White Paper

Toward 6G Core Architecture Using an Inline Service Mesh

Authors

SeongJun Lee (seoul.lee@sk.com), B5G/6G Core R&D Architect, SK Telecom

DongJin Lee (dongjin@sk.com), B5G/6G Core R&D Architect, SK Telecom

JoongGunn Park (clark.park@sk.com), Core Architect, SK Telecom

Brendan Ryan (brendan.ryan@intel.com), Senior Software Engineer, Intel Corporation

Michal Kobylinski (michal.kobylinski@intel.com), Software Engineer, Intel Corporation

Chetan Hiremath (chetan.hiremath@intel.com), Senior Principal Engineer, Intel Corporation

Table of Contents

Executive summary

The service mesh has become a core component of cloud-native platforms which has been introduced as wireless service providers deploy Core Network systems based on the new 5GS standard from 3GPP. While the service mesh complements the cloud-native approach of decomposing the new 5G Core network functions into stateless microservices which are deployed as cloud native virtual network functions (CNFs) on container platforms, it has an undesirable side effect in that it adds latency to control plane processing. This latency overhead is caused by the current industry standard approach of using a service mesh sidecar proxy container in microservice pods. This has become a pain point that needs to be addressed in the deployment of 5G Core network functions (NFs) on cloud-native platforms. If this latency problem can be solved, the service mesh has the potential to provide greater benefits to 5G Core deployments, especially as we move forward to 6G and AI/ML integrated Core Network.

This paper describes a viable way to eliminate latency overhead introduced by the service mesh by adopting a new inline approach to replace the sidecar proxy method. Empirical performance data is provided for a commercial-grade cloud-native 5G Core stack. This solution achieves up to 70%[1] latency reduction for transactions between the session management function (SMF) gateway and packet data unit (PDU) session microservices and a 33%[2] reduction in gateway CPU usage. The paper further explores how such a low-latency service mesh could be further improved to provide greater benefits to 5G and 6G Core deployments, such as incorporating the functionality of the new service communication proxy (SCP) NF specified by 3GPP, and potentially leveraging AI/ML techniques to enable more resilient and reliable wireless cores.

1 Introduction and background

As the LTE/5G and Beyond 5G (B5G) Core network adopts a cloud-native architecture, network functions (NFs) have been restructured and decomposed into "microservices" which are deployed as containers on cloud-native platforms with Kubernetes orchestration. Current LTE/5G and B5G service-based architecture (SBA) implements the NFs to utilize an individual service-based interface (SBI) to communicate with each other, allowing the system to be more flexible and agile. As technologies are moving from B5G to 6G Core networks, industries and standardized development organizations (SDOs) are adapting NFs and creating enhanced features based on this new cloud-native architecture.

With a rise of emerging services such as XR/metaverse, satellite, AI/ML and energy-efficient networking, mobile network operators (MNOs) and telecommunication equipment manufacturers (TEMs) are expected to attempt to increase time-to-market (TTM) for such new features by introducing continuous-integration/deployment (CI/CD) for commercial deployments. This means finer-grained service and feature controls which, along with the increasing number of 3GPP NFs, will require a robust and performant microservices communication infrastructure for a viable solution.

A relevant factor is that a service communication proxy (SCP) was standardized in 3GPP to support efficient communication among multiple NFs by indirect communication with delegated discovery. This means that consumer NFs do not perform any discovery or selection, but instead just provide the necessary discovery and selection parameters required to find a suitable producer NF for a service request. The SCP uses the request address and the discovery and selection parameters in the request message to route the request to a suitable producer instance. The SCP can also perform discovery with a network repository function (NRF) to obtain the discovery result.

Along with the SCP, a new component known as the service mesh has also been introduced for optimal interoperability between the many microservices in cloud-native deployments. As well as its core function of optimal endpoint discovery, the service mesh has come to include many other features, such as observability, filtering, load balancing, failure management and security.

With such features, the hope was that the 6G Core network using a service mesh architecture could enable more efficient and flexible NFs without impacting end-to-end signaling performance such as latency. However, the current 3GPP standardization of the 5G Core (5GC) and accompanying shift to cloud-native deployments have led to increased latency for control plane operations, of which the following are the key contributors:

- **Decomposition of NFs into microservice CNFs encapsulated in pods:** Latency is added due to the inter-communication between these microservice CNFs, generally via HTTP/2 REST APIs over a container network interface (CNI) such as Calico. Note that this would not be required in a more monolithic NF application deployed as a virtual network function (VNF) or physical network function (PNF).
- **Use of a service mesh sidecar proxy for communication between CNFs:** This adds additional latency to 5GC control plane microservice intercommunication due to extra traffic routing between the Linux kernel, proxy sidecar container and application container.
- **3GPP Release 18 and 19 require more interaction between the NFs:** These standards require additional NF interactions related to 5G non-standalone (NSA, 5G with LTE) and other new ones such as NRF interactions to discover other NFs and unstructured data storage function (UDSF) interactions to maintain the stateless nature of the NF pod-level microservices. This further adds to control plane latency for core cloud-native deployments.

This paper from SK Telecom and Intel mainly addresses how additional latency contributed by the service mesh could be addressed for 6G.

Currently, the most popular service mesh being used for cloud-native core deployments is Istio, and the most common deployment configuration is to use a sidecar proxy such as Envoy in microservice pods. This interfaces with the Istio service mesh controller and serves as a proxy for microservice applications to communicate over the service mesh with other microservices.

While offering benefits, such a service mesh proxy also adds some overhead to communication latency between microservices. For inter/intra NF(s), this latency is critical and needs to adhere to strict timing requirements for 3GPP call-procedure operations such as user equipment (UE) registration and PDU session establishment. The decomposition of NFs into microservices has already increased this latency, and further delays due to traversal through a sidecar proxy makes latencies more challenging. A thorough understanding of the traffic path and how individual NFs communicate is important to improve the design of the service mesh.

Some alternative service mesh approaches are also being proposed, such as Ambient mesh, which removes the need for a sidecar proxy by providing a common L7 processing layer with a secure overlay. However, it does not fully address the latency overhead issue and has another potential shortcoming in that it could become a single point of failure.

This paper outlines an attempt to observe, measure and improve service mesh performance by implementing an inline service mesh for real cloud-native 5GC CNFs, empirically measuring the performance benefits, and finally discussing the results with a view to providing recommendations for B5G and 6G cloud-native deployments.

The paper also explores how the inline approach may complement other 6G features such as service mesh based SCP and the use of AI for platform resilience.

In the following sections, we will consider the service mesh improvements which are needed for the 6G Core.

1.1 Service mesh based SCP

As new and emerging service features are developed and distributed in smaller individual functional units, the need to perform communication between each function more efficiently becomes more important. A service mesh based SCP may partially address this by providing indirect communication between NF services, as illustrated in Figure 1, showing Model C and Model D. The NF uses a “delegating procedure” to direct the SCP to perform common NF features such as load balancing, failure management and redundancy so the NF can give more focus to NF service logic, thereby reducing development and verification time of NF features.

