On Hardware Programmable Network Dynamics with a Chemistry-Inspired Abstraction

Massimo Monti, Manolis Sifalakis, Christian F. Tschudin, and Marco Luise

***Abstract*—Chemical algorithms (CAs) are statistical control algorithms described and represented as chemical reaction networks. They are analytically tractable, they reinforce a deterministic state-to-dynamics relation, they have configurable stability properties, and they are directly implemented in state-space using a high-level –visual– representation. These properties make them attractive solutions for traffic shaping and generally the control of dynamics in computer networks.**

In this paper, we present a framework for deploying chemical algorithms on Field Programmable Gate Arrays (FPGA). Besides substantial computational acceleration, we introduce a low-overhead approach for hardware-level programmability and reconfigurability of these algorithms *at runtime*, and without service interruption.

We believe that this is a promising approach for expanding the control-plane programmability of Software Defined Networks (SDN), to enable programmable *network dynamics*. To this end, the simple high-level abstractions of chemical algorithms offer an ideal *northbound* interface to the hardware, aligned with other programming primitives of SDN (*e.g.,* flow rules).

***Index Terms*—Chemical algorithm, Programmable networks, Software defined networking, Traffic shaping, FPGA.**

I. Introduction

Research and engineering efforts in *Software Define Networks* (SDN) and *Network Function Virtualisation* (NFV) propose solutions to make the Internet infrastructure more volatile and software programmable. Although SDN research initially focused on control-plane operations, more recently attention has also shifted towards the data-plane functionality. Efforts in this direction [1], [2], [3] propose FPGA technology as the means to provide a volatile data-plane for a SDN switch. A main objective is software-configurable network dynamics.

Network dynamics is a loosely defined term for various transient phenomena emerging in a queueing network. These phenomena arise from the way packet flows multiplex, aggregate, and access shared network resources (memory at the intermediate nodes and link capacities). Common network functions for controlling network dynamics include flow scheduling, traffic shaping and policing, differentiated services, and active queue management (AQM).

Initial submission 2014. This work has been supported in part by Swiss National Science Foundation grant #132525.

M. Monti is with Elettronica Monti, P.zza G.Rossa 15, 56024 Ponte a Egola (PI), Italy (e-mail:massimo@elettronicamonti.it). This work was conducted as part of his PhD thesis at the University of Basel.

M. Sifalakis is with the IBM Zurich Research Lab, CH-8803 Rüschlikon, Switzerland (e-mail:emm@zurich.ibm.com). He was previously in the Dept. of Mathematics and Computer Science, University of Basel.

M. Luise is a Prof. in the Information Engineering Department, University of Pisa, V. Caruso 16, 56126 Pisa, Italy (e-mail:marco.luise@iet.unipi.it).

C. Tschudin is a Prof. in the Department of Mathematics and Computer Science, University of Basel, Spiegelgasse 1, 4051 Basel, Switzerland (e-mail:christian.tschudin@unibas.ch).

In this paper, we propose the use of FPGA technology to create a programmable data plane, on which one can implement functions for controlling network dynamics. For the first time, we pair an effective *overlay technology* on FPGAs and an unusual representation for algorithmic logic derived from *chemical reaction network* engineering. The achievement is high performance and representation simplicity, two desirable properties for network dynamics control functions. While a "*chemical*" representation of algorithms seems unconventional at first glance, upon use, it becomes an intuitive process of describing control systems directly in state-space (real chemical reaction networks are dynamical systems). As we demonstrate in this paper, this approach has two notable advantages: *(i)* an accurate and straightforward mapping of a mathematical model to an algorithm and its implementation on programmable hardware, and *(ii)* short conception-to-deployment timescales on FPGA fabric. The approach we propose is very much aligned with the way how SDN solutions orchestrate topology changes nowadays.

Specifically in this paper we extend previous work [4], [5] in the topic of *Chemical Algorithms* (CAs), and make the following new contributions:

1) Auto-generation of parallelisable, fast-to-deploy implementations of mathematical models, without requiring intermediate *translation* to a discrete program (nor HDL code for hardware programming).
2) *Runtime* re-programming and re-configuration on FPGAs, which is more lightweight than *partial-reconfiguration* technology [6], [3], and simpler than compiler-dependent execution environments, e.g., [7].
3) Simple, expressive representation of algorithms for controlling network dynamics, consistent with SDN rule-based programmability yet extended to data plane functions.

The remaining of the paper is structured as follows. In Sect.II, we discuss the needs that motivated this work. In Sect.III, we summarise the theory of CAs, focussing on key aspects that influenced the design of our hardware-programmability framework. In Sect.IV and Sect.V, we present our system design and we provide a validation of various programmability-related aspects on our prototype with Xilinx FPGA devices. Among others, we show-case various "real-world" examples of network dynamics control functions. In Sect.VI, we revisit our main contributions in more detail and in the context of the related literature. In Sect.VII, we conclude the paper.

II. Motivation and Requirements

Programmable hardware, based on FPGA technology, is an appealing solution for enabling a reconfigurable data-plane. However, SDN-style programmability, as well as "on-the-fly" deployment of functions for network dynamics control, sets-out challenges that make this task not straightforward. These challenges mainly involve finding expressive primitives for algorithmic logic, engineering the runtime interface to the programmable fabric, as well as achieving narrow timescales and small resource overhead for dynamic deployment.

Formally, algorithms for controlling network dynamics implement functions of multiple inputs, driven by events in the system's *fast* (data) path. Despite being data-plane functions, architecturally, they do not need to be embedded in the packet processing pipeline *per-se*. Instead, simple event hooks along the data-path can provide a sufficient and lightweight integration interface (*e.g.,* packet enqueue and dequeue processes). Typically, such hooks are already available as packet monitors/meters in suites like OpenFlow [8], OpenVSwitch [9] and Netfilter [10]. Such a decoupling allows these (often computationally intensive) functions to execute independently and in parallel to the packet processing workflow. As far as the dynamic composition and management of data-path pipeline are concerned (*e.g.,* FIB configuration, routing, packet header processing, *etc.*), these functions do not introduce a significant complexity to existing SDN primitives.

Nevertheless, network dynamics functions control the temporal behaviour of flows as they traverse a node's data path. Thus, functions' operational resolution must be sufficient to avoid introducing undesired latency overheads to the various stages of the packet-processing pipeline. To achieve high performance (ideally, higher speeds than the line speed of the network interface), network dynamics control functions have to exploit dedicated processing resources and to be hosted on acceleration hardware (FPGA, ASIC, *etc.*).

Two main questions, however, emerge. *(i)* Can we achieve the timescales for on-the-fly customisation that SDN-style network re-configuration requires? *(ii)* Can we balance simplicity versus algorithmic expressibility at the *northbound* interface, to effect SDN-programmability? The former is a hot topic of current research on FPGA technology (e.g. programmable overlay IP cores, accelerator virtualisation, partial reconfiguration, etc). Often, sought solutions are not lightweight. For example, partial reconfiguration in the SDNet [3] framework from Xilinx duplicates spatial resources on chip to implement a dual-image/buffer solution, in order to meet timescales for on-the-fly TCP-IP stack re-configuration. Regarding the tradeoff between simplicity and expressibility of algorithms, current SDN primitives seem at first glance overly restrictive for expressing state-machines or describing the temporal and differential behaviour of a control algorithm. On the other hand, a compile time API in Hardware Description Language (HDL) renders implementation of algorithms tedious and slow. Moreover, a discrete programming based on a high-level language, *e.g.,* [11]-[14] (as in High-level Synthesis, or with OpenCL), always brings to multiple implementations, often characterised by different/incompatible temporal behaviour and resolution.

III. Chemical Algorithms (CAs) and the Control of (Network) Dynamics

Chemical Algorithms, or *Chemistry-inspired Algorithms* (CAs) refer to a class of stochastic algorithms whose logic is described and implemented as a chemical reaction network. Inputs, outputs and internal states are represented by concentrations of molecular species, and their (mathematical) relationships are represented by reaction rules. Such a representation offers a few very simple primitives that can be implemented on hardware logic using only integer arithmetic (addition, multiplication).

The operation of CAs obeys to kinetics laws of Chemistry (the Law of Mass Action and conservation laws), which dictate and influence the behavioural characteristics of the algorithm. Thanks to these laws, CAs are *robust*, *analysable*, and guarantee *deadlock-free operations*. CAs are dynamical systems that continuously process event signals, and respond consistently to errors or perturbations [4], [15], [16]. CAs are stochastic but with a deterministic average behaviour that can be accurately described by a system of equations, directly derived from its (graphical) representation as a reaction network [4], [16]. For this reason, numerous interesting projects have formalised the chemical metaphor as a general computational and programming model *e.g.,* [17]-[22].

