Evolutionary Autonomous Networks

Paul Harvey*, Alexandru Tatar, Pierre Imai, Leon Wong and Laurent Bringuier

Rakuten Mobile Innovation Studio, Japan
E-mail: paul.harvey@rakuten.com; alexandru.tatar@rakuten.com; pierre.imai@rakuten.com; leon.wong@rakuten.com; laurent.bringuier@rakuten.com
**Corresponding Author*

Received 30 September 2020; Accepted 17 March 2021;
Publication 02 June 2021

Abstract

The communication networks of today can greatly benefit from autonomous operation and adaptation, not only due to the implicit cost savings, but also because autonomy will enable functionalities that are infeasible today. Across industry, academia and standardisation bodies there has been an increased interest in achieving the autonomous goal, but a path on how to attain this goal is still unclear.

In this paper we present our vision for the future of autonomous networking. We introduce the concepts and technological means to achieve autonomy and propose an architecture which emerges directly through the application of these concepts, highlighting opportunities and challenges for standardisation. We argue that only a holistic architecture based on hierarchies of hybrid learning, functional composition, and online experimental evaluation is expressive and capable enough to realise true autonomy within communication networks.

Keywords: Autonomous networks, closed loop, computer networks, architecture, evolution, online experimentation.

Journal of ICT Standardization, Vol. 9_2, 201–228. *River Publishers*
doi: 10.13052/jicts2245-800X.927

1 Introduction

The importance and impact of networking and communication systems on our everyday lives is enormous. Almost every minute of our waking day is somehow influenced (or controlled) by our phones and *smart* devices [1]. This influence is expected to grow dramatically in the near future due to the proliferation of automotive [2], wearable [3], and other emerging IoT-related applications [4] which are set to permeate every aspect of our existence. This substantial transformation of human society relies on ever-more advanced communication networks to enable the near-instant transportation of massive amounts of data from the billions of connected devices [5–7].

One of the most challenging research directions in communications networks is to build *autonomous networks*, i.e., networks that can run with little or no human involvement. This paradigm, which triggered much interest from academia, industry and standardization bodies, will allow networks to better operate in today's dynamic environment including, but not limited to, the abilities to self-operate, self-heal, and self-optimise [8].

The purpose of this paper is to present our own vision for the autonomous network of the future, one based on ***evolutionary computing*** combined with ***online trail-and-error experimentation*** to achieve ***guided emergent behaviour***. Here, guidance is provided by technologies, such as ontology, reducing direct human involvement. By adopting this approach, we seek to address the primary challenge of improving the networks' ability to cope with uncertainty, a strong limitation of current approaches to autonomous networking that are predominantly use-case based.

Our approach operates at a higher level of abstraction compared to traditional management and network orchestration (MANO) functionality or individual use cases; we describe a system that is responsible for the management and orchestration of the MANO itself or use cases, reducing (or replacing) the role of human in these situations. We describe the principles that guide our pursuit of a (semi) autonomous network in Section 2 as well as an architecture based on these principles in Section 3. Throughout the discussion, we highlight necessary standardisation activities that will require industry-wide consensus.

2 Principles of Evolutionary Autonomy for Networks

An autonomous network cannot come into existence in a deus-ex-machina approach, but instead will have to gradually *evolve* from existing

technologies. Having motivated *why* autonomous networks are necessary, we now introduce *how* academic research will enable fully autonomous networks in the future, and partially today.

2.1 Guiding Principles of an Autonomous Network

Autonomous networks are not a new idea of the telecommunications industry. The topic has been investigated in networking research for decades, led to several independent large research initiatives [8–11], and includes fields far beyond networking, such as *Cybernetics* [12]. Clark et.al's knowledge plane for the Internet [13] is a seminal work in the field which defines the concepts of *edge involvement*, *global perspective*, *compositional structure*, *unified approach*, and *cognitive framework*.

Based on these concepts, existing efforts, research, and our previous work [14, 15], we derive the following five principles we deem necessary for achieving true autonomy:

1. **Holistic approach:** Autonomous networks require comprehensive access to information about the network (within the domain it operates in). We extend this requirement to encompass access to all network functionality, based on a decentralized network architecture, that could benefit from autonomous control (e.g. network components), as well as all control, optimization, and adaption functionality deployed therein.
2. **Abstraction and Genericity:** Autonomous networks require the network and all control loops that co-ordinate its operation to be abstracted by interfaces or composed of functional building blocks. We extend this to encompass the interaction between different modules (Section 3.4.1) in the network.
3. **Hybrid Intelligent System:** Autonomous networks require the ability to make informed decisions. As these decisions become more complicated – such as those performed by highly-skilled engineers – cognition becomes a pre-requisite. To take steps towards achieving this cognition, a combination of symbolic (structured knowledge and relationships) and connectionist (artificial neural networks) [16] approaches will be used, as opposed to any single technology alone.
4. **Functional composition:** Autonomous networks require a self-reflective design and the ability to adapt and improve their performance based on previous network operation and behaviour. Functional composition – the online construction of network components based on

available building blocks – enables a system to be more adaptive to new situations and to more easily integrate new technology.

5. **Experimental Evaluation:** Autonomous networks require online experimentation to evaluate the potential of new solutions or configurations in the operational environment, as modern communication networks are too complex for theoretical or simulation-based reasoning [17].

Based on the above, the need for standardisation is easily identified: standardised data models to facilitate holistic (i.e., domain agnostic) information sharing, clear specification and description languages to facilitate interaction among functional building blocks, shared ontologies for use by intelligent decision making, clear taxonomies of use cases to facilitate clear functional decomposition, as well as descriptions of simulations and network components to facilitate experimentation and canary testing. We thus believe there is a need for consensus from the community to achieve interoperable autonomous networks. A view shared by the recent formation of the ITU pre-standardisation focus group on autonomous networks.[1]

In the remainder of this section, we introduce several technologies that will enable us to realise these principles, along with a brief overview of how we intend to utilise them in our framework design, detailed in Section 3.

