chain. (see Flow 3: SSLA Secure Storage for more details)
RawEvidenceCommit	Commit contract for raw trace evidence without enclave involvement. Stores evidence in the form {`dbPointer, vrouterID, timestamp, signature`}. Ensures immutability and auditability of raw inputs used for later trust assessment.	**Router Element** → **RawEvidenceCommit**: Router Elements append raw traces on-chain via the SCB to ensure auditability, data veracity, and immutability of the underlying evidence (see Flow 4: TNDI Raw Traces Secure Storage for more details).
RawEvidenceQuery	Read-process path for raw trace evidence. Allows retrieval by `vrouterID` and time ranges, returning pointers and signatures for verification.	**External Entity** → **RawEvidenceQuery**: Auditors / Services or internal actors retrieve raw evidence for verification and auditing purposes (see Flow 5: Query TNDI Raw Traces and Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions for more details).
OracleJobRequest	Trigger contract for fetching Trustworthiness Claims. Receives oracle job requests and emits `OracleRequested (jobId, vrouterID, params...)`. This event initiates the oracle workflow.	**Local TAF** → **OracleJobRequest**: Upon finalized assessment output, the Local TAF submits a trigger request on the SCB that emits an event to initiate fetching and processing by the Oracle Worker (see Flow 6:TNDI Trustworthiness Claims Secure Storage (Pull Mode - from Local TAF) for more details).
LocalTrustAssessment Commit	Creation of objects for trust assessment records (e.g., Trustworthiness Claims). Commit for the claims keyed by `vrouterID`. Stores/ appends `localTrustLevel[]` and `localTAFSignatures[]` only. Global fields are left empty. Enforces enclave authorization (`ORACLE_A`) and anti-replay.	**Local TAF** → **OracleJobRequest** → **Oracle Worker**: Local TAF triggers the event; Oracle Worker listens, fetches data, verifies it, performs computations, and commits the local Trustworthiness Claims on-chain (see Flow 6:TNDI Trustworthiness Claims Secure Storage (Pull Mode - from Local TAF) for more details).
GlobalTrustAssessment Commit	Update commit for trust assessment records (e.g., Trustworthiness Claims). Appends `globalTrustLevel[]` and `globalTAFSignatures[]` to an existing trust assessment record for the same `vrouterID`. Enforces enclave authorization (`COMPUTE_B`), versioning, and anti-replay.	**Global TAF** → **Computation Worker**: Global TAF provides additional Trustworthiness Claims inputs; Computation Worker updates the previously initialized local Trustworthiness Claims to create a combined local and global trust bundle for a specific `vrouterID` (see Flow 7: TNDI Trustworthiness Claims Secure Storage (Push Mode - from Global TAF) for more details).

Contract Name	Description	High-level flow
TrustAssessmentQuery	Read-process path for trust assessment records. Supports historical queries, retrieval of local vs. global trust components, and versioned access if enabled.	**Global TAF** → **TrustAssessmentQuery**: Global TAF queries historical trust assessments previously stored from Local TAF to support aggregation, correlation, or validation workflows. (see Flow 8: Query Historical Trustworthiness Claims for Trust Evolution Management for more details)
SSLAQuery	Read-process path for SSLA records. Supports query by serviceID or sslaID and optional retrieval of latest compliance state.	**External Entity** → **SSLAQuery**: External systems query SSLA compliance status for monitoring or reporting purposes. (see Flow 9: Trust Exposure Layer for Trust Route Configuration Abstractions for more details)
WorkerRegistry	Registers and manages authorized SGX enclaves (off-chain workers). Stores enclave identity, role (`ORACLE_A` / `COMPUTE_B`), and status (active/revoked). Acts as the on-chain enforcement point for which enclaves are allowed to perform oracle or compute operations.	**Gatekeeper** → **WorkerRegistry**: The Gatekeeper interacts with this contract to populate, update, and enforce enclave registration and lifecycle rules as needed.
ARPolicy (helper contract)	Helper policy contract used by the EnclaveRegistry. Defines which enclave identities are allowed and under which conditions.	**Gatekeeper** → **ARPolicy** → **EnclaveRegistry**: Used by the Gatekeeper to enforce access control and registration policies on-chain.
AccessControlManager (whitelisting)	Central role and permission manager. Defines who can register enclaves, trigger oracle jobs, commit raw evidence (SCB role), manage trust policies, and pause contracts.	**Governance** / **cross-flow**: Enforces access control across all flows and contracts.

5.5 Beynd Zero Trust: Controlled Data Exposure and Query-time Transformation by Harmonzing TCs

Many of the evidence attributes handled by the system are sensitive in nature and are not meant to be revealed in full precision to all the consuming actors. Therefore, the system architecture provides support for controlled data exposure through the application of transformation functions to sensitive trust evidence before it is provided to the requesters. This is in line with the least privilege principle, where each actor is provided with only the level of detail that is required for its specific purpose while minimizing the chance of sensitive information being leaked across administrative or operational domains.

The transformation functions are used at query time only when needed. Currently, in the system architecture, all queries are passed through the Security Context Broker (SCB). The SCB enforces access control policies and response-shaping logic before the data is provided to the requester. Depending on the identity, role, and domain of the requester, the system provides raw anchored data, partial transformation data, or fully abstracted trust indicators. This is achieved without the need to store data redundantly on-chain or fragment trust records.

A concrete example of this mechanism applies to trustworthiness claims produced by local and global Trust Assessment Frameworks. Trust assessment records may contain fine-grained numerical attributes such as uncertainty, belief, disbelief, and appraisal, as well as multi-dimensional global trust components such as resilience and availability. These numerical values are meaningful for internal analysis and aggregation but are not necessarily appropriate for all consumers. Instead, the system can transform these values into coarse-grained trust classifications such as HIGH, MEDIUM, or LOW for a given assessed object (for example, a specific vrouterID).

This capability is particularly important in multi-operator and multi-domain environments. Different Mobile Network Operators (MNOs), orchestrators, and control-plane components may need to ingest trust information originating from other domains while being subject to distinct disclosure policies. For example, an external orchestrator may only require a high-level trust posture to make placement or routing decisions, whereas an internal component may be authorized to access more detailed trust vectors. Query-time transformation allows the same anchored trust data to be safely reused across domains without exposing sensitive internal assessments.

In cases where there are multiple trust dimensions in a global trust assessment, the transformation functions will operate on each trust dimension independently. Due to the fact that not all trust dimensions are necessarily required to be present in every update, there has been a focus on ensuring that the transformation logic can operate effectively in such scenarios. In certain scenarios, it will be possible to aggregate the dimension-level transformations with the global appraisal value associated with each dimension in order to produce a final result, without necessarily disclosing the underlying numerical trust vectors.

Through this approach, it will be possible for trust assessments to be updated incrementally over a particular period of time without compromising confidentiality, as well as ensuring interoperability and compliance with policy. An external entity will be able to access trust outcomes derived from on-chain anchored data without necessarily having access to sensitive trust evidence. In future versions of this system, it will be possible for a dedicated enclave component responsible for confidential querying as well as response shaping to be responsible for query-time transformations.

Chapter 6

Segment Routing (SR) Policy Construction and Enforcement

Routing can be influenced by various constraints, including Quality of Service (QoS), policy, and cost. In MPLS, GMPLS, and Segment Routing (SR) networks, constraint-based routing plays a key role in Traffic Engineering (TE). It calculates the optimal path for data to travel through the network and defines the route for each configured Label-Switched Path (LSP).

Traditionally, path computation was handled either by a management system or at the head end of each LSP. However, in large, multi-domain networks, this process can become highly complex, often requiring more computational resources and network visibility than individual network elements typically possess. At the same time, it must remain more dynamic and responsive than what a centralized management system can usually provide. Therefore, a **Path Computation Element (PCE) is a functional entity responsible for calculating optimal paths for individual services or groups of services within a network.**

6.1 Exploring the Path Computation Element Protocol (PCEP) for Routing Configuration and Management

A PCE is a network component capable of determining optimal routes for traffic flows through a network by considering topology information, resource availability, and various constraints such as bandwidth, latency, QoS, and policy requirements. [16, 22] Originally developed to offload computationally intensive path computation functions from routers in MPLS Traffic Engineering networks, the PCE has significantly expanded its role and capabilities over the past two decades. [34]

6.1.1 Architectural Components

There are different components related to PCE and the architecture involved in its implementation. Their definitions can be found in the following list:

- **PCE Server**: The computational entity that maintains a Traffic Engineering Database (TED) containing network topology, link states, available bandwidth, and other relevant metrics. The PCE applies sophisticated algorithms to compute optimal paths subject to specified constraints [33, 30].
- **Path Computation Clients (PCCs):** Network devices such as routers that consume path computation services. PCCs can request paths from the PCE or delegate control of their Label Switched Paths (LSPs) to the PCE for centralized management [15].