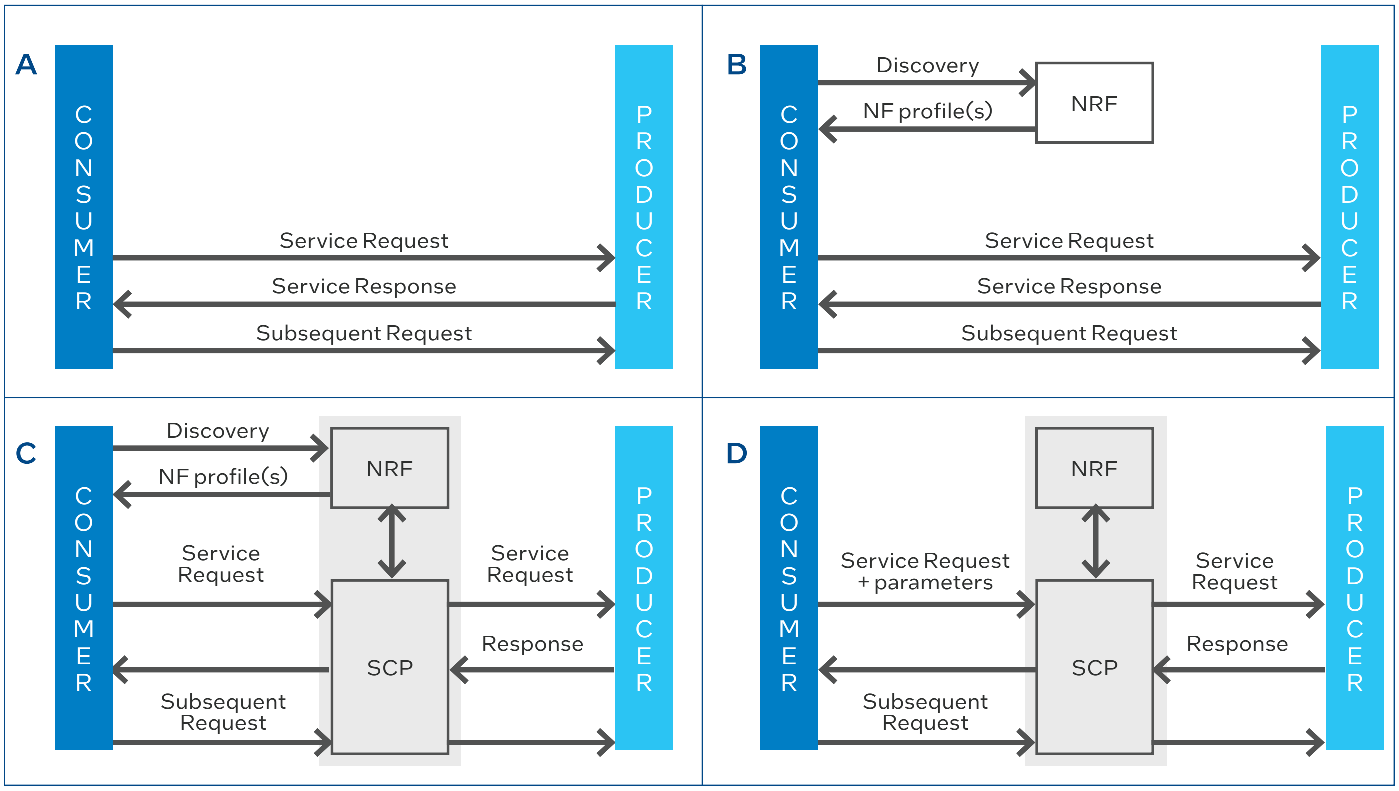

Figure 1. Service Communication Proxy deployment Model C and D from 3GPP 23.501(Rel-18/19)

In the deployment Model D, "Indirect communication with delegated discovery," a consumer NF's initial discovery/selection and reselection of producer NFs are all processed by the SCP functionality. In this deployment model, NFs no longer need to search for potential peer NFs to perform communication or handle load balancing, prioritization and overload handling, reducing common NF overhead. Additionally, because all NF messages are handled by the SCP in a distributed manner, a unified monitoring and tracing approach is also possible. As such, a service mesh with enhanced functionalities can appropriately provide capabilities of an SCP.

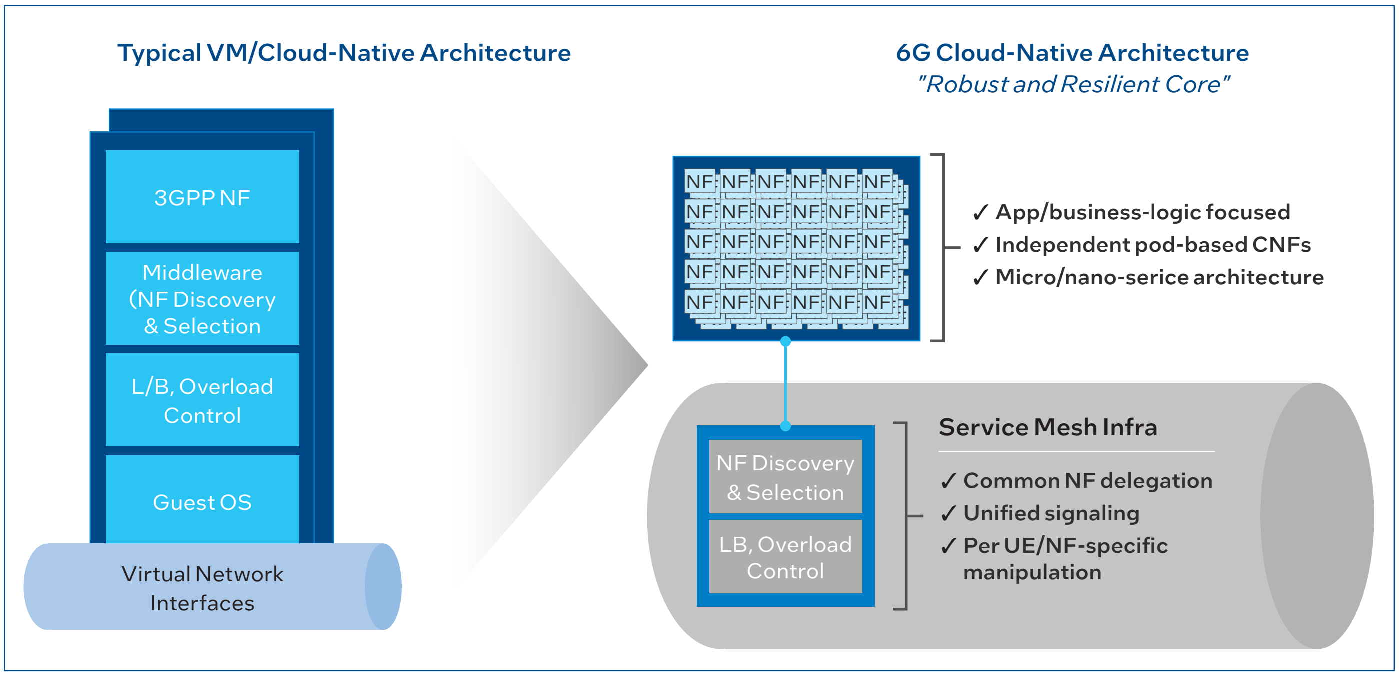
Typical VM/Cloud-Native Architecture
3GPP NF
Middleware (NF Discovery & Selection
L/B, Overload Control
Guest OS
Virtual Network Interfaces
6G Cloud-Native Architecture
"Robust and Resilient Core"
NF
✓ App/business-logic focused
✓ Independent pod-based CNFs
✓ Micro/nano-serice architecture
Service Mesh Infra
NF Discovery & Selection
LB, Overload Control
✓ Common NF delegation
✓ Unified signaling
✓ Per UE/NF-specific manipulation

Figure 2. Traditional vs. 6G cloud-native architecture

Figure 2 illustrates a potential 6G cloud-native architecture, "Robust and Resilient Core," evolved from a traditional architecture. The 3GPP NF's common features, such as discovery and selection, load balancing and overload control, are delegated to the service mesh layer. This Mode D deployment of the SCP can be implemented using the Istio service mesh. All the NF procedures and messages are passed through the service mesh layer, simplifying the communication channel and reducing burdens on the NFs.

Essentially the SCP can be considered a type of service mesh with some additional functionality for core NF discovery and selection. This enables a robust and resilient Core network because it can:

- Support distributed NF services providing interoperability and resiliency without causing a single point of failure.
- Control individual NF service rules dynamically for various inter/infra-NF communications, allowing flexibility and agility of the NF operations.
- Configure intelligently, such as with zero-touch automation, to handle NF service traffic load balances, overload protections, retries, bypasses and suspensions accordingly to the NF service status with the MNO's policies.
- Produce visibility and observability, allowing dynamic in-depth analysis of NF procedures from traces and logs, which can also be used to verify and understand behaviors of emerging service features in real time.
- Provide a canary release in a commercial environment for speedy development and deployment of new features.

In summary, the service mesh-based SCP simplifies cloud-native architecture that can handle an increasing number of NFs and their communications, including configurations of service/message routing, load balancing, prioritization and monitoring, thereby making it a robust and resilient core network.

The inline service mesh approach described in this paper can be enhanced to support service mesh-based SCP, thus enabling a robust and resilient core network without sacrificing any performance or efficiency.

1.2 Leveraging AI/ML technology for enhanced Core resilience

As an MNO, SK Telecom adheres to making a robust and resilient core the fundamental design rule for the 6G Core network; the primary focus is protecting end customers from any possible service quality degradation. As shown in Figure 3, SK Telecom foresees that a 6G Core network must be able to fully meet cloud-native principles, especially including monitoring, overload protection, recovery and redundancy. As such, NFs and their interrelated individual pods must be measured in depth for their processing latency, jitter and packet loss for analysis in real time. Any faults or erroneous behaviors should immediately trigger actions such as buffering, load (re)balancing, and self-healing. Furthermore, NF pods must have multiple replicas and be able to scale in/out dynamically, and they need to be monitored and visualized end-to-end for their inter/infra-NF-pod communications.

