PICCOLO

CELTIC-NEXT

Deliverable D1.1

Use Cases, Application Designs and Technical Requirements

Editor:	Dennis Grewe – Bosch Research, Robert Bosch GmbH
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public
Contractual delivery date:	March 31, 2021
Actual delivery date:	March 30, 2021
Suggested readers:	Representatives of network operators, industry automation, vision processing, car manufacturers, automotive part suppliers
Version:	1.0
Total number of pages:	46
Keywords:	In-Network Compute; Computing on Network Infrastructures; Distributed Computing; 5G; Vision Sensing; Industrial Internet of Things; Connected Vehicles; Network Operator

Abstract

This document presents the Piccolo use cases from the domains of telecommunications, vision sensing, industrial Internet of things and connected vehicles. Based on the introduction of the use cases, limitations of today's solutions are presented and technical directions for the Piccolo system are discussed. Furthermore, we describe the application designs and technical requirements for the Piccolo system based on the use cases.

About the Project

Project Coordinator: Philip Eardley, BT
Technical Manager: Dirk Kutscher, University of Applied Sciences Emden/Leer
Start date of project: October 2020
Duration: 24 months

Disclaimer

This work was done within the EU CELTIC-NEXT project PICCOLO (Contract No. C2019/2-2). The project is supported in part by the German Federal Ministry of Economic Affairs and Energy (BMWi) and managed by the project agency of the German Aerospace Center (DLR) (under Contract No. 01MT20005A). The project also receives funding awarded by UK Research and Innovation through the Industrial Strategy Challenge Fund. The project is also funded by each Partner.

Impressum

[Full project title] Piccolo: In-Network Compute for 5G Services
[Short project title] PICCOLO
[Number and title of work-package] WP1. Applications, Use-Cases, Proof-of-Concept Prototypes
[Number and title of task] T1.1. Use Cases, Application Designs and Technical Requirements
[Document title] D1.1 Use Cases, Application Designs and Technical Requirements
[Editor: Name, company] Robert Bosch GmbH
[Work-package leader: Name, company] Dennis Grewe, Robert Bosch GmbH

Copyright notice

Executive summary

This document presents Project Piccolo's work on Use cases and requirements. It describes a public report of the Piccolo project.

The document presents Piccolo use cases considered from a range of potential use cases. We have looked at 4 broad use cases:

- **Vision Processing** – led by Sensing Feeling
- **Connected and Automated Driving** – led by Robert Bosch GmbH
- **Network Operations and Management** – led by BT.
- **Smart City and Industry** – led by Peer Stritzinger GmbH

Based on the use case descriptions, we analyse the requirements under the headings:

- functional requirements and features for a Piccolo component (WP2) and a Piccolo infrastructure (WP3)
- non-function requirements and features for a Piccolo component (WP2) and a Piccolo infrastructure (WP3)
- structural & management requirements – the spread of functionality between the application and 'base' functionality provided by Piccolo, incl. aspects such as monitoring, policy and intent-based management of the overall system

Finally, we selected directions for two shorter-term Proof-of-Concept prototypes of the domain vision sensing and automotive, as well as the industrial Internet of things.

List of Authors

Company	Author
ARM Ltd.	Chris Adeniyi-Jones
British Telecommunications plc	Philip Eardley, Andy Reid, Peter Willis
Fluentic Networks Ltd.	Ioannis Psaras, Alex Tsakrilis
InnoRoute GmbH	Andreas Foglar, Marian Ulbricht
Robert Bosch GmbH	Dennis Grewe, Naresh Nayak, Uthra Ambalavanan
Sensing Feeling Ltd.	Jag Minhas, Dan Browning, Chris Stevens
Stritzinger GmbH	Peer Stritzinger, Sascha Kattelmann, Stefan Timm, Mirjam Friesen
Technical University Munich	Jörg Ott, Raphael Hetzel, Nitinder Mohan
University of Applied Science Emden/Leer	Dirk Kutscher, Laura al Wardani

Table of Contents

Executive summary . . . 2
List of Authors . . . 3
List of Figures . . . 6
Abbreviations . . . 7

1 Introduction 8

2 Motivation 10
2.1 Feasibility Discussion . . . 11
2.2 Example . . . 12
2.3 Technical Motivations and Challenges . . . 13

3 Piccolo Use Cases 17
3.1 Stakeholder . . . 17
3.2 Telco: CV-OAM: Continuous Verification Operations, Administration and Maintenance 18
3.2.1 Problem statement . . . 19
3.2.2 Potentials for in-network compute . . . 20
3.2.3 Key requirements and challenges . . . 20
3.2.4 Stakeholders . . . 21
3.2.5 Other use cases considered . . . 21
3.3 Telco: Radical network use case . . . 22
3.3.1 Problem statement . . . 22
3.3.2 Potentials for in-network compute . . . 23
3.3.3 Key requirements and challenges . . . 23
3.3.4 Stakeholders . . . 24
3.4 Vision Sensing: In-Vehicle, Real-Time Behavioural Risk Monitoring . . . 25
3.4.1 Problem Statement . . . 25
3.4.2 Potentials for In-Network Compute . . . 26
3.4.3 Key Requirements & Challenges . . . 26
3.4.4 Stakeholders . . . 27
3.4.5 Other use cases considered . . . 27
3.5 Industrial Applications: Plug & Produce for Industrial Production Systems . . . 28
3.5.1 Problem Statement . . . 28
3.5.2 Potentials for In-Network Compute . . . 30
3.5.3 Key Requirements & Challenges . . . 30
3.5.4 Stakeholders . . . 31
3.5.5 Other use cases considered . . . 31
3.6 Automotive: Offloading Driving Logic for Automated Driving Vehicles . . . 32
3.6.1 Problem Statement . . . 32
3.6.2 Potentials for In-Network Compute . . . 33
3.6.3 Key Challenges . . . 34

3.6.4 Stakeholders 34
3.6.5 Other use cases considered 35

4 Application Requirements for Piccolo 36
4.1 Requirements for Piccolo Node 36
4.2 Requirements for Piccolo Infrastructure 37
4.3 Security and Privacy Requirements 39
4.4 Requirements for Resource Management 40

5 Conclusion 42

List of Figures

1 Latency/Capacity Requirements for Different Applications 12
2 Sensing Feeling Vision Processing 13
3 Visual processing of images sent from the vehicle to the cloud 26
4 Conveyor belt based production line using IEC 61499 28
5 IEC 61499 IDE showing a composite function block 29
6 Decomposed driving logic to be offloaded from the vehicle 33
7 Piccolo scenario in which driving logic can be offloaded into the infrastructure . . . 34
8 Example: Compute and Communication Infrastructure for Vision Processing 38

Abbreviations

ADAS Advanced Driving Assistance Systems

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

ASIL Automotive Safety Integrity Level

CDN Content Delivery Network

CPU Central Processing Unit

CV Continuous Verification

CV-OAM Continuous Verification Operations, Administration and Maintenance

GPU Graphics Processing Unit

FPGA Field Programmable Gate Array

ICN Information-Centric Network

IoT Internet of Things

LDM Local Dynamic Map

MANO Management and Orchestration

MEC Mobile Edge Computing

MES Manufacturing Execution System

MPLS Multiprotocol Label Switching

MTU Maximum Transmission Unit

NAT Network Address Translation

NFV Network Functions Virtualisation

OAM Operations, Administration and Maintenance

PLC Programmable Logic Controller

RTT Round Trip Time

RMI Remote Method Invocation

SDN Software-Defined Networking

SLA Service Level Agreement

VPE Visual Processing Engine

VPN Virtual Private Network

1 Introduction

Piccolo will develop new solutions for in-network computing that remove known and emerging deficiencies of edge/fog computing. This document presents Project Piccolo's initial work on use cases and requirements. We stress that these are interim results produced half way through the first year of a 2 year project. Our motivation is that the centralised cloud computing model in use today has difficulty handling new and emerging applications. Ever-more powerful user and IoT devices are producing enormous amounts of data - too much to send into the cloud for centralised processing, and further the round trip time is too large for the stringent latency requirements of some applications. Also, there are increasing concerns about leaving data privacy at the mercy of big cloud operators. Shifting from centralised to in-network compute can alleviate these concerns and thereby open up new horizons for application development and create new infrastructure markets. Piccolo is also motivated by the desire that network operations become much more automated. Optimisation, maintenance, verification and so on will happen with closed-loop, autonomous control on short timescales, whilst taking account of application requirements, policy constraints and resources. These motivations are explored in more detail in Chapter 2.

The project's first activity was to consider a wide range of potential use cases for in-network computing, under four broad categories that reflect our industrial Partners' key interests:

- **Vision Processing** – led by Sensing Feeling
- **Connected and Automated Driving** – led by Robert Bosch GmbH
- **Network Operations and Management** – led by BT
- **Smart City and Industry** – led by Peer Stritzinger GmbH

The Annex **??** presents this wide selection of use cases.

In Chapter 3 of this document we present a narrowed list of the most promising use cases. We selected each one on commercial and technical grounds:

- exploitability, especially as judged by the lead Partner for the use case and thus reflecting their commercial plans and beliefs about how 5G services are developing
- skills match, so the Consortium collectively has a reasonable chance of fulfilling the work-to-do
- technical benefit, an informal assessment of where in-network compute can offer a real advantage compared to normal cloud computing and (emerging) 5G Mobile Edge Computing (MEC).

The selected, specific use cases will be developed during the remainder of the Piccolo project, with the aim of at least two proof-of-concept prototypes:

- In-vehicle, real-time behavioural risk monitoring. By combining in-vehicle metrics such as

speed and acceleration with scene telemetry (driver fatigue, nearby pedestrians - derived from vision processing), technology can help us provide a real time risk measurement for a vehicle and display this information directly to the driver or to a fleet manager potentially thousands of miles away.

- Offloading driving logic for automated driving vehicles. High costs arise from the safety-criticality combined with the increasing complexity of the software. Our use case proposes to decompose the driving logic into smaller micro-functions and to offload some of them into the edge infrastructure outside the vehicle.

- Adaptive production line setups in the context of Industry 4.0. Conveyor belt networks in modern production facilities are driven by distributed software systems which need to rapidly adjust to changing production targets. We introduce a technology stack for Plug & Produce capable industrial production systems.

- More flexible 'Continuous Verification' of services and networks, as part of a telco operator's Operations, Administration and Maintenance. It adapts to the specific Service Level Agreements, policy, network conditions, and the results of previous tests.

In addition, we will research a "radical" telco network that has been designed from scratch according to Piccolo principles so that it is as automated as possible (with human intervention confined to those things that require human astuteness). Output will be somewhat longer-term research (theoretical and practical), rather than prototyping.

Chapter 4 presents the "features" that a Piccolo system should support. We derived them by thinking about what is needed to achieve the use cases (both ours and others beyond the scope of our project, for example in the health IoT area), in particular the required functionality and its distribution and management. We are using the "features" in order to guide the work in our two technical work-packages (WP2 on "Piccolo Node" and WP3 on "Piccolo Infrastructure"), which supports both the shorter-term Proof-of-Concept prototypes and the longer-term research activity.

Overall, the Piccolo project seeks a distributed computing platform that can leverage different kinds of underlying infrastructure in order to cater to various business needs and user preferences, and that provides an open platform for future applications. Our vision is that "compute" will become integrated into the network and storage fabric:

Every network node will provide secure processing and storage for third party application and network functions, using a "Functions as a Service" or "serverless" programming paradigm readily accessible to application developers and third party service providers.

Finally, Chapter 5 concludes the document and highlights the key challenges of the use cases and the important requirements for the technical workpackages.

2 Motivation

Piccolo embraces computing as a first-order principle in networked systems, aiming at a novel approach of integrating computing and networking, and to illustrate the approach in several use cases.