2.2 The Role of Intent

Many efforts have devoted a great deal of attention to the specification and handling of intent in guiding the purpose and operation of various aspects of the network [18–21]. Here, intent is defined by the IETF [22] as an abstract high-level policy used to operate the network:

> *at a level of abstraction that is much higher than that of typical configuration parameters, for example, "optimise my network for energy efficiency".*

The translation of such an abstract, high-level policy into a concrete low-level configuration, heuristic, or program that can be executed is not possible without additional knowledge. For example, through a pre-configured mapping of high-level policies to low level actions an automated agent is **told** what actions to take to optimize an objective.

[1] https://www.itu.int/en/ITU-T/focusgroups/an/Pages/default.aspx

Alternatively, optimisation techniques (such as reinforcement learning and genetic algorithms) can be used to tackle the above. These techniques usually depend on a fitness or reward function, which provides a measure of the suitability of a proposed solution. In other words, these approaches judge how well the intent is realized by a potential solution and use it to guide their incremental **search** for a better solution.

Abstractly, these two approaches can be compared as top-down vs. bottom-up. Given that an *autonomous* agent is expected to come up with a solution on its own without being explicitly told the mapping necessary to achieve its goal, we embrace a search-based bottom-up approach as it naturally reflects the concept of adaptation *by design*.

In our system, intent is therefore expressed as a combination of utility function and boundary conditions that constrain or guide the search process in achieving the overall policy. This approach enables dynamic and 'autonomous' optimisation of the employed heuristics to match the intent.

Sections 3.5 and 3.6 discuss how the above is realised in practise through the composition of software modules into *controllers*, which are responsible for some domain of operation in the network. Hardware composition is currently beyond the scope of this work.

2.3 The Control Loop

All manifestations of autonomy closely resemble the *feedback loop* [12], also known as the *autonomic control loop* [22, 23] (Figure 1) or MAPE-K

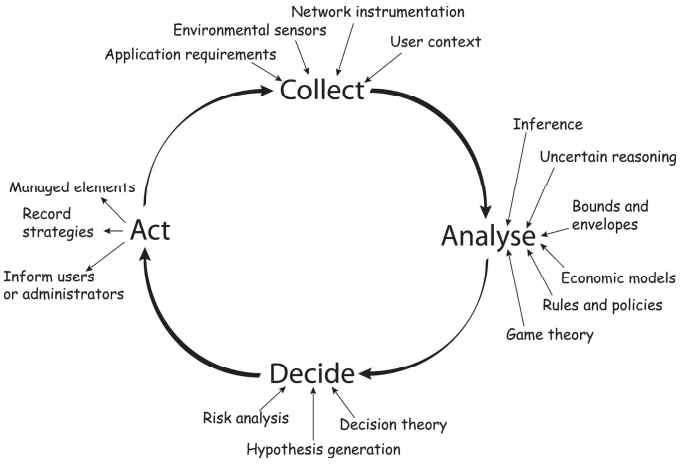

Figure 1 Autonomic control loop [23].

loop [25]. The *collect/sense* stage gathers and aggregates sensor data or other information from which the *analyse* stage derives an understanding of the current system and environment state. Combining this state and the predicted outcome of potential actions enables decisions of how to act in the *decide* stage, which are then applied during the *act* stage. These stages may execute independently, as in the OODA loop [26]. Given the complexity of the envisioned autonomous network, a single control loop is likely to be insufficient, hence we propose a *hierarchy* of control loops in Section 2.4.

2.4 The Control Hierarchy

Biological systems hierarchically control the actions of their subordinate entities. For example, brain functions in the frontal cortex make higher level decisions to run away from a threat but cannot implement them without the help of the cerebellum, which translates "run towards the door" into actions to be performed by the legs. The cerebellum in turn does not control the actions performed to counteract sudden events which need immediate actions. The latter is the domain of reflexes, which are hardcoded and able to respond faster than the cerebellum or the even-slower frontal cortex ever could.

The rationale for this split is that higher-level functionality is more costly and takes longer. Offloading these functions to less complex parts of the brain increases reaction speed and frees more advanced areas to handle more challenging tasks. The same challenges led the robotics field to adopt hierarchical layered architectures [27], as well as military and (some) corporate structure to make long-term strategic decisions at a higher rank than tactical or short-term ones.

Our vision for an autonomous network reflects this concept by providing multiple control loops in a hierarchical structure. We intend to model the process of how the cerebellum directs the actions of a leg, which can be overridden when needed by a reflex, within our network: for example, a higher-level controller assigns weights per data centre (DC), which are used to distribute application instances among machines in that DC.

In our design, however, the hierarchy is not limited to optimizing operations. The structure and composition of the underlying control loops themselves can be modified and improved by higher-ranked control loops. Furthermore, we do not employ a single, linear order of layers, but instead use a directed graph. In our approach, several subordinate loops can be controlled by one or multiple higher-ranking loops, responsible for different optimization aspects as described in Section 3.3.

2.5 Functional Composition

Modular designs that allow run time construction of functionality from functional building blocks have a long history in a large variety of computing systems [28–30] or networks [31–33]. Of note are the larger ANA [34] and 4WARD [35] projects where the framework could control runtime rebinding of communication channels between functional blocks [36].

By decomposing the logic currently being used in the management and operation of the network to functional (Lego) building blocks, our approach would seek to assemble and reassemble these blocks into closed loop controllers at runtime (see Section 3.5.1), reducing human involvement. In this way, the interaction with the network becomes decoupled from the autonomy that drives it, and the specific technologies used. Several existing standardised approaches support the description of such building blocks [37–40].

2.6 Artificial Cognition

Research into artificial thinking and reasoning is spread across many different areas of science and can be split into two distinct approaches: development of systems that think *rationally* or that think *like a human* [41]. This has led to cognitive architectures [42, 43].

Artificial Neural Networks (ANN) [41] are the main connectionist [44] approach. Despite limited pattern classification and function approximation capabilities [45], Fukushima's Cognitron [46] inspired convolutional neural networks and consequently the advent of Deep Learning [47]. Advances in computational power and data have enabled huge performance increases in several beat? application areas for ANNs, enabling Deep Reinforcement Learning to best humans at board (go, chess) and video games [48].