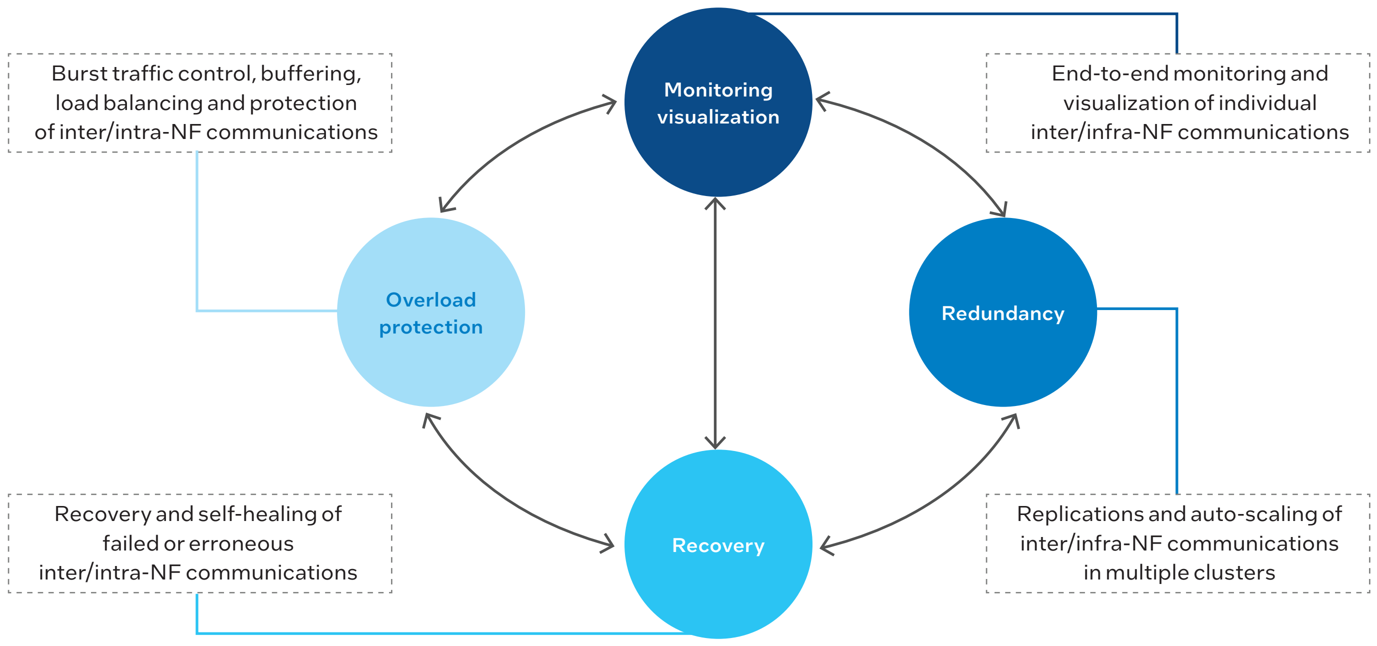
Burst traffic control, buffering, load balancing and protection of inter/intra-NF communications
Monitoring visualization
End-to-end monitoring and visualization of individual inter/infra-NF communications
Overload protection
Redundancy
Recovery and self-healing of failed or erroneous inter/intra-NF communications
Recovery
Replications and auto-scaling of inter/infra-NF communications in multiple clusters

Figure 3. Fundamentals of a robust and resilient core

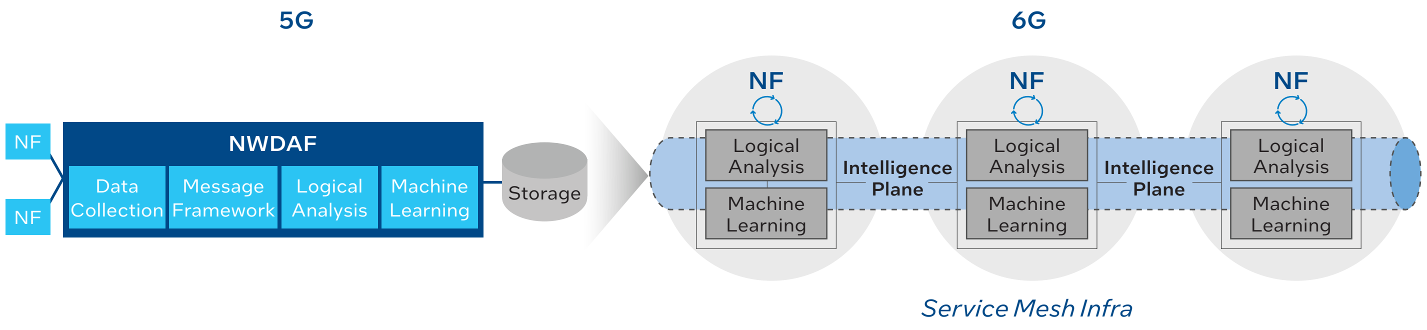

Figure 4. AI native Core network redesign

SK Telecom recently published a 6G whitepaper[3] which contains views on 6G key requirements, evolution methodology, development directions and promising 6G use cases. It also describes lessons learned and improvements to make the network more robust and resilient along with functional segmentation and distributed processing necessary for AI-native network evolution. With a higher number of NFs and higher performance capacities being demanded along with more complex network interconnections, AI assistance is an essential element toward a 6G network.

A 6G Core network also requires even lower latency and higher speed while maintaining reliability, thus AI-based optimizations should be applied to meet such demands. For example, as shown in Figure 4, individual NFs should have logical analysis and AI/ML entities to locally optimize key performance indicators (KPIs) and have ways for inter/intra-NF communications to communicate with each other via an intelligence plane for globally-optimized KPIs. Such an architecture would require a higher number of connections between pods, and hence it is apparent that a service mesh-based SCP could be the appropriate choice towards an AI-enhanced core network.

It is reasonable to consider that the service mesh could play an important part in an AI-assisted 6G Core, and this could be achieved with the type of inline service mesh described in this paper.

1.3 The need for a more flexible service mesh

The current 3GPP SCP is limited to support only inter-NF communications using service-based interface (SBI) over HTTP/2. Also, current service mesh architecture does not support protocols other than, for example, HTTP/2 and gRPC. This means that the current service mesh may not be able to work on NFs that have legacy interfaces and protocols such as S5/8, S11, N4 (UDP) and S1, N2 (SCTP), making it a lot more difficult to harmonize into a single unified service mesh network.

It is important to note that NFs in the future, in the 6G and 6G Core eras, would be more disaggregated and decomposed to even smaller-sized functions. Some studies suggest that the future NFs, including RAN and UE, will support SBI natively, making it more unified. We are likely to see more and more studies on how inter/intra-NF communications and their different decompositions affect performance, flexibility and resiliency for the next generation architecture. For instance:

- They potentially provide a way of mapping gRPC to the 3GPP HTTP/2 REST API for SBI communications.
- This could enable NFs with gRPC support to use gRPC for all control plane communications as the SCP could potentially enable gRPC-encapsulated SBI messages to be mapped HTTP/2 REST.
- They can reuse Kubernetes-based PaaS/IaaS toolkits to support self-healing, in-service software upgrade and scale-in/out.

1.4 Inline approach needed to reduce service mesh overhead

SK Telecom's SA commercial cloud-native Core network with sidecar proxy was measured based on busy-hour call attempts (BHCA). As shown in Figure 5, it was observed that a 5GC CNF's call-processing latency (e.g., 3GPP PDU session establishment procedure) increases with increasing number of gRPC connections between the pods.

Depending on the BHCA requirements, pod resource allocations and work-thread synchronizations could reduce such overhead (e.g., queuing). Some reduction of the 3GPP procedure latency can also be achieved by using parallel processing as connections between pods are multiplexed.

However, as more gRPC connections are created, an increasing number of the service mesh sidecar proxies are injected, too, which can dramatically degrade the 3GPP procedure latency (PDU session establishment) as shown in Figure 5. Even if the CNF application's logic and configurations are further tuned, the nature of the service mesh proxy (i.e., terminating connections) adds overhead due to kernel-space processing, the details of which will be explained later.

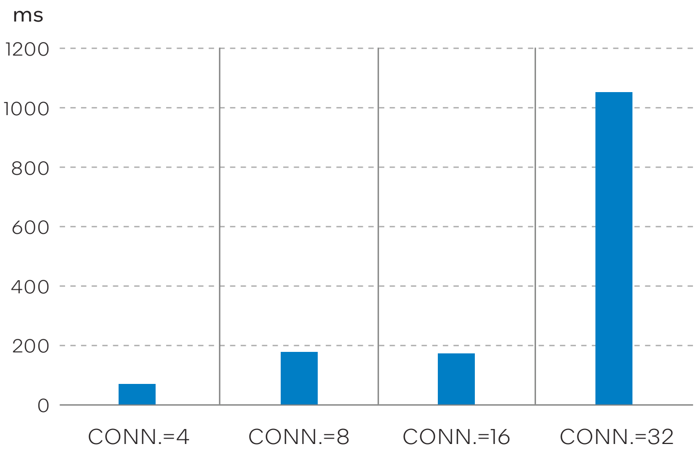

Figure 5. Latency related to connections between pods with a service mesh proxy in an SKT commercial system

The sidecar proxy approach, as shown in Figure 6, does reduce the burden to the microservice business logic of handling communications, traffic management, load balancing, circuit breaking, scaling, observability and monitoring. However, it has one significant problem, which is increased latency in pod-to-pod communications. Reducing the latency while keeping the cloud-native flexibility in the core system is one of the important research areas that needs to be studied, especially from the MNO's perspective to provide low latency end-to-end connectivity service to end users.