Today, services are becoming ever-more dominated by large cloud service providers such as Google that run their own cloud/Content Delivery Network (CDN) infrastructure (e.g., Google Cloud [1]). We expect that, by default, emerging technologies such as cyber-physical connectivity, networked virtual reality, and data analytics will lead to further on-going centralisation of cloud and network infrastructure. This will limit the ability to innovate to the dominant cloud service providers. It also induces technical problems such as:

- Data privacy is at the mercy of a few big cloud operators whose operations are difficult to audit and control.
- High-volume sensing and the huge number of Internet of Things (IoT) devices lead to a ”reverse Content Delivery Networking” problem, potentially exhausting the upload capacity (e.g., [2]).
- The advantage of low latency of next generation 5G networks is lost if requests have to travel deep into the cloud and back.

Companies wanting to deploy innovative distributed applications have a tough choice between staying local (private cloud), relying on limited processing offered by CDNs (mostly static delivery) or deploying in public clouds with the associated privacy, latency and bandwidth limitations.

In the future we expect billions of sensors and IoT devices, each generating and/or collecting potentially large amounts of data. We don’t really want to shift it all to the central cloud, process the data there, and shift the result back to (a subset of) the nodes. Local processing of the sensor information greatly reduces the load on the network, and the shorter, more controlled network path enhances the quality of service. This rationale for Piccolo’s in-network computing is very similar to the reasons why CDN caches are places inside a telco operator’s network – to reduce the operator’s network costs and to improve the customer’s quality of experience.

From a privacy perspective, it is sub-optimal to send off-net the data from all the billions of sensors, especially when much of it will reveal personal information, for example relating to health and location. Further, use cases such as vehicular ”electronic horizon” (e.g., Continental eHorizon [3], or Bosch Connected Horizon [4]) are more powerful if all the nearby vehicles contribute their information to a unified database, but each company must be confident that it is not leaking to its rivals commercially-sensitive details about its operations and users. Piccolo offers a way ahead – the functions can be run locally, under the control of a specific enduser, with only the result (and not the raw data) being shipped out.

Today’s technologies such as Network Functions Virtualisation (NFV) [5], Software-Defined Networking (SDN) [6] and (proposed) slicing are quite static processes with long-lived entities, long-

term resource lockdown and statically-defined low-level policies, and relatively manual management. Piccolo offers the prospect of automated flexibility; the goal is to run automated, closed-loop control for verification, maintenance and overall optimisation on short timescales, taking account of application requirements, the nature of the processing job required, policy constraints, and the nature and amount of the available resources.

So our belief is that Piccolo's approach can alleviate the technical problems, reduce the centralisation and so allow a fresh wave of innovation at the edge (and open up the market to new entrants):

- Application back-ends are transformed from tightly knit and closely orchestrated deployments ("micro-management") into an assembly of service components that run on a continuum of networking, computation and storage resources.
- The latency-critical parts of applications can run on Piccolo's in-network computing, so removing the "time-of-flight" problem.
- Data can be filtered and anonymised before they are shipped back to the central cloud, which alleviates the data upload problem.
- Long-term resource lock-down is replaced by agile dynamic function invocations guided by configurable policies ("management by objectives") expressed by data owners, operators of managed services, and infrastructure carriers.

2.1 Feasibility Discussion

Figure 1 shows an estimate about the importance for various applications of the improvements in latency and bandwidth load from edge computing (Piccolo can be seen as an extreme version of edge computing, where potentially every node is enabled with compute and storage as well as networking capability.) The "feasibility region" (which is double-hashed in the Figure) is where the reduction in latency and bandwidth is both useful and achievable, but today contains a minority of the application areas. We can expect this region to grow in the future: downwards as 5G and full fibre bring lower latencies and leftwards as more devices grow the network load.

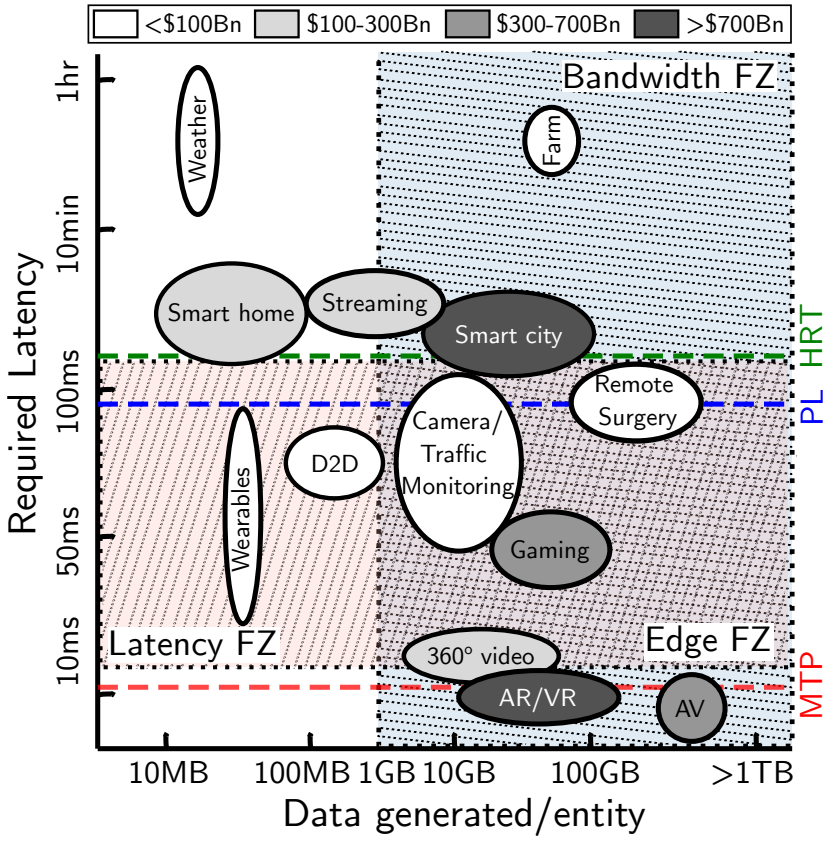

Figure 1: Latency/Capacity Requirements for Different Applications [7]

2.2 Example

In Section 3 we describe Piccolo's use cases. We illustrate Piccolo's motivations using one of our use cases: *Vision Processing*, as depicted by Figure 2. In today's deployment solution, the analytics engine is running as a Linux container service in the centralised cloud, whilst the Visual Processing Engine (VPE) is on the camera. It may be desirable, for privacy as well as for performance reasons, to decompose the analytics function and shift some components onto in-network infrastructure close to the video source, where it may also be easier to take advantage of data from the vehicle subsystem. It may be desirable to decompose that function and shift some components closer to the video sources. Dependent on the use case demands, the VPE can flexibly run on the in-network infrastructure.

This (initially) static data flow graph could be changed dynamically at run time. For example, as more cameras are added or switched on, it may be required to run more than one instance of the VPE. From a cost-perspective, it would be beneficial if the system was modular enough so that only the overloaded component would have to be duplicated, i.e., it must be possible to decompose the overall system sufficiently and then re-arrange the data flow graph accordingly.

From a operational perspective, it would be expected that the application logic that this network of functions (edge-node, VPE, analytics engine) represents can be instantiated on different actual

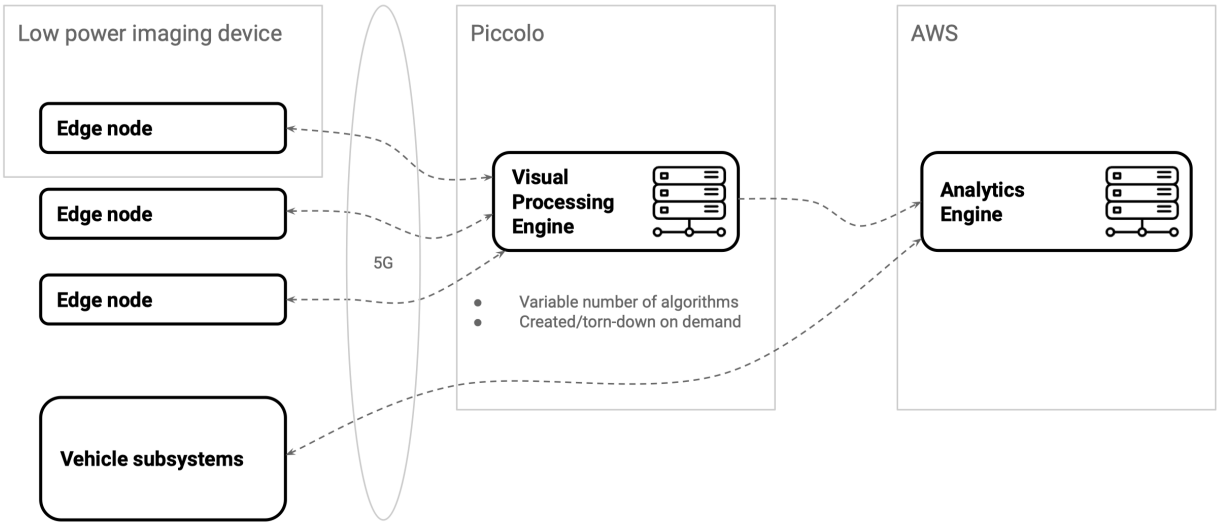

Figure 2: Sensing Feeling Vision Processing

networks, e.g., in a data centre, in an edge deployment, in hybrid settings etc. Moreover, it should be possible to extend or otherwise adapt the system dynamically, e.g., by adding new machine learning models for the VPE.

From a developer perspective, the specific deployment parameters (where the system gets instantiated, how it is scaled at run-time etc.) should not be relevant. However, there may be specific requirements that a developer may want to express for a system deployment, for example specific types of execution environments, affinity of components etc.

Piccolo is conceived as a system that supports creating such distributed applications that can

- be composed of heterogeneous types of components (running on different execution environments, implemented in different programming languages etc.);
- be instantiated in different actual networks;
- employ different kinds of communication interactions (data streaming, remote-method invocation etc.); and that can
- decouple the application logic from run-time and other deployment configuration.

2.3 Technical Motivations and Challenges

Today edge- or in-network computing typically refers to server (infrastructure) virtualisation. For example, in the telecommunications domain, ETSI MEC [8] is extending cloud computing and NFV [5] concepts to make compute infrastructure available at the network edge, whilst in the industrial IoT

domain, notions such edge or fog computing as put forward by the Industrial Internet Consortium [9] and other groups are also mostly concerned with infrastructure and limited platform virtualization.

While the ability to leverage virtualised infrastructure can help to run virtual servers at the edge of the network, it is not sufficient for the lightweight, distributed computing environment that Piccolo targets. In the following, we will contrast the state-of-the-art and its shortcomings with Piccolo's ambition for a distributed computing platform that can leverage multiple underlying virtual infrastructures (such as MEC) but that is fundamentally agnostic with respect to the specific infrastructure(s).

Limited In-Network Computing Today

To date, in-network compute resources cater for operators' own processing or are aimed at slicing for specific verticals as proposed in 5G. Slicing lacks scalability because it reserves resources for tenants regardless of their traffic load: when traffic is below peak (most of the time), resources are wasted. Piccolo shifts away from the current reservation-based model of in-network compute towards a model that resembles packet switching. Moreover, the present approaches (upon which slicing is built) either limits where in-network computing happens – namely in dedicated servers at the edge or in data centres – or constrains how complex compute operations are supported – namely the limited expressiveness of packet/flow matching and steering operations available using software-defined networking (OpenFlow [6], P4 [10]). Piccolo will bring compute/storage capabilities to a broader set of networking elements, blurring the distinction between servers and network nodes to offer an infrastructure for flexible provisioning of all necessary resources.