However, despite impressive results in certain areas, a general approach to new (never before seen) situations is still limited at best. Many researchers now acknowledge the limitations of Deep Learning approaches and explore potential solutions to advance the state of the art. For example, meta-learning [49] explores how to overcome inductive bias in learning algorithms through automatic optimization. Of particular interest to us is the idea of using a hierarchy of learning systems, where each layer learns how to improve the performance of the underlying layer through, for example, Genetic Programming [50]. This concept is applied recursively throughout the hierarchy, from top to bottom.

Ultimately, artificial intelligence is still extremely far from matching the cognitive abilities of a human being, and creating a human-like artificial

intelligence can be considered the holy grail of AI research. With this in mind, our framework is designed to be modular and decoupled to be as accepting as possible for future AI technology.

Even partial autonomy will allow us to reduce costs as well as increase the capabilities of the network and is therefore worth pursuing. We believe that the current state of AI research enables the creation of a *semi-autonomous* network through a combination of semantic/symbolic/connectionist/emergent approaches, as appropriate for the application context. In other words, no single AI/ML technology is sufficient to address all concerns of an autonomous network, requiring a "right tool for the right job" mentality: If context and logical relationships can reasonably be provided by an engineer, such as the relationship between analogue signals and temperate in a sensor, this can be leveraged by means of symbolic reasoning. This is similar to the concept of an expert system. Whenever this is infeasible, due to the amount of data, variation, or lack of knowledge about relationships, such as how exactly an intrusion attempt would manifest itself in the network, then we will focus on connectionist approaches – data-driven learning.

2.7 Metaheuristic Optimisation

Many, if not most, optimization problems encountered in computer networks can be expressed by means of a utility function. A utility function provides a measure of optimality for a potential solution, state, or configuration setting.

For difficult (high-dimensional, non-convex) or unknown search spaces, which cannot exhaustively be explored, common heuristics, such as hill-climbing [41] or simulated annealing [41], are not applicable. In such cases, the introduction of randomicity both helps to traverse the potentially vast search space of the network and also deals with uncertainty in cases where the underlying relationships between actions and outcomes are either unknown or difficult to reason about. Stochastic meta-heuristic approaches offer a flexible way of finding a sufficiently good, but not necessarily optimal, solution. Many examples of such approaches are based on, or inspired by, biological processes: genetics [41], evolution [51], ant colony behaviour [52], or biased random exploration [53].

As all these approaches – in the context of our approach described in Section 3 – share common inputs (e.g., current state, (partial) history information, utility, previous configurations) and outputs (e.g., new state or sets of states to explore) modularization and use as building blocks for

functional composition is straight-forward. By layering these approaches in inter dependent control loops, meta-meta-heuristic optimization is possible and is applied in our framework.

From a standardisation perspective, we consider several challenges: designing an architecture with a common abstraction layer, specifying the interface points with the underlying network, deciding on the state representation (and input/outputs) and in building appropriate protocols for inter-control loop communication. Given the wide variety of network contexts in which these control loops will operate, a one-size-fits-all approach is infeasible and further exploration will be required to find the appropriate technology.

2.8 The Case for Experimental Online Evolution

In 1997 the Internet was already too complicated to feasibly simulate [17] and the same claim can easily be extended to today's massive telecommunication networks. Even if the system itself was not complex, environmental interaction can easily make it so [54], especially considering that the traffic carried on these networks is predominantly human-initiated and -controlled.

Sentient beings overcome similar problems through online experimentation and continuous learning [55]. Similar approaches have discovered the best networking protocol to deploy in the network [56], the best radio parameters [57], found the best bit-rate for wireless communication [58], or reinvented modern technology [59].

The trade-off between exploration and exploitation, is a major issue for online search heuristics [41]: exploiting only the current knowledge can maximize immediate payoffs but can also penalize the agent for failing to explore even better options. The multi-armed bandit problem has been a particular focus of study, which led to strategies such as ε-greedy, Bayesian approaches, or approximate methods. In general, however, the best strategy is highly dependent on the scenario and a general solution for the full reinforcement learning case has yet to be found [60].

One additional caveat is the possibly exponential growth of the search space that needs to be explored experimentally if optimization is to be performed on multiple interdependent layers. The optimal strategy to this problem again depends on the peculiarities of the problem set.

Our framework design employs online experimentation of potentially better solutions for optimization problems (see Section 3.6). Note that the experimentation process is also subject to online optimisation. The use of online experimentation enables positive and negative evolutionary-driven

exploration to be addressed while at the same time reducing the human in the loop. The precise mechanism to drive this process is a topic of research.

3 A Framework For Evolutionary Autonomous Networks

Guided by the concepts and technologies above, we now present our vision for a future autonomous management of a telecommunication network. We describe how the key elements of our approach are embodied in our framework. To provide a clear narrative, engineering-focused topics are deferred.

3.1 Module: The Building Block of Functional Composition

The functional building block is a key element of our approach to autonomy and realized within our framework as a *module*, Figure 2.A module consists of a *software* element and corresponding *composition information*. As discussed above, in this work we focus on software operation. This structure enables us to compose controllers (described in Section 3.2) out of compatible modules using the process described in Section 3.5.

The software section represents the operational logic of a module and consists of three parts: *code*, *parameters*, and *API*. Code represents the logical operation of the module and is expressed as software. The scope, size, and choice of software technology is defined by the designer and may be of arbitrary size. Parameters are used to initialise and configure the operation of the code, for example, how long to wait before a timeout. Finally, there is a well-defined API which enables this code module to be interacted with.

The API is identified by a unique ID together with an arbitrary number of *optional* tags to specify the module's functionality. Examples include *lossless* or *lossy* for an audio codec API. Importantly, the API ID combined with the optional tags constitute a "contract" for composability and interoperability.

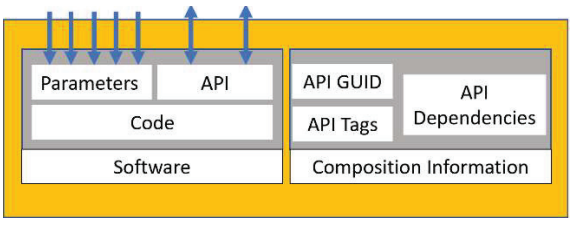

Figure 2 A module.