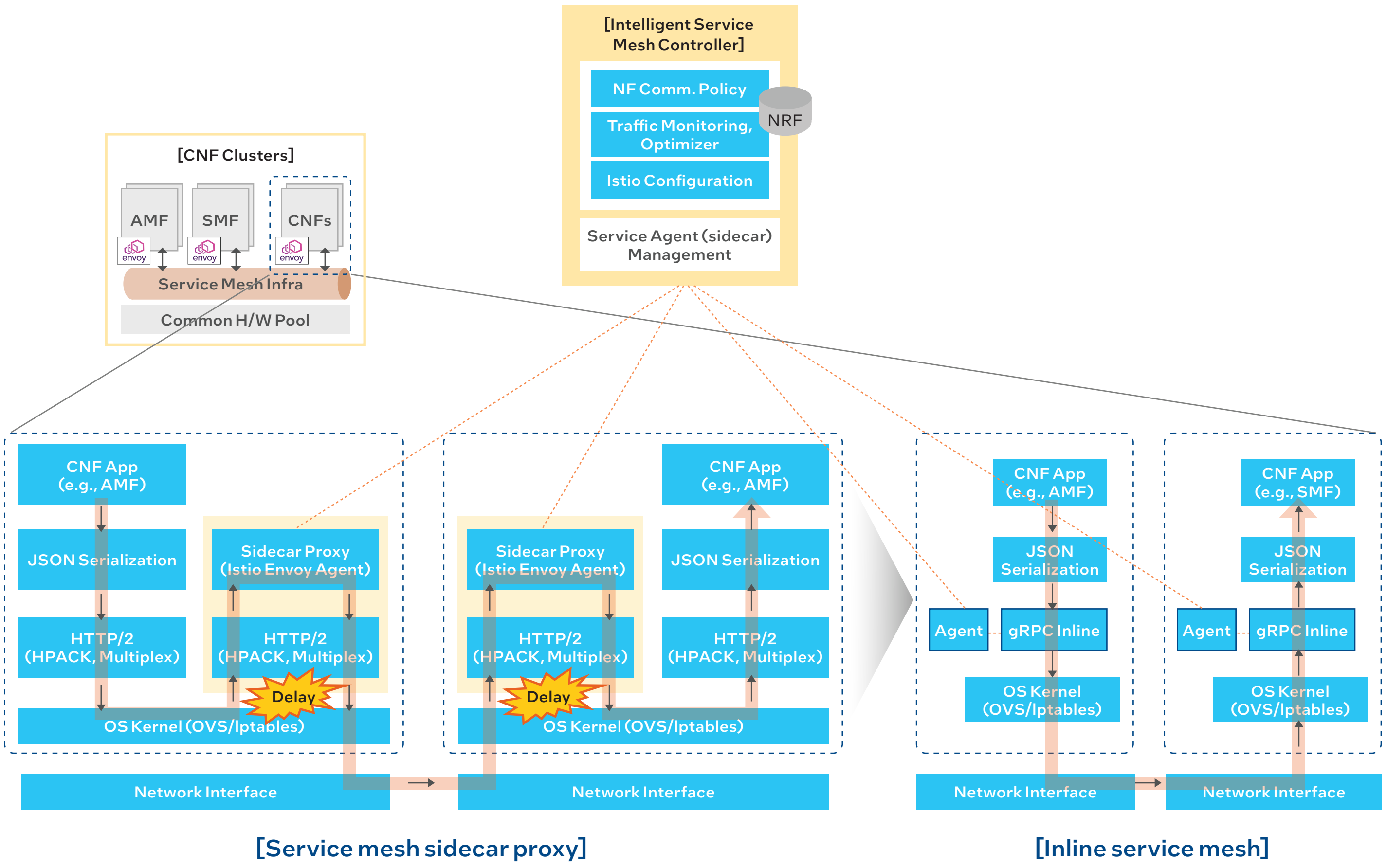

Figure 6. Moving from a service mesh with sidecar proxy to inline design

Kernel IP tables are manipulated to redirect incoming TCP/IP traffic to a sidecar proxy, which in turn executes routing decisions based on xDS which is in turn controlled by Istio. A sidecar proxy introduces extra kernel-space processing, which adds to latency incurred for the NF's pod-to-pod communications. See the article titled, "Sidecar injection, transparent traffic hijacking, and routing process in Istio explained in detail,"[4] for a detailed description of the Istio routing process for an injected service mesh sidecar proxy.

On the other hand, for an inline approach without a sidecar proxy, the logic executing the routing decisions based on xDS is in the application container, thereby making the redirection of TCP/IP traffic unnecessary and reducing the extra kernel-space processing. Thus, reduced latency is expected for the HTTP/2 messages moving in and out of pods.

Support has been added to gRPC for an xDS interface which enables it to interact with an Istio service mesh controller. Likewise, support has also been added to Istio for deployment of pods in Kubernetes with a gRPC agent which brokers an Istio controller interface for application containers.

In order to make use of the Istio gRPC-based inline service mesh feature, applications must be compiled with gRPC libraries in order to interact with the Istio gRPC agent. While this removes the need for a sidecar proxy, it may require the application to be adapted to use gRPC for communication with other application pods.

The remainder of this paper describes an example of how 5GC control plane CNFs may be adapted to use an inline service mesh. Furthermore, the benefit of an inline service mesh to 5GC control plane performance and efficiency is proven with empirical benchmarks taken from a lab deployment the Intel FlexCore reference 5GC cloud-native stack.

2 Adapting 5GC CNFs for an inline service mesh

FlexCore is Intel's cloud-native 5G Core, which was used as an example of how to adapt such a stack for an inline service mesh. This was then evaluated to determine the performance and efficiency benefits of such an inline service mesh when compared with the predominant sidecar proxy approach.

The following is a description (from top to bottom) of the layers of the reference 5G Core stack, as shown below:

- MANO (management and orchestration) layer, which is composed of a GUI for deployment of 5GC CNFs.
- 3GPP (Release 16.2) standards-compliant 5GC NFs developed with a microservices design from the start.
- Common integration layer (CIL) for abstraction of 5GC CNFs from south-bound PaaS and north-bound MANO.
- PaaS based on Cloud Native Computing Foundation (CNCF)-certified open-source cloud-native software for platform services such as logging (Elasticsearch, Fluentd, Kibana), tracing (Jaeger), monitoring (Prometheus and Grafana) and service mesh (Istio, Envoy).
- CaaS also based on CNCF-certified open-source cloud-native software for containerization such as Kubernetes for orchestration, Helm for package management, Calico and Multus for cloud-native networking, Docker as a container engine and containerd for the container runtime.

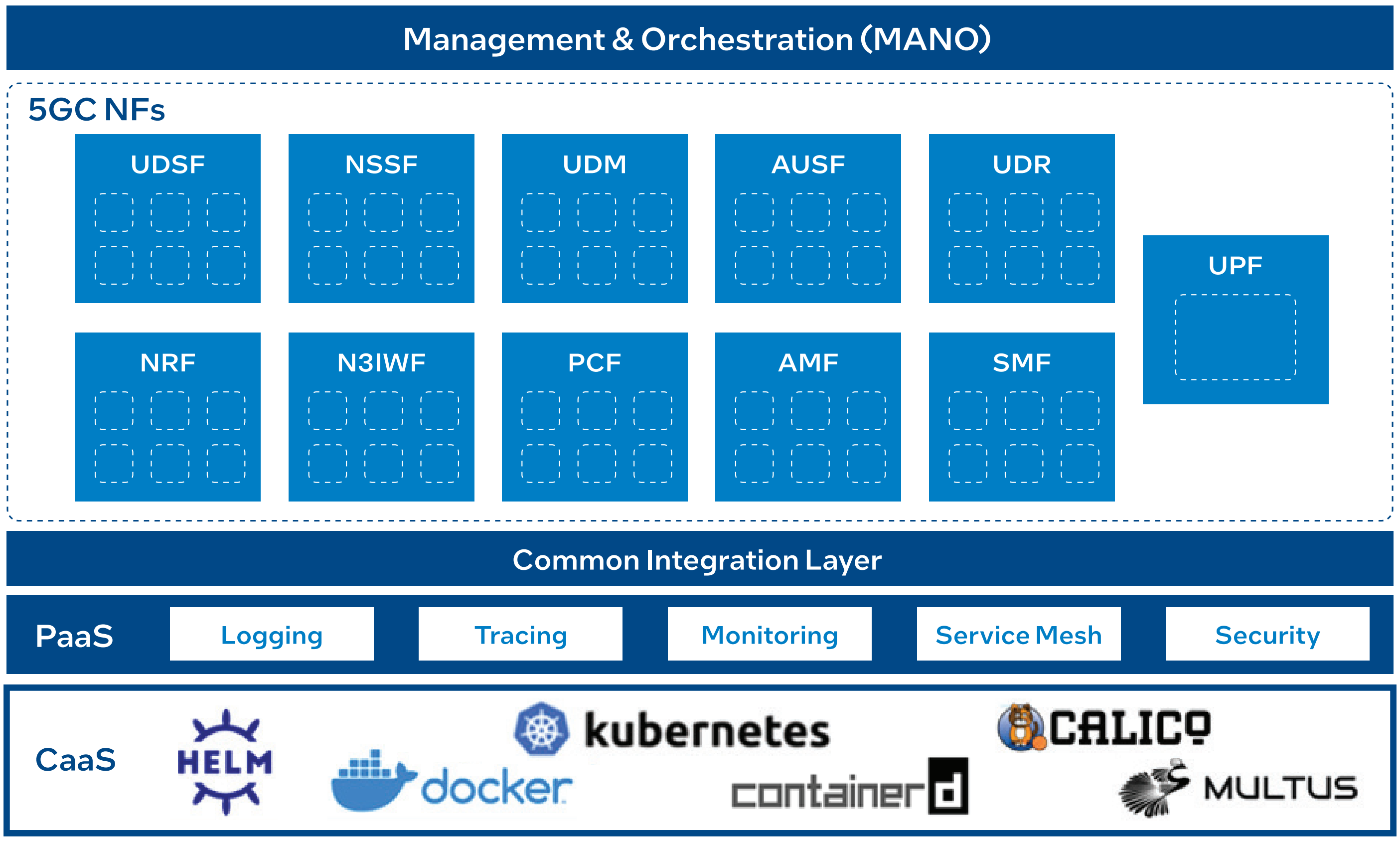

Figure 7. Overview of the FlexCore reference cloud-native 5G core stack

2.1 Design and implementation of a gRPC-based inline service mesh

The following are the key steps required to adapt the reference 5G Core stack to use an inline service mesh:

1. Modified 5GC NF microservice message transfer implementation to use gRPC and its xDS feature.
   - Implemented a gRPC shim layer for this to avoid any modification to application code.
   - Included support to optionally use HTTP/2 or gRPC through runtime environment variables.