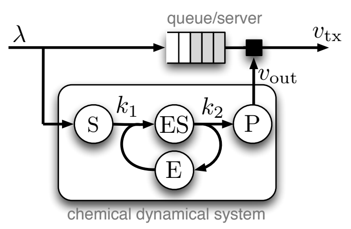
A. Representation of CAs

Instead of state diagrams or pseudocode that describe a sequential logic, the logic of CAs is (suitably) expressed (and visualised) in *drawings* of chemical reactions among molecular species (*e.g.,* Fig. 1(a), rounded-corner square). The species represent the algorithm's inputs, outputs, and internal state variables. The reaction network diagram encodes the parameters that control the behaviour of the system (reaction coefficients and reactant stoichiometric coefficients). A reaction captures a causal relationship between the system's state-variables (reactants and products). Formally, a reaction network (and therefore a CA) is represented by a set $\mathcal{S}$ of molecular species (variables), and a set $\mathcal{R}$ of reaction rules in the general form

$$r \in \mathcal{R}: \quad \sum_{s\in\mathcal{S}} \alpha_{r,s} s \xrightarrow{k_r} \sum_{s\in\mathcal{S}} \beta_{r,s} s \qquad (1)$$

This equation specifies how reactant molecules interact to create product molecules. For a reaction r, k_r is a constant parameter, known as *reaction coefficient*, which regulates the relative speed of the reaction (more details later). Parameter $\alpha_{r,s}$ is the *stoichiometric reactant coefficient*, specifying the number of molecules of a species $s \in \mathcal{S}$ consumed by reaction r. Similarly, parameter $\beta_{r,s}$ is the *stoichiometric product coefficient*, specifying the number of molecules of a species $s \in \mathcal{S}$ produced by reaction r.

A simple example that illustrates a chemical traffic control algorithm is shown in Fig. 1. As we demonstrate in the following, similar to the traditional Token Bucket (TB) scheme, this chemical mechanism can be used to control the



(a) System

$$r_1 : \mathrm{S} + \mathrm{E} \xrightarrow{k_1} \mathrm{ES}$$
$$r_2 : \mathrm{ES} \xrightarrow{k_2} \mathrm{E} + \mathrm{P}$$

(b) Reactions

Fig. 1. `Rnet1`: The enzymatic reaction network used as a traffic rate controller (pacing and rate capping). CA's input is connected to a queue's arrival process and the CA's output controls the queue's service process.

service process of a queue and rate cap the outgoing traffic up to a predefined, adjustable threshold. The service process, implemented with a CA, is graphically shown in Fig. 1(a) and its logic is formally described by reactions r_1 and r_2 in Fig. 1(b). For each enqueued packet (or certain amount of bytes), a molecule of species S is created. The dequeueing and transmission of a packet is authorised by the execution of reaction r_2, which implies the production of a P molecule and the consumption of an ES molecule. The production of ES molecules in turn is controlled by reaction r_1, and depends on S molecules (arrivals of packets in the queue) and the availability of E molecules, which embody tokens. Molecules of species E (tokens) are replenished from the separation of ES molecules at the rate at which reaction r_2 occurs. Overall, the effective queue service policy is *non work-conserving*: the queue is not served as fast as possible; its service is instead regulated by the relationship between rates of reactions r_1 and r_2 (as explained in the next section).

B. Operation and Dynamical System Aspects

Dynamics of CAs (when and which reaction is executed) are regulated by the *Law of Mass Action* (LoMA). The LoMA [23] states that the average rate $v_r(t)$ of occurrence of a chemical reaction $r \in \mathcal{R}$ is proportional to its reactant concentrations:[1]

$$v_r(t) = k_r \prod_{s \in \mathcal{S}} c_s^{\alpha_{r,s}}(t) \ , \tag{3}$$

where $c_s(t)$ is the amount of molecules of species $s \in \mathcal{S}$ at time t ($c_s(t)$ can be seen as a time-continuous, discrete-valued signal that the system processes), and k_r is the coefficient that regulates the reaction speed (regulating the relationship between molecular mass and rate). Reactant concentrations affect the reaction speed in a non-linear way, based on the stoichiometric reactant coefficients; the sum $\sum_{s \in \mathcal{S}} \alpha_{r,s}$ of reactant coefficients of a reaction r is known as *reaction order*.

The LoMA couples the state and the dynamics of the system, and plays a key role in CAs (as a self-adaptive internal scheduler). For example in the (enzymatic) rate controller in Fig. 1, the effectiveness of the loop (E–ES) to control the transmissions (generation of P molecules) stems from the strict relation that the LoMA imposes between the current state of the system (how many transmissions have been authorised and how many packets await in the queue) and the rate along the E–ES loop. By comparison, work-conserving scheduling disciplines would cause tokens to loop infinitely accelerated, making the mechanism ineffective to shape and limit the traffic.

The other operational principle behind the automatism of the control loop is the *mass-conservation* law, which states that the total sum of molecular concentrations along a loop remains constant if *(i)* the total number of molecules consumed by reactions along the loop is equal to the total number of molecules produced, and *(ii)* all concentrations along the loop are altered only by reactions involved in this or another loop. It follows that, in the (enzymatic) rate controller in Fig. 1, the number of tokens $c_{\mathrm{E}} + c_{\mathrm{ES}} = e_0$ is conserved. This limits the maximum number of P molecules that can be generated per second, and thus enforces a rate cap to the packet transmission.

C. Modelling and Analyzability

In CAs, the dual relationship between system state and dynamics guarantees an exact/accurate mathematical description of the system. This makes signal- and control-theory viable tools to analyse the behaviour of the algorithm.

Specifically, the behaviour of each CA is mathematically expressed as a fluid model, *i.e.,* a set of Ordinary Differential Equations (ODEs) of the form

$$\dot{\mathbf{c}}(t) = \mathbf{\Xi} \cdot \mathbf{v}(\mathbf{k}, \mathbf{c}(t)). \tag{4}$$

The term on the left-hand side represents the vector of state changes (concentration variations), whereas the right-hand side specifies how reactions effect these changes. The stoichiometric matrix $\mathbf{\Xi}$ captures the topology of the reaction network, whereas the reaction rate vector $\mathbf{v}$ encodes the speeds of all reactions, by combining reaction coefficients $\mathbf{k}$ and concentrations $\mathbf{c}$ according to the LoMA in (3). For example, referring back to our rate controller in Fig. 1, and given the reaction set in Fig. 1(b), the resulting system of ODEs is

$$\begin{bmatrix} \dot{c_{\mathrm{S}}}(t) \\ \dot{c_{\mathrm{E}}}(t) \\ \dot{c_{\mathrm{ES}}}(t) \\ \dot{c_{\mathrm{P}}}(t) \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot [k_1 c_{\mathrm{S}} c_{\mathrm{E}} \ \ \overbrace{k_2 c_{\mathrm{ES}}}^{v_{\text{tx}}} \ \ \lambda]^T \tag{5}$$

where the term $k_2 c_{\mathrm{ES}}$ reflects the rate of reaction r_2 and thus the dequeueing/transmission rate v_{tx}.

From the developer's perspective, the stoichiometric matrix $\mathbf{\Xi}$ provides the means to program any certain CA, and the reaction coefficient vector $\mathbf{k}$ represents the means to calibrate/tune it. The concentration vector $\mathbf{c}(t)$ then represents changes in the CA's state, as the system evolves over time.

From (5), it follows (by solving the homogeneous system for the steady state) that, as long as $\lambda < e_0 k_2$, the concentration c_{S} remains stable and the steady-state transmission rate v_{tx}^* follows the packet arrival rate λ (see [24] for more details). On the other hand, by applying the mass conservation law ($c_{\mathrm{E}} + c_{\mathrm{ES}} = e_0$), one obtains the Michaelis-Menten (biochemical) equation:

$$v_{\text{tx}}^* = k_2 c_{\mathrm{ES}} = \overbrace{e_0 k_2}^{v_{\max}} \frac{c_{\mathrm{S}}}{(k_2/k_1) + c_{\mathrm{S}}} \ ,$$

[1]The rate value found in (3) can be regarded as a simplified value quantifying the propensity a_r of a reaction r to occur.

from which it follows that, when $\lambda > e_0 k_2$ and thus when c_S grows without bounds, the transmission rate v_{tx}^* grows asymptotically towards the rate cap of $v_{\max}$, given by the product of the terms e_0 and k_2. The ratio k_2/k_1 controls how fast the rate limit is enforced.

From the transient/sensitivity analysis in [24], it also stems that the control algorithm has a low-pass filtering behaviour. The cut-off frequency is directly controllable through k_2-coefficient (*i.e.,* higher k_2 values lead to higher cut-off frequencies; the outgoing traffic from the system is more bursty).