- **Traffic Engineering Database (TED)**: A comprehensive repository of network state information that the PCE uses for path computation. The TED is populated through Interior Gateway Protocol (IGP) extensions, Border Gateway Protocol Link State (BGP-LS), or direct interfaces to network management systems [34].
- **Path Computation Element Protocol (PCEP) stack**: The communication layer that enables request-response interactions between PCCs and PCEs, supporting both stateless and stateful operation modes [34].

The different components are typically used in the following manner: The PCE operates through the PCEP, a TCP-based protocol standardized in RFC 5440[34] that enables communication between a PCC—typically a router or network controller—and the PCE server. PCEP allows PCCs to request path computations from the PCE and enables the PCE to provide computed paths back to requesting clients [34].

6.1.2 Capabilities

Since the first implementations did not store the network state, a fundamental advancement in PCE technology is the stateful PCE model, standardized in RFC 8231 [12]. Unlike traditional stateless PCEs that simply compute paths upon request without retaining information about established LSPs, stateful PCEs maintain a comprehensive view of all active LSPs in the network and their resource consumption [37]. Stateful capabilities include:

- **LSP State Synchronization:** The PCE maintains real-time awareness of all LSP states across the network, enabling it to make globally optimal path computation decisions that account for existing traffic patterns and resource utilization[16].
- **LSP Delegation:** PCCs can delegate control authority over their LSPs to the PCE, allowing the PCE to update paths, adjust bandwidth allocations, and implement policy changes without requiring explicit requests from the PCC [16].
- **Active vs. Passive Modes:** In passive mode, the PCE monitors LSP states and responds to computation requests. In active mode, the PCE can proactively modify LSP parameters and attributes through Path Computation Update (PCUpd) messages, enabling dynamic traffic optimization [16].

Active PCE with status is an interesting approach since, in addition to having the TED constantly updated, it is capable of communicating proactively with the PCCs.

6.1.3 Integration with Segment Routing

A major area of PCE state-of-the-art development is its integration with Segment Routing (SR), particularly SR Traffic Engineering (SR-TE). Segment Routing simplifies traffic engineering by encoding forwarding instructions as ordered lists of segments in packet headers, eliminating the need for per-hop state maintenance and complex signaling protocols [28, 20, 22].

Furthermore, Segment Routing's Flexible Algorithm capability, combined with PCE, enables networks to define multiple logical topologies over the same physical infrastructure, each optimized for different metrics (IGP cost, TE metric, latency) or constrained by specific link attributes. The PCE can compute paths within specific Flex-Algo planes and program appropriate segment lists, enabling service differentiation and multi-tenant network slicing.

6.1.4 Flexible Algorithm

Flexible Algorithm (Flex-algo) is a mechanism that allows IGPs—specifically IS-IS and OSPF—to compute constraint-based paths autonomously by defining multiple optimization objectives beyond traditional shortest-path routing. Each Flex-algo is characterized by a Flexible Algorithm Definition (FAD) consisting of three essential parameters:

- **Numeric ID**: A unique number ranging from 128 to 255 per network.
- **Calculation Type**: The algorithm used for path computation, typically Shortest Path First (SPF) or constrained SPF.
- **Metric Type**: The optimization criterion, which can be IGP metric (default), Traffic Engineering (TE) metric, minimum unidirectional link delay (as defined in RFC 8570)[17], or bandwidth-based metrics.
- **Constraints**: A set of topological restrictions such as affinity-based link inclusion/exclusion, bandwidth thresholds, or maximum delay limits that prune links from consideration during path computation.

Flex-algo constraints will exclude those routes that do not have the desired terms in the network and trust agreement, and the result will be a subset of possible routes for making the connection. The downside of this technology is that it must be deployed by a network administrator or be preconfigured. That is why combining PCE with Flex-something allows us to automate this process.

6.1.5 Hierarchical PCE

Another useful feature of this component is that hierarchical architecture can be used. The Hierarchical PCE (H-PCE) architecture, defined in RFC 6805, establishes a parent-child relationship among PCEs [19]. A parent PCE coordinates path computation across multiple domains by interacting with child PCEs responsible for individual domains. The parent PCE maintains an abstract topology of the multi-domain network and can compute end-to-end paths by requesting domain-specific path segments from child PCEs.

On the one hand, the Hierarchical PCE (H-PCE) architecture provides significant advantages for multi-domain path computation by enabling optimal end-to-end paths across domains while preserving administrative autonomy and confidentiality. The Parent PCE (P-PCE) coordinates domain sequencing and applies network policy without requiring detailed knowledge of child domain internals, which protects proprietary topology information—particularly important when domains are operated by different commercial entities. This architecture offers a scalable path computation scheme since each Child PCE (C-PCE) only manages its own domain topology while the P-PCE works with abstracted domain connectivity, reducing computational complexity compared to flat approaches. H-PCE integrates well with frameworks like ACTN (Abstraction and Control of TE Networks), where it can receive abstract topologies from Provisioning Network Controllers to enable coordinated multi-domain service delivery. Additionally, the stateful H-PCE model allows the P-PCE to trigger global reoptimization based on LSP state changes and coordinate PCE-initiated LSP segments that are stitched together for end-to-end connectivity [13].

On the other hand, H-PCE architecture introduces several challenges, primarily around scalability at the Parent PCE level—if all Child PCEs report all LSPs in their domains, the P-PCE can become overwhelmed, necessitating careful filtering to only report LSPs directly involved in hierarchical operations. Multi-domain operations inherently require information exchange across domain boundaries, creating significant security and confidentiality risks that must be mitigated through mechanisms like path-keys

and strict authorization controls. The P-PCE has limited direct visibility into child domains, which can constrain its ability to perform fine-grained optimization and may result in suboptimal paths compared to solutions with full topology visibility. Coordination complexity increases with recursive hierarchies (where PCEs delegate to parent PCEs, which may further delegate upward), and maintaining consistent state across multiple levels adds operational overhead. Furthermore, the architecture depends on proper configuration of trust relationships and authorization policies — unauthorized reports must be dropped, and management organizations must carefully control which PCEs can participate in hierarchical procedures [13].

6.2 CASTOR Customized Path Selection & Enforcement: Optimizing Traffic Engineering with SRv6 Flex-Algo & PCE Extensions

Existing PCEP specifications lack mechanisms to explicitly signal and negotiate SR-Algorithm capabilities and constraints. This limits the ability of PCEs to make informed path computation decisions based on the specific SR-Algorithms supported and desired within the network. The absence of an explicit SR-Algorithm specification in PCEP messages implied no specific constraint on the SR-Algorithm to be used for path computation, effectively allowing the use of SIDs with any SR-algorithm.

A primary motivation for these extensions is to enable the PCE to leverage the path computation logic and topological information derived from Interior Gateway Protocols (IGPs), like Flexible Algorithms. Aligning PCE path computation with these IGP algorithms enables network operators to obtain paths that are congruent with the underlying routing behavior, which can result in segment lists with a reduced number of SIDs.

In a practical manner, the PCE performs Flexible Algorithm path computation and policy enforcement based on topology information stored in its TED [34]. The TED is expected to be populated with necessary information, including Flexible Algorithm Definitions (FADs), node participation, and ASLA-specific link attributes through standard mechanisms such as Interior Gateway Protocols (IGPs) with Traffic Engineering extensions or BGP-LS [31].

CASTOR's approach to this problem is to treat the baseline PCE behaviour as a fallback mode. When an SSLA or SLA contract violation is detected and the system must react with minimal delay, the PCE computes a new route directly from the current topology and Flex Algo Definition (FAD) constraints, without waiting for fresh optimization results. This fast-reaction mode, in which the PCE behaves as a conventional TE controller driven purely by FAD and topology information, is referred to as "vanilla PCE".