Heavyweight Lifecycle Management in Infrastructure Virtualisation

NFV and SDN achieve resilience and scaling for their virtualised functions as part of the virtualisation technology. However, this requires heavyweight lifecycle management, with a lot of state in the NFV Management and Orchestration (MANO) [11] layer and in the VIM layer; it makes NFV orchestration quite a static process with long-lived entities; this model is inherited by 5G slicing proposals. We argue that a better approach is to achieve resilience and scaling by instantiating fresh instances of a function. This is directly inspired by the way that the Internet achieves reliability over unreliable packet networks, by using end-to-end mechanisms – and contrasts to the circuit switched approach which makes the circuit itself reliable, tightly manages the network resources associated with each individual flow and keeps flow-state in the network. Packet switching is 'stateless' and is one of the key factors in the success of the Internet – a similar packet-like approach to virtualisation is critical to enable the success of in-network computing.

Technical Challenges for In-Network Computing

Piccolo's observation that function-as-a-service is an appropriate model to provide computation is hardly novel, as shown by the plethora of offerings from all public clouds. Turning the function-as-a-service model into the basis for in-network compute, however, requires solving three key challenges:

- *Moving beyond stateless functions and RESTless Application Programming Interfaces (APIs)* – these are great for clouds, but inefficient when applied at packet level and insufficient to implement even basic in-network functionality (e.g. caching).
- *Ensuring strong isolation* despite more powerful APIs, while at the same time having a platform that can support tens of thousands of active functions on commodity hardware. Compared to the state-of-the-art, this requires supporting ten times more tenants on the same hardware.
- Optimizing the placement of functions by *jointly considering compute and network resources simultaneously*, as opposed to individual optimization today.

Serverless Computing Not General Enough

Serverless computing, as seen in AWS's Lambda [12] for instance, enables an application to run without having to be concerned about provisioning or managing servers; the application writer can therefore focus on programming the 'function as a service'. One limitation is that it only allows functions that provide a response directly back to the user (single ended functions). We envisage many useful 'functions as a service' that actually require chains or even meshes of (micro) functions, most likely spread over several locations. Such systems would also allow for function mobility or re-instantiation, for example when moving a certain computation process closer to large input data sets. Theoretical computer science has modelled such systems using process/agent calculi (such as Milner's π Calculus [13] and bigraph formalism [14] which we are considering leveraging pragmatically.

Need for Adequate Programming Abstractions and APIs

Another question relates to the language used to express functions. General purpose languages used today are the default choice, but they may not be the best choice for some of the specialized processing in-network functions may implement. Packet processing, for instance, can be handled very efficiently by domain-specific languages such as eBPF [15] or P4 [10] – Piccolo will embrace such languages to improve the efficiency of specific types of functions.

Building efficient APIs to handle state is the final ingredient needed to enable expressive functions in Piccolo; the challenge is finding the right balance between ease of use on one side (e.g. strong consistency guarantees) and scalability on the other.

Gap Between Networking, Computation, and Storage

Networking and computing are currently optimised separately but using this approach for in-network compute would be highly inefficient: depending on Central Processing Unit (CPU) and the backbone network, processing an image could be faster at the edge or in the network core; an optimal placement decision cannot be taken by looking at a single resource in isolation. Piccolo relies on *joint optimisation* to reduce function invocation latency and enable running the entire infrastructure at higher resource utilization levels.

Insufficient Consolidation of Compute Platforms

There are several key emerging technologies for in-network computing which Piccolo builds on to achieve its goal of supporting millions of functions. Perhaps the most important is programmable hardware, which is enabling much more flexibility in the data plane, in terms of the function implemented, and the overhead and latency associated with virtualisation. Other relevant work includes uni/microkernels (e.g., MirageOS [16] and ClickOS [17]), extended finite state machines and the somewhat more mature Linux containers.

Security and Isolation

The scale, heterogeneity, and value of the data processing to be supported by Piccolo point to requirements on security (ensuring the right people execute the right computations), privacy (ensuring that those computations respect the scale and potential intimacy of the data involved), and accountability (ensuring there is a basis for explaining the decisions taken about data processing).

The Piccolo architecture aims to support multiple service providers and billions of users running code on a shared infrastructure. This infrastructure must thus be secured: the right code must be run by the right users, and the multi-tenancy of the infrastructure requires that one computation cannot maliciously interact with another.

3 Piccolo Use Cases

This chapter illustrates the use cases envisioned for the overall Piccolo system using four different use case domains, namely *telecommunications*, *vision processing*, *industrial IoT*, and *connected vehicles*. This is not meant as a detailed analysis, ready to be implemented (as the project will not implement all use cases, but rather some exemplary demos), but rather to guide the discussion of existing solutions within the context of Piccolo and to derive requirements for the Piccolo system.

3.1 Stakeholder

The envisioned Piccolo system consists of different stakeholders interacting with the overall system. This section lists the basic stakeholders considered.

Device Owner

This is the end-customer who eventually will have paid the cost of the system. He or she wants to interact with services and applications. The *device owner* has a strong interest in security and privacy and wants to control access to data from their device. Examples of a *device owner* in the context of Piccolo include a vehicle owner, a manufacturing company that owns machines to produce products, etc.

Infrastructure Provider / Network Operator

The *Infrastructure Provider / Network Operator* develops or customizes the devices hosting the Piccolo system (e.g., potentially installed into a cellular base station, or vehicle). It is expected that this persona also operates or manages operation of some additional infrastructure components like an application/function store or a billing/payment system. It is the goal of this stakeholder to generate revenue by operating and controlling the ecosystem.

Application/Service Developer

The *application/service developer* develops applications (which might be composed of smaller services) and solutions for the Piccolo system. The applications/services may be offered to the end customer using the Piccolo system, or can be part of a specific service functionality from another provider, suppliers, or third parties such as a *device owner*, a *network operator* or *service provider*. The developer expects well-defined APIs to develop applications and services for the Piccolo system. This also includes easy access to data, e.g., to a vehicle or manufacturing machine. Furthermore, the

application/service developer expects a simple integration of its applications into an existing ecosystem.

Services Provider

This role operates services that may include any application within the system and offers them to *device owners*. The *Service Provider* may be identical to the *Infrastructure Provider / Network Operator* or just use the infrastructure provided by them.

3.2 Telco: CV-OAM: Continuous Verification Operations, Administration and Maintenance

The context for the use case is BT's strategic objective to automate network management, with the purpose to reduce the cost of operating networks whilst enabling networks to offer high service availability and agility to deliver new innovative services.

Continuous Verification (CV) is used today to check that the network is performing satisfactorily. It involves the generation of a regular test packet, in an IP network typically this is a "ping" created once per second. The receiver monitors the pings to derive performance characteristics, such as connectivity failure and the loss rate. The problem is that CV is configured by the network designer and is unlikely to be modified during the lifetime of the network. It is typically implemented in Application-Specific Integrated Circuits (ASICs) which makes its behaviour difficult to modify and so it runs at a fixed, slow rate (1 per second, say), as a compromise between overloading the network equipment and gathering enough information to be useful. CV does not monitor the performance of the end to end path (only of a link) and does not diagnose the fault condition (only detects failure).

In this use case the objective is to develop a more flexible approach to Continuous Verification Operations, Administration and Maintenance (CV-OAM). For example:

- Adapt the CV rate according to the Service Level Agreement (SLA), so that problems are noticed more rapidly on services with the most demanding SLAs.
- Increase the CV rate in response to network conditions, so there is better resolution when a problem is suspected, but less load is generated where there are no issues.
- Go beyond simple pings. For example, the IPPM working group at the IETF has defined many metrics [18].
- Have the choice of (follow-up) test depend on the results of the (previous) test.
- Get the true end-to-end performance view, by running CV agents in the end hosts, which is of particular importance in light of the rise of end-to-end encryption, at the application and at the

IP transport level (TLS, QUIC).

We also want to extend this approach such that the network itself builds its own catalogue about what equipment it has and how it's connected. Today the information is derived from a design-time, off-line process, which invariably has mistakes and hence self-verification would be valuable.

3.2.1 Problem statement

One use of Operations, Administration and Maintenance (OAM) protocols is the CV or testing of the network. CV involves the generation of regular test packets which check the performance of the network - for example with IP networks this would be a continuous "ping". Connection oriented CV measurements are not a problem e.g. for SDH, Optical Transport Network and carrier Ethernet. CV for connection-oriented services can be automatically activated when the connection is set-up, statically configured but directly tied to the provision of the connection. For MPLS (RFC3031 [19]), LSP-ping (RFC4379 [20]) or BFD (RFC5884 [21]) generates the CV flows but these are not tied to the provisioning of the LSP (Label Switch Path). Point to multipoint LSPs (RFC4875 [22]) add complexities (square law growth of possible connections) that Piccolo type processing could help with. The LSPs to be measured are within the network only.

CV protocols run continuously and are typically not event triggered. CV is done at a slow rate e.g. 1 per second and is simple or implemented in ASICs which makes its behaviour difficult to modify. CV is configured at a fixed rate by the network designer and unlikely to be modified during the lifetime of the network. This makes it difficult to support dynamic SLAs as the CV rate cannot be automatically adapted to the resolution required by the SLAs. In a more intelligent network we could imagine the CV rate being adjusted in response to network conditions, such that the network OAM can focus in higher resolution on the issues, and generate less load where there is no issues. Further we could imagine the CV tests being much more than simple pings if the network had a programmable CV capability e.g. monitor DNS responses.

There are 2 types of CV packets for connectionless networks e.g. IP: Those that are extra packets generated within and often terminated within the network and those that are inserted into the customers' or clients' packets sometimes call "in-situ OAM" packets [23]. We have already seen proof of concepts of in-situ OAM packets generated and processed by P4. This method may impact the behaviour of the packet e.g. increase packet size and cause Maximum Transmission Unit (MTU) problems. *CV protocols today only detect a failure but do not diagnose the fault condition.* In a more intelligent network the CV failure could trigger actions that lead to a diagnosis, e.g. deeper probing of performance and examination of network state.

In-situ OAM and passive measurements are much harder in the context of encryption of end to end customer sessions and the transport-layer protocol (such as with QUIC [24]; the network operator cannot see retransmissions, sequence numbers etc,) (QUIC includes the (unencrypted) "spin" bit, which allows a limited measurement of round trip time for experiments. There are various proposals in the IETF to expand this to 4 bits to allow measurement of: precise Round Trip Time (RTT), round-

trip packet loss and one-way packet loss.) Putting Piccolo functions on the end (customer) devices might allow every path (TCP/QUIC session) to be monitored (see functional testing). The Piccolo framework could include, as part of hosting the function, a monitor that is visible to the Piccolo network infrastructure. There is an opportunity for devices to communicate using Piccolo functions to aid diagnosis.

In summary, the limitations of today's solution are:

- Today every link in a network can be monitored but it's difficult to measure the performance of every possible end to end path because of the square law growth of possible connections.
- CV protocols are fixed at a slow rate (typical the maximum is 1 per second), as a compromise between overloading the network equipment and gathering enough information to be useful. The CV rate does not adapt to network conditions, events or user behaviours. A CV protocol failure can generate an event e.g. link-down.
- CV protocols are not programmable so only perform simple fixed checks. The information gathered does not adapt to network conditions, events or user behaviours
- CV protocols detect failure but do not diagnose the fault condition.
- Event correlation and analysis is done centrally which scales linearly with the number of nodes monitored.