IV. Enabling CAs on FPGA Technology

Having discussed CAs from a formal perspective, we can now describe how to exploit the introduced concepts in order to engineer a generic programmable hardware platform for CAs, on FPGA technology. The underlying intent is having a means to programmatically express functions to control network dynamics in an FPGA-based data-plane.

The fundamental abstraction we sought to provide is a "chemical engine", which serves two important purposes. One is to hide low-level hardware logic description inside "chemical" primitives, exploited to obtain a high-level representation for CAs by means of reaction networks. The other is to divide the programming/deployment time of CAs based on a two-level configuration process.

At the lower level (level-1), configuring a chemical engine pre-allocates chemical resources on the FPGA and creates an execution environment, as an *overlay* core [2], [25], [26]. Practically, a large enough grid of chemical resources are reserved (hardware logic circuitry on the FPGA). Level-1 configuration process thus involves automatic RTL synthesis and bitstream generation, followed by "slow-path" programming at *device initialisation*. This, as any execution environment, provides all background functionalities for accommodating different CAs and embodies internally the chemical kinetics "rules" for executing them. Furthermore, by resorting to partial reconfiguration technology [27], [28], device initialisation with multiple chemical engines can be effected without power-cycling and while other accelerated functions are still running on the FPGA.

At the higher level (level-2), the actual programming of CAs is effected as a configuration task which assigns part of the chemical resources and interconnects them in the corresponding reaction network. This involves the setting of values in memory-mapped registers on the FPGA, for the number of species, the initial values for their concentrations, and the set of reactions with their coefficients. These resources can be re-assigned or modified at any time (through a new level-2 configuration), in order to implement another CA. Level-2 configuration is the essence of *fast runtime* programmability of CAs, since it makes hardware programming a mere update of register values within the overlay core of the chemical engine. How fast level-2 modifications are effected depends on the access-time to registers on the specific FPGA device and is determined by the latency characteristics of the employed I/O interface and protocol (*e.g.,* GPIO, PCIe, buffered I/O, *etc.*).

Very frequent updates of CAs may interfere with the convergence time to new steady-states of the implemented control-system (CA). However, in such a case, the stability properties of a CA are straightforward to infer from chemical kinetics analysis [4], and the required queue characteristics are thereafter easy to profile. In this way, the accuracy in executing CA is not compromised by its implementation. Moreover, although not addressed in our prototype implementation, introducing queuing and batch buffered updates can address the potential issues of frequent updates.

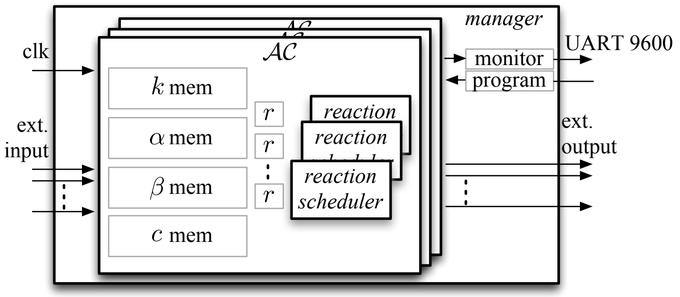


Fig. 2. Block-diagram illustrating the main components of the chemical middleware platform for programming CAs on FPGA hardware.

Conceptually, the instantiation of a CA (at level-2) within the chemical engine (configured at level-1) implements a so-called *Artificial Chemistry* [29] $\mathcal{AC} = \{\{\mathcal{S}\}, \{\mathcal{R}\}, \mathcal{A}\}$. At level-1, the chemical engine provides the LoMA reaction scheduling logic $\mathcal{A}$ in the execution environment. At level-2, CA instantiation provides the structural information of a species set $\{\mathcal{S}\}$ and a reaction set $\{\mathcal{R}\}$.

A. Prototype Platform Overview (Level 1)

The key building blocks (operational modules and functional structures) of our platform are shown in the block diagram of Fig. 2. Runtime operation is divided across three main nested modules: *(i)* the `manager` module, *(ii)* the $\mathcal{AC}$ module that implements one or more *chemical engines* as part of the CA execution environment, and *(iii)* the `reaction-scheduler` module (LoMA core) that implements the reaction algorithm $\mathcal{A}$ that schedules reactions for execution.

The `manager` module may serve simultaneously (taking advantage of the hardware parallelisation) more than one $\mathcal{AC}$ modules, each hosting a separate CA. It handles the I/O for each $\mathcal{AC}$ module by mapping input and output signals (events such as packet arrivals) to specific species of a CA. It also serves the monitoring of the CA's state by logging concentration values of selected species.

An $\mathcal{AC}$ module represents the principal container of instantiated CAs in hardware. It maps in memory the functional data structures for the structural representation of a CA – *i.e.,* species concentrations, stoichiometric reactant and product coefficients, and reaction coefficients. Values in these structures, which are runtime accessible, provide the inputs to the addressing logic embedded in the $\mathcal{AC}$ module, in order to inter-connect the various parts of the CA and orchestrate its execution. For example, the values of the stoichiometric memories α-`mem` and β-`mem` decode the addresses of reactant and product concentrations of each reaction. Similarly, the

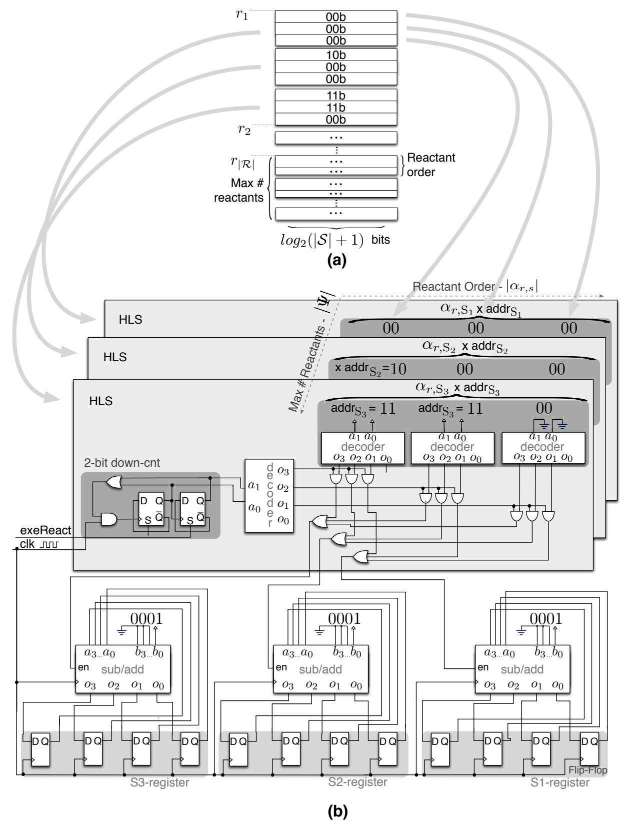


Fig. 3. Addressing logic for updating reactant concentrations (analogous for product concentrations). (a) 3D organisation of the stoichiometric memory of reactants: number of reactions $\rightarrow$ reaction order $\rightarrow$ reactant order. (b) Circuitry schematic related to the stoichiometric memory (example for maximum 3 species, 3 reactants per reaction, up to 3rd order reactants).

reaction coefficients stored in k-mem control the computation of the *next reaction time* according to (3).

The reaction-scheduler module (LoMA-core) computes the propensity of a reaction from its reactant concentrations and from the reaction coefficient, and produces as output the time at which each reaction has to be executed.

B. Reaction Network Instantiation (Level 2) – CA topology

Individual species are implemented as registers each using a chain of flip-flops, whose size determines the maximum value (as a power of 2) that a concentration can assume.

Reactions provide information about which species participate as reactants, which as products, and in what quantities (respective stoichiometric coefficients). The stoichiometric information of reactants and products is divided in two respective 3D array structures that store address information of species (Fig. 3(a)). This information is used by the chemical engine in order to update the right species concentrations whenever a reaction takes place. The size of these arrays reflects the maximum chemical resources available to the user for instantiating CAs. In the following description, we confine our discussion to the operations involving the reactant species only. An analogous description involves product species with the sole difference that logic elements for addition replace those of subtraction (Fig. 3(b)).

The stoichiometric array α-mem of reactants (Fig. 3(a)) is dimensioned by the maximum number of reactions (1D), by the reaction order (2D), and by the reactant order (3D). One filled record implies a 1st-order reactant, and activates once its nesting reactant (indexed at 2nd-level), in a specific reaction (indexed at the 1st-level). Two filled records with the same species address implies a 2nd-order reactant, which activates twice the nesting reactant indexed at the 2nd-level, and so forth.