During normal operation, however, traffic engineering decisions are expected to follow the trust- and network-aware recommendations produced by the Optimization Engine. In this steady-state mode, the PCE is augmented with an enforcement extension that consumes the Segment Routing (SR) policies and configurations generated by the Traffic Engineering Policy Engine, ensuring that the optimized, trust-compliant paths are consistently installed on ingress and border routers. This modular extension, which decouples CASTOR's optimization logic from vendor-specific PCE implementations and allows the optimization engine to evolve independently of underlying routers, is referred to as "PCE Extension".

As shown in Figure 6.1, as an initial step, the Network Controller must apply the relevant configuration to the router in order to establish connectivity with the network and standard configurations. Then, when a new path is required, the TE Policy Engine sends a request to the Optimization Engine. Once the optimal graph has been computed, it is returned to the TE Policy Engine, which translates it into device-level configurations that can be enforced by the PCE Extension and installed on the routers in the topology. This configuration is then passed to the Facility Layer, which acts as middleware, handling message management and routing them to the appropriate components. This design accommodates heterogeneous deployments, as a network administrator may operate multiple, redundant PCE instances within the same

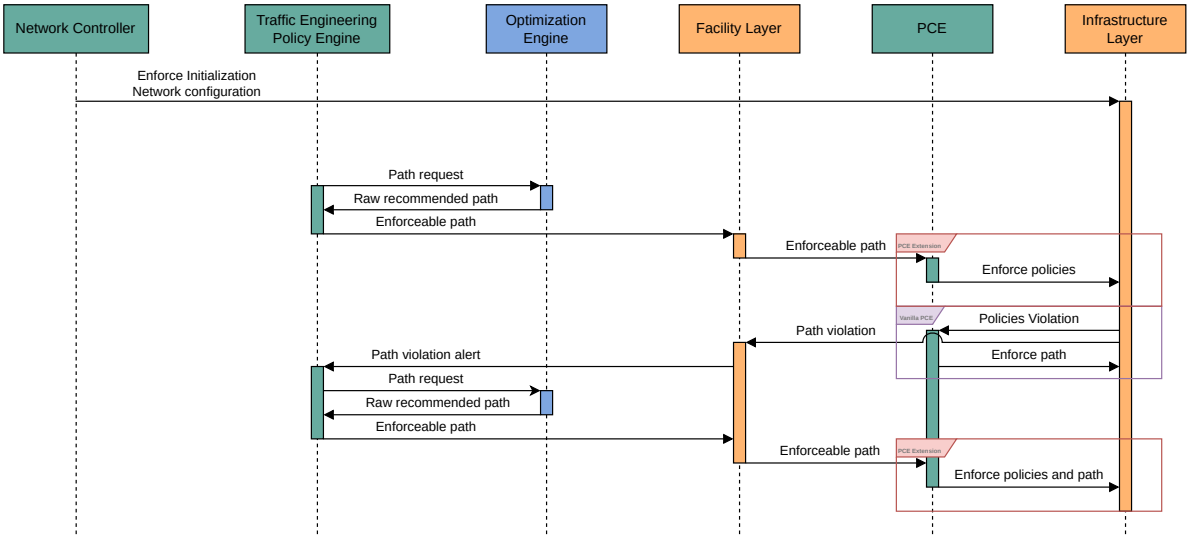

Figure 6.1: PCE sequence diagram

topology. Finally, the PCE Extension enforces the configuration and applies the negotiated (S)SLAs along the selected paths.

When a topology change causes a policy violation and disrupts communication between source and destination, the SSLA requirements must be re-evaluated against the updated network state. As an immediate reaction, the vanilla PCE computes and installs a new path that still complies with the constraints encoded in the current Flex Algo Definition (FAD), restoring basic connectivity. Subsequently, the Facility Layer, once it detects the updated Topology Graph, requests fresh path recommendations from the CASTOR traffic engineering and optimization components, re-running the procedure described for initial path instantiation. The CASTOR-derived Segment Routing policies then overwrite the interim routes provided by the vanilla PCE, ensuring that the final paths once again satisfy the full set of trust- and performance-related SSLA objectives in PCCs.

In this way, CASTOR achieves a manufacturer-agnostic traffic routing architecture by decoupling the trust-aware optimization logic from vendor-specific control-plane implementations, offering excellent modularity and versatility. This architectural approach enables operators to improve and evolve the Optimization Engine—the core intelligence responsible for computing trust-aware paths—independently of the underlying router hardware or software, without incurring dependency on proprietary vendor solutions.

By leveraging standardized traffic engineering interfaces and protocols such as the Path Computation Element (PCE) protocol and IETF Segment Routing mechanisms, CASTOR ensures that trust-aware routing policies can be consistently enforced across heterogeneous, multi-vendor network infrastructures. This vendor-agnostic design not only facilitates broader adoption across diverse network environments—from legacy enterprise networks to modern edge and cloud infrastructure—but also future-proofs the framework against technological obsolescence, allowing operators to seamlessly integrate new router models or replace existing equipment without disrupting the trust-aware traffic engineering capabilities or requiring extensive reconfiguration.

Chapter 7

Configuring Network Service Topology with SR Flex-Algo Slicing: A Running Example

7.1 Routing Protocols and Source Routing in SR Slicing

Multiprotocol Label Switching (MPLS) has enabled Service Providers (SPs) to offer a wide selection of Quality of Service (QoS)-enabled Virtual Private Networking (VPN) services. Historically, BGP and LDP were used to deploy Layer 3 and Layer 2 VPNs, IS-IS or OSPF as Interior Gateway Protocols (IGPs) for fast next-hop convergence, and LDP, BGP or RSVP-TE for transport label distribution.

However, the need to reduce the complexity and the number of protocols in the core network, along with the rise of Software-Defined Networking (SDN), has driven the development of Segment Routing (SR) with its two main flavors: SR-MPLS and SRv6 (RFC 8402). In Segment Routing, source routing is used by allowing the ingress router to push a label or segment stack that defines the entire route the packet will take through the network. IGPs have been extended to carry segment (label) information in Link State Advertisements (LSA), so LDP and RSVP-TE aren't needed for label distribution or traffic engineering anymore. This is the primary reason behind the selection of SR as the main TE paradigm for validating the capabilities of CASTOR towards trustworthy service provisioning.

In single-domain, flat link-state IGP networks, all routers share the same complete view of the network. Therefore, the ingress router can compute the full segment/label stack to encapsulate and transport VPN traffic by itself.

In SR-MPLS service provider networks, Provider Edge (PE) routers will connect to customer equipment. PEs offer either Layer3 routing services using dedicated virtual routing and forwarding tables (VRFs), or Layer2 forwarding via point-to-point or multipoint services. All these services are tagged with BGP extended communities and distributed among participating PEs via MP-BGP (e.g., VPNv4, VPNv6, EVPN) full-mesh or using route reflectors (RR). Provider (P) routers are core transit routers, part of the same IGP as the PE routers, offering label-based forwarding. They do not need to carry customer service routing information in BGP. Customer Edge (CE) devices are part of the customer infrastructure, and do not participate in SR or the provider's IGP. For Layer 3 services, they may run routing protocols with the PE routers, under each customer VRF instance (requires VRF-aware routing protocol configuration on PEs). In that case, customer routes will be redistributed into MP-BGP to reach the other PEs.

In multi-domain or hierarchical networks under a single administrative domain, some destinations may not be visible to the ingress router within its link-state database (LSDB). For SRv6, since segments are IPv6 addresses, route summarization and redistribution can be used to reach these destinations. However, for SR-MPLS, an external helper like an SDN controller or Path Computation Element is needed to compute paths and segment/label stacks. The SR PCE can receive topology information from IS-IS, OSPF, or BGP Link State Address Family. As mentioned in Chapter 6, redundant PCE designs are also supported,