Modules can depend on APIs provided by other modules, where the same API can be provided by multiple different modules – in other words the same interface with different implementation behind it. Equally, differently parametrised instances of the same module can be utilised concurrently. In this way, inter-module dependencies are implicitly inferred via the APIs (and tags) used. We currently assume that software modules are user-defined: software generated modules can be accepted by our approach, but the generation of such modules is left for later discussion. To help clarify the role of the module, examples of potential modules may include, aggregation functions, DNS configuration interfaces, gathering Kubernetes statistics, an entire deep neural network (DNN) model, a single layer of a DNN model, etc. By analogy, just as a Java object is a container of logic, activity, or hardware interaction, so too is a module. This flexibility is intended so that:

- The burden of adopting future technologies is reduced
- Controllers (Section 3.2) can be created of arbitrary size and scope, where this scope can be that of a network function, network control function, or any other purpose, including the higher-level meta-evolution and self-reflection discussed in Section 3.3.

From this perspective, the 'code' of a module is not as important as the description of the module itself, as well as the interfaces and standardisation of how such modules are composed and interact.

Given the above, modules can now be programmatically (or automatically) composed together to create *controllers*, discussed below.

3.2 Controller: An Instantiation of the Control Loop

The control loop (Figure 1) is manifested in our framework as a *controller*, Figure 3. A controller is responsible for the control, operation, or optimisation of some task or domain within our network. Like modules, the scope is user-defined.

The four control loop phases (*sense, analyse, decide, act*) are present within a controller. Each controller phase operates on an independent time scale from the others. For example, sense can be a continuous process, whereas analyse is likely to operate only occasionally to interpret collected data. Decisions are either periodic or triggered by changes in the environment, and actions are in response to decisions. Each controller phase is a composition of modules and can be represented as a directed graph of nodes, where each node is one module instance. The root of this graph is the *sink*:

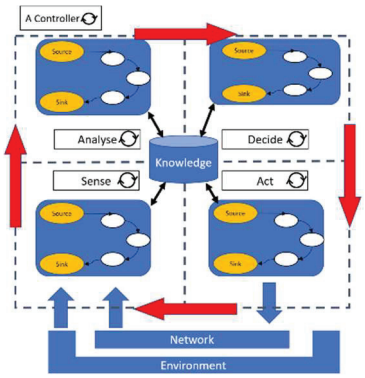

Figure 3 A generic controller architecture.

an abstract module which represents the required inputs of the next phase in the control loop to ensure that all necessary data is present. The next phase contains a corresponding *source* module, which provides access to the output of the previous phase. A detailed description of what exactly is required for each phase is provided by the *controller specification* (Section 3.4.3), in conjunction with additional requirements derived from the modules present within this next phase.

The vertices of this graph represent the dependencies between the modules. As different module combinations are possible, and different modules possess different dependencies, arbitrarily complex graphs (and hence controllers) are possible. It is the manipulation by means of creation, rearrangement, replacement, and configuration changes of module compositions that enables our framework to adapt to both new and evolving situations (see Section 3.5).

N.B.: our framework currently requires a user to provide a metric to measure the *fitness* of a controller by means of a utility function. Like modules, we do not prohibit auto-generation of utility functions but defer this to future work.

Finally, all controller phases share access to persistent knowledge through a *knowledge base*. This is necessary to understand previous choices and their consequences, changing system state, and to ease synchronisation across distinct update periods. The latter points being necessary to be able to checkpoint and restart controllers in the face of inter-node migration and error. The exact knowledge to be kept is dependent on the specific controller but can be utilized by separate controllers of the same or different types if applicable.

This related to the notion of holistic information, where data is used beyond the domain for which it is collected. The knowledge store is an eventually consistent distributed data store, which we will also discuss separately.

3.3 Controller Hierarchy: A Layered Approach to Control

Our approach uses flexible, runtime-defined hierarchies of controllers, which consist of *operation controllers* (OC) and *evolution controllers* (EC).

OCs either directly control network elements or supervise other operation controllers. Examples include heuristics for network load balancing, resource distribution, job scheduling logic, or anomaly detection and mitigation technologies.

ECs optimize and adapt the composition and configuration of other controllers to achieve the best possible utility under the current operation conditions. This is an exploratory task and is referred to as *evolution* (see Section 3.5). ECs can be under the supervision of other (meta) ECs and thus evolve themselves, as explained below.

Figure 4 separates controllers into conceptual layers by the roles they perform. Practically, all controllers are arranged in a single hierarchy, where all leaf nodes are OCs.

OCs may be controlled by distinct higher-level OCs which supervise and direct the operation of subordinate OCs. For example, leaf OCs may optimise some activity in data centres, but the bounds within which they operate may

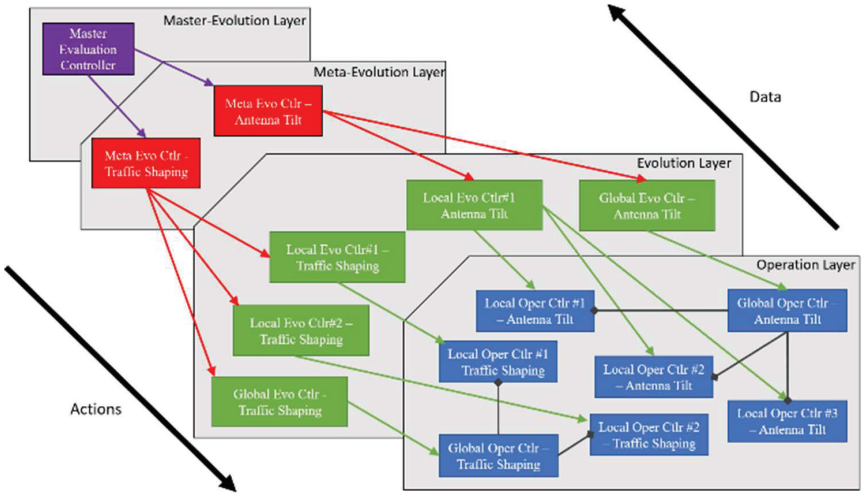

Figure 4 Example controller hierarchy split by layer.

be defined by a regional OC, above which a global OC may be deployed. OCs cannot influence the controller graph or the evolution process.