As illustrated in Fig. 3, the structure of the stoichiometric array reflects the fact that processing for each indexed reactant (2nd-level) is active in separate *Hardware Logic Slices* (HLS). Multiple HLSs are engaged in parallel computations of the CA, such that reactions that involve reactants of 1st-order (*e.g.,* $S_1 + S_2 + S_3 \rightarrow \ldots$) update their species' concentrations in simultaneous steps. At the same time, reactants of 2nd-order or higher engage in processing within an HLS in pipelined fashion. The number of *active* address-records at the 3rd-level (encoding the reactant order) enumerates how many consecutive processing steps are required to complete the full update of the reactant's state, during the execution of the reaction. 3rd-level address-records directly enable a respective number of decoder elements within each reactant's HLS. Upon reaction execution (exeReact signal transient), decoders get active in sequence by a step-down counter. The address stored in each address record of the stoichiometric array is fed to the decoder so as to trigger a subtraction operation on the respective species concentration. As a result of this process, a reaction of the sort $3S_1 \rightarrow \ldots$ is computed in a number of steps that reflects the reactant order $(S_1) + (S_1) + (S_1) \rightarrow \ldots$ (where each parenthesis pair denotes a single processing step).

The *maximum* number of indexable reactions (1D), reaction order (2D) and reactant order (3D) records is part of the chemical resource reservation that takes place at the time of (re-)initialisation of the FPGA with a (partial) bitstream for the chemical engine. For example, the addressing logic of Fig. 3(b) refers to a resource reservation (maximum allocations) for 3 species with concentration size up to 15 molecules, and one indexable reaction with at most 3 reactants/products per reaction, and of up to 3rd order each. For 3 species, 2-bit addresses are needed to resolve access to their registers (S3, S2, S1), each of which is 4-bit wide (number of flip-flops in each register); thus serving concentration size values $\leq$ 15. The corresponding reactant stoichiometric array structure (see Fig. 3(a) for reaction r_1) can index reactions (1st-level), of maximum 3 reactants (2nd-level), each in turn of up to 3rd-order.

For a reaction of the form $2S_3 + S_2 \rightarrow \ldots$ (Fig. 3), the two reactants S_2 and S_3 occupy two 2nd-level records (out of the three available). The one corresponding to S_3, which is a 2nd-order reactant, has two 3rd-level records (out of three available) filled with the species address 11b of the S_3 register. By analogy, for the 1st-order reactant S_2, only one 3rd-level record (out of three available) is filled with the species address 10b (see Fig. 3(a)).

When the reaction is active (exeReact-signal), the reactant species are processed at separate HLSs in parallel, e.g. S_3 will be processed at the first HLS, S_2 at the second, while the third HLS will remain unused since there are only two reactants. Within each HLS, e.g. for each reactant, the 2-bit species address stored in each 3rd-level record of the stoichiometric array is input to one decoder. For reactant S_3, its address `11b` appears in the inputs of two of the three decoders. The output of each decoder is processed in subsequent steps of the step-down counter and activates (`EN`-input) a subtracter that decrements by 1 molecule (in every step) the contents of the respective species register. In fact, this reduces the concentration of S_3 by 2 in two steps, and respectively the concentration of S_2 by 1 in one step. Overall, the discussed hardware logic computes $2S_3 + S_2 \to \dots$, as $(S_3 + S_2) + (S_3) \to \dots$.

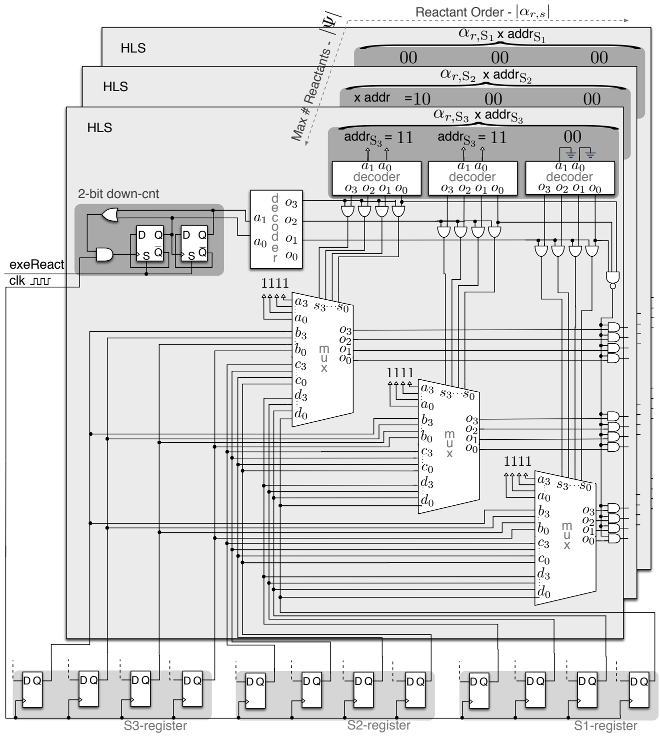
C. *Reaction Scheduling (Level 2) – CA Dynamics*

Reactions execute in real-time according to a time-schedule as described in (3) (LoMA). Computing the reaction-times schedule is the most costly operation, in terms of hardware logic. After a reaction has fired, and the update of species concentrations of reactants and products has completed, a computation is triggered for the *next reaction time slot* of each dependent reaction (*i.e.,* reactions whose reactant concentrations have been modified). For a reaction r, this requires to compute the propensity, *i.e.,* the product of reactants' concentrations $c_s^{\alpha_{r,s}}$ and the reaction coefficient k_r, according to (3). The reaction coefficients $\mathbf{k}$ are stored in a separate bank of registers.

To engage the relevant (reactant) species for computing the propensity of each dependent equation, we use the hardware logic design in Fig. 4. As with the addressing logic for updating the concentrations in the previous section, we rely on information from the reactant stoichiometric array to index across HLSs and decoders. However, in this case, the output of each decoder selects inputs of a multiplexer chain. At every step of the counter, one multiplexer outputs the address of the decoded species register (for $s_3 \dots s_0 =$ `1000` it forwards the value of the $S3$-register, for $s_3 \dots s_0 =$ `0100` the value of the $S2$-register, and for $s_3 \dots s_0 =$ `0010` the value of the $S1$-register), or the fixed value `1111` for the identity element of the multiplication.

Outputs from each HLS contribute to the computation of the power for each reactant concentration $c_s^{\alpha_{r,s}}$ (*e.g.,* $c_{S_3}^2$), while the combination of the outputs across HLSs contribute to the computation of the product of reactants' terms $\prod_{s\in\mathcal{S}} c_s^{\alpha_{r,s}}$ (*e.g.,* $c_{S_3}^2 c_{S_2}$). To complete the computation of the propensity, these values alongside the reaction coefficient k_r, are input to a multiplier module. Depending on the desired trade-off between logic density and computation speed, this operation can be performed by a single multiplier in as many as $|\Psi| \times |\alpha|$ time steps ($|\Psi|$ being the maximum possible number of reactants, *i.e.,* reaction order), and accounting for the additional multiplication by k_r), or by up to $|\Psi|$ parallel scaled-multipliers in as few as $|\alpha|$ time steps.

Generating the new time schedule requires *(i)* computing the reciprocal of the propensity value of the reaction that was just executed, and *(ii)* possibly rescaling the old propensity value of all its dependent reactions, so as to update their time schedules according to new reactant concentrations.



Reactant Order - $|\alpha_{r,s}|$
Max # Reactants - $|\Psi|$
HLS
HLS
HLS
α_{r,S_1} x addr$_{S_1}$
00 00 00
α_{r,S_2} x addr$_{S_2}$
x addr =10 00 00
α_{r,S_3} x addr$_{S_3}$
addr$_{S_3}$ = 11
addr$_{S_3}$ = 11
00
a_1 a_0 decoder o_3 o_2 o_1 o_0
2-bit down-cnt
decoder
exeReact
clk
1111
mux
S3-register
S2-register
S1-register

Fig. 4. Addressing logic for selecting concentrations to compute reaction propensities (example for a single reaction and for maximum 3 species, 3 reactants per reaction, up to 3rd order reactants).

The propensity and reaction-time computations can also be parallelised by means of $|\mathcal{R}|$ separate `reaction-scheduler` modules (*i.e.,* from one per $\mathcal{AC}$ up to one per reaction).

D. *Realisation on Xilinx Spartan-6 FPGA Family*

We originally prototyped the FPGA overlay framework for CAs discussed so far on a relatively small low-cost low-end FPGA device: the Xilinx Spartan-6 XC6SLX9 [30] on the Avnet Spartan-6 LX9 MicroBoard. For the `reaction-scheduler` module, we have used a Xilinx single-precision floating-point IP core [31] (IEEE-754 Standard compliant, $\sim \pm 2^{127}$ dynamic range, $\sim 2^{-23}$ resolution).