NOTE: If an NF microservice already supports gRPC, it may still need to be modified to enable use of the gRPC xDS feature to enable Istio inline service mesh support.

2. Modified the PaaS layer to use an Istio version with support for a gRPC-based inline service mesh.
   - The original cloud-native platform was using Istio v1.8.2, so that had to be upgraded to Istio v1.13.0 for a gRPC-based inline service mesh.
3. Modify 5G Core deployment helm charts to enable 5GC NF microservice pods to use the gRPC-based inline service mesh.
   - Modified microservice pod spec Istio annotations to use Istio grpc-agent instead of sidecar as shown below. The holdApplicationUntilProxyStarts annotation is also required for a gRPC-based inline service mesh.

Istio annotation for use of Envoy sidecar proxy:

```
sidecar.istio.io/inject: "true"
```

Istio annotation for use of a gRPC-based inline service mesh:

```
inject.istio.io/templates: grpc-
agent

proxy.istio.io/config:
'{"holdApplicationUntilProxyStarts":
true}'
```

Figure 8 below is a high-level view of how a gRPC-based inline service mesh was implemented for 5G Core CNF pods. A gRPC shim layer was implemented to avoid any modification to application code and thus reduce the effort needed to support gRPC.

The following are the key points of the design:

- The API presented by the HTTP/2 upper layer to NF microservice applications is unchanged.
- The HTTP/2 upper layer is modified to support gRPC message transfer using a gRPC shim as well as HTTP/2 message transfer using the nghttp2 library.
- The gRPC shim encapsulates the gRPC core and presents an API to the HTTP/2 transport upper layer for gRPC client and server transactions.
- The gRPC shim manages the xDS interface with the Istio service mesh controller.
- gRPC metadata is used to transport 3GPP SBI (HTTP/2) headers.
- The SBI HTTP/2 payload, e.g., MIME, JSON or Protobuf, is copied to gRPC messages as binary data.

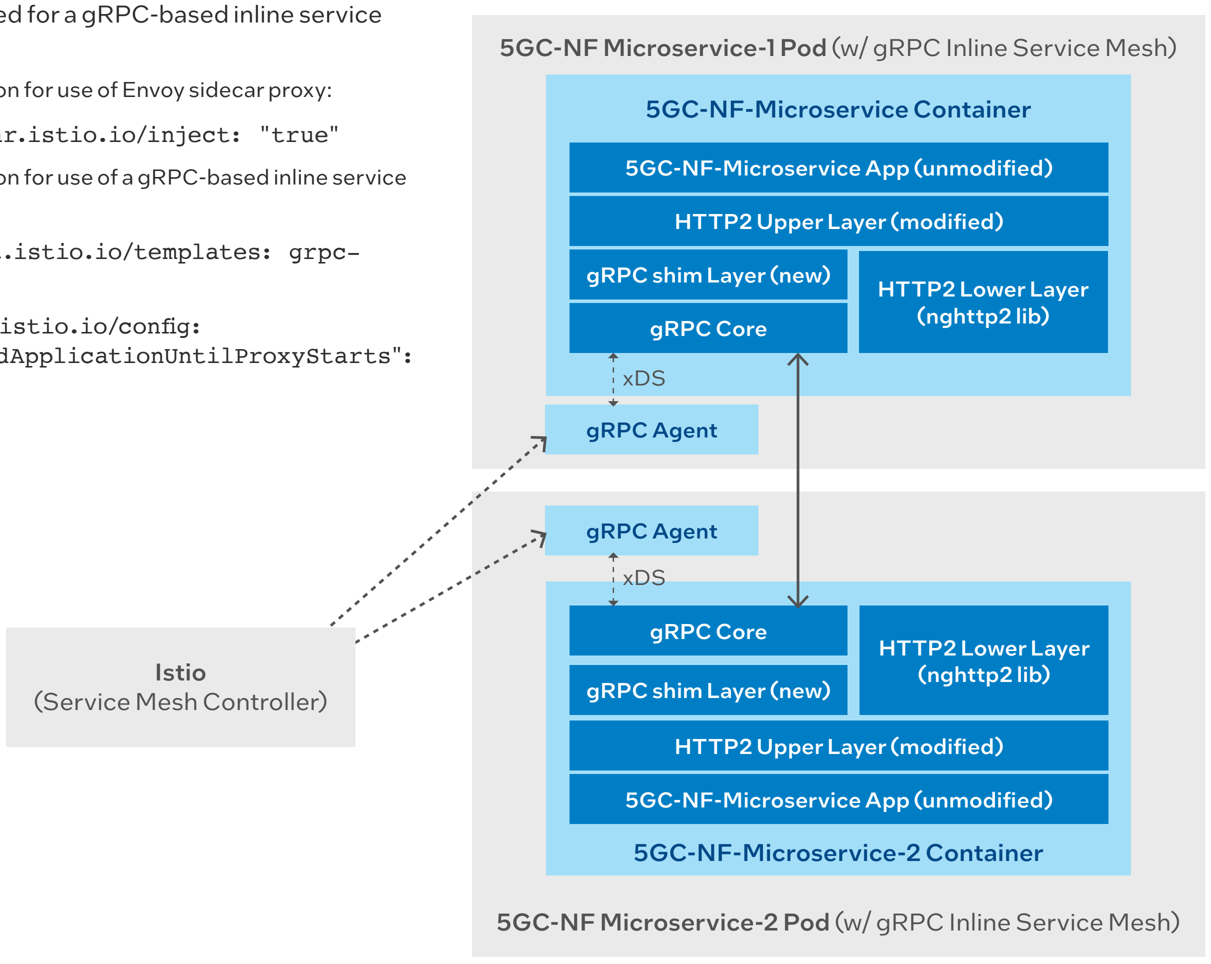
5GC-NF Microservice-1 Pod (w/ gRPC Inline Service Mesh)
5GC-NF-Microservice Container
5GC-NF-Microservice App (unmodified)
HTTP2 Upper Layer (modified)
gRPC shim Layer (new)
gRPC Core
HTTP2 Lower Layer (nghttp2 lib)
xDS
gRPC Agent
gRPC Agent
xDS
gRPC Core
gRPC shim Layer (new)
HTTP2 Lower Layer (nghttp2 lib)
HTTP2 Upper Layer (modified)
5GC-NF-Microservice App (unmodified)
5GC-NF-Microservice-2 Container
5GC-NF Microservice-2 Pod (w/ gRPC Inline Service Mesh)
Istio (Service Mesh Controller)

Figure 8. 5GC microservice design to support an inline service mesh

3 Evaluation

The Intel FlexCore reference cloud-native 5G Core stack with Spirent Landslide 5G traffic generator was used to evaluate the benefit of an inline service mesh for 5GC control plane performance. This provides a realistic benchmark using a commercial-grade 5G Core control plane.

3.1 Benchmarking setup and methodology

Figure 9 shows a high-level overview of the benchmarking setup in Intel's lab environment. A Spirent Landslide 5G traffic generator was used with a two-node OCP cluster. Both OCP worker nodes are using 4th Generation Intel® Xeon® Scalable Processors and hosting CaaS/PaaS and 5GC NF microservices. For worker-node configuration details, refer to Appendix A: Intel FlexCore benchmarking environment details.

As described in the previous section, 5GC NF microservices were adapted to use a gRPC-based inline service mesh for empirically benchmarking the inline service mesh approach for 5G Core deployments.

The inline service mesh was enabled for SMF microservices, and the impact on transaction latency between SMF-Gateway and SMF-PDUSession microservices was measured. Specifically, the latency of Session-Management Context-Create requests from SMF-Gateway microservice transmission to the SMF-PDUSession microservice receipt was measured, as shown in Figure 10.