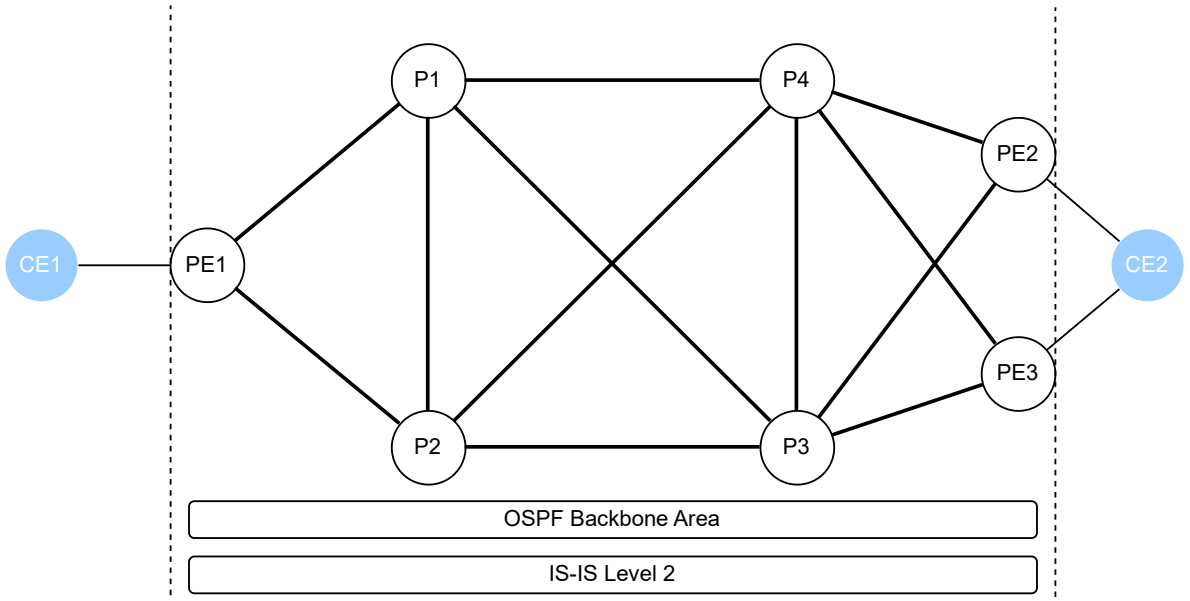

Figure 7.1: Single-domain flat IGP networks

with various approaches, including PCE-to-PCE communication or headend routers having connections to multiple PCEs. For redundancy to work in the latter case, the PCEs involved need to have the same topology information.

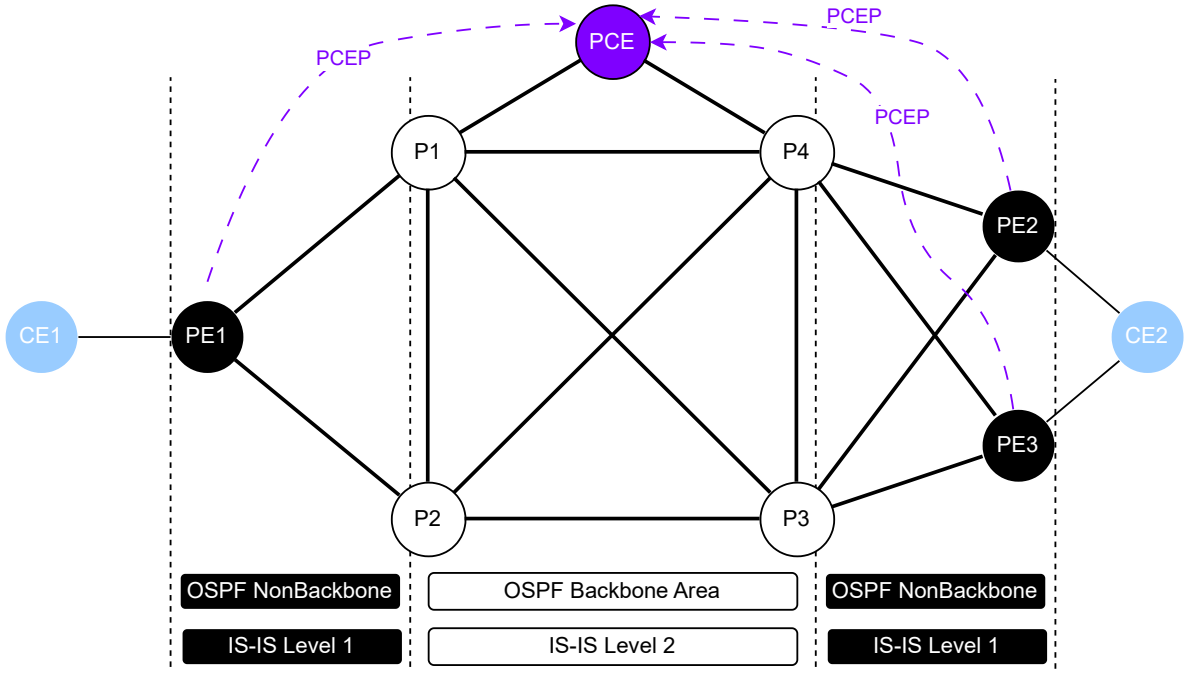

Figure 7.2: Cross-domain IGP networks

The Segment Routing Policy Architecture (RFC 9256) brings Traffic Engineering features to Segment Routing (SR-TE). Traffic can now be steered onto specific network paths, optimized for latency, bandwidth, or other KPIs. In CASTOR, we examine how this set of requirements can be extended so as to incorporate trust aspects as well.

On a headend router, an SR Policy is defined by a Color and an Endpoint - usually the egress PE for the BGP VPN services. When a BGP VPN route is marked with a Color extended community (RFC 5512, sec. 4.3), and a matching (Color, Endpoint) SR Policy exists on the headend, traffic is automatically steered along the active policy path (Automated Steering). SR Policy Colors can

be mapped to a Service Level Agreement (SLA), or specific intent. As already mentioned, in CASTOR, SR Policy Colors can be also mapped to a specific Path Profile that can satisfy services with distinct network- and trust-related requirements.

One of the benefits of using SR-TE over RSVP-TE is the removal of path state from the core network control plane. The ingress router decides on the actual path, and pushes a segment stack to match the intent. Intermediate only need to do data plane segment Push/Pop/Swap operations.

For large networks, SR Policy configurations can become hard to maintain. On-Demand Next-Hop (ODN) enhances SR-TE, allowing the headend router to assign Segment Routing (SR) paths to traffic flows as needed. The ODN/AS solution allows for full automation of SR-TE policies, and integration with BGP for customer service route distribution. If no valid SR Policies exist for a service route, classical forwarding to the BGP next-hop Prefix SID is used.

By using Affinity Link Colors, an operator can mark certain links to be excluded or included in the SR paths. Link Colors are advertised in the IGP LSAs as bit maps, where each color corresponds to one bit in the map. Naming schemes are locally significant, so network operators have to ensure the affinity name to bit position mappings are consistent across the network. A link can have multiple colors, with multiple bits set in the advertised bit map (see RFC 3630 sec. 2.5.9, RFC 5303 sec. 3.1 for 32-bit maps, and RFC 7308 for Extended Administrative Groups). Affinity Link Colors should not be confused with SR-TE Policy Colors! Multiple Affinity Link Colors can be used in include/exclude statements, defining constraints for the IGP or PCE SPF path computations.

IGP Prefix-SIDs are used to send traffic on the shortest path to a node or prefix. This shortest path is usually the accumulated IGP path link metric cost. However, multiple shortest paths may be identified if we were to optimize for different metrics: IGP cost, TE cost, or delay. Flexible Algorithm enhances SR and SR-TE, allowing network operators to deploy multiple logical topologies (Flex-Algos) on top of the same physical network. Each Flex-Algo can run its own Shortest-Path First (SPF) optimization objective and set of constraints. Prefix-SID segment lists, with each list belonging to a different Flex-Algo, can be used to steer traffic to the same egress PE loopback address.

With SR-TE, explicit segment lists for specific destinations can be defined by an operator or a controller. Whether explicit or PCEP-delegated paths are used, they need to pass validation checks on the ingress router. For example, networking vendors might check if the first SID in the list is reachable before accepting the path and programming the Forwarding Information Base (FIB).

When customer services are deployed, BGP routes are distributed among edge routers, with each PE retaining only those routes whose Route Target extended communities match its import policy [11]. Before programming the FIB and being able to forward traffic to any destination PE, the ingress PE must resolve the transport paths corresponding to the destination PE next-hop + color community tuple. To do this, the router looks in the local configuration for either on-demand next-hop (ODN) policies or per-next-hop policies.

Color-aware per-next-hop policies allow configuring multiple candidate paths with different preferences. For instance, one can configure explicit segment lists with higher priority, and use either locally computed paths (e.g., via IS-IS and flex-algos) or PCE-computed paths as lower-priority dynamic options. This type of policy can also be used to map the BGP color community (SLA intent) to a specific Flex-Algo ID.