Conversely, ECs decide when and how to evolve the controllers they are responsible for. Unlike controller module compositions, each EC in the hierarchy graph may define its own dependencies: each EC defines the specification and number of controllers in the layer directly below itself, either via specification (Section 3.4.3) or programmatically at runtime. Thus, ECs may apply one or more independently evolved OCs locally, regionally, or globally at their discretion, compare the results, and decide which approach is most efficient, based on their utility function.

The OCs hierarchy enables the separation of local *reactive* decisions from *considered* global decisions. For example, a single base station controller can quickly adjust its tilt based on the number and conditions of connected devices, however, the global controller can get feedback from many local controllers and provide more general policy decisions at a larger temporal and geographical granularity. Thus, our framework offers the same underlying concepts that centralized and distributed SON [61] provides, such as the deployment of agents in each cell to co-ordinate and minimize interference.

Whether separation into multiple OCs is sensible or not depends on the use case and application environment. Hence, we allow ECs to decide their sub-graphs as it enables them to autonomously try out different operational separations of control and choose the right one for the current situation.

As well as OCs, ECs may control other ECs in a role we call a *meta evolution controller* (MEC). Having a hierarchical ordering of ECs enables us to apply different evolution approaches depending on the use-case and operational environment. For example, the strategy for in-data centre resource allocation may differ from the regional strategy (different time scale, explicit allocation to machines vs weights per application group).

The mapping of MEC to EC follows a similar logic. As the ideal number and role of ECs depends on the use case and application environment, it is the prerogative of the MEC to decide (i.e., evolve) the composition of the hierarchy below itself.

Figure 4 illustrates the potential of hierarchical control separation via two use cases: traffic shaping and antenna tilt optimization. For traffic shaping, a global OC decides the high-level weight allocations per location, and two local OCs shape the traffic, obeying the global weights. These OCs are independently evolved by their corresponding local or global EC. For tilt, all three local OCs are evolved by one EC. All ECs are evolved by

a corresponding MEC, which in turn are evolved by the *master evolution controller* (Section 3.5.2) This is one potential configuration amongst many.

Initially, it is the engineers' decision to decide whether an EC may try out different hierarchies of OCs and/or ECs, or whether the hierarchy should remain static and evolution should be limited to controller composition. The correct degree of freedom given to the system is a topic of future work.

3.4 Description Language: Meta-Data for Symbolic Reasoning, Controller Composition, and Use Case Specification

The description language steers the functional composition and derivation of meaning (symbolic reasoning) from sensor data. It provides a normalisation layer that enables our system to programmatically understand and reason about the modules and sensors provided. Additionally, it allows us to specify constraints for controllers, controller hierarchy branches, and utility functions, and thus to add new use case-specific controller (hierarchies).

3.4.1 Module description

Our framework composes controllers from modules by specifying module capabilities (e.g., sensing, compression, aggregation, configuration), configurable parameters, and interfaces using a description language. Such specification may be achieved in many ways [62–64]. Given space limitations, we defer description of our specific approach. Having each module provide a standard description enables equivalent but different modules to be interchanged programmatically.

Irrespective of the technology or description used, the standardisation of the specification of modules and their capabilities is a clear necessity to achieve interoperable autonomous networking.

3.4.2 Sensor description

There are two types of description for sensors. The first concerns the symbolic description of the sensors (e.g., thermistor, packet probe, energy meter), as well as the data types that they produce (e.g., degrees centigrade, packet loss, joules). Our framework requires the developer to provide this information via a specification. To assist, our framework will support a taxonomy of sensor types and data. Existing work in sensor networks [65] and more recently IoT [66] have made efforts to classify both sensor types and data by problem domain to better exploit the right tool for the right job. Our framework will do the same; just as modules are symbolically described, so too must

sensors within the context of our taxonomy. By doing so (1) sensors from one domain can be reused in another, (2) equivalent but different sensors can be interchanged, (3) classification can guide the process of "good" module compositions within a controller, where sensor access is encapsulated by a module (4) classification can help automate the process of sensor data aggregation and later reuse of this aggregation between similar sensors classes.

The second description type concerns the inference of meaning from the raw sensor data. In this case, the use of taxonomies combined with *ontologies* will enable these relationships to be inferred [67].

3.4.3 Controller description

The constraints that guide which controllers are required to be present in the framework for a particular use case are also specified by a description language. As explained in Section 3.2, the conceptual structure of the controller is constant and always consists of the *sense, analyse, decide*, and *act* phases. The connections between the phases are provided through the corresponding *source* and *sink* modules of each stage. Figure 4 shows such constraints as applied to a load balancing use case, in which one evolution controller is responsible for the evolution of two distinct load balancing operation controllers. Note that only the required outputs are specified explicitly. The required inputs are derived directly from the requirements of a composition that provides the needed outputs. These input requirements can also lead to additional output requirements for preceding phases.

The utility metric used to evaluate all controllers under an evolution controller can also be defined either in software (via a module provided for this purpose) or directly within the controller specification by means of a simple mathematical syntax. In accordance with the holistic approach (see Section 2.1) we take, any available information can be used for the utility estimation purpose, be it sensor data, module output provided by other controllers, or statistics gathered directly from the network.

3.5 Composition & Online Evolution

One of the key strengths of our framework is the ability to adapt and improve the operation of controlled network entities, but also to self-*evolve*. Specifically, to adapt and improve its own functionality. This is achieved by manipulation of controller module compositions and hierarchy topologies.

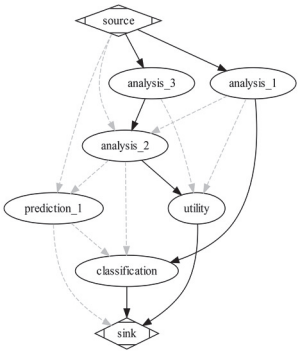

Figure 5 Composition #2.