Latency was measured by timestamping in SMF-Gateway and SMF-PDU-Session microservices. The SMF-Gateway request-sent timestamp is passed as a header field and used at the request-received point in the SMF-PDUSession microservice to measure latency between the transmission and receipt of this control plane message between microservices. Thirty SMF-PDUSession pod replicas were used to represent a typical configuration for 5GC SMF processing.

For context, the round-trip latency of the Session-Management Context-Create request to the response is approximately 50 ms,[5] as measured in the AMF-Comm microservice shown below. Of this latency, the time spent in the microservice application logic is approximately 60%, and the time spent in communications between microservices is approximately 40%.

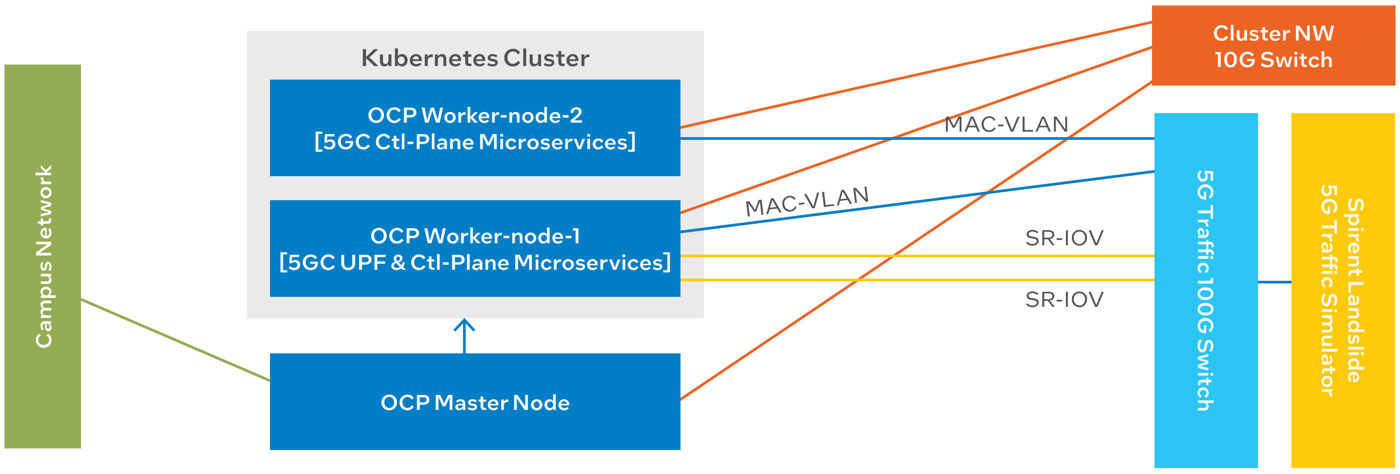
Campus Network
Kubernetes Cluster
OCP Worker-node-2 [5GC Ctl-Plane Microservices]
OCP Worker-node-1 [5GC UPF & Ctl-Plane Microservices]
OCP Master Node
Cluster NW 10G Switch
MAC-VLAN
MAC-VLAN
SR-IOV
SR-IOV
5G Traffic 100G Switch
Spirent Landslide 5G Traffic Simulator

Figure 9. Cloud-native 5G core performance benchmarking setup

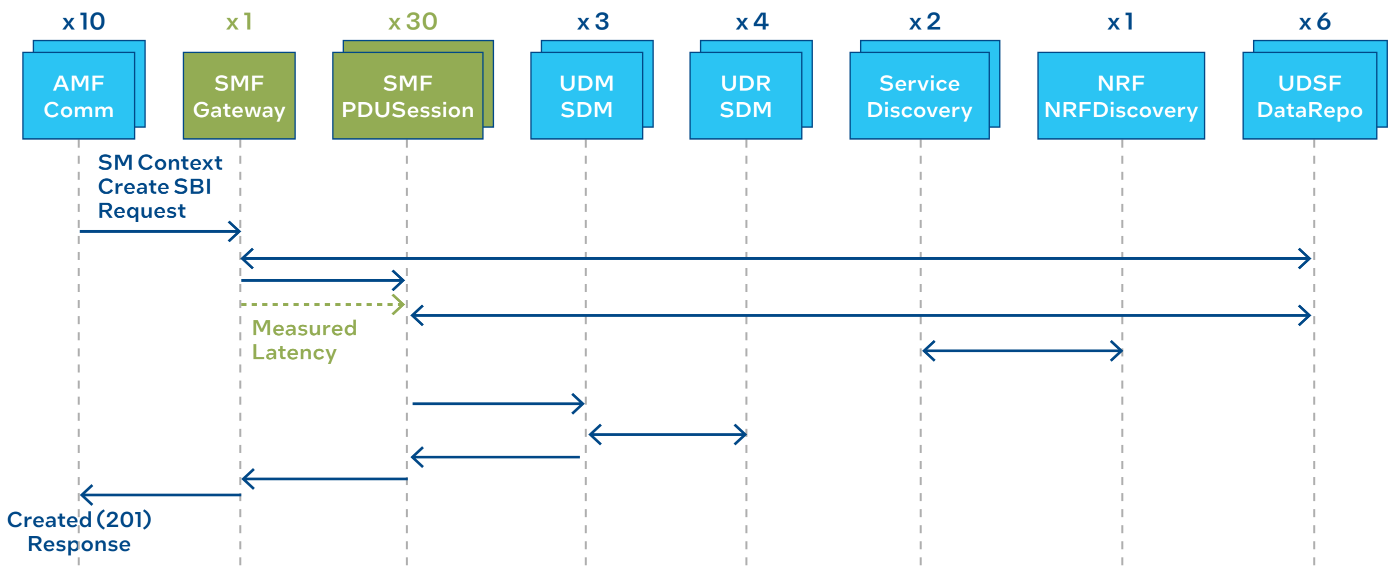

Figure 10. Measuring latency between SMF GW and PDU-Session microservices

3.2 Benchmarking results

Performance of the SMF was compared for gRPC-based control plane transactions using an Envoy sidecar proxy versus an inline service mesh. The SMF was configured with 30 x PDU-Session processing pods.

Latency and CPU usage were measured for an increasing Session-Management Context-Create request rate which was driven by a Spirent Landslide 5G traffic generator.

Figure 11 shows significant reduction of control plane transaction latency and CPU usage for an inline service mesh in the case of a high number of SMF-PDU-Session microservice instances.

At high request rates (1500 requests/sec), the latency is over **3x higher** with an Envoy proxy as compared with an inline service mesh. It was also observed that latency measurements for the Envoy proxy exhibit **higher jitter levels** than an inline service mesh.

It should be noted that the maximum traffic rate of 1500 SMF requests per second was based on deployment of a single *SMF-Gateway* microservice pod which routes SMF SBI messages to an *SMF-PDUSession* microservice deployment with thirty pod replicas. By configuring the *SMF-Gateway* microservice deployment for multiple pod replicas and scaling the *SMF-PDUSession* microservice pod's proportionately, it is reasonable to assume that, with sufficient CPU and memory resources available, that the request rate should scale proportionately and maintain the same latency level as shown above.

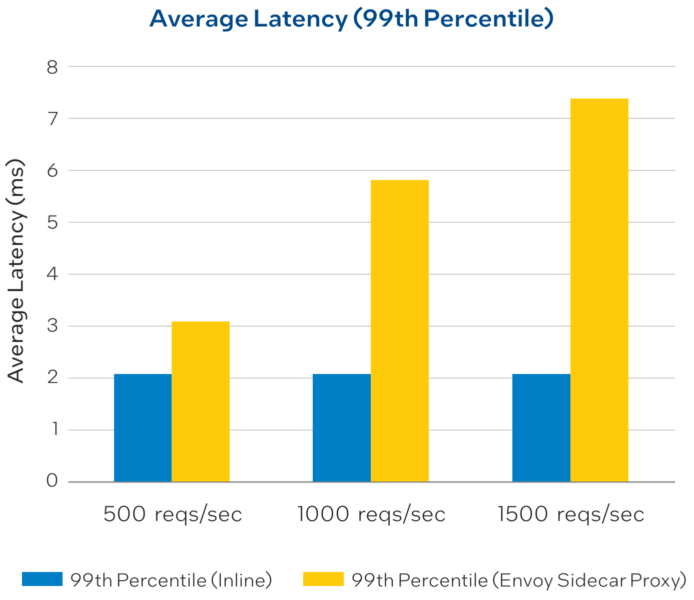

Figure 11. Average latency between SMF gateway and PDUSession microservices

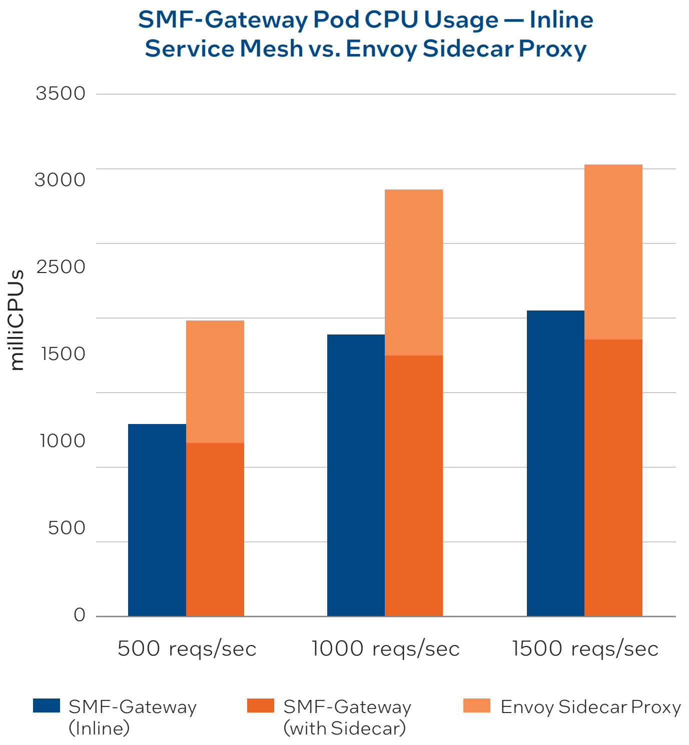

Figure 12. Average CPU usage of an SMF-Gateway microservice pod — inline service mesh vs. envoy sidecar proxy