Color-aware on-demand next-hop policies simplify the router configuration; however, they are not compatible with explicit segment lists. Flex-Algo ID mappings are also supported in this model.

The selected highest-preference path must pass internal validation (e.g., label stack viability, adjacency/IGP reachability, policy constraints). Once validated, it is programmed into the FIB so that customer traffic can be forwarded between the source and destination sites. A similar path selection and FIB-programming process occurs in the reverse direction.

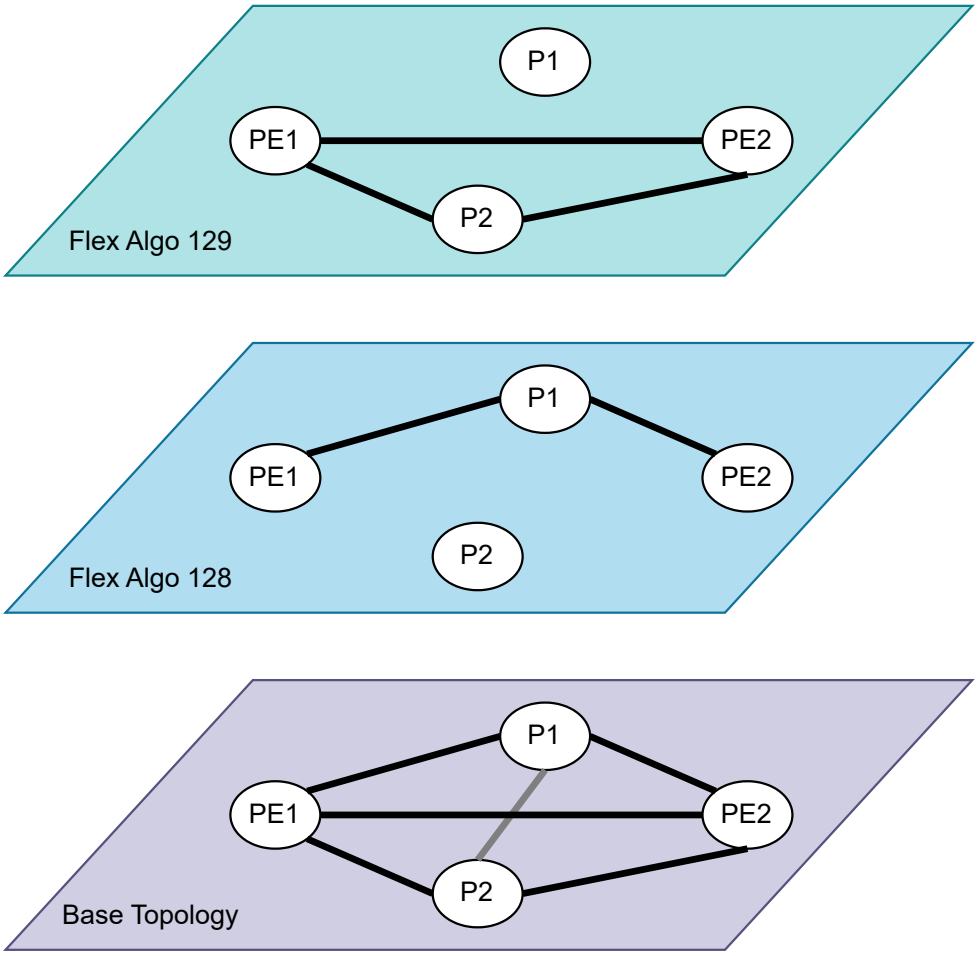

Figure 7.3: Example of overlay Flex-algo topologies on top of a base topology at the infrastructure layer.

7.2 Updating the Routing table

Provider (P) and Provider Edge (PE) Routers in a link-state IGP topology will go through neighbor discovery, adjacency establishment, and link-state database exchanges.

In SR-MPLS, IGPs will advertise a special type of segment ID - Node SIDs — which are used to represent a router's loopback address or node identity in the topology. A Node SID is a global segment that identifies a specific router, and allows any other router to forward packets toward that node using along the shortest path.

Each router takes a snapshot of its own connected prefixes, interfaces and neighboring routers, adds a sequence number to this snapshot, and floods it within the IGP domain. All routers within an area or flat domain will share the same view of the network, and run the Shortest-Path First (SPF) algorithm to select best paths to other nodes.

For end customer service advertisement and discovery on Provider Edge (PE) routers, BGP is the protocol of choice. It is capable of scaling to millions of VPN routes, across address-families such as IPv4 VPN, IPv6 VPN, L2VPN EVPN. Provider routers (P) typically do not need to run BGP since they're not doing any customer aggregation.

From a topology perspective, PE routers will usually be dual-homed to different P routers, for increased uplink resilience. For link loss detection, SP routers will rely on physical layer events such as optical Loss of Signal (LOS), or packet-based control-plane checks via Bidirectional Forwarding Detection (BFD). A link-loss detection event will trigger the IGP to take a newer snapshot of its state and interfaces, put a newer sequence number on this snapshot, and flood it across the IGP domain/area. All routers in the topology, and PCEs importing the updated topology will process the newest LSAs (with the higher

sequence numbers), and do SPF recalculations. A similar event-driven OSPF LSA or IS-IS LSP update takes place when Link Affinity Colors change on a node, since these are expressed as bit maps in the LSA/LSP. The RIB updates will be propagated to the FIB, which is pushed to the linecards to update Data Plane forwarding.

When the IGP best path towards a specific node changes from one uplink to another, vendors usually implement platform-customized hierarchical Forwarding Information Base (FIB), so PE routers only need to update a pointer to the newest best egress interface. Convergence times for VPN services remain the same no matter how many Layer 2 or Layer 3 VPNs are deployed, because there's no time wasted updating millions of routes one-by-one.

When a PE node fails, its loopback address will become unreachable after the IGP flood and recalculation. Vendors will use BGP next-hop tracking to trigger Routing Information Base (RIB) route withdrawals immediately, without waiting for keep-alive timers to expire. RIB updates will trigger a FIB update, and the linecards will be reprogrammed with the newest information.

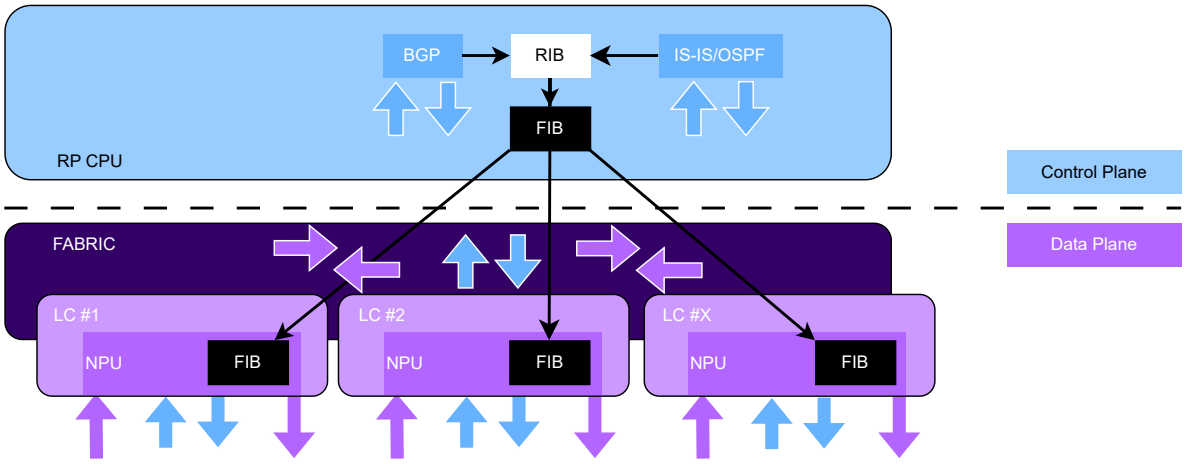

Figure 7.4: How IGP and SR-TE Policy events reach the RP/RSP and trigger FIB updates that propagate to the linecard ASICS/NPUS

7.3 Instantiating an SR-MPLS TE policy

When building or upgrading a network to support modern technologies like Segment Routing or Flex-Algo, certain steps have to be taken:

- deciding on the business requirements, the types of services (i.e., path profiles) to be delivered, scalability, SR-MPLS vs. SRv6, IS-IS vs. OSPF
- network and geographical topology assessment, choosing the optical transport specs
- networking equipment selection for P and PE routers.
- interop testing, if needed

Before the implementation phase, Loopback and link IPv4 addressing is defined per node, along with IS-IS NET. Defining an always-on loopback-type address defined on each network node helps ensure that BGP next hops do not change during PE uplink failures, providing greater predictability and stability for routing protocols.