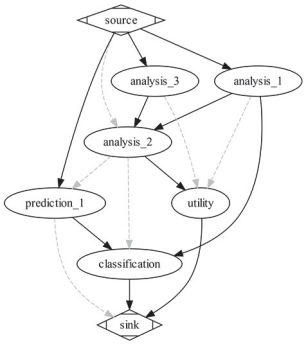

Figure 6 Composition #1.

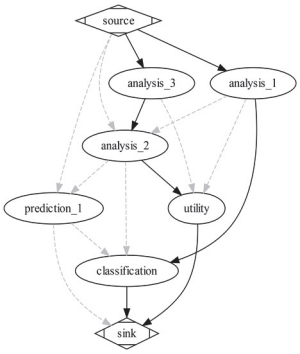

Figure 7 Search domain.

3.5.1 Controller evolution

As previously described, controllers are broken into four phases, where each phase is a composition of modules.

The set of all potential valid compositions (of module instances and parameter configurations) defines the search space for the evolution process. The composition process is illustrated in Figures 5–7. These figures show a number of modules, where the dotted lines represent all potential composable configurations of their APIs – the search domain. The module names are indicative of the logic contained within them, such as analysis, prediction, classification, or utility function. The logic of a module is irrelevant to the structural composition of the controller: only a module's API is relevant at this point (Section 3.1).

The search domain is represented by all possible paths from source to sink. Figures 5–7 show possible valid configurations. The evolution process can utilize any valid solutions within the search domain to instantiate a controller. How to search is discussed in Section 3.5.3. Hierarchy Evolution.

ECs are allowed to define how the sub-graph of controllers below them is composed *at runtime*.

The root controller is known as the *master evolution controller*. It is instantiated first and notifies the framework regarding the composition of the sub-graph below it, which is then instantiated recursively for each nested sub-graph. When an EC decides to change the composition of the sub-graph it manages, the above process is applied to that sub-graph alone.

The set of all potentially valid controller graphs (like the module graphs) constitutes the search space to be traversed by the evolution process. Given the large potential search space, iterative search is used. Specifically, we instantiate a limited number of controllers, evaluate their performance, and refine our search process based on the gained knowledge. Similarly, we iteratively generate one controller hierarchy and let it operate unmodified until an EC decides to change the sub-hierarchy. How to search is discussed in Section 3.5.3.

3.5.2 Traversing the compositional search space

Composition of controllers and the controller hierarchy is an iterative process. The search space that needs to be explored is complex and high-dimensional, where the number of dimensions is not static. For controller composition, every connection between module instances and configuration parameter constitutes one dimension. If a different module is chosen, the number and types

of parameters and dependencies changes. Likewise, for controller hierarchies, each EC can define the controllers of its own sub-graph and hence the number of dimensions.

Building on our previous work of discovering optimal solutions in such complex environments [15], we will enhance and apply the techniques developed to both controller composition and hierarchy evolution. The former case is straight-forward, but the latter case, requires future research.

3.6 Online Experimentation

With the ability to evolve controllers programmatically, our framework can now automatically generate *new* controllers. However, to understand their utility or fitness as applied to some domain of control we require *online experimentation*.

Within our framework, online trial-and-error experimentation is the responsibility of the experimentation *manager*. As described previously, we adopt a multi-layered approach to experimentation: First, new controllers are *sanity checked* to detect logical mistakes, such as a controller using a light sensor module where no such sensor exists, even though the module is available. The use of taxonomies and ontologies (common sense) can be used to assist in this. Next, candidate controllers are tested in simulation to estimate their utility. While not perfect, simulation tools, such as ns-3 [68], or Naos [69], can serve as indicators of potential success or failure. Finally, network trials see the experimentation manager responsible for limiting the (physical and temporal) scope of experiments, with gradual expansion based on a particular controller's measured (or estimated) utility.

Additionally, the experiment manager acts as coordinator for different concurrent controller experimentations as requested by multiple ECs or MECs. Intuitively, not all experiments can be run concurrently as conflicts and false utility may be observed due to experiments interfering. A telecommunications network is a large interconnected system meaning that interference is inevitable, however, the experiment manager seeks to limit this ensuring that experiments are *fair*, *meaningful,* and *representative* of the actual operation environment.

An experiment consists of the controller to test, its parameter configurations, utility functions, current experimental scope, and results. These results are used to determine if the candidate controller should replace an existing controller. Based on the provided information, potential conflicts in experimentation can be inferred.

4 Conclusion

In this paper we present our vision for reaching true autonomy in the communication networks of the future and propose an approach towards realizing this goal. Our vision is based on several core principles and key technologies, which together outline the road towards an autonomous, self-evolving architecture. This architecture will enable autonomy, will be flexible enough to encompass arbitrary future technologies, and will be concise enough to be deployable in actual networks. We believe that our focus on a holistic and evolution-based approach, which can *on its own* adapt to, and optimize for, *any* use case, is superior to both one-size-fits-all efforts and hand-crafted use case-specific designs.

Our future work will consist of implementing this framework not only for the purpose of experimentation, but for production deployment with the aim of enriching the first fully-deployed virtualized telecommunication network. As we learned through our efforts on autonomous network stack evolution, we expect the road towards this goal to be full of rocks and pitfalls, but by leveraging the experience we gained to overcome these issues, we aim to achieve the world's first truly *autonomous network*.

References

[1] M. Harris, *The End Of Absence: Reclaiming What We've Lost in a World of Constant Connection*. Harper Perennial, 2014.
[2] R. Stahlmann, A. Festag, A. Tomatis, I. Radusch, and F. Fischer, "Starting European field tests for Car-2-X communication: the DRIVE C2X framework," in *18th ITS World Congr. and Exhibition*, 2011, p. 12.
[3] J. V Jacobs et al., "Employee acceptance of wearable technology in the workplace," *Appl. Ergon.*, vol. 78, pp. 148–156, 2019.
[4] M. Jalasri and L. Lakshmanan, "A Survey: Integration of IoT and Fog Computing," in *2018 2nd Int. Conf. Green Comput. and Internet of Things (ICGCIoT)*, 2018, pp. 235–239.
[5] G. Davis, "2020: Life with 50 billion connected devices," in *2018 IEEE Int. Conf. Consum. Electron. (ICCE)*, 2018, p. 1.
[6] 3GPP, "Public Warning System (PWS) requirements," 2015.
[7] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile internet," *IEEE J. Sel. Areas Commu.*, vol. 34, no. 3, pp. 460–473, 2016.