As shown in Figure 12, the total SMF-Gateway CPU usage is approximately **1 virtual core more** with an Envoy proxy as compared to an inline service mesh. It should also be noted that an SMF-Gateway microservice pod with an Envoy proxy requires reservation of additional cores based on the Envoy concurrency setting, so that fewer cores are available for other pods.

4 Summary and recommendations

With a rise of emerging services such as XR/metaverse, satellite, AI/ML, energy-efficient networking and the transition to 6G Core networks, NFs will likely need to support new service features. With the increasing demand for these new service features, it is expected that CNFs are going to be decomposed into even more microservices, which perform 5GC processing via a unified communication infrastructure for flexibility and reliability. The service mesh that connects these microservices will thus be critical for performance and may also provide a means to introduce new features such as AI-based analysis and inference for improved reliability.

The 6G Core architecture from the 3GPP is expected to be standardized starting with Releases 20 and 21. Currently much study and research is being conducted into enhancing this.

The following are the key conclusions and recommendations from the work described in this paper by SK Telecom and Intel, which investigated the use of an inline service mesh to reduce control plane latency and CPU usage in cloud-native 5GC deployments.

(1) An inline service mesh reduces control plane latency and CPU usage for cloud-native 5G Core deployments

Based on a comparison of measurements of average latency for proxied and inline service mesh configurations, it was shown that there is a significant reduction in latency and CPU usage for message transfer between two microservices. This is especially so in the case of a gateway microservice forwarding incoming SBI messages to an NF's core processing microservice, such as the SMF-Gateway and SMF-PDUSession communication benchmarked in this paper.

Furthermore, the end-to-end latency reduction for a control plane operation depends on the number of microservices that must be traversed to complete the operation. Each microservice hop benefits from a latency reduction when using an inline service mesh with the result that the aggregate latency reduction is even more significant.

It was also shown that there is significant CPU usage reduction when a service mesh proxy container such as Envoy does not need to be deployed for every 5GC NF microservice container.

While service mesh proxy approaches such as Envoy support a rich set of features such as circuit breaking, tracing, logging and embedding analytics, such features may also be added for a gRPC-based inline service mesh.

(2) A generic message-transfer API should be provided for 5GC NF microservices, abstracted from the underlying protocol such as HTTP REST or gRPC

For the purpose of this paper, 5G Core NF microservices were adapted to use a gRPC-based inline service mesh by implementing a shim for an alternative gRPC message transfer path to the existing nghttp2 library and Envoy proxy path.

The shim approach avoided any change to microservice application code which significantly reduced the effort and time for gRPC integration into the 5G Core stack.

However, such an approach may add additional performance overhead in the gRPC adaptation. A recommended improved approach is to use a generic API that supports independent paths of execution for gRPC and HTTP-REST.

(3) Possible 6G design for an inline service mesh

Although gRPC is based on HTTP/2, a gRPC endpoint cannot communicate **directly** with a plain HTTP-REST endpoint, which means that currently gRPC cannot be used for 3GPP SBI communication between NFs; it can only be used within the boundaries of an NF between its microservices as illustrated in Figure 13.

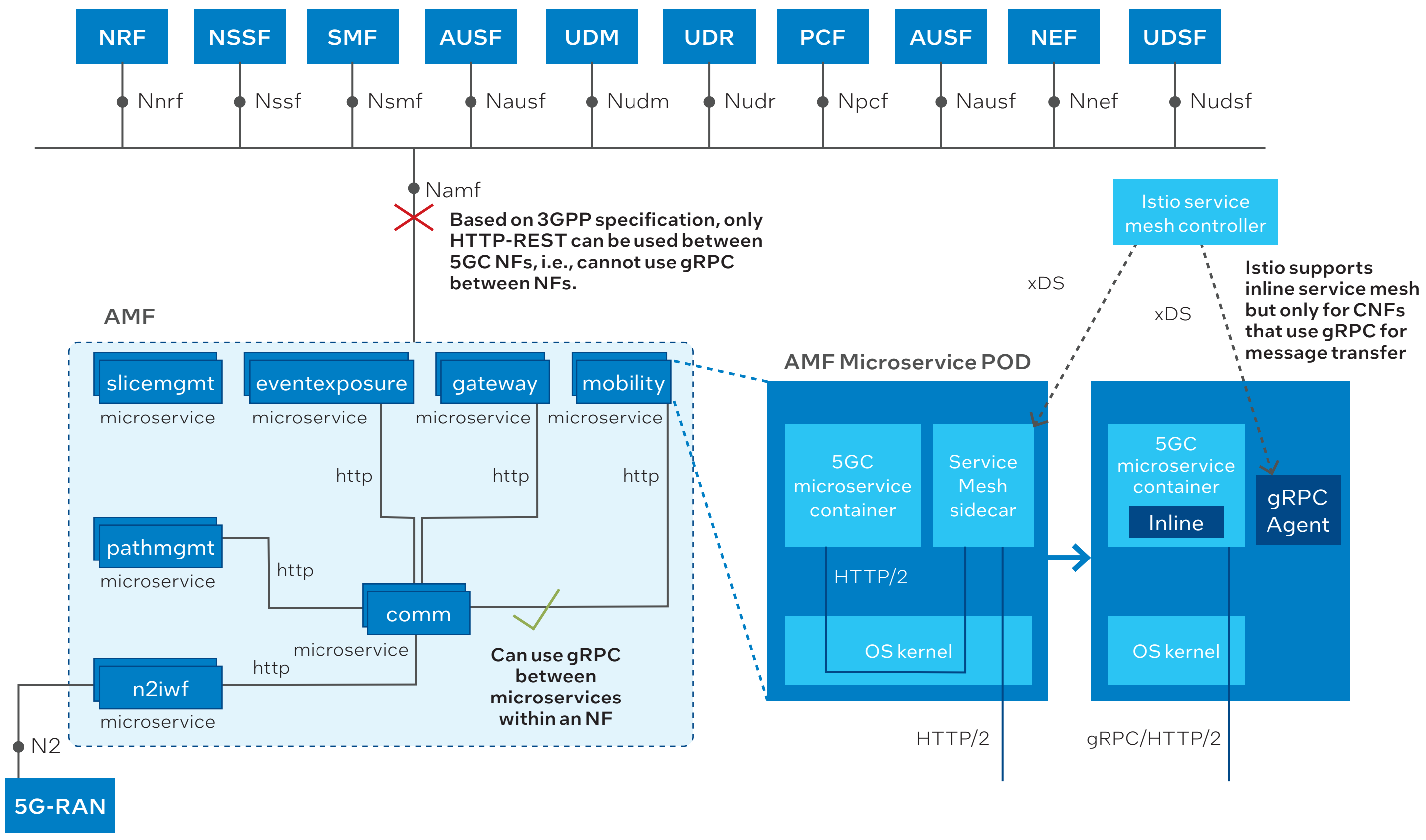
NRF
NSSF
SMF
AUSF
UDM
UDR
PCF
AUSF
NEF
UDSF
Nnrf
Nssf
Nsmf
Nausf
Nudm
Nudr
Npcf
Nausf
Nnef
Nudsf
Namf
Based on 3GPP specification, only HTTP-REST can be used between 5GC NFs, i.e., cannot use gRPC between NFs.
Istio service mesh controller
xDS
xDS
Istio supports inline service mesh but only for CNFs that use gRPC for message transfer
AMF
slicemgmt microservice
eventexposure microservice
gateway microservice
mobility microservice
http
http
http
pathmgmt microservice
http
comm microservice
http
n2iwf microservice
Can use gRPC between microservices within an NF
N2
5G-RAN
AMF Microservice POD
5GC microservice container
Service Mesh sidecar
HTTP/2
OS kernel
5GC microservice container
Inline
gRPC Agent
OS kernel
HTTP/2
gRPC/HTTP/2

Figure 13. Using an inline service mesh in a 5G Core control plane deployment

While this offers reasonable latency and CPU usage benefits, this could be improved further if interoperability was possible between gRPC and HTTP-REST endpoints and inline service mesh communication was supported for HTTP-REST-based microservices. Figure 14 shows a prototype design for a potential solution where gRPC/HTTP/2 mapping could be built into 5G Core microservices to enable inline service mesh communication for all control plane messages.