3.2.2 Potentials for in-network compute

The potential is that CV-OAM becomes a software function built into the in-network compute. An operator would be able to specify CV rules/expression (invariants etc.) at an abstract level for the network as a whole and at run-time, instead of on a device-by-device basis at design time. The OAM would automatically break down the CV rules into "sub-routines" and distribute them across the set of monitoring/verification nodes within the in-network compute, leveraging execution platforms all over the network. Then the rules would automatically be updated to adapt to the actual network conditions, events and user behaviours. Another aspect is that in-network compute would enable local analysis and filtering of results (instead of transporting all results to a central site for processing), and it could even allow several nodes near each other to cooperatively diagnose and fix problems, through the use of AI.

3.2.3 Key requirements and challenges

The key requirements for CV-OAM are:

- Capability to specify the CV-AM protocol at an abstract level e.g. its syntax and behaviour, not just the message rates.

- Capability to adapt the CV-OAM according to SLAs, measured results, network conditions or events, and customer behaviours.
- Capability to place Piccolo functions (which implement the CV-OAM) on end users/customers devices, CPE and network operator equipment.

However, there are several key challenges within this scope. Today's standardised CV-OAM protocols are very limited, as they assume 'lowest common denominator' network equipment. Piccolo requires that network designers can create their own CV-OAM protocols using a relatively abstract interface to define test specifications, and the follow-on processes of anomaly detection and repair. The context is that of considerable heterogeneity in the network equipment and execution environments (P4, Field Programmable Gate Arrays (FPGAs), X86 etc), whilst "whiteboxes" permit flexibility in the choice of operating system and protocols. Traditionally, knowledge of network inventory and SLAs is considered outside the scope of the CV-OAM system. But this is a pre-requisite for self-diagnostics and automatic anomaly detection (perhaps using ML), as envisaged by Piccolo.

3.2.4 Stakeholders

The stakeholder/personas of the use case are:

- **Device Owner** – prefer to have a service that is continuously monitored for performance and availability (whilst respecting privacy).
- **Network Operator** – check that any provisioned services are working properly or to raise alarms if a service fails and to aid diagnostics. Network architects have to consider the impact of self-diagnostics on the scalability of the network e.g. design to have enough resources available for self-diagnostics at the required scale and rates.
- **Service Provider** – will use CV information to isolate or identify where faults are.

3.2.5 Other use cases considered

The following user stories have been identified for improving the network operations of the current network, in consultation with the BT teams responsible for network operations and architecture:

- CV-OAM - see above
- In-network Sandbox
- Self-diagnostics: Adaptive Event Filters
- Digital Twin Support

- Abstraction of Customer Network Data

We selected the CV-OAM for the Piccolo use case, partly because its implementation best fits the skills of consortium members and partly because it is a valuable stand-alone first step.

3.3 Telco: Radical network use case

By re-thinking from scratch about the design of a telco network, we can look at more radical approaches. We want an automated network, so that human intervention is confined to those things that require human astuteness. The fully automated network would adapt to the policies of the operator and their tenants. In light of these, and the applications and their traffic, the network optimises usage of its resources, jointly considering all types of resources (especially compute, storage and networking). It self-verifies what elements there are (nodes, links etc), how they're configured, and how they're performing (fine-grained telemetry). It provides closed-loop control for scaling and resilience mechanisms, to deal with faults and performance issues. These factors imply that software autonomously runs most aspects of a network's management, control and data planes.

3.3.1 Problem statement

The operational issues of today's networks are created by:

- The lack of visibility of network state, or too much state in the network.
- Networks having very limited self-verification.
- The use of human centric APIs rather than machine centric APIs.
- IP address management and configuration, which is a challenge and bottleneck.
- The lack of configuration verification in-network before activation; more than just a syntax check is required.
- Functions and protocols do not include self-integrity testing.
- No "what-if" diagnostics being built in.
- Lack of insight into customers' quality of experience.
- Lack of inherent traffic steering for Edge Compute.
- The considerable time and effort needed to negotiate, design, plan and test the installation of Edge Compute functions or CDN caches.

All the above: generates operational costs; delays network deployments and upgrades; requires net-

work equipment and software upgrades to be extensively tested; reduces service availability due to human configuration errors and manual diagnostic processes; and leads to significant over-provisioning to allow for outages causes by maintenance upgrades.

3.3.2 Potentials for in-network compute

We want an automated solution to these problems. This requires much more accuracy and timeliness in understanding about the network elements: what there is, how they're configured and how they're performing. It requires richer control at instantiation: distribution of application and service functionality, and joint optimisation of usage of the different resources. It also needs closed-loop lifecycle management of scaling, resilience and so on. Overall, we need more functionality, of all sorts, deep inside the network. Hence the Piccolo concept is that every node in the network has compute and storage capability, in addition to the usual networking.

3.3.3 Key requirements and challenges

Here we provide a little more information about a few aspects.

Today 60% of all ISP network traffic originates from a CDN cache and is mostly video traffic. Every CDN provider (Akamai, Cloudflare and so on) and every OTT service provider (such as Netflix and Google) installs their own overlay caching infrastructure, with a few caches inside the network operators. The lack of scalable multicast and native in-network storage means there is little alternative. Initiatives such as ETSI MEC, transparent proxies and operator owned CDNs have failed, due to technical and commercial reasons. Live streaming is also important and, in the near future, new interactive formats and virtual reality. Hence the requirement is that Piccolo natively supports application-agnostic caching and point-to-multipoint communication. This capability should be extended to apply to functions as well as content. Users request the content or function by name, and then the network finds the best copy.

We need to mitigate the current complexity of IP address management and inclusion of IP addresses in the configuration state (of networks and applications). The ID scheme should remove issues of Network Address Translation (NAT) (overlapping addresses and names), partitioning of name-spaces and resources. It needs to consider privacy and security. Hence the requirement is that Piccolo functions have a consistent name that is decoupled from the network address. The address should be "discovered" via a dynamic network directory, which solves the problems of edge resource discovery.

Today's networks are designed for essentially a flat network service which requires considerable complexity to make them support Virtual Private Networks (VPNs) (for example, Cisco Multiprotocol Label Switching (MPLS) VPN manual [25] shows the complexity of a very simple MPLS VPN configuration). Vertical stove piped solutions are common, leading to a lack of reuse of service definitions. Hence the requirement is that Piccolo allows the construction of a service from a hierarchy of

micro or nano services. The construction should address subscription, authorisation, AAA, resource partitioning and isolation. A hierarchical name-space is needed.

Today telemetry is an ‘over the top’ service, which makes it complex to measure Key Performance Indicators and a customer’s Quality of Experience. Hence the requirement is that a Piccolo function includes telemetry, within the function itself, which ensures that the telemetry "fate shares" with the function. As well as performance measurement, it assists the function to self-verify and self-check. Separately the telemetry is used for automated protection and repair. In the wider context, the network can check itself.

Current systems expect to maintain consistency within the system which leads to compromises. Piccolo’s design should not assume total consistency. Instead the end to end application is much better placed to guarantee consistency, which allows novel approaches to scaling and resilience.

3.3.4 Stakeholders

The stakeholder/personas of the use case are:

- **Network Operator** – may be required to implement it (in a truly dis-aggregated network equipment model, network operators (with sufficient R&D capability) could implement independently of equipment vendors
- **Service Provider** – need to understand how to use the Piccolo features.

3.4 Vision Sensing: In-Vehicle, Real-Time Behavioural Risk Monitoring

While technology is always advancing, safety has always been at the forefront for all parties involved. When it comes to transportation, this is even more true. Development of self driving cars, and smarter vehicles in general, leaves us putting a lot of trust into these automated systems which the end user will likely not understand.

Today's vehicles come equipped with a myriad of sensors. By combining these in-vehicle metrics such as speed and acceleration with scene telemetry (driver fatigue, nearby pedestrians), technology can help us provide a real time risk measurement for a vehicle and display this information directly to the driver or to a fleet manager potentially thousands of miles away.

3.4.1 Problem Statement

Technological advancements in vehicle software and hardware in recent years has significantly improved the safety of road vehicles, but the driving behaviour of individuals still plays a huge role. The primary cause of vehicle accidents remains fatigue, a metric that should be easily identified by vision sensing. Whether this is displayed in facial prompts, or if it as serious as head drooping, sensors can pick up on this and act accordingly.

Calculating this risk index can potentially look different for different vehicles, as the technology inside the vehicles themselves will vary. The tight coupling of software and hardware of these vehicles means that the accessible data is usually inflexible. However, by collating all available information in the cloud, the data can be fed through dynamic algorithms which can be updated without needing any access to the vehicle itself.

Processing video footage can be offloaded from the vehicle, so no extra computation capabilities are required at the edge. This processing can instead be done in an execution node, allowing implementation for new devices to be relatively cheap. Instead of an entire Visual Processing Engine, the vehicle just needs a transmitter for the images, drastically reducing the cost.

Joining the data from the vehicle data with scene telemetry allows the computation to take into account a multitude of factors. If the vehicle operator is showing signs of fatigue while driving at high speeds, this is significantly more dangerous than driving at high speeds while focused.

With multiple vehicles in a similar space the risk isn't necessarily tied to just one vehicle. The speed of other nearby vehicles, as well as their positioning, could cause an increase in risk that potentially wouldn't be picked up by one vehicle's data. By utilising local nodes, the data from many vehicles could be assessed together and any anomalies identified.

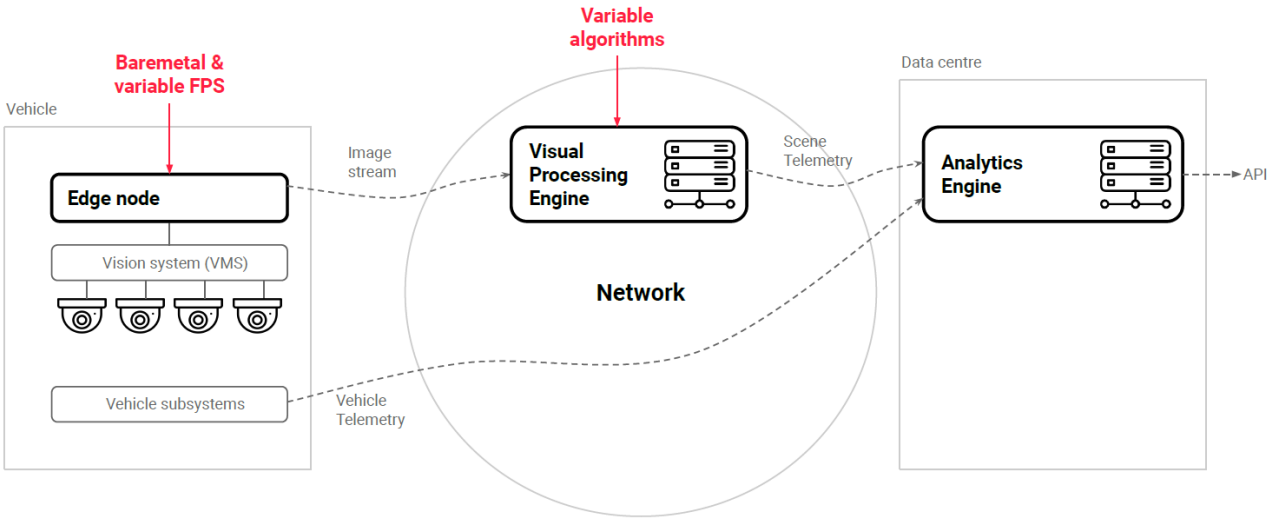

Figure 3: Visual processing of images sent from the vehicle can be carried out as a Piccolo function, with further analysis done in the Cloud

3.4.2 Potentials for In-Network Compute

There are a multitude of factors to calculating the current risk of a vehicle, and the vehicle itself is just the beginning. Past this, road conditions and weather systems should be considered. Knowledge of these factors should be known by a nearby node, without having to rely on systems inside each vehicle. This shared data can then be factored into any calculation process as necessary.