For the experiments presented here, we have configured the CA framework with the chemical resource reservation shown in Tables I and II.[2] A single chemical engine can host up to 255 species in 8 reactions of up to 8 reactants/products of maximum 8th-order. For most of the network dynamics functions we have implemented, reactants/products go up to 2nd order, and individual reactions rarely involve more than 3-4 reactants and 1-2 products each. Thus, reservation shown in Table I is sufficient for implementing most of network dynamics control functions.

The c-`mem` of species concentrations is 16-bit wide. It is initialised with $00...0b$, except for the first (reserved) position set to $00...1b$. Concentrations interfacing to input/output

[2]Source VHDL code can be retrieved from the URL http://cn.cs.unibas.ch/projects/HWAC/.

parameter	value	description
$\|\mathcal{R}\|$	8	max number of reactions
$\|\Psi\|$	8	max number of reactants/products
$\|\mathcal{S}\|$	255	max number of species
$\|C\|$	16bit	max concentration value/size ($2^{\|C\|}-1$)
$\|\alpha\|$	8	max reactant stoichiometric coefficient value
$\|\beta\|$	8	max product stoichiometric coefficient value
$\|k\|$	32bit	reaction coeff. size (single-precision floating point)

TABLE I
CHEMICAL MIDDLEWARE PLATFORM RESOURCE RESERVATION PROGRAMMED ON THE XC6SLX9 FPGA FOR OUR EXPERIMENTS.

table	size		
c-mem	$\|\mathcal{S}\|$ pos	x	$\|C\|$ bit
α-mem	$(\|\mathcal{R}\|$ x $\|\Psi\|$ x $\|\alpha\|)$ pos	x	$log_2(\|\mathcal{S}\|)$ bit
β-mem	$(\|\mathcal{R}\|$ x $\|\Psi\|$ x $\|\beta\|)$ pos	x	$log_2(\|\mathcal{S}\|)$ bit
k-mem	$\|\mathcal{R}\|$ pos	x	$\|k\|$ bit

TABLE II
CAPACITIES OF CA MEMORIES

events are updated in batch according to a molecules-per-event ratio. The α-mem and β-mem store the stoichiometric array structures, and the k-mem stores single-precision floating-point values of reaction coefficients. Because of limited space on the XC6SLX9 chip, there is a single reaction scheduler module that makes use of four floating-point modules (DSP48E1s): two multipliers and two divisors.

V. EVALUATION

The following evaluation of the CA framework aims to *(i)* validate the *runtime* re-programmability of CAs using FPGA hardware, *(ii)* quantify the performance benefits/costs from enabling FPGA-based acceleration of CAs, *(iii)* assess their deployability in real network environments, given the capabilities of available FPGA devices on the market, and *(iv)* assert the expressibility of CAs to implement various functions for controlling network dynamics.

A. Experiment Setup

We used CAs to control the service process of the egress queue of a standard Linux host (Linux, Kernel 3.8.6), and thereby control the dynamics of outgoing traffic. To this end, we isolated a traffic class in a separate FIFO queue (`tc` facility). The arrival process of that queue provided the input events for the CA: for each enqueued packet, an amount of molecules corresponding to the number of bytes in the packet was added to an input species S in the chemical engine. The service process of the queue was controlled by the concentration of output species P: for each P-molecule produced, a fixed number of bytes were allowed to leave the queue at packet granularity (given enough molecules to match the byte-size of the packet at the front of the queue, the packet was dequeued and transmitted). In both cases, the molecules-to-bytes ratio was kept fixed at 1 mol/KB.

To interface the chemical engine on the FPGA (LX9 board) with the queue management subsystem of the linux kernel, we use the parallel port together with the `Parapin` kernel module. This allowed us to use the parallel interface signals for custom I/O (*i.e.,* handling interrupts at the port pins, and accessing directly port registers), and wire them to one of

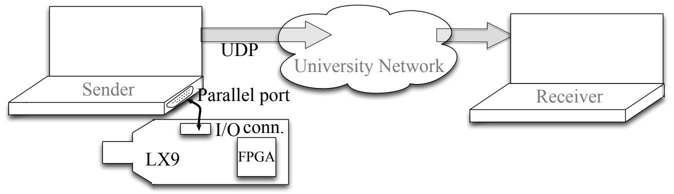


Fig. 5. Experiment setup to rate control PC's egress traffic by means of CAs. The FPGA hosting the chemical engine was connected to the parallel interface of the sender host for facilitating the signalling between the CA and the queue-management subsystem of the linux kernel.

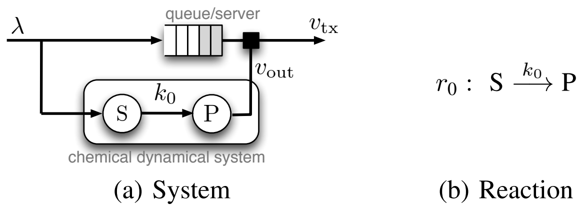


Fig. 6. `Rnet2`: Simple reaction network enforcing the LoMA (3) as a queue service process, so as to implement a traffic pacer.

the LX9's I/O connectors. With this setup, it was possible to produce/process interrupts and exchange events every 100 ns.

The traffic exchanged was host-to-host, over a high-speed switched network (Fig. 5). The results and graphs that follow concern UDP traffic produced with the `iperf` tool (client side running on the controlled node). With TCP traffic, results are inherently more difficult to illustrate because of the coupling between the two control-loops in the protocol state machine.

B. Runtime Programmable Hardware – Proof of Concept

We began by instantiating a simple CA (Fig. 6) that paces packet transmissions by a variable time delay. This rather basic reaction network directly imposes LoMA (3) behaviour as the queue service policy. The CA "program" is essentially the following reaction network specification:

$$\begin{gathered} \mathcal{S} = \{\text{S}, \text{P}\}, \quad \mathcal{R} = \{r_0\} \\ c^0_{\text{S}} = c^0_{\text{P}} = 0, \quad k_0 = 10 \text{ s}^{-1} \end{gathered} \tag{6}$$

In the first $5s$ of the experiment (see Fig. 7), the output rate followed the average load of the queue. At the same time, the burstiness of the arrival process was filtered and smoothed out, by setting the cut-off frequency via k_0-coefficient (=$10s^{-1}$).

After $t = 5s$, we replaced `Rnet2` with `Rnet1` in the $\mathcal{AC}$ (enzymatic rate controller of Fig. 1), by directly loading (while the system ran) the following CA specification.

$$\begin{gathered} \mathcal{S} = \{\text{S}, \text{E}, \text{ES}, \text{P}\}, \quad \mathcal{R} = \{r_1, r_2\} \\ c^0_{\text{E}} = 25 \ Kmol, \ \ c^0_{\text{S}} = c^0_{\text{ES}} = c^0_{\text{P}} = 0, \quad k_1 = 1 \ (\text{mol}{\cdot}\text{s})^{-1}, \ \ k_2 = 20 \text{ s}^{-1} \end{gathered}$$

The values chosen for c^0_{E} and k_2 set the rate cap at 0.5 GBps.

At $t = [6.5s, 14s]$ (Fig. 7), a new burst of UDP transmissions increased the load above the predefined rate cap. The output rate ramped-up to the cap rate, and remained at that limit until the burst ended. A third round of UDP transmission started at time $t = 19s$. The load had a mix of high-frequency

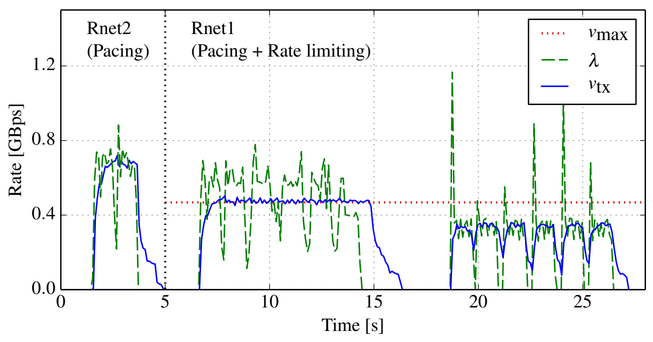


Fig. 7. Traffic shaping effects of 2 programmed CAs: Between t=[0-5s] the chemical engine was programmed with `Rnet2`, then between t=[5-27s] the chemical engine was re-programmed with `Rnet1`. λ is the input rate (load presented by the network layer), $v_{\max}$ is the rate limit set by `Rnet1`, v_{tx} is the output rate (actual transmissions authorised by the CA).