This is intended as a proposal for further inline service mesh development which builds on the initial work described in this paper. It is proposed to develop a library for inline service mesh communication which could be built into 5GC NF microservices.

The following are some key points related to this design proposal:

- A generic API is presented to 5GC control plane microservices, i.e., no HTTP/2 specifics.
- Microservices will have the capability to use gRPC or HTTP/2 for client requests depending on the target network function.
- The message transfer upper layer provides adaptation to the required protocol and higher-level functionality to augment the grpc-core and HTTP/2 third-party libraries.
- Libraries from the open-source gRPC project may be used for the gRPC lower layer.
- An open-source HTTP/2 library such as nghttp2 may be used for the HTTP/2 lower layer.
- A thin layer is required to listen for incoming server requests and to map to HTTP/2 or gRPC request handling, e.g., based on content type.

NOTE: This may also be possible by using separate ports for HTTP/2 and gRPC listener threads, assuming a mechanism is in place for external NFs to determine which port to use. In this case, the request mapping layer is not required.

- Support must be added for HTTP/2-based inline service mesh communication — the chosen HTTP/2 implementation must interact with a service mesh agent to broker the xDS interface with a service mesh controller such as Istio, similar to how this is supported by gRPC.

(4) Another consideration for 6G cloud-native architecture — adding intelligence via the service mesh

The 6G Core will become a highly distributed AI structure as network data analytics function (NWDAF) service is inherent in each NF. Logical analysis and machine learning components will be two essential components for each NF, and an intelligence plane will be constructed through interconnection among NFs.

Data collection, message processing and storage functions will be performed by the NFs themselves. With an intelligence plane, information from all network layers can be collected, analyzed and inferred, and AI-based data learning can be performed with ultra-low latency across the entire network area.

Additionally, AI-based analysis and inference results are shared through the intelligence plane to ensure high reliability, and parameters and network configuration information that were previously managed statically are now dynamically auto-configured and optimized.

Further study in this area and the way that the service mesh is leveraged also has great significance for 6G.

Looking forward, Intel and SK Telecom will continue to jointly collaborate on software and hardware R&D toward optimal architecture of the B5G and 6G Core.

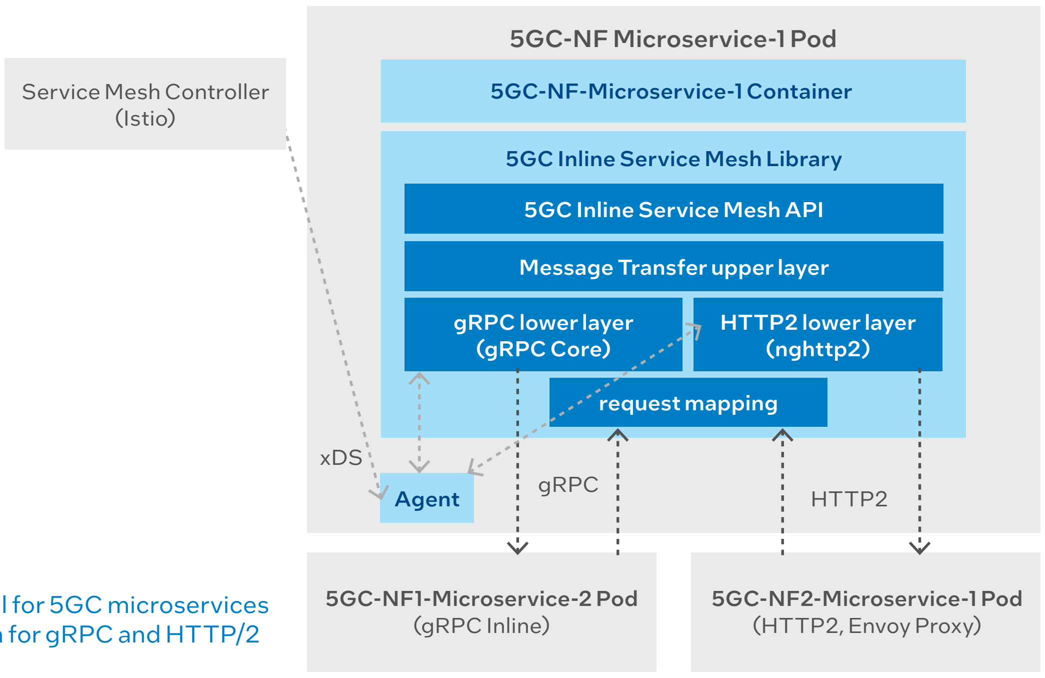
Service Mesh Controller (Istio)
5GC-NF Microservice-1 Pod
5GC-NF-Microservice-1 Container
5GC Inline Service Mesh Library
5GC Inline Service Mesh API
Message Transfer upper layer
gRPC lower layer (gRPC Core)
HTTP2 lower layer (nghttp2)
request mapping
xDS
Agent
gRPC
HTTP2
5GC-NF1-Microservice-2 Pod (gRPC Inline)
5GC-NF2-Microservice-1 Pod (HTTP2, Envoy Proxy)

Figure 14. Design proposal for 5GC microservices with an inline service mesh for gRPC and HTTP/2 message transfer

5 Appendix A: Intel FlexCore benchmarking environment details

Table 1. FlexCore worker-node configuration

Hardware	
Platform	Intel® Server System M50FCP2UR (Fox Creek Pass)
CPU	4th Gen Intel Xeon 6438Y+ Dual Processor, 2.0 GHz, 32 Cores
Memory	8 x 32GB DDR5 (256 GB Total)
Hard Drive	2 x 1TB Intel SSD
Network Interface Card	2 x Intel Ethernet Converged Network Adapter XXV710 25GbE Dual-port
Software	
Host OS	RHEL 8.6, Linux Kernel 4.18
CaaS/PaaS	RedHat OpenShift 4.12, Kubernetes 1.25
Service Mesh	Istio 1.14.1, gRPC 1.43
FlexCore SW	23.07 5GC NF images and deployment charts

Acronyms

3GPP	Third Generation Partnership Project (for Development of Mobile Core Technical Specifications)
5G	Fifth Generation (of Mobile Core Standards)
5GC	5G Core
6G	Sixth Generation (of Mobile Core Standard)
AI	Artificial Intelligence
AMF	Access & Mobility Management Function
B5G	Beyond 5G
CaaS	Containers as a Service
CI/CD	Continuous Integration / Continuous Development
CIL	Common Integration Layer
CNCF	Cloud Native Computing Foundation
CNF	Container Network Function
CNI	Container Network Interface
CPU	Central Processing Unit
gRPC	Google Remote Procedure Call (Protocol)
HTTP/2	Hypertext Transfer Protocol 2
KPI	Key Performance Indicators
L7	Layer 7 (OSI Model)
LTE	Long Term Evolution
ML	Machine Learning
MANO	MANagement and Orchestration
MNO	Mobile Network Operator
NF	Network Function
NRF	Network Repository Function
NSA	Non-Stand-Alone
NWDAF	NetWork Data Analytics Function
OCP	Openshift Container Platform
PaaS	Platform as a Service
PDU	Packet Data Unit
PNF	Physical Network Function
RHEL	RedHat Enterprise Linux
SBA	Service Based Architecture
SBI	Service Based Interface
SCP	Service Communication Proxy
SDO	Standards Development Organization
SDM	Subscriber Data Management
SMF	Session management Function
TCP/IP	Transmission Control Protocol / Internet Protocol
TEM	Telecoms Equipment Manufacturer
TTM	Time To Market
UDM	Unified Data Management
UDR	Unified Data Repository
UDSF	Unstructured Data Service Function
UPF	User-Plane Function
VNF	Virtual Network Function
xDS	(List of) Discovery Services

intel + SK telecom

Legal Notices and Disclaimers

[1] Up to 70% latency reduction is based on data presented in Figure 11 : 7ms latency is reduced to 2ms at 1500 reqs/sec (= approx. 70%).
This specifically relates to the 99th percentile average latency of SM-Context-Create requests between SMF Gateway and PDU-Session microservices at 1500 requests/sec

[2] Up to 33% reduction in Gateway CPU usage is based on data presented in Figure 12 : 3000 milli-cpu's is reduced to 2000 milli-cpu's at 1500 reqs/sec (= 33%)
This specifically relates to the average CPU usage of the SMF Gateway micro-service at 1500 PDU-session establishment reqs/sec.

[3] SK Telecom 6G white paper, https://www.sktelecom.com/en/press/press_detail.do?idx=1575

[4] Jimmy Song, "Sidecar injection, transparent traffic hijacking, and routing process in Istio explained in detail," https://jimmysongio.medium.com/sidecar-injection-transparent-traffic-hijacking-and-routing-process-in-istio-explained-in-detail-d53e244e0348

[5] 50ms response time for SessionManagement Context-Create responses as measured in the test environment described in Figures 9 and 10.

0124/BB/MESH/PDF 358251-001US