Figure 3 highlights how the architecture could look for such a solution. In this case, there would be multiple vehicles with multiple cameras each communicating with the same Visual Processing Engine running in the network, which would then forward the data inference results onto the Cloud. By running this engine in the network, it is easy to access and to update, allowing for full control over what Edge devices can reach it and what algorithms can be run on the data. It also provides much more control over the hardware being used, as it removes the variable nature of the in-vehicle setup. This centralised service ensures all connected devices get the best level of service available, regardless of any restrictions at the Edge. The vehicle itself also benefits from this offloading as it means less computation is required, resulting in less power consumption and potentially lower hardware costs.

3.4.3 Key Requirements & Challenges

Vehicles, by design, are typically going to be travelling long distances in one journey. This could mean that during one journey, a vehicle could travel in and out of range of multiple nodes. In such a scenario, new nodes must be discovered and any required algorithms would have to be configured to start when needed. Previous data could be needed by the newly assigned nodes, and any security credentials would need to be maintained.

At all points of data transfer, the user must be confident that their data is secure. Other parties should

not be able to access data travelling between the vehicle and the node, between different nodes, or between nodes and the cloud. Similarly, while image inference is taking place on the node, it's important that no other applications running can have an impact on this data flow.

The nodes themselves will undoubtedly have variable usages. At peak time on a motorway, a node would be servicing significantly more vehicles than it would in the same space early on a Sunday morning. Similarly, a node servicing a country lane would also have significantly less traffic. The nodes would need to be able to handle these changes in demand and make use of an appropriate amount of available resource, and pass this off to other functions if applicable.

On a larger scale, different countries will likely have some variations in what is considered dangerous. Whether this is due to differences in driving law, different signage or just different culture, this is a consideration that will significantly affect the risk index calculated.

3.4.4 Stakeholders

The stakeholder/personas of the use case are:

- **Device Owner** – End users will want their video and accompanying data streamed securely to the Piccolo system. They will also want guaranteed service wherever they are on the road, along with a simple one time setup.
- **Application Developer** – The application should be able to easily communicate with various endpoints and have a simple workflow for data transfer. Minimal changes should be required to switch from an edge/cloud based system into a Node based system.
- **Service Provider** – The service provider will want control over what algorithms can be deployed for use and what vehicles will have access to these.

3.4.5 Other use cases considered

Further user stories are discussed in the context of Piccolo and the domain of 'vision processing' (cf. Annex **??** for further details):

- **Sensory assistant for people with Autistic Spectrum Disorders** – a smart wearable assistant that provides real-time sensory cues to the wearer about the nature of their social interactions with others.
- **Multiple 'zero cost' edge devices** – a very cheap low-compute power edge device that acts purely as an image transmitter.

3.5 Industrial Applications: Plug & Produce for Industrial Production Systems

Within the context of the Fourth Industrial Revolution (Industry 4.0) companies are getting more and more aware of the importance of having adaptable production lines, mostly made of standardized building blocks like conveyor belts, which can be rearranged for different production setups such that different products (with varying quantities) can be produced with little overhead in the actual restructuring of a production line.

Since such Smart Factories are very much driven by distributed software components with local control loops there is a need to (re-)configure such systems based on the purpose of each component - leading to the notion of Plug & Produce capable production systems.

3.5.1 Problem Statement

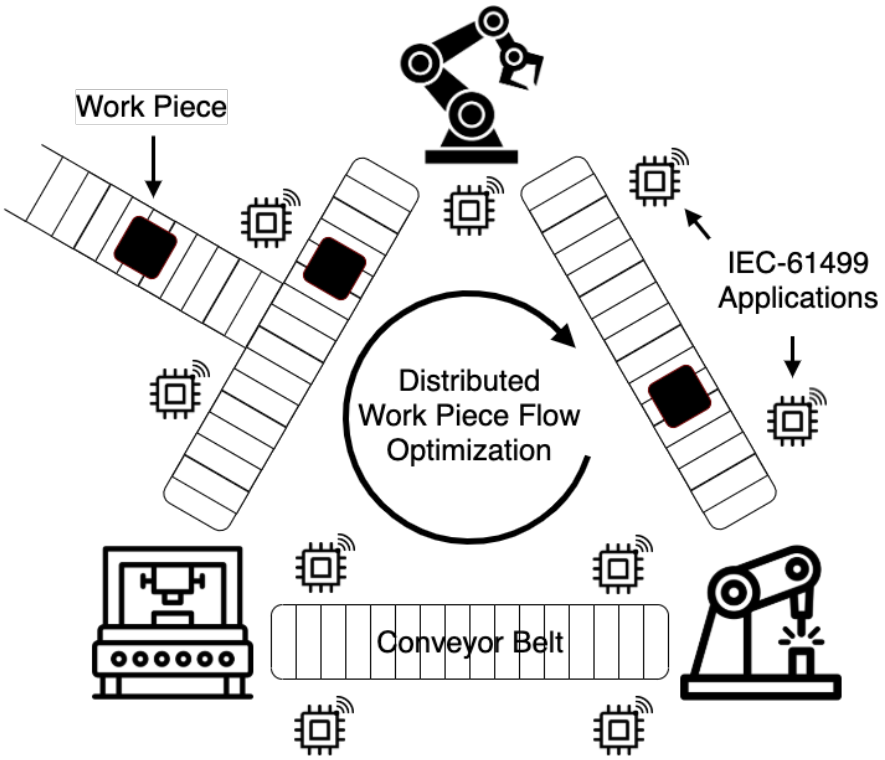

Figure 4: Conveyor belt based production line using IEC 61499

So far industrial applications with a focus on local control loops are based on e.g. Programmable Logic Controllers (PLCs) using the standard IEC 61131 [26]. Networking between PLCs is realized using field bus protocols. The execution mode of these PLCs is cyclical and hence also cyclical field bus protocols are applied, which are wasteful with respect to scarce bandwidth since data is always transferred, even when there is no change in the application itself. Also the synchronisation of PLCs

is supposed to be done by the application developer that leads to long development cycles and a hardly controllable complexity for larger distributed applications.

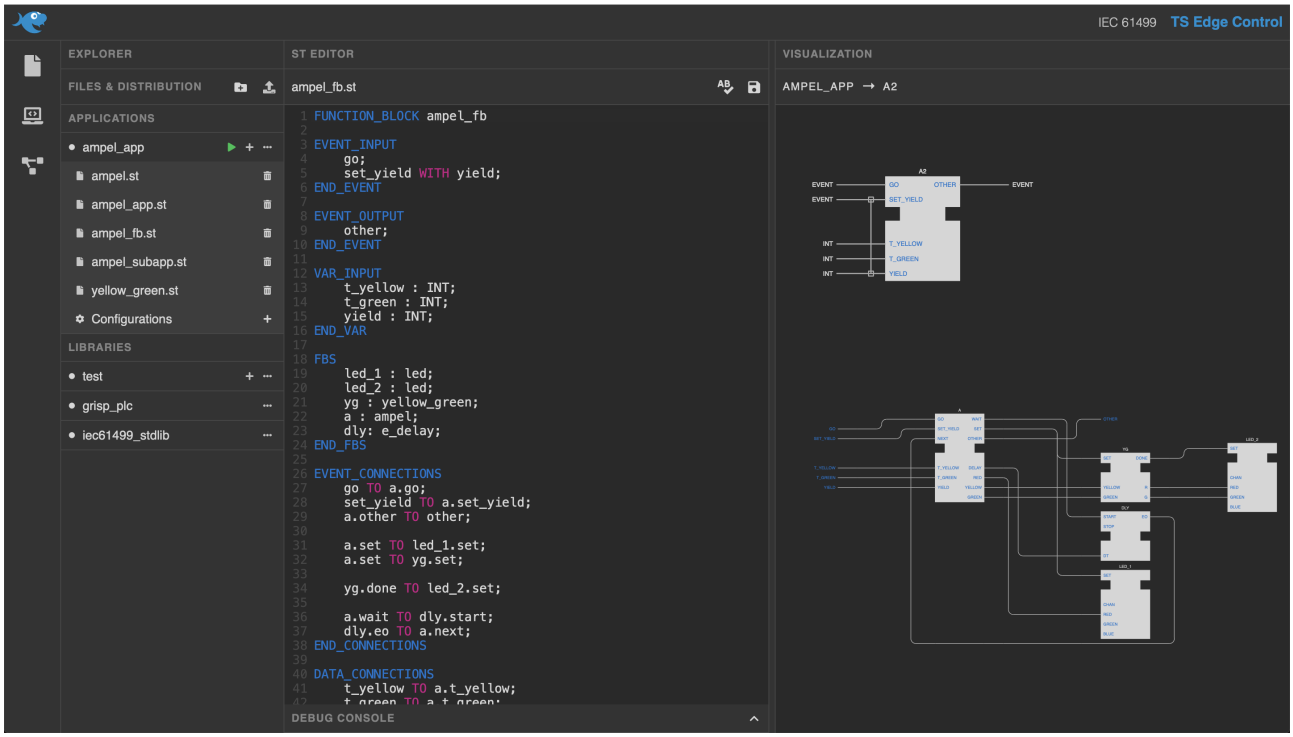

Figure 5: IEC 61499 IDE showing a composite function block

A new standard for distributed PLC programming is IEC 61499 [27], which introduces an asynchronous, event based communication model. However, this model still demands a manual placement of function blocks into computational resources and targets local applications with maximum 100 nodes. For future distributed application in industrial automation there is a need for nested control loops with a combination of a demand for low latency over meshed networks that can scale to multiple orders of magnitude.

Further, production systems are usually governed by a Manufacturing Execution System (MES), which does resource planning, telemetry data acquisition and general process control. Such a MES can be a local machine in the factory hall but also be a service in the cloud. Having such a central point-of-failure is a massive problem since it poses a threat to production downtime. Hence, the traditional functionality of a MES should be distributed over the mesh network to keep crucial information local and available. This also allows optimization efforts in terms of distributed online planning between (groups of) controller nodes for resource and work piece flow.

Also the network connectivity between PLC nodes is currently usually not wireless. Cable management poses a significant effort and potentially longer ramp-up periods for factories. Hence, there is a need for network connectivity based on wireless technologies such as 5G and UWB.

3.5.2 Potentials for In-Network Compute

IEC 61499 applications can be represented by dataflow networks, which can become very complex depending on the use case. The nodes in these networks are IEC 61499 function blocks, some of which have specific requirements for its placement on a physical network. Using Piccolo concepts we want to dynamically map these complex dataflow networks onto physical networks, respecting the mentioned constraints.

In-network computing offers a combination of compute and network infrastructure on all levels, from cloud over 5G base stations to local microcontrollers, and a transparent management of function placement and migration within the network. Such a management of functions in particular involves the properties and capabilities (compute resources, access to certain data or sensors, etc.) of a network node and the distribution of functionality (IEC 61499 function blocks) can reflect on this. Hence, the Piccolo platform offers the foundation for automatic installation or placement of software components for industrial production systems based on IEC 61499.

Additionally, in-network computing enables concepts which require topology awareness on each node. This is not only important for the previously mentioned function placement but also for application level optimizations within a distributed MES, e.g. traffic optimization for conveyor belts. Such optimizations operate on the overall network topology.

3.5.3 Key Requirements & Challenges

- Since node properties need to be advertised throughout the network to other nodes there is a need for primitives to describe these node capabilities in a first step. Further, such a node description needs to be distributed to other nodes in a consistent way such that the whole information base of each node is converging towards the same topology and node properties descriptions.
- It should be possible to inject system level components from any place at the production facility. There is no centralized management logic to handle the placement of components. Hence, orchestration mechanisms are needed to dynamically move components to their proper place in the network.
- Low-level functionality is described in terms of IEC 61499 applications and such a graph of function blocks needs to be mapped to a network topology. This requires a distributed model interference such that the dataflow network described through a IEC 61499 model is preserved in the target network. Also conditions like e.g. neighborhood of components should be preserved as much as possible to ensure e.g. latency conditions.
- A high-level distributed control layer needs to be introduced to replace the classic MES (which is a single point of failure). This control layer can e.g. steer the work piece flow and is supposed to leverage distributed (online) planning algorithms. A 'Digital Twin' of the overall system

plays a crucial role here as a data foundation for control layer optimization.