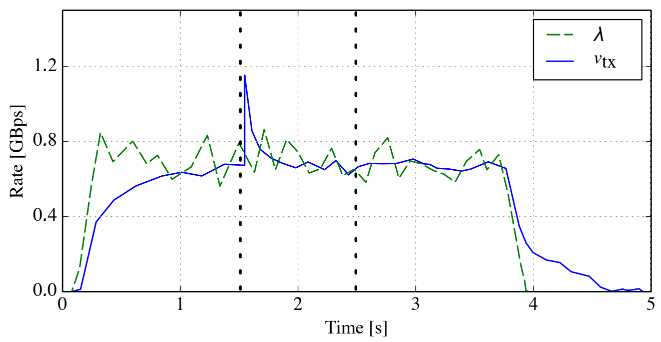


Fig. 8. Programmable dynamics and service consistency: The input traffic pattern λ was the input of a CA configured as `Rnet2`, initially with $k_0 = 10$ s^{-1}, at $t_1 = 1.5$ s with $k_0 = 20$ s^{-1}, and at $t_2 = 2.5$ s with $c_S = 500$ mol, $k_0 = 10$ s^{-1}. v_{tx} is actual transmissions authorised by the CAs. A discontinuity is visible at t_1 but not at t_2. Vertical dashed lines highlight when k_0 was modified.

and low-frequency bursts, but this time it did not exceed the rate cap. In this case, the CA worked as a pacer, *i.e.,* the transmission rate followed closely the slow fluctuations of the arrival rate, but very high-frequencies were filtered out.

We used again `Rnet2` (Fig. 6) to demonstrate fine grained runtime modifications of individual CAs, as well as system state consistency with regard to modifications. Fig. 8 shows the test results. While traffic was arriving at the queue with a constant mean rate, we modified the existing CA by first setting $k_0 = 20$ s^{-1} at $t_1 = 1.5s$. This caused a noticeable discontinuity (step) in the transmission rate. At $t_2 = 2.5s$, a second modification reverted back $k_0 = 10$ s^{-1}, but at the same time we also modified the state of concentration c_S, such that $\lim_{x\to 0^+} c_S(t_2 + x) \cdot k_0(t_2 + x) = \lim_{x\to 0^+} c_S(t_2 - x) \cdot k_0(t_2 - x)$ (so as to preserve the instantaneous reaction rate). In contrast with the first modification, this one did not cause any inconsistent behaviour at the queue. Such carefully accounted modifications draw from the analysability and predictability of CAs, which indeed can help confining potential consistency pitfalls in the queue state. Sound knowledge of the dynamics of the implemented control system (see [4]) helps to understand, foresee and contain transient interruptions of the implemented service due to perturbations, caused either from algorithm changes or other external factors (e.g. other system management tasks).

In summary, so far we validated that new algorithms can be programmed/deployed and modified. We specifically focused on system consistency during service.

C. Quantifying Gains from Hardware Acceleration

When functions for network dynamics control are integrated in the packet processing pipeline, they introduce a computational overhead for the CPU, which adds to the processing delay of the fast-path. Even worse, if because of this overhead CPU reaches 100% utilisation and cannot compensate for the packet arrival rates anymore, delay variance in the processing pipeline can start interfering with the nominal operation of the implemented traffic/queue control function, with potential effects not foreseeable by the algorithm model. This penalty for enabling traffic/queue control functions is avoided as soon as the computation is offloaded to an accelerator (or to a separate CPU core in a multicore system), and as long as the accelerator is not slower than the packet processing pipeline. As we have already stated, one of the advantages of CAs is such an implicit decoupling, as CAs rely only on input/output event hooks to the data path.

To quantify the benefits from hardware acceleration of CAs we measured the computational overhead when executing CAs in CPU, within the linux kernel. This overhead is the "saving" from moving CAs on the FPGA.

Using the ChemFlow software (from the experiments of [4], [5]), we were able to isolate the CA (algorithmic) computations in a separate kernel thread, for which we track the utilisation of the CPU. This measurement excludes other queue management and packet processing tasks. We performed measurements for two CAs of different complexity, `Rnet1` and `Rnet2` (`Rnet1` has double amount of species and reactions than `Rnet2`). For each CA we measured the impact of different packet arrival rates (input events). The results are shown in Fig. 9, grouped by the rate of input events. Beyond a certain rate (1M mol/s) the CPU utilisation increases dramatically, eventually at the expense of other (application) tasks in the system. Doubling the amount of chemical resources does not double the load, but nevertheless increases it significantly ($\sim 10\%$). We note that aside of all benefits, implementing dynamics control functions as CAs is computationally expensive [4], [5], due to the frequency of reaction-scheduling.

In the case of the FPGA implementation, these computations are offloaded to an external hardware and do not steal resources to the CPU. Still implementations of CAs on FPGAs have limits in terms of maximum handleable rates. Clocking our device at 80 MHz and using 1 mol/KB resolution, this limit is approximately at 800 Mbps when two reactions are involved in the CA, and at 1.6 Gbps when there is one reaction involved. However, in an implementation tailored for scalable performance (*i.e.,* with parallel reaction schedulers), the CAs' complexity would not affect the performance.

The reported rates from our low-end device do not seem to fit the needs of today's high-end core Internet routers, still

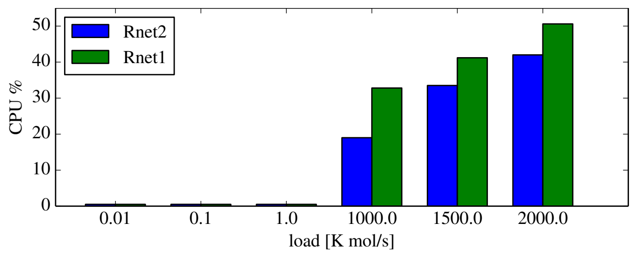


Fig. 9. CPU utilisation when executing directly on the main CPU (as software task in kernel space of the OS) the CAs in `Rnet1` and `Rnet2` with different input loads. Monitored for 20 seconds and then averaged separately for each input load (until 1M mol/s load the host CPU utilisation is near zero).

	$\lvert\mathcal{R}\rvert$ = 2	$\lvert\mathcal{R}\rvert$ = 4	$\lvert\mathcal{R}\rvert$ = 8	$\lvert\mathcal{R}\rvert$=32
# Slice Registers	1'338	1'533	1'922	4'290
# Slice LUTs	3'071	3'464	3'931	7'838
# Occupied Slices	1'145	1'340	1'792	4'398
# DSP48E1s	4	4	4	4

TABLE III
LOGIC RESOURCE REQUIREMENTS ON XC7K325T FPGA FOR A CHEMICAL FRAMEWORK WITH UP TO $\lvert\mathcal{R}\rvert$ REACTIONS AVAILABLE.

they are indicative for current last-mile and edge connectivity. Moreover, newer top-of-the-range FPGAs together with a high-bandwidth host interface (*e.g.,* PCIe Gen3, NVlink, *etc.*) are suitable to serve the needs of high-end core Internet routers (as asserted by the numbers of the next section).

D. Performance as a Function of Logic Resources

In this section, we report on three potentially concerning features of the hardware platform we propose.

1) Logic resources: With the chemical resource provision of Table I, one is able to express (mathematical) models described by a system of up to 8 ODEs with a maximum of 256 8th-order terms, using 16-bit variables. For network dynamics control functions this corresponds to rather big and expressive models. On Xilinx XC6SLX9 FPGA (the 2nd smallest device of the Spartan-6 family), this implementation consumed around 70% of the slice LUTs. On a XC7K325T FPGA (device on NetFPGA-1G-CML board), the exact same framework instantiation barely utilises 1% of the slice LUTs available. Indicatively, in Table III we provide summary reports produced by the Xilinx EDA tool, for the amount of logic resources consumed on the XC7K325T FPGA when we scaled up the maximum accommodated number of reactions $|\mathcal{R}|$ (number of ODEs) under the same design. Note that increasing the maximum amount of chemical resources available for CAs does not change the requirements in terms of logic resources of the LoMA scheduler (LoMA scheduler is one and, to be implemented, needs 722 Slices, 798 Slice Reg, 2334 LUTs, 79 LUT-RAM, and 4 DSP48E1s on the XC7K325T FPGA).

2) Speed performance: There is substantial room for improvement in terms of parallelisation and pipelining (trading logic resources). Specifically, by employing a LoMA scheduler per reaction, each reaction benefits from a completely independent processing path, eliminating the latency from inter-reaction re-scheduling. Furthermore, by using many parallel DSPs for each LoMA scheduler further reduces the number of clock-cycles in the computation of the propensities. We experimented with these optimisations on the larger of the two devices we had available (XC7K325T FPGA). By allocating a separate LoMA core to each reaction in the configuration of column 4 of Table III, the number of clock-cycles for re-scheduling the reactions dropped from $\sim$1600 to 52, while the logic resource budget increased to 30'055 slice registers, 79'016 slice LUTs, and 128 DSPs. By additionally changing the pipeline of the LoMA core, we attained a further reduction to only 24 clk-cycles, while the utilised logic resources increased to 54'154 slice registers, 113'495 slice LUTs, and 320 DSPs. This still amounts to less than 50% of the logic resources available on the XC7K325T FPGA. With such a performance-optimised configuration, the CA controller can theoretically deal with up to 216 Gbps rates, still considering 1 mol/KB as resolution and an FPGA clock of 80 MHz, as we did in Sect.V-A.