A minimal initial config for SR-MPLS looks like this, with BGP needed only on PE routers (Listing 7.1):

Listing 7.1: Initial configuration of interfaces in router PE2

```

hostname xrd-2
!
tcp path-mtu-discovery
interface Loopback0
 no shut
 ipv4 address 100.100.100.102 255.255.255.255
!
interface GigabitEthernet0/0/0/0
 no shut
 mtu 9114
 ipv4 address 100.102.104.102 255.255.255.254
!
interface GigabitEthernet0/0/0/1
 no shut
 mtu 9114
 ipv4 address 100.102.106.102 255.255.255.254
!


router isis 1
 net 49.0100.0100.0100.0102.00
 log adjacency changes
!
 address-family ipv4 unicast
  metric-style wide
  advertise passive-only
 !
 interface Loopback0
  passive
  address-family ipv4 unicast
   !
 !
 interface GigabitEthernet0/0/0/0
  circuit-type level-1
  point-to-point
  address-family ipv4 unicast
   metric 1
   fast-reroute per-prefix
   fast-reroute per-prefix ti-lfa
   !
 !
 interface GigabitEthernet0/0/0/1
  circuit-type level-1
  point-to-point
  address-family ipv4 unicast
   metric 1
   fast-reroute per-prefix
   fast-reroute per-prefix ti-lfa
   !
 !
!

```

```
router bgp 100
 bgp router-id 100.100.100.102
 address-family vpnv4 unicast
 !
 address-family vpnv6 unicast
 !
 address-family l2vpn evpn
 !
 neighbor-group RR
  remote-as 100
  update-source Loopback0
  address-family vpnv4 unicast
  !
  address-family vpnv6 unicast
  !
  address-family l2vpn evpn
  !
 !
 neighbor 100.100.100.108
  use neighbor-group RR
 !
!
```

The next step for SR-MPLS would be defining the Flex-Algo topology cost functions, link metrics and affinities. The same template is used across all P and PE routers. As shown in Listing 7.2, this configuration showcases an example of encoding of high-level trust-related requirements (e.g., HIGH_TRUST is encoded in the affinity's register bit position 253) as a label that can be used to define various Flex-Algo Definitions. In IOS-XR, the Segment Routing Global Block (SRGB) for SR-MPLS starts at label 16000, so a prefix-sid index of 101 maps to label 16101, 102 maps to 16102, and so on.

Listing 7.2: Initial configuration for all routers in the topology. It specifies the available Flex-algo definitions and adds an initial assignment of labels in the supported interfaces of this router.

```

router isis 1
    net 49.0100.0100.0100.0102.00
    affinity-map HIGH_TRUST bit-position 253
    affinity-map HIGH_ENCR bit-position 254
    affinity-map HIGH_RELIAB bit-position 255
    affinity-map LOW_TRUST bit-position 0
    affinity-map LOW_ENCR bit-position 1
    affinity-map LOW_RELIAB bit-position 2
    flex-algo 128
        metric-type te
        advertise-definition
        affinity exclude-any LOW_TRUST
    !
    flex-algo 129
        advertise-definition
        affinity exclude-any LOW_RELIAB LOW_TRUST
    !
    flex-algo 130
        metric-type delay
        advertise-definition
        affinity exclude-any LOW_TRUST
```

```
    !
    flex-algo 131
        metric-type bandwidth
        advertise-definition
        affinity exclude-any LOW_TRUST
    !
    address-family ipv4 unicast
        metric-style wide
        microloop avoidance segment-routing
        advertise passive-only
        advertise link attributes
        segment-routing mpls sr-prefer
    !
    interface Loopback0
        passive
        address-family ipv4 unicast
            prefix-sid index 102
            prefix-sid algorithm 128 index 202
            prefix-sid algorithm 129 index 302
            prefix-sid algorithm 130 index 402
            prefix-sid algorithm 131 index 502
        !
    !
    interface GigabitEthernet0/0/0/0
        circuit-type level-1
        point-to-point
        affinity flex-algo LOW_TRUST LOW_ENCR
        address-family ipv4 unicast
            bandwidth-metric flex-algo 201
            te-metric flex-algo 1001
            metric 1
            fast-reroute per-prefix
            fast-reroute per-prefix ti-lfa
        !
    !
    interface GigabitEthernet0/0/0/1
        circuit-type level-1
        point-to-point
        address-family ipv4 unicast

        bandwidth-metric flex-algo 201
        te-metric flex-algo 1001
        metric 1
        fast-reroute per-prefix
        fast-reroute per-prefix ti-lfa
        !
    !
!
performance-measurement
    interface GigabitEthernet0/0/0/0
        delay-measurement
        advertise-delay 1000
    !
!
```

```
interface GigabitEthernet0/0/0/1
    delay-measurement
        advertise-delay 1000
        !
    !
!
```

BGP Layer 3 or Layer 2 VPN services can now be provisioned on PEs, with service routes marked with a color community corresponding to a list of predefined SLAs or templates from a Service Catalog (example in Listing 7.3).

If a customer requests a simple any-to-any (full-mesh) VPN service, a single VRF is typically sufficient on the PE routers. For hub-and-spoke VPNs or other custom topologies, multiple VRFs may be required. In such cases, multiple Route Targets (RTs) are usually referenced in the import/export configuration of each VRF to control which routes are shared between which VPN sites.

VPN services (L3VPN/L2VPN) do not map one-to-one to SLA levels; instead, a VPN provides logical separation and reachability, while SLAs are layered on top using QoS classes and options. A basic VPN usually comes with standard availability and best-effort performance, while higher tiers add multiple traffic classes (e.g., voice, business-critical, best-effort), each with its own delay, jitter, loss, and bandwidth objectives. These classes are enforced by mapping customer markings to internal MPLS/Segment Routing QoS, possibly using SR-TE or SR policies and different protection/redundancy options, so a single VPN can carry traffic with several SLA levels at once.

BGP color communities can be used to signal different SLA or intent levels for customer routes (e.g., gold/silver/low-latency). Each PE tags customer prefixes with a specific color, and the remote ingress routers map that color to a particular SR policy or TE class with defined performance characteristics (latency, loss, protection).

Listing 7.3: Example on how to associate a specific Service with a Path Profile.

```

vrf 100
    address-family ipv4 unicast
        import route-target
            100:100
        !
        export route-policy RP100
        export route-target
            100:100
        !
    !
    address-family ipv6 unicast
        import route-target
         100:100
        !
        export route-policy RP100
        export route-target
            100:100
        !
    !
!

vrf 200
    address-family ipv4 unicast
        import route-target
```

```
            100:200
        !
        export route-policy RP200
        export route-target
            100:200
        !
    !
    address-family ipv6 unicast
        import route-target
            100:200
        !
        export route-policy RP200
        export route-target
            100:200
        !
    !
!
```

Regarding SR-TE, a mix of on-demand next-hop with automated-steering, and per-next hop policies can be used. In the example below (Listing 7.4), sid-algorithm represents the Flex-Algo ID, and higher preference is given to the explicit paths compared to the PCE-delegated paths. Color 100 uses the base topology (Flex-Algo 0).