3.5.4 Stakeholders

The stakeholder/personas of the use case are:

- **Device Owner** – The production facility wants to add or move stations that join the network and the production line without (much) configuration changes or the need to add new cable connections.
- **Infrastructure Provider / Network Operator** – The production facility provides a flexible wireless and secure network that can easily adjust to geographical changes of the nodes.
- **Application Developer** – PLC programmers develop IEC 61499 applications.

3.5.5 Other use cases considered

The topology aware distributed IoT platform of this use case was also discussed as basis for a smart city platform with different potential user stories:

- Ubiquitous traffic measurement – For traffic planning and optimisation, high quality traffic data is required. The measuring process and the data density could be improved a lot using a lot of sensors connected and communicating in a smart way.
- Parking Management – Sensors on smart street lights could recognize free parking spaces and make this information available on a smartphone app.
- Cost efficient tracking of on-street equipment – Street light sensors can get the positions from one another, and build their own map, rather than having someone input the coordinates of each street light.

The smart factory use case is built with the same topology aware IoT platform and selecting it over smart cities was a business decision.

3.6 Automotive: Offloading Driving Logic for Automated Driving Vehicles

Today, connected and autonomous driving is one of the major technological drivers in the automotive domain. It is already evident that automated driving will heavily rely on information sharing from in-vehicle components as well as external sources (e.g., communication infrastructure, or cloud backends) [28].

As of today, automated driving functions are deployed as part of the Advanced Driving Assistance Systems (ADAS) systems in a distributed fashion within the automotive E/E-architectures. Such functions are highly safety critical and have to adhere with stringent Automotive Safety Integrity Level (ASIL) requirements which range from verification of program logic up to execution of functions on certified hardware.

Each new functionality being implemented in an ADAS comes with a significant cost. On the one hand the hardware must be provisioned and certified for these functionalities, on the other hand the software becomes increasingly complex. The complexity can be directly translated into increasing costs for the various certification processes.

The state of the art mentions several approaches to reduce this complexity, one of the most promising ones being decomposition of the driving logic into smaller functions and offloading them into the infrastructure. To keep the costs low, this use case proposes to offload parts of driving functions, especially the non-safety-critical ones, to the infrastructure outside the vehicle.

3.6.1 Problem Statement

Instead of implementing full fledged services on “expensive boxes" deployed within the vehicle, the decomposition of driving logic into smaller functions describes an interesting alternative (cf. Figure 6). While the minimal driving functions of an ADAS system will still remain in the car (or in its close proximity), other functions (e.g., enhanced driving functions on dedicated and certified hardware) can be offloaded to execution nodes located within the infrastructure or in a cloud infrastructure (e.g. for tele-operated driving services).

Existing technologies foresee use of edge deployments (e.g. dedicated MEC servers - see MECView project [29]) for such infrastructure assisted driving. However, these deployments must be provisioned for handling up to peak traffic, e.g., at a busy crossing, and thus, the significant computing resources remain unused during the remaining time.

This use case seeks to utilise the concepts from in-network computing for efficient utilization of computational and networking resources. Driving functions from a vehicle need not be necessarily offloaded to dedicated edge servers. The network determines at run time the suitable execution nodes (between the edge and the cloud) for a function depending on its criticality and the real-time requirements. Additionally, to satisfy the safety requirements of automotive domains, functions can be

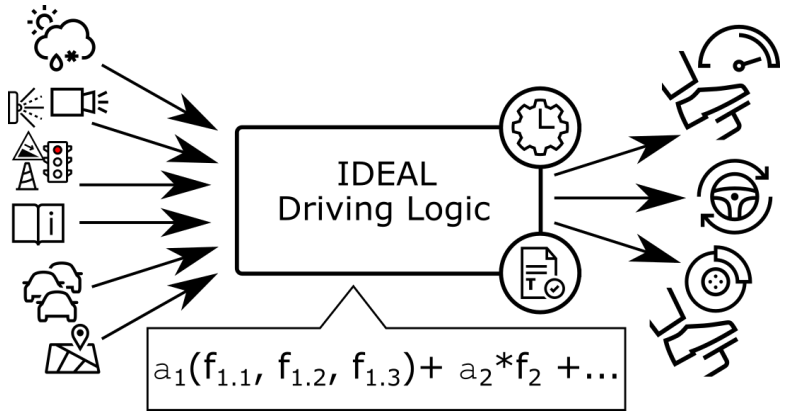

Figure 6: Decomposition of the driving logic into sub-functions which can be offloaded

executed at multiple locations in the network and the results can be compared before being returned to the consumer.

3.6.2 Potentials for In-Network Compute

Advances in data-oriented networking principles like Information-Centric Networks (ICNs) simplify the discovery of services, the access to resources (e.g., compute, data, storage) as well as the dissemination of data in mobile scenarios [30], [31]. Furthermore, advances in virtualization and programmable networking technologies create new opportunities for application aware networks (e.g., simplify compute result dissemination). Those networks promise to prepare, process, and analyse data (e.g., from vehicles) directly on the communication path.

Based on the Local Dynamic Map (LDM) [32] technology, automated driving vehicles require an environmental model generated periodically within the cloud and verified against in-vehicle data (incl. road topology, localization, position of obstacles and moving objects, odometry, acceleration data).

In order to utilize the presented advances, Piccolo programming abstractions will simplify the development of distributed vehicle applications and enable the flexible deployment of driving logic/LDM logic within an execution infrastructure by ensuring certain application specific requirements (e.g., maximum end-to-end latency) and assist the network to process data flows towards cloud backends [2]. Figure 7 illustrates an automotive Piccolo scenario of a flexible deployment of driving logic/LDM logic in the communication infrastructure.

Furthermore, the Piccolo infrastructure will assist in the generation of environmental models by offloading parts of the generation of the "vehicle's horizon", e.g., by augmenting the view with data from external sources and computed within the infrastructure.

As part of the planned Piccolo proof-of-concept prototype, aspects like data-oriented networking principles as well as the role of programmable data planes for automotive applications will be investigated.

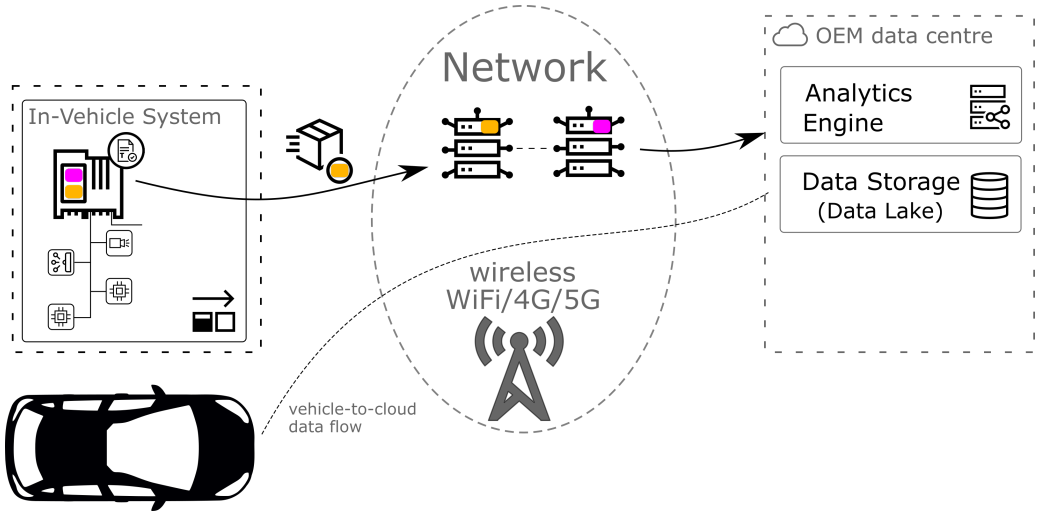

Figure 7: Piccolo scenario in which driving logic can be offloaded into the infrastructure. Potentially any node in the network providing compute capabilities and guarantees to fulfill the offloading requirements can serve the offloading request.

3.6.3 Key Challenges

The key challenges to be tackled to implement this use case are as follows:

- The mobility of vehicles means the node to offload any of the driving function must be discovered/determined rather swiftly. Furthermore, mechanisms for quick execution of functions (access of computing and networking resources) and returning of results to the corresponding vehicle, even if the vehicle has now moved from its original location are highly desirable.
- Existing programming abstractions are unsuitable for developers to describe driving functions which can, if required, be decomposed into smaller functions which communicate with each other and can be offloaded flexibly into the infrastructure.
- Securing offloaded functions against malicious attacks and ensuring that returned results are trustworthy require extending the state-of-the-art.
- While not safety critical, the offloaded functions still be able to provide high availability and reliability by means of redundant computations if required.
- While the driving functions which are offloaded may not be safety critical, they are bound to have (soft-)real-time requirements, which the infrastructure must strive to fulfill. E.g. receiving driving recommendations for a stretch of road after already driving by is not ideal.

3.6.4 Stakeholders

The stakeholder/personas of the use case are:

- **Device Owner** – the vehicle owner wants to see that privacy and security is ensured.
- **Service Provider** – the service provider (might also be manufacturer, tier one supplier, etc.) provides monitoring hooks to the vehicle to monitor certain automated driving functions.
- **Infrastructure Provider** – offers compute/network resources for service providers to deploy application logic closer to the vehicles and to bill the service provider.

3.6.5 Other use cases considered

As part of the introduction of the 'automotive 'offloading" use case scenario, further user stories are discussed between Bosch business units to contribute to Piccolo (cf. Annex **??** for further details):

- **Vehicle Data Analytics** – efficient gathering and pre-processing of in-vehicle data directly on the delivery path towards cloud backends.
- **Infrastructure-assisted Advanced Driving Assisted Systems** – based on the Electronic Horizon technology (e.g., [3, 4]).
- **Collaborative Perception** – to avoid hazardous situations (e.g., hidden end of traffic jam, icy roads, approaching emergency vehicle, etc.).
- **Tele-operated Driving** – dynamic service/function deployment of driving functions to assist automated driving functions (e.g., automated valet parking [33], stop vehicle safely - 'limp home mode').

4 Application Requirements for Piccolo

This chapter introduces the requirements for the Piccolo ecosystem based on the use cases described in Section 3. This includes the requirements valid for the individual node in the system executing a Piccolo functions – the Piccolo Node – as well as requirements for the entire Piccolo infrastructure. Please see Piccolo Del2.1 [34] and Piccolo Del3.1 [35] for further information on the Piccolo Node and the Piccolo Infrastructure respectively.

In addition, features that applies to both Piccolo Node and Piccolo Infrastructure have been identified, namely (i) *security and privacy*, and (ii) *management of resources incl. planning, orchestration, and service deployment*. The follow subsections introduce the requirements in detail.

4.1 Requirements for Piccolo Node

The Piccolo ecosystem consists of two important parts: (i) nodes implementing Piccolo principles as well as the (ii) infrastructure interconnecting the nodes with each other to a overall system. In order to tackle the challenges set by the use cases introduced in Section 3, a Piccolo node has to provide **well-defined interfaces** for both local and system-wide management interactions of compute and storage resources. Those interfaces allow for the following features required by the use cases:

- **Discovery**: A Piccolo node has to support discovery mechanisms for the node itself. For example in the smart factory use case (cf. Section Section 3.5), discoverability of node capabilities and the resulting topology allows for topology aware deployment of functions. The mechanisms should be as general enough to allow any type of computing platform to join the piccolo ecosystem.