3) Resolution in processing events: Our single-scheduler implementation can handle events occurring approximately every 10 μs, 5 μs, and 2.5 μs, for the nominal XC6SLX9 FPGA clock frequencies of 40 MHz, 80 MHz, and 160 MHz respectively. In the XC7K325T device, the device can be clocked at up to 650 MHz, enabling our prototype to process events at 615 ns resolution. In this case, a design with multiple parallel LoMA schedulers achieves roughly 27 ns-resolution.

E. Algorithmic Expressibility of CAs

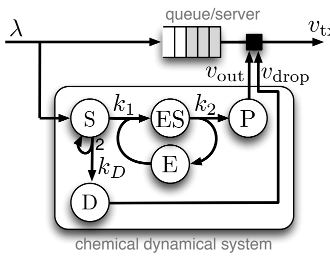
Algorithmic expressibility of CAs has been demonstrated with implementations for congestion control in [4], rate control and distributed resource access in [5]. Here, we extend the range to include an AQM scheme and an implementation for traffic prioritisation/conditioning (and thereby the examples recorded in the literature cover most typical applications of network dynamics control functions).

1) CAs for AQM: An extension of the enzymatic rate controller scheme in Fig. 1 suffices to turn the CA into an AQM scheme with a packet dropping tendency analogous to RED [11]. As shown in Fig. 10, the extension involves one additional reaction (r_3) and one additional species (D), whose concentration regulates the drop process at the queue.[3]

Reaction r_3 (much slower than r_1) occasionally "samples" the queue fill-level (*i.e.,* concentration of species S). If the queue size starts growing (*i.e.,* too-slowly dequeued packets or too high arrival rates), r_3 accelerates fast (as a 2nd-order function of the queue size), creating drop tokens (D) that in turn remove packets from the queue. As the queue size decreases, r_3 quickly recovers its low speed and slows-down packet drops.

Fig. 11 validates experimentally this behaviour in a simple scenario where `iperf`-generated variable bit rate (VBR) UDP traffic goes through a queue controlled by this CA. The upper rate limit of the enzymatic controller was set to 0.4 Gbps, representing the maximum desired link utilisation (condition under which no queue is built-up). The UDP traffic was admitted to the queue initially at 0.2 Gbps and then at 1 Gbps,

[3]Discussing the specific dropping policy (*e.g.,* head- or tail-dropping) is outside the context of this paper's focus. Anyway, both dropping policies can be facilitated in the CA system.



(a) System

$$r_1 : \mathrm{S} + \mathrm{E} \xrightarrow{k_1} \mathrm{ES}$$
$$r_2 : \mathrm{ES} \xrightarrow{k_2} \mathrm{E} + \mathrm{P}$$
$$r_3 : 2\,\mathrm{S} \xrightarrow{k_D} \mathrm{S} + \mathrm{D}$$

(b) Reaction set

Fig. 10. Rnet3: The enzymatic reaction network (Rnet1) can be extended to be used as a AQM scheme. The CA has two outputs: species P controls the departure process, and species D regulates the drop process of packets from the queue. The scheme guarantees a maximum transmission rate of packets while keeping the queue size low.

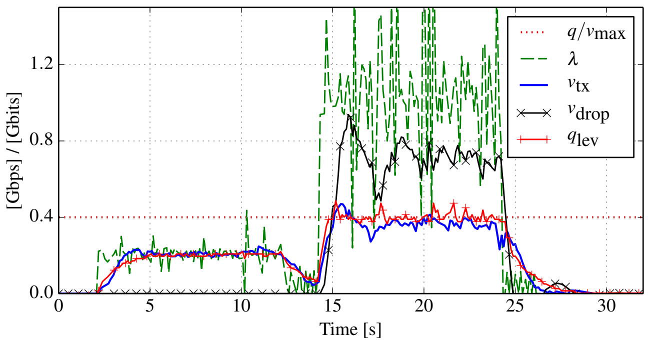


Fig. 11. Experimental result of an AQM-style chemical controller. λ, v_{tx}, v_{drop} are rates of enqueued traffic, transmissions, and drops as shown in Rnet3 in Fig. 10). q/v_{max} is the maximum queue level and the related maximum transmission rate (set via k_2-e_0) and q_{lev} is the level of the queue.

during different phases of the experiment ($\sim$ $[2s, 13s]$ and $\sim[14s, 25s]$). The drop rate (black line) remained effectively zero under low-load conditions (first phase). But as soon as the rate cap was reached and the queue started building up (second phase), the drop-mechanism kicked in and drained the queue at a pace synchronised (no phase lag) with the queue fill-level variations.

Note, that while the CA operates on the queue size (S species), its configuration is in terms of a throughput cap (0.4 Gbps) at the queue (close analogy to “automatic adjustment” in modern AQM schemes [12], [13]).

2) CAs for traffic prioritisation: By combining the distributed rate control scheme presented in [5] with the CA for AQM of the previous section, we created a CA for weighted, or proportional, fair-queuing (Fig. 12). The servers of the participating queues in the scheme (typically corresponding to distinct classes of traffic) were controlled by identical reaction sub-networks, which nevertheless shared tokens in molecular state (aggregate of species P_i feeds back to each T_i). Coefficients $k_{2,i}$ at each sub-network configure the relative proportional bandwidth share of each queue. The outputs of these queues aggregate at a single egress queue, which is then controlled by the last stage of the CA, a sub-network implementing the AQM in Sect.V-E1.

Without delving into analysis details (due to space limitation),[4] we show an experimental validation in Fig. 13. The service processes of three intermediate queues and the egress queue were controlled by the reaction network Rnet4. The top plot in Fig. 13 shows the constant bit rate (CBR) admission of UDP traffic to all three queues with iperf, in two phases: (1) $t < 10s$, and (2) $t \geq 10s$. The second plot in Fig. 13 shows fair-sharing, and the last two plots demonstrate weighted (proportional) fair-sharing, enforced by choosing different $k_{2,i}$ configurations.

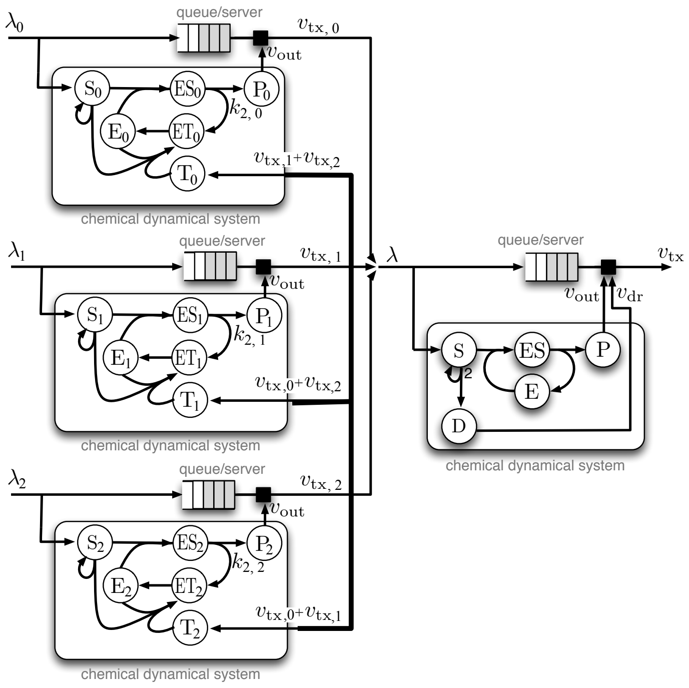

λ_0
λ_1
λ_2
queue/server
$v_{tx,0}$
$v_{tx,1}$
$v_{tx,2}$
v_{out}
$v_{tx,1}$+$v_{tx,2}$
$v_{tx,0}$+$v_{tx,2}$
$v_{tx,0}$+$v_{tx,1}$
$k_{2,0}$
$k_{2,1}$
$k_{2,2}$
S_0 ES_0 P_0 E_0 ET_0 T_0
S_1 ES_1 P_1 E_1 ET_1 T_1
S_2 ES_2 P_2 E_2 ET_2 T_2
chemical dynamical system
λ
v_{tx}
v_{dr}
S ES P E D
2

Fig. 12. Rnet4: The combination of Rnet3 with the distributed rate controller scheme in [5] leads to a CA capable of weighted/proportional fair-queuing. Priorities are configurable via $k_{2,i}$.