Listing 7.4: Segment Routing policy encoded in the ingress (Head-end) PE router

```
segment-routing
    global-block 16000 18000
    traffic-eng
        candidate-paths
            all
                source-address ipv4 100.100.100.102
            !
        !
        segment-list MYSEGLIST1
            index 10 mpls label 16106
            index 20 mpls label 16104
            index 30 mpls label 16103
            index 40 mpls label 16101
        !
        segment-list MYSEGLIST2
            index 10 mpls label 16206
            index 20 mpls label 16204
            index 30 mpls label 16203
            index 40 mpls label 16205
            index 50 mpls label 16201
        !
        on-demand color 100
            dynamic
                pcep
                    metric
                        type igp
                    !
                !
            !
        !
```

```
        on-demand color 200
            dynamic
                pcep
            !
            constraints
                segments
                    sid-algorithm 128
                !
            !
        !
        on-demand color 300
            dynamic
                pcep
            !
            constraints
                segments
                    sid-algorithm 129
                !
            !
        !

        policy PC100
            color 100 end-point ipv4 100.100.100.101
            candidate-paths
                preference 100
                    dynamic
                        pcep
                            metric
                                type igp
                            !
                        !
                    !
                !
                preference 300
                    explicit segment-list MYSEGLIST1
                        weight 300
                    !
                !
            !
        !
        policy PC200
            color 200 end-point ipv4 100.100.100.101
            candidate-paths
                preference 100
                    dynamic
                        pcep
                    !
                !
                constraints
                    segments
                        sid-algorithm 128
                    !
                !
                preference 300
```

```
                    explicit segment-list MYSEGLIST2
                        weight 300
                    !
                !
            !
        !
        pcc
            pce address ipv4 100.100.100.107
        !
    !
!
```

7.4 CASTOR TE Policy Enforcement strategies

The previous sections of this chapter focus on the vanilla TE process, covering aspects of initial router configuration, definition of routing policies, and service deployment. Through this analysis we can arrive to two critical observations with respect to the realization of the overarching CASTOR framework: (i) there is not a single TE paradigm that fits all network provisioning requirements, and (ii) the selection of the appropriate mechanism to enforce a new routing policy is directly dependent on various reasons, including the characteristics of the network topology and the network elements' capabilities. For example, if the underlying data plane consists of legacy network elements incapable of processing Segment Routing instructions, the system must dynamically fall back to a traditional paradigm like RSVP-TE to ensure the policy is enforced. Furthermore, even when the infrastructure fully supports the Segment Routing paradigm, the specific enforcement mechanism still depends on the underlying fabric. Enforcing a policy in a modern IPv6 environment allows the framework to utilize SRv6, taking advantage of native IPv6 extension headers and route summarization. Conversely, enforcing that same policy over a traditional IPv4/MPLS data plane requires the framework to utilize SRv4 (SR-MPLS), translating the routing instructions into standard MPLS label stacks.

CASTOR's capabilities for securely measuring (see D3.1 [7]) and assessing (see D4.1 [5]) the trustworthiness of the underlying forwarding plane are agnostic to the preferred TE strategy. However, this flexibility does not easily extend to the selection and enforcement of optimal routing policies. In fact, CASTOR positions the TE Policy Engine to decide which is the appropriate mechanisms to enforce a new routing policy to the underlying routers. As already highlighted in Section 7.1, the validation of CASTOR's ability to engrain trust in TE policies relies on the Segment Routing paradigm. However, this approach is adaptable; for example, the TE Policy Engine could be seamlessly extended to encode policies into an RSVP-TE configuration.

The process of identifying the optimal routing policy for a particular SLA is a well-explored and established domain in traffic engineering (see Section 7.1). Routers can natively compute the optimal IGP path towards an intended destination or invoke a controller entity (e.g., PCE) in case of hierarchical or cross-domain communication. In addition to that, technologies such as Flex-Algo enhance the capabilities of a network element to simultaneously maintain the optimal path towards other elements under varying set of TE requirements. Therefore, it becomes clear that modern network equipment has inherent capabilities to support TE optimization challenges that can accommodate a variety of network-related performance requirements.

However, with the increasing demand for security and trust assurances in the routing plane - as seen in IETF's initiative on Trusted Path Routing (see Engineering Story-V) and the SCION framework (see Chapter 2) - it remains unclear whether (and how) existing, and native to the router software stack, optimization solutions can accommodate both network and trust requirements. For instance, it is possible

to reduce the problem of co-enforcing trust and network requirements using Flex-Algo configuration, provided that they can be expressed as a set of multiple constraints and a single TE metric. In fact, the example configuration of Listing 7.2 illustrates how the possible trust attributes are encoded as affinity bits (line 4-9), whereas line 49 showcases how the trust evaluation of each node and link is encoded as part of the respective interface configuration (e.g., interface GigabitEthernet0/0/0/0 is labelled with a Flex-Algo affinity bit of LOW_TRUST).

But does transforming multi-objective problems into a single metric for a Flex-Algo definition truly suffice for solving any arbitrary co-enforcement problem? What happens when the network and trust requirements cannot be reduced to a single TE metric? Given the aforementioned questions and the overarching generic optimization problem formulated in D4.1 [5], it becomes apparent that there is **a level of complementarity between native optimization mechanisms and the CASTOR Optimization Engine**. This level of complementarity in the possibilities of deriving optimal TE decisions opens up the path for two critical explorations when it comes to trust-aware routing configuration in the forwarding plane:

- First, CASTOR envisions exploring whether the recommended output of the CASTOR Optimization Engine can be expressed as a dedicated Flex-Algo Definition that can equivalently accommodate the established SSLA requirements. Intuitively, this exploration requires the expression of all envisioned objectives into a single TE metric (e.g., through a scalarization function) and the conversion of any other requirement as a TE constraint.
- Second, In the context of deriving the optimal recovery and restoration scheme, relying solely on explicit paths may introduce a window-of-opportunity for compromises, especially between recalculation periods. Therefore, identifying a trust-enhanced Flex-Algo Definition that can act as a robust backup strategy is of crucial importance. This ensures that if an active path fails, traffic falls back to a topology that maintains network and trust requirements within accepted SSLA margins, bridging the gap until the CASTOR Optimization Engine provides a fresh set of recommendations to be enforced.

All these considerations will be initially evaluated in dedicated Proof-Of-Concept (PoC) scenarios that will shed light into the challenges and limitations of each approach. Details on concrete evaluation planning for the relevant PoCs on the TE approaches to be adopted in CASTOR are reflected in D6.1 [6].

Chapter 8

Summary and Conclusions

This deliverable presents foundational concepts and core challenges of incorporating trust as part of traffic engineering decision making. It sets the scene for the functional specification of the core control services under an administrative domain, focusing on the inner challenges of the orchestration layer and the complementary entities that enable its operation in a decentralized and auditable manner. Starting from the high-level characteristics of the overall CASTOR framework and the envisioned control plane infrastructure, we draw a high-level comparison with the SCION project which re-designs control logic in order to incorporate trustworthy establishment of cross-domain communication. This comparison highlights the key added value that CASTOR introduces by securely monitoring and assessing the trustworthiness of network elements, paths, and entire domains, thereby establishing runtime trust guarantees on the underlying communication fabric that are essential prerequisites for supporting overlay services of mixed criticality. Secondly, this document presents a clear identification of the main challenges - in the form of Engineering Stories - that shall guide the functional specification and development of the relevant artifacts that shape the CASTOR Orchestration Layer.

Subsequently, this deliverable presents the high-level architecture of the Orchestration Layer envisioned within CASTOR. This overview delineates the key interactions and responsibilities of the various subcomponents that enable not only the trust-aware provisioning of the network element lifecycle but also the trust-enabled traffic engineering process for application services. Additionally, Chapter 5 provides an initial technical description of the CASTOR DLT. This component supports the operation of one or more administrative domains in a transparent, auditable, and immutable manner, while simultaneously facilitating the exposure of abstracted trust insights to authorized external entities via the Trust Exposure Layer.

Finally, the deliverable addresses the diversity of Traffic Engineering (TE) strategies and details how CASTOR incorporates trust into routing policy provisioning. First, it highlights how Path Computation Element (PCE) technology can be leveraged to enforce trust-aware TE policies, exploring potential extensions to existing PCE functionalities. Second, through a detailed example of provisioning Segment Routing (SR) TE policies, the text demonstrates CASTOR's capacity to integrate trust insights into the forwarding plane configuration across various scenarios where distinct TE capabilities are required.

Overall, this deliverable presents a foundational blueprint of the CASTOR Orchestration Layer. This aims to steer the design and development activities of the first release of the orchestration functionalities that will be employed to deploy and evaluate the overall CASTOR framework in the context of the use case applications and scenarios. With a view towards the final release of the Orchestration Layer that spans beyond the boundaries of a single administrative domain, this deliverable outlines initial challenges with respect to cross-domain service provisioning that spans across distinct domains with potentially different trust assumptions and trust models.