- **State Handling**: Piccolo node interfaces need to offer mechanisms to store state of functions as well as the access to state regarding migration scenarios. For example the automotive use case, offloaded functions to Piccolo infrastructure nodes should be able to fetch the latest function state to ensure functional operation.

- **Interoperability & Flexibility**: Interfaces on a Piccolo node should allow platform (runtime) and programming language independent interaction between functions. Lightweight implementations (of interface) allow to interact with a variety of types of nodes including embedded nodes as well as powerful compute nodes. Furthermore, it includes the support of 'legacy function' in which existing application logic can be hosted on Piccolo nodes without high adaption effort.

- **Hardware Heterogeneity**: Piccolo considers different types of hardware to host functions. Dependent on the use case domain. Those hardware range from specific hardware to accelerate the performance (e.g., CPU, FPGAs, or ASICs) as well as different types of compute platforms ranging from micro-controller, over micro-processors up to Graphics Processing Unit (GPU)

(e.g., for image processing).

- **Multi-tenancy**: Piccolo node interfaces should enable different application providers to execute functions in parallel on the same node without adversely compromise each other for resources. Those interfaces should ensure that reusable program logic (e.g., represented as functions) are carefully controlled and managed.

Piccolo Deliverable D2.1 describes the Piccolo node properties for distributed computing and corresponding technical requirements in more detail.

4.2 Requirements for Piccolo Infrastructure

In the Piccolo ecosystem, applications are foremost distributed applications, and they require a distributed computing infrastructure supporting: (i) **communication between functions**, (ii) **resource management** – such as allocating computing resources on nodes (see Section 4.1), and (iii) **run-time management and optimisation** – such as re-acting to changing load, performance requirements etc.

Communication Between Functions: Based on the the Vision Processing use case (cf. Section 3.4, Figure 8 illustrates an example consisting of different function instances (*RTSP video feed*, *Edge node*, *Visual Processing Engine*, and *Analytics Engine*. We call these functions *actors* in the following (as the more general term). These actor instances reside (i.e., run) on different Piccolo nodes in a given deployment. There can be different types of interactions between the instances, e.g.,

- **Remote Method Invocation (RMI)** – for invoking a specific method on another actor
- **Data Streaming** – Real-Time data streaming from live media sources, or (more general) data flows processing
- **Publish-Subscribe** – one actor subscribing to data events/updates on another actor
- **Data-Set Synchronization** – a set of actors that share a common data structures, keeping it synchronised in the presence of state changes;

In Piccolo, those interaction types are supported by additional features, e.g., supporting address/location independence and different types of transport protocols to simplify the exchange of data between these function actors. This list may not be complete, and some interaction types may be implementable using other interaction types as building blocks.

Resource Management: Resource management involves the management and allocation of available infrastructure resources (incl. compute, network, storage), e.g., by selecting a Piccolo node for executing a functions (or a compute slot on a node) and allocating it to a specific actor instance.

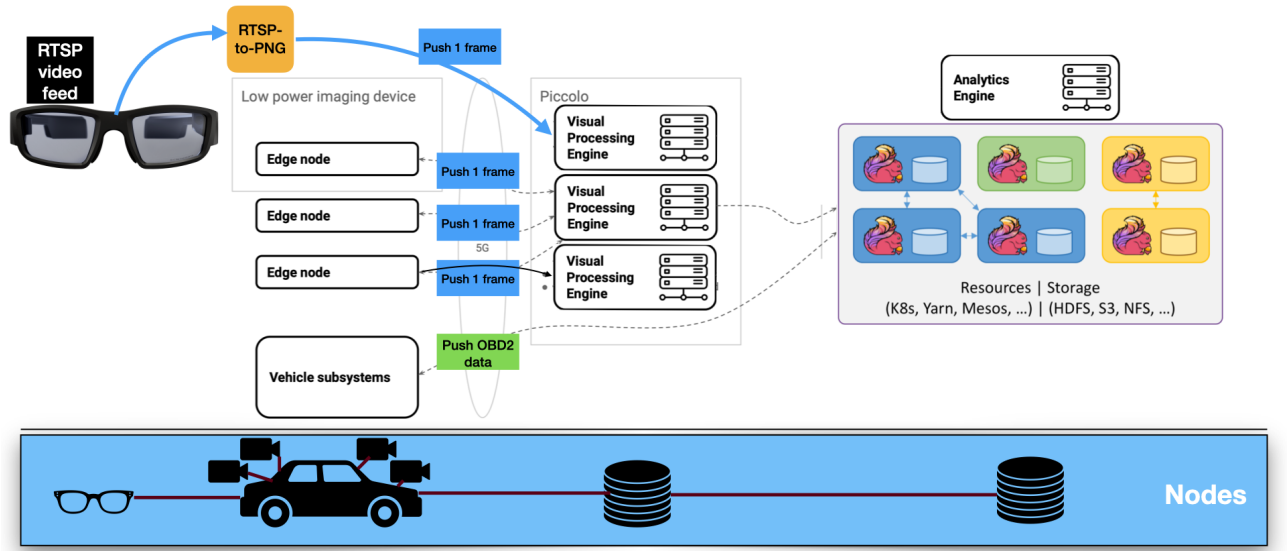

Figure 8: Example: Compute and Communication Infrastructure for Vision Processing

For example, when bootstrapping a distributed application, the actors (actor instances, depending on the required level of parallelism) get instantiated on Piccolo nodes within the infrastructure, able to communicate to each other using interaction types as described above.

The Piccolo ecosystem explicitly supports heterogeneous infrastructure (node platforms) and different types of actors. For example, not all actors will necessarily have to be instantiated explicitly. There can bare-metal, single-purpose systems that provide a specific functionality and that become part of a Piccolo distributed application deployment. In that case, this single-purpose system would still communicate with other actors for the different interaction types, but would not require dynamic instantiating.

Dependent on the use case scenario and the deployment, the resource management mechanisms in Piccolo can be provided in different ways, i.e., by using (logically) centralised resource managers such as Kubernetes for spatial deployments or as decentralised approaches for wide area deployments.

Runtime Management: Runtime management considers a special part of resource management (see above). In order to support distributed applications, the Piccolo infrastructure has to support mechanisms to react to changes in the infrastructure (e.g., dynamic load, or connection losses due to mobility of a node). In such cases, a running Piccolo system must be adapted, e.g., by scaling out actors, by restarting an actor, or by instantiating a new actor.

These dynamic changes should be transparent from a user perspective (in the sense that they happen automatically), and they may require changes with respect to communication relationships, connectivity etc.

We anticipate Piccolo to have visibility into available resource at runtime and leverage this information for making runtime management decisions. Again, this information could be obtained and

managed by a central entity or be shared in different (e.g., decentralized) ways.

Piccolo Deliverable D3.1 described the architecture properties for distributed computing and corresponding technical requirements in more detail.

4.3 Security and Privacy Requirements

Besides the mentioned requirements for a Piccolo node and the Piccolo infrastructure, both parts of the Piccolo ecosystem have to support several **security and privacy** features to prevent for security threats, privacy risks and unauthorized access to Piccolo functions and corresponding data. Requirements of these domains must be met with the same firmness at the edge as they are at core networks.

Moving outside of the data centers, the spectrum of attacks is increased as data is spread across many more locations at the same time. Piccolo execution nodes are located outside of highly secure operator data centres. Both end users and function/application providers need security and privacy assurances backed up by tools such as constant monitoring of edge nodes, reliable access logs and appropriate authentication procedures. However, the expected heterogeneity of Piccolo nodes (cf. Section 4.1), and therefore, the different supported node capabilities describe a challenging task regarding security and privacy.

In principle, the Piccolo ecosystem should address the following features to application providers, end-users and operators in terms of security and privacy:

- **Authentication and Access Control**: provide authentication and access control measures to ensure that only legitimate entities are accessing sensitive information, or invoke and manipulate critical operations of the Piccolo node. The communication between the different entities in the Piccolo ecosystem should be signed.
- **Confidentiality**: ensure that only the data owner and user(s) can access private information. Unauthorized entities should not acquire data, function code or secrets of an application(s) in memory, on disk or on the network. The communication between the different entities in the Piccolo ecosystem should be encrypted regardless of the transport protocol used.
- **Code and data integrity**: provide mechanisms that prevent unauthorized parties (even systems) to modify data at rest, in transit and at use. Additional mechanisms allow for the detection of such modifications. Absence of integrity auditing measures could affect the user's privacy and trust towards the system.
- **Accountability**: The accountability principle requires controllers and processors to take responsibility for their processing activities. Any action in terms of processing and data manipulation needs to be justified and comply with specific policies. The Piccolo system should ensure that there is a basis for explaining the decisions taken about data processing and task outsourcing.

- **Platform attestation**: Platform attestation refers to a system's capability to deem trustworthy by other systems with which it must interact, or to in turn be deemed trustworthy by those other parties. Piccolo should provide mechanisms that prove to a third-party that the compute-platform is trusted and can provide certified hardware. Such certification reports should be available through the Piccolo system to application developers.

- **Rogue actor prevention**: Malicious adversaries may deploy applications that advertise false data to the network in order to compromise sensitive information, inject false information, disrupt the network operations. Piccolo should use monitoring mechanisms to detect unauthorised parties that try to eavesdrop user's data or act as man-in-the-middle and intercept traffic.

- **Node and secrets protection**: Cryptographic key material should be frequently rotated according to a specific strategy. In case of a compromised Piccolo Node, it should not have the authority to affect any other Piccolo node or any other component of the system, especially applications with sensitive information running on the node.

Piccolo node interfaces should address the presented security and privacy features. They enable different application providers to execute functions in parallel on the same node without adversely compromise each other for resources. Reusable program logic (e.g., represented as functions) must be carefully controlled and managed since they create a serious vulnerability and contravene confidentiality through possible data leakage.

4.4 Requirements for Resource Management

The last category of requirements describes features for management of resources within a Piccolo ecosystem including planning, orchestration, and service deployment.

General management features from the perspective of enterprises, application providers and end users that use a Piccolo system, are:

- **Fair Access to Data and Services**: In-network computing can be used by anyone as easily as connecting to the Internet is today. There are likely to be hundreds of service providers, who register thousands of applications that are used by millions of end users and devices. Piccolo should allow new classes of applications and new types of users. Fair access will be crucial to create a thriving economy of data and services. The Piccolo architecture should call for an architectural approach that avoids central entities or platforms which may be able to control market access.

- **Location-independence & Seamless Integration**: Distributed applications should be able to seamlessly use in-network computation without worrying about where it is executed and how – just as endpoints do not worry about how their packets are delivered by the Internet today. This "serverless" or "functions as a service" (FaaS) programming model is similar to Amazon's Lambda today, but enables more expressiveness and functionality, particularly about capabili-

ties that are inherently ”in-network” such as caching or distributed machine learning.

- **Strategy-based management**: the enterprises, application providers and end users can express their high-level demands in a human-user-friendly manner. These demands express objectives about 'traditional' SLA factors such as latency, including the impact of the compute as well as the networking aspects, and non-functional requirements such as privacy and security (see Section 4.3). Included here are the high-level trade-offs, for instance: Which factors matter more to me? What is my price sensitivity?

- **Resource Utilization & Monitoring**: the enterprises, application providers and end users can request telemetry information at an appropriate level of granularity and timeliness, concerning performance and faults and so on (and constrained by the SLA).

In addition, special management features required by special groups of entities in the Piccolo system (e.g., the operator of the telco network and the smart factory network) are as follows:

- **Business-based management**: the network operator expresses their business-aware, high-level goals in a human-user-friendly manner (by contrast, traditionally policies are written at a low-level). The network automatically coordinates the strategies of the operator and the multiple tenants.