[8] O. Babaoglu et al., "The Self-Star Vision," Springer, Berlin, Heidelberg, 2005, pp. 1–20.
[9] M. Smirnov, "Autonomic Communication—Research Agenda for a new Communication Paradigm. Company whitepaper," *Fraunhofer Inst. Open Commu. Syst., Berlin, Ger.*, 2004.
[10] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, 2003.
[11] P. Demestichas, V. Stavroulaki, D. Boscovic, A. Lee, and J. Strassner, "m@ANGEL: autonomic management platform for seamless cognitive connectivity to the mobile internet," *Commu. Mag. IEEE*, vol. 44, no. 6, pp. 118–127, Jun. 2006.
[12] N. Wiener, *Cybernetics*. Technology Press, 1949.
[13] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications – SIGCOMM '03*, 2003, p. 3.
[14] P. Imai and C. Tschudin, "Practical online network stack evolution," in *Proceedings – 2010 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop, SASOW 2010*, 2010, pp. 34–41.
[15] P. Imai, "Exploring Online Evolution of Network Stacks," University of Basel, Switzerland, 2015.
[16] P. Smolensky, "Connectionist AI, symbolic AI, and the brain," *Artif. Intell. Rev.*, vol. 1, no. 2, pp. 95–109, 1987.
[17] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," in *Proc. the 29th Conf. Winter Simul.*, 1997, pp. 1037–1044.
[18] 3GPP, "Telecommunications Maangement: Study on Scenarios for Intent Driven Management Services for Mobile Networks," 2020.
[19] A. Clemm, L. Ciavaglia, L. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Overview," 2019.
[20] T. Szigeti, D. Zacks, M. Falkner, and S. Arena, *Cisco Digital Network Architecture: Intent-based Networking for the Enterprise*. Cisco Press, 2018.
[21] A. Rafiq, M. Afaq, and W.-C. Song, "Intent-based networking with proactive load distribution in data center using IBN manager and Smart Path manager," *J. Ambient Intell. Humaniz. Comput.*, pp. 1–18, 2020.
[22] M. Behringer et al., "RFC 7575: Autonomic Networking: Definitions and Design Goals," 2015.
[23] J. Mitola, "Cognitive Radio Architecture Evolution," *Proc. IEEE*, vol. 97, no. 4, pp. 626–641, Apr. 2009.

[24] S. Dobson, E. Bailey, S. Knox, R. Shannon, and A. Quigley, "A first approach to the closed-form specification and analysis of an autonomic control system," in *ICECCS '07: Proc. 12th IEEE Int. Conf. Engineering Complex Computer Systems*, 2007, pp. 229–237.
[25] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer (Long. Beach. Calif).*, vol. 36, no. 1, pp. 41–50, 2003.
[26] J. Boyd, G. T. Hammond, and A. U. (U.S.). Press, *A Discourse on Winning and Losing*. Air University Press, Curtis E. LeMay Center for Doctrine Development and Education, 2018.
[27] E. Gat, R. P. Bonnasso, R. Murphy, and A. Press, "On Three-Layer Architectures," in *Artif. Intell. and Mobile Robots*, 1997, pp. 195–210.
[28] D. M. Ritchie, "A Stream Input-Output System," in *AT&T Bell Laboratories Technical J.*, 1984, pp. 311—324.
[29] R. G. Guy et al., "Implementation of the Ficus replicated file system," in *USENIX Conf. Proc.*, 1990, vol. 74, pp. 63–71.
[30] P. Harvey, *A linguistic approach to concurrent, distributed, and adaptive programming across heterogeneous platforms*. 2015.
[31] N. C. Hutchinson and L. L. Peterson, "The X-Kernel: An Architecture for Implementing Network Protocols," *IEEE Trans. Softw. Eng.*, vol. 17, no. 1, pp. 64–76, 1991.
[32] S. Patarin, S. Patarin, M. Makpangou, M. Makpangou, and S. Pat, "Pandora: A Flexible Network Monitoring Platform," in *Proc. USENIX 2000 Annu. Tech. Conf.*, 2000, p. 200.
[33] F. Ogel, S. Patarin, I. Piumarta, and B. Folliot, "C/SPAN: a self-adapting web proxy cache," in *Autom. Comput. Workshop. 2003. Proc.*, 2003, pp. 178–185.
[34] "Autonomic Network Architecture Project."
[35] "The FP7 4WARD Project."
[36] M. Harris et al., "Digital Transformation Initative Telecommunications Industry," *communicate*, Jan-2019. [Online]. Available: https://ieeexplore.ieee.org/document/8399482/. [Accessed: 05-Apr-2019].
[37] A. Brogi, J. Soldani, and P. Wang, "TOSCA in a Nutshell: Promises and Perspectives," in *Service-Oriented and Cloud Computing*, 2014, pp. 171–186.
[38] K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz, "Verifying Safety Properties with the TLA$^+$ Proof System," in *Proceedings of the 5th International Conference on Automated Reasoning*, 2010, pp. 142–148.
[39] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2012.