List of Abbreviations

Abbreviation	Translation
AR	Access Rule
AS	Autonomous System
ATL	Actual Trustworthiness Level
BR	Boarder Router
BGP	Border Gateway Protocol
BGP-LS	Border Gateway Protocol Link State
CA	Certification Authority
CC	Confidential Computing
CoCo	Confidential Computing paradigm
COLIBRI	COntroLling Inter-domain Bandwidth Resource Allocation
DLT	Distributed Ledger Technology
DRKey	Dynamically Reachable Key
FAD	Flex-Algo Definition
HL	High-Level
HIN	Health Info Net
H-PCE	Hierarchical PCE
P-PCE	Parent PCE
IGP	Interior Gateway Protocol
ISD	Isolation Domain
KBS	Key Broker Service
LSP	Label-Switched Path
MAC	Message Authentication Code
MANO	Management and Orchestration
MPLS	Multi-Protocol Label Switching
MR Enclave	Measurement Reference Enclave
ODN	On-Demand Next-hop
ONOS	Open Network Operating System
PoC	Proof-of-Concept
PCB	Path Construction Beacon
PCC	Path Computation Client
PCE	Path Computation Element
PKI	Public Key Infrastructure
PCEP	Path Computation Element Protocol
QoS	Quality of Service
RIB	Routing Information Base
RoT	Root of Trust
RTL	Required Trust Level
SR	Segment Routing

SCB	Security Context Broker
SDN	Software-defined Networking
SGX	Software Guard Extensions
SID	Segment Identifier
SLA	Service-level agreement
SPF	Shortest Path First
SSFN	Secure Swiss Finance Network
SSLA	Secure Service Level Agreement
SSLO	Security Service Level Objective
SCION	Scalable, Control, and Isolation on Next-Generation Networks
SITDS	Service Intent Translation & Decomposition Service
TAF	Trust Assessment Framework
TCB	Trusted Computing Base
TPR	Trusted Path Routing
TE	Traffic Engineering
TEE	Trusted Execution Environment
TLEE	Trust Logic Evaluation Engine
TNDE	Trust Network Device Extension
TNDI	Trusted Network Data Infrastructure
VC	Verifiable Credential
WP	Work Package
vRouter	Virtual Router

Bibliography

[1] Ably. What is pub/sub? the publish/subscribe model explained. https://ably.com/topic/pub-sub, 2026. Accessed: 2026-02-04.

[2] Abdeljalil Beniiche. A study of blockchain oracles. *arXiv preprint arXiv:2004.07140*, 2020.

[3] H. Birkholz, E. Voit, C. Liu, D. Lopez, and M. Chen. Trusted Path Routing. https://www.ietf.org/archive/id/draft-voit-rats-trustworthy-path-routing-11.html, January 2025.

[4] CASTOR. Operational landscape, requirements and reference architecture - initial version. Deliverable 2.1, The CASTOR Consortium, 11 2025.

[5] CASTOR. Architectural specification of castor continuum-wide trust assessment framework. Deliverable 4.1, The CASTOR Consortium, 2 2026.

[6] CASTOR. Castor integrated framework, use case analysis & interim report on poc verification results. Deliverable 6.1, The CASTOR Consortium, 3 2026.

[7] CASTOR. Conceptual architecture of castor trusted computing base & composable attestation model specification. Deliverable 3.1, The CASTOR Consortium, 02 2026.

[8] CASTOR. Implementation of castor policy enforcement in (cross-) domain continuum topologies. Deliverable 5.2, The CASTOR Consortium, 06 2026.

[9] Shinyoung Cho, Romain Fontugne, Kenjiro Cho, Alberto Dainotti, and Phillipa Gill. Bgp hijacking classification. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 25–32, 2019.

[10] "Cisco". Ios xrd. https:www.cisco.com/c/en/us/products/collateral/routers/ios-xrd/ios-xrd-ds.html#Productoverview, 2025. Accessed: January 2026.

[11] Cisco Systems, Inc. Understand Segment Routing Traffic Engineer Policy Path Validation Criteria. https://www.cisco.com/c/en/us/support/docs/routers/asr-9000-series-aggregation-services-routers/223048-understand-segment-routing-traffic.html, 2025. Cisco Technical Support Documentation, Document ID 223048, updated May 19, 2025.

[12] Edward Crabbe, Ina Minei, Jan Medved, and Robert Varga. Path Computation Element Communication Protocol (PCEP) Extensions for Stateful PCE. RFC 8231, September 2017.

[13] Dhruv Dhody, Young Lee, Daniele Ceccarelli, Jongyoon Shin, and Daniel King. Hierarchical Stateful Path Computation Element (PCE). RFC 8751, March 2020.

[14] S. K. Ezzat, Y. N. Saleh, and A. A. Abdel-Hamid. Blockchain oracles: State-of-the-art and research directions. *IEEE Access*, 10:67551–67572, 2022.

[15] Adrian Farrel and Seisho Yasukawa. Path Computation Clients (PCC) - Path Computation Element (PCE) Requirements for Point-to-Multipoint MPLS-TE. RFC 5862, June 2010.

[16] Adrian Farrel, Quintin Zhao, Zhenbin Li, and Chao Zhou. An Architecture for Use of PCE and the PCE Communication Protocol (PCEP) in a Network with Central Control. RFC 8283, December 2017.

[17] Les Ginsberg, Stefano Previdi, Spencer Giacalone, David Ward, John Drake, and Qin Wu. IS-IS Traffic Engineering (TE) Metric Extensions. RFC 8570, March 2019.

[18] "Grafana". Grafana documentation. https://grafana.com/, 2025. Accessed: January 2026.

[19] Daniel King and Adrian Farrel. The Application of the Path Computation Element Architecture to the Determination of a Sequence of Domains in MPLS and GMPLS. RFC 6805, November 2012.

[20] Daniel King and Haomian Zheng. Applicability of the Path Computation Element to Inter-area and Inter-AS MPLS and GMPLS Traffic Engineering. RFC 8694, December 2019.

[21] "Kubernetes". Kubernetes documentation. https://kubernetes.io/, 2025. Accessed: January 2026.

[22] Zhenbin Li, Shuping Peng, Mahendra Singh Negi, Quintin Zhao, and Chao Zhou. Path Computation Element Communication Protocol (PCEP) Procedures and Extensions for Using the PCE as a Central Controller (PCECC) of LSPs. RFC 9050, July 2021.

[23] Sahil Minhas, Ritik Jaswal, Ankita Sharma, and Sanjay Singla. Revolutionizing networking: A comprehensive overview of intent-based networking. In *2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP)*, pages 463–468, 2024.

[24] Secure Swiss Finance Network. https://www.scion.org/ssfn-scion/. Accessed: 2026-01-23.

[25] NOX. https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/. Accessed: 2026-01-29.

[26] ONOS. https://opennetworking.org/onos/. Accessed: 2026-01-29.

[27] OpenDayLight. https://www.opendaylight.org/. Accessed: 2026-01-29.

[28] Udayasree Palle, Dhruv Dhody, Yosuke Tanaka, and Vishnu Pavan Beeram. Stateful Path Computation Element (PCE) Protocol Extensions for Usage with Point-to-Multipoint TE Label Switched Paths (LSPs). RFC 8623, June 2019.

[29] POX. https://github.com/noxrepo/pox. Accessed: 2026-01-29.

[30] Jean-Louis Roux and Gerald Ash. Path Computation Element (PCE) Communication Protocol Generic Requirements. RFC 4657, September 2006.

[31] Ketan Talaulikar. Distribution of Link-State and Traffic Engineering Information Using BGP. RFC 9552, December 2023.

[32] The Prometheus Authors. Prometheus documentation. https://prometheus.io/, 2025. Accessed: January 2026.

[33] JP Vasseur, Adrian Farrel, and Gerald Ash. A Path Computation Element (PCE)-Based Architecture. RFC 4655, August 2006.

[34] JP Vasseur and Jean-Louis Le Roux. Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440, March 2009.

[35] Eric Voit, Henk Birkholz, Thomas Hardjono, Thomas Fossati, and Vincent Scarlata. Attestation Results for Secure Interactions. Internet-Draft draft-ietf-rats-ar4si-09, Internet Engineering Task Force, August 2025. Work in Progress.

[36] Y. Xian, L. Zhou, J. Jiang, B. Wang, H. Huo, and P. Liu. A distributed efficient blockchain oracle scheme for internet of things. *IEICE Transactions on Communications*, 107(9):573–582, 2024.

[37] Xian Zhang and Ina Minei. Applicability of a Stateful Path Computation Element (PCE). RFC 8051, January 2017.

[38] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. In *2011 IEEE Symposium on Security and Privacy*, pages 212–227, 2011.

[39] Liehuang Zhu et al. SDN controllers: A comprehensive analysis and performance evaluation study. *ACM Comput. Surv.*, 53(6), December 2020.