- **Automated resource management**: Automation of network operations is the key strategic goal. The function/application must be distributed across the end device, Piccolo's in-network compute and the central cloud, using a chain or even mesh of composed services. The distribution and placement jointly considers all the different types of resource: networking, processing (both general purpose and specialised compute) and storage, and perhaps other resource types such as energy. This contrasts with today's approach of individually optimising each resource. The **joint optimisation** needs to be cognisant that the compute and networking resources are highly heterogeneous, as well as any prioritisation or quality of service techniques that can be employed.

- **Life-cycle management**: Automation also applies to life-cycle management, in particular to handle scaling and resilience.

- **Human judgment**: Human intervention should be restricted to where automation falls short and our strengths over a computer's can solve an issue: we are adaptable, flexible and astute. For example, if the policies of the operator and the various tenants are contradictory or impossible to achieve collectively, then human input is required to negotiate and resolve the tensions and incompatibilities.

All of the mentioned resource management requirements must help towards a networked system that is simpler and cheaper to design, build and operate.

5 Conclusion

We highlight in this document, interesting and relevant opportunities for the Piccolo project by introducing use cases from four different domains, viz., *Network Operations and Management*, *Vision Processing*, *Smart City and Industry*, and *Connected and Automated Driving* (cf. Section 3).

Instead of following today's centralized compute and networking model – as presented by the cloud computing model – the presented use cases demand alignment of compute and networking resources inline with the distributed nature of emerging applications in the IoT.

Each of the presented use case identifies general challenges and limitations in their development and deployment solutions (cf. Section 2) including additional management overheads for compute and network resources discovery, misaligned resource utilization due to missing interfaces to operate on compute and network more efficiently, or inflexible deployment solutions due to static configurations.

We also presented an informal assessment of where in-network compute can offer a real advantage compared to normal cloud computing and (emerging) edge computing.

The key challenges for a distributed compute and networking infrastructure like Piccolo have been identified and discussed based on the presented use cases. These include:

- **Communication models** – to support different types of interactions between compute instances including *data stream processing*, *publish/subscribe*, *remote method invocation*, etc.
- **Programming abstractions** – to describe compute and network node capabilities required by a set of distributed program functionalities.
- **Mobility Support** – to distribute compute results efficiently in the network even for highly mobile participants as in connected vehicle environments.
- **Location independent discovery of compute nodes** – to simplify the discovery of compute nodes independent of the location (e.g., deployment in the network, or geo-location).
- **Distributed Management** – to allow scaling of applications (vertically and horizontally) using distributed control mechanisms to void bottlenecks or single-point-of-failures.
- **Security and Privacy** – to handle compute inputs, secure the execution of compute tasks and ensure that the compute results is correct, valid, verified and can be trusted.
- **Safety** – to provide high availability and reliability by means of redundant computations.

By re-imagining and addressing the challenges of the presented use cases, Piccolo introduces several opportunities to improve system performance and be an enabler for practical future deployments. For example, programming abstractions will simplify the development of emerging distributed applications in the IoT. Piccolo management mechanisms will take care about the deployment of applications

on valuable execution nodes in the network by addressing application specific demands (e.g., time-criticality, mobility, etc.) and scale compute and network resources on demand. Due to the flexible on-demand deployment of applications, the resulting efficient handling and dissemination of compute requests/results will ensure that connected devices get the best level of service available. The acceptance of Piccolo solutions within the market are increased by taking aspects such as security, privacy and safety as a forethought into the Piccolo design.

In order to reveal these opportunities, a list of technical functional/non-functional requirements have been identified for the technical work packages in Piccolo. Besides, the spread of functionality between the application and 'base' functionality provided by Piccolo, including aspects such as monitoring, policy and intent-based management of the overall system, are discussed in *structural & management requirements* (cf. Section 4). A list of important requirements for the Piccolo ecosystem include:

- **Piccolo Node** – definition of interfaces of a Piccolo node to support simple compute node discovery, multi-tenancy, support of hardware and runtime heterogeneity, as well as to ensure interoperability and flexible deployment of distributed functions across several Piccolo nodes.
- **Piccolo Infrastructure** – the support of a variety of communication protocols to realize different types of interactions between compute nodes (e.g., remote method invocation, publish/subscribe, etc.), as well as well-defined management interfaces for efficient resource allocation and runtime management (e.g., to react to changes in the infrastructure).
- **Security and Privacy** – prevention of security threats, privacy risks and unauthorized access to Piccolo functions and corresponding data.
- **Socio-economical Requirements** – the support of fair access to data and applications, assist in seamless use of in-network computations, and to ensure that application requirements (functional and non-functional) are met.

Finally, we selected directions for shorter-term Proof-of-Concept prototypes including:

- a prototype of the *In-Vehicle, Real-Time Behavioural Risk Monitoring* (cf. Section 3.4) by developing aspects of vision processing and in-vehicle data access (cf. Section 3.6).
- a prototype of the *Continuous Verification Operations,Administration and Maintenance* (cf. Section 3.2) as a stand-alone first step.
- a prototype of the *Plug & Produce for Industrial Production Systems* (cf. Section 3.5) as a joint work of Piccolo partners from industry and academia.

References

[1] Google LLC. *Google Cloud CDN*. 2021-3-22. Mar. 2021. URL: https://cloud.google.com/cdn.

[2] H. Moustafa, E. M. Schooler, and J. McCarthy. “Reverse CDN in Fog Computing: The lifecycle of video data in connected and autonomous vehicles”. In: *Proceedings of the IEEE Fog World Congress (FWC)*. 2017, pp. 1–5. DOI: 10.1109/FWC.2017.8368540.

[3] Continental AG. *Continental eHorizon*. 2021-3-04. Mar. 2021. URL: https://www.continental-mobility-services.com/en-en/products/ehorizon/.

[4] Robert Bosch GmbH. *Connected Horizon*. 2021-3-04. Mar. 2021. URL: http://www.bosch-softtec.com/connected_horizon.html.

[5] European Telecommunications Standards Institute. *Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV*. 2021.

[6] Nick McKeown et al. “OpenFlow: Enabling Innovation in Campus Networks”. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), 69â74. ISSN: 0146-4833.

[7] Nitinder Mohan et al. “Pruning Edge Research with Latency Shears”. In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. HotNets ’20. Virtual Event, USA: Association for Computing Machinery, 2020, 182â189. ISBN: 9781450381451. DOI: 10.1145/3422604.3425943. URL: https://doi.org/10.1145/3422604.3425943.

[8] European Telecommunications Standards Institute. *ETSI Multi-access Edge Computing (MEC)*. 2021.

[9] Industrial Internet Consortium. *Website Industrial Internet Consortium*. Accessed on: March 22nd, 2021. 2021. URL: https://www.iiconsortium.org/.

[10] Pat Bosshart et al. “P4: Programming Protocol-Independent Packet Processors”. In: *SIGCOMM Comput. Commun. Rev.* 44.3 (2014). DOI: 10.1145/2656877.2656890. URL: https://doi.org/10.1145/2656877.2656890.

[11] European Telecommunications Standards Institute. *ETSI GS NFV-SOL 005: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point*. Tech. rep. 2018.

[12] Amazon Web Services, Inc. *AWS Lambda*. Accessed on: March 22nd, 2021. 2021. URL: https://aws.amazon.com/de/lambda/.

[13] Robin Milner. “Functions as processes”. In: *Mathematical Structures in Computer Science* 2.2 (1992), 119â141. DOI: 10.1017/S0960129500001407.

[14] The Bigraphical Model. *The Bigraphical Model*. URL: https://www.cl.cam.ac.uk/archive/rm135/uam-theme.html.

[15] Cilium, Inc. *eBPF project website*. Accessed on: March 22nd, 2021. 2021. URL: https://ebpf.io.

[16] Linux Foundation. *MirageOS project page*. Accessed on: March 22nd, 2021. 2021. URL: https://mirage.io/.

[17] Eddie Kohler et al. “The Click Modular Router”. In: *ACM Trans. Comput. Syst.* 18.3 (2000), 263â297. URL: https://doi.org/10.1145/354871.354874.

[18] *IP Performance Measurement, working group at IETF*. URL: https://datatracker.ietf.org/wg/ippm/documents/.

[19] *Multiprotocol Label Switching Architecture*. URL: https://www.rfcreader.com/#rfc3031.

[20] Kireeti Kompella and George Swallow. *Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures*. RFC 4379. Feb. 2006. DOI: 10.17487/RFC4379. URL: https://rfc-editor.org/rfc/rfc4379.txt.

[21] Thomas Nadeau et al. *Bidirectional Forwarding Detection (BFD) for MPLS Label Switched Paths (LSPs)*. RFC 5884. June 2010. DOI: 10.17487/RFC5884. URL: https://rfc-editor.org/rfc/rfc5884.txt.

[22] Dimitri Papadimitriou, Seisho Yasukawa, and Rahul Aggarwal. *Extensions to Resource Reservation Protocol - Traffic Engineering (RSVP-TE) for Point-to-Multipoint TE Label Switched Paths (LSPs)*. RFC 4875. May 2007. DOI: 10.17487/RFC4875. URL: https://rfc-editor.org/rfc/rfc4875.txt.

[23] Frank Brockners, Shwetha Bhandari, and Tal Mizrahi. *Data Fields for In-situ OAM*. Internet-Draft draft-ietf-ippm-ioam-data. Work in Progress. Internet Engineering Task Force, Feb. 2021. 45 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-data.

[24] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-34. Work in Progress. Internet Engineering Task Force, Jan. 2021. 207 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34.

[25] *Configuring a Basic MPLS VPN*. URL: https://www.cisco.com/c/en/us/support/docs/multiprotocol-label-switching-mpls/mpls/13733-mpls-vpn-basic.html.

[26] International Electrotechnical Commission. *IEC 61131 Programmable Controllers - Parts 1-10*. 2003.

[27] International Electrotechnical Commission. *IEC 61499 Function blocks - Parts 1-4*. 2012.

[28] J. Wang, J. Liu, and N. Kato. “Networking and Communications in Autonomous Driving: A Survey”. In: *IEEE Communications Surveys Tutorials* 21.2 (2019), pp. 1243–1274. DOI: 10.1109/COMST.2018.2888904.

[29] MEC-View project website. *Mobile Edge Computing Based Object Detection for Automated Driving*. 2021-2-12. Feb. 2020. URL: http://www.mec-view.de/.

[30] G. Grassi et al. “VANET via Named Data Networking”. In: *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2014, pp. 410–415. DOI: 10.1109/INFCOMW.2014.6849267.

[31] Dennis Grewe et al. “Information-Centric Mobile Edge Computing for Connected Vehicle Environments: Challenges and Research Directions”. In: *Proceedings of the Workshop on Mobile Edge Communications*. MECOMM ’17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, 7â12. ISBN: 9781450350525. DOI: 10.1145/3098208.3098210. URL: https://doi.org/10.1145/3098208.3098210.

[32] L. Andreone et al. *Safespot final report. Technical Report D8.1.1*. Tech. rep. Accessed on: January 31st, 2021. 2010. URL: http://www.safespot-eu.org/documents/D8.1.1_Final_Report_-_Public_v1.0.pdf.

[33] Robert Bosch GmbH. *Connected Parking: Automated Valet Parking*. 2021-2-12. Feb. 2021. URL: https://www.bosch.com/stories/automated-valet-parking/.

[34] Piccolo Project. *Piccolo Node Definition*. Deliverable D2.1. 2021.

[35] Piccolo Project. *Architectural Invariants for Distributed Computing*. Deliverable D3.1. 2021.