[40] IETF, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," 2010.
[41] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach, 2nd edition," Prentice-Hall, 2003, pp. 1–5.
[42] J. E. Laird, *The Soar Cognitive Architecture*. The MIT Press, 2012.
[43] F. E. Ritter, F. Tehranchi, and J. D. Oury, "ACT-R: A cognitive architecture for modeling cognition," *WIREs Cogn. Sci.*, vol. 10, no. 3, p. e1488, 2019.
[44] M. Minsky, "Logical versus Analogical or Symbolic versus Connectionist or Neat versus Scruffy," *AI Mag.*, vol. 12, no. 2, pp. 34–51, Apr. 1991.
[45] M. Minsky and S. Papert, "Perceptrons: An essay in computational geometry," *MIT Press*, 1969.
[46] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biol. Cybern.*, vol. 20, no. 3–4, pp. 121–136, 1975.
[47] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.
[48] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A Survey of Deep Reinforcement Learning in Video Games," *arXiv Prepr. arXiv1912.10944*, 2019.
[49] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artif. intell. Rev.*, vol. 44, no. 1, pp. 117–130, 2015.
[50] J. Schmidhuber, "Ultimate cognition à la Gödel," *Cogn. Comput.*, vol. 1, no. 2, pp. 177–193, 2009.
[51] T. Bäck, *Evolutionary Algorithms in Theory and Practice - Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.
[52] M. Dorigo and G. Di Caro, "Ant colony optimization: a new metaheuristic," in *Proc. 1999 Congr. Evolutionary Comput. – CEC99 (Cat. No. 99TH8406)*, 1999, vol. 2, pp. 1470–1477.
[53] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.
[54] H. A. Simon, "The Sciences of the Artificial," 3rd ed., Cambridge, MA: MIT Press, 1996, pp. 51–52.
[55] D. Wood, J. S. Bruner, and G. Ross, "The Role of Tutoring in Problem Solving," *J. Child Psychol. Psychiatry*, vol. 17, no. 2, pp. 89–100, 1976.
[56] J. J. Ramos-Munoz, L. Yamamoto, and C. Tschudin, "Serial Experiments Online," in *ACM SIGCOMM Comput. Commu. Review*, 2008, vol. 38, no. 2, pp. 31–42.

[57] G. D. Troxel et al., "Adaptive Dynamic Radio Open-source Intelligent Team (ADROIT): Cognitively-controlled Collaboration among SDR Nodes," in *Netw. Techn. for Software Defined Radio Networks, 2006. SDR '06.1st IEEE Workshop*, 2006, pp. 8–17.

[58] J. C. Bicket, "Bit-rate selection in wireless networks," Massachusetts Institute of Technology, 2005.

[59] E. Real, C. Liang, D. R. So, and Q. V Le, "Automl-zero: Evolving machine learning algorithms from scratch," *arXiv Prepr. arXiv2003.03384*, 2020.

[60] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction, 2nd ed," MIT Pressl, 2017, pp. 19–35.

[61] 3GPP, "36.902: Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Self-configuring and self-optimizing network (SON) use cases and solutions," 2008.

[62] O. Faldik, R. Payne, J. Fitzgerald, and B. Buhnova, "Modelling system of systems interface contract behaviour," *arXiv Prepr. arXiv1703.07037*, 2017.

[63] B. Porter, M. Grieves, R. R. Filho, and D. Leslie, "REx: A Development Platform and Online Learning Approach for Runtime Emergent Software Systems," *12th USENIX Symp. Oper. Syst. Des. Implement. (OSDI '16)*, pp. 333–348, 2016.

[64] M. Harris et al., "Digital Transformation Initative Telecommunications Industry," *communicate*, Jan-2019.

[65] R. Jurdak, C. V. Lopes, and P. Baldi, "A framework for modeling sensor networks," in *Proc. Building Softw. for Pervasive Comput. Workshop at OOPSLA*, 2004, vol. 4, pp. 1–5.

[66] V. Rozsa et al., "An Application Domain-Based Taxonomy for IoT Sensors.," in *ISPE te*, 2016, pp. 249–258.

[67] M. Eid, R. Liscano, and A. El Saddik, "A novel ontology for sensor networks data," in *Proceedings of 2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, CIMSA 2006*, 2006, pp. 75–79.

[68] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator," in *Modeling and Tools for Netw. Sim.*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34.

[69] Forsk, "Naos Overview." 2020.

Biographies

Pierre Imai is Head of the Research & Innovation Department at Rakuten Mobile and has been working for the Rakuten Institute of Technology as Principal Scientist since 2016. His research and work experience includes the fields of networking & telecommunications, self-optimizing & autonomous systems, internet-scale distributed systems, artificial intelligence & machine learning, and robotics. Before joining Rakuten, he was improving the content distribution and active network monitoring systems down to network protocol level at Google, working on future internet and telecommunications research at NEC Labs Europe, and participated in multiple EU and national research projects such as ANA and Onelab. He is especially interested in building systems that can learn and improve themselves, for example, a network stack that autonomously re-composes and re-configures itself to reach close-to-optimal performance based on the current state of the network and traffic – as he did for his PhD, on a research grant from the Swiss National Fund.

Paul Harvey received his doctorate in Computing Science from the University of Glasgow focusing on heterogeneous adaptive systems. He possesses extensive experience in academia, including multiple international collaborations and positions, and is currently bridging academic and industrial

research. He is one of the original founders of the Autonomous Networks Research & Innovation Lab in Rakuten Mobile Japan and is passionate about pursing collaborative and open research.

Alexandru Tatar is a researcher at Rakuten. He received an Engineering Degree from Polytechnic University of Bucharest and a Ph.D. from UPMC Sorbonne University, France. His work focuses on autonomous networks and on studying user behaviour in e-commerce.

Leon Wong is the Industry Research Collaboration Lead for Rakuten Mobile Autonomous Networking Research & Innovation Lab. He has over 15 years experience in Telecommunications and IT, as a Subject Matter Expert in OSS, Solution Architect and Lead Engineer for international projects. Prior to joining Rakuten Mobile, he was responsible for building GPU High Performance Computing clusters for Rakuten Technology Division.

Laurent Bringuier graduated from Paris-Sud University with a maîtrise degree in electrical and computer engineering. Laurent has over 20 years experience in the telecommunication industry working in a range of companies from small start-up operators to large international vendors. In that time, he gained experience as a network engineer, OSS IT architect, and OSS project manager on network and infrastructure resource management. He transitioned to R&D, working in Rakuten Institute of Technology as a research project manager on various applied machine learning projects from inception to conclusion. Now, Laurent is one of the original members of the Autonomous Networks Research & Innovation Dept in Rakuten Mobile Japan. Here he is leveraging his pragmatic telco knowledge and experience in applied research management to contribute to the creation of an autonomous network.