In the first phase, the total aggregate admission rate (at the intermediate queues) did not exceed the configured 2 Mbps-limit at the egress queue. All flows claimed and received what they needed from the available bandwidth. In the second phase, the total aggregate admission rate exceeded by far the rate limit and prioritisation kicked in. The share each flow received is (statistically) proportional to the weights expressed as $k_{2,i}$ parameters.

VI. Discussion and Related Work

We review the three main contributions of this work in light of our evaluation results. By comparison to current literature, we highlight the benefits of the CA-based framework.

A. Algorithms on Hardware: Models vs Discrete Programs

Traditionally, the implementation of algorithms on FPGA devices implies a time-consuming tedious process that requires competence in logic design and in hardware description languages such as VHDL [32] and Verilog [33]. It is therefore undertaken only by a small community of designers (much

[4]The analysis is a straightforward application of the theory in [4].

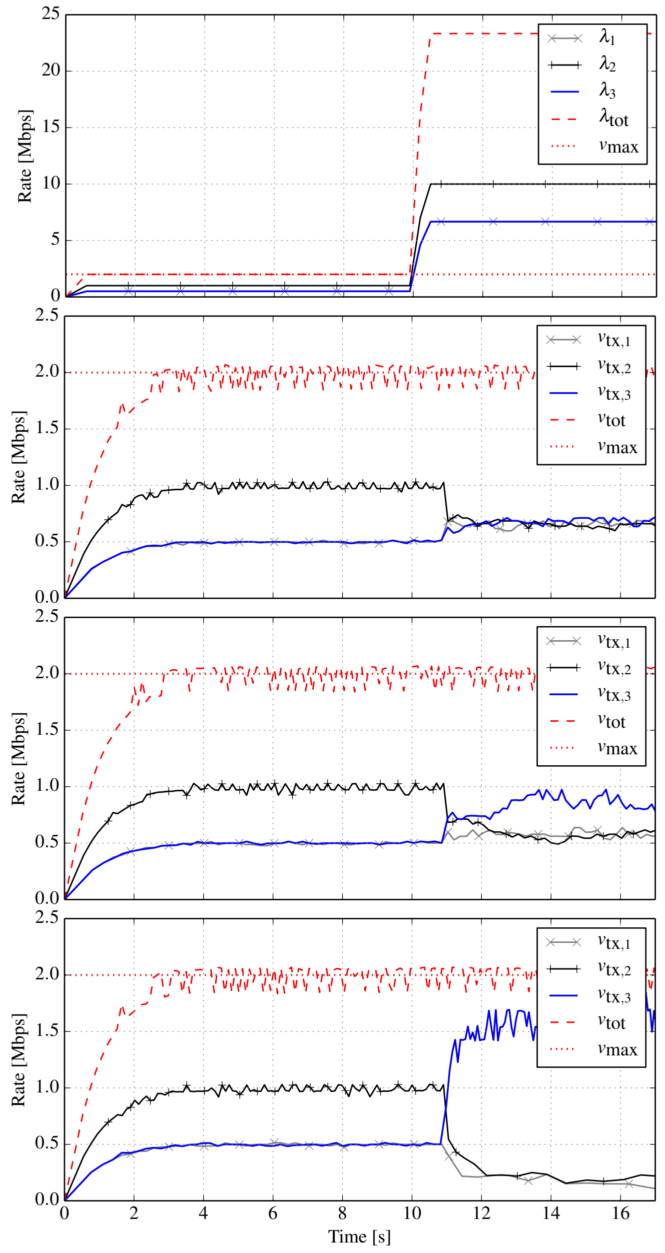


Fig. 13. Prioritisation of traffic classes via $k_{2,i}$-values. λ_i is the load at the i-th queue, λ_{tot} is the total load ($\sum_i \lambda_i$), v_{max} is the configured maximum output rate, $v_{\text{tx},i}$ is the dequeueing rate related to the i-th queue, and v_{tot} is the aggregate output rate ($\sum_i v_{\text{tx},i}$), see CA in Fig. 12. Curves λ_1 and λ_3 overlap and appear as one.

more restricted than the one of software programmers). More "modern" ways of programming FPGAs try to elevate the level of abstraction with high-level frameworks like OpenCL [7], [34], [35] and High Level Synthesis [36], [37] or by using visual primitives for data-flow-based programming such as in [38]. These new approaches bring to task modularity, implementation homogeneity and code verification. However, these approaches rely on static compilation and synthesis every time an algorithm is changed. They are also typically engineered to first translate sequential constructs of *discrete programs* into sequential finite state machines (FSM) in HDL or register-transfer logic (RTL), and eventually generate a configuration (bitstream) for programming the FPGA device.

While this hybrid programming has recently boosted the popularity of FPGAs as an attractive solution for computation acceleration (*e.g.,* alternative to GPUs), it also poses some important limitations. The implementation of a mathematical-model-based algorithm as a discrete program implies essentially a serialisation process (algorithm expressed as a sequence of steps). This inevitably compromises some, or occasionally many, facets of parallelisability available in the original model. Moreover, as this mapping is not deterministic, some versions may suit better than others the automated heuristic process of generating a parallelised RTL designs.

By contrast, the approach presented in this paper directly maps mathematical models to RTL, without going through the intermediate serial transformation into a discrete program. Parallelisability is thus not limited by the translation of a discrete program. Furthermore, there is no heuristic process that may give better results for some implementations than others. Instead, an algorithm instantiation has access to all possible degrees of parallelisation that its expression as a model allows. The algorithm is only limited by resource logistics during deployment (resource economy *vs.* performance), not implementation limitations.

The approach shown in this paper benefits from the expressibility of mathematical (ODEs) models, directly derived from chemical reaction networks. We exploit this simple representation, in terms of operators, encoding of dependencies, and parallel computation paths, in order to create a generic addressing fabric pre-installed on the FPGA (as an overlay execution environment). Thereafter an algorithm instantiation is an activation of the right parts of this addressing fabric. This mapping is deterministic, and fully parallelised. Parallelisability allows local algorithmic interventions on parts of the program during execution time (by updating register or memory data), without compromising system integrity (of course, algorithmic integrity can be compromised).

B. *Re-Programmable data-plane on FPGAs*

There is a mismatch between typical deployment timescales in SDN (*e.g.,* flow table updates) and deployment latencies of algorithms on FPGAs. Compiling a bitstream can easily take hours. Downloading it on the device takes several minutes, and re-programming may require reseting (often re-powering). There are two approaches in practical use today trying to overcome these constraints.

1) Partial Reconfiguration (PR): PR technology [27], [28], initially available on Xilinx devices, has the philosophy of software plugins. A bitstream (FPGA program) is divided in two parts: the "skeleton" with hooks and APIs at the RTL level, and a set of *partial bitstreams* (PBs) that can be interchanged on the skeleton hooks according to these APIs. The whole process is faster than downloading the entire bitstream and does not require to reset the device. Unlike software plugins, the spatial dimensions (logic resources) and the connectivity of the PBs need to be confined in advance. This may limit the functionality PBs may implement, and challenges their compactness. More importantly, the deployment timescales do not match those of typical SDN updates.

2) Overlay-based approach: These approaches avoid hardware re-programming altogether and make use instead of sophisticated IP cores called *overlays*, which implement a more generic *execution environment*. Such IP cores may provide tailored functionality for some class of functions [2] or a certain task [25], or may involve general purpose processing elements, such as *soft* processors [39] or DSPs [26], for executing code written in a high-level programming environment. Unlike PR technology, this approach is very attractive for fast (runtime scale) deployment of accelerator functions, and does not require the tedious process of synthesising and generating (partial) bitstreams. Overlay cores typically come with domain-specific language (DSL) or ELF compilers/interpreters, which allow programming and prototyping using high-level language frameworks. Overlay IP cores represent a fast alternative to conventional re-programmability on FPGAs with bitstreams.

In this paper, we present in fact a two-level approach. The chemical framework with the chemical engines is an overlay IP core, on which CAs instantiate accelerated model functions. An instantiation of our framework *per-se* with a certain chemical resource reservation can be implemented and programmed as a partial bitstream at much coarser deployment timescales. Multiple of these bitstreams may co-exist or replace one another. Finally, in contrast with typical use of overlay IP cores to date, programming CAs on the chemical engines does not require static compilation and allows partial modifications of a running algorithm (interactively).

C. Software Defined Network Dynamics

As SDN technologies have started maturing, configurability and programmability of data-plane functions has also started to attract engineering attention, with a number of projects focusing on programmable network dynamics [2], [3], [40]-[44] (not all of the cited works are within the SDN community but they share the same objective).

The NetFPGA project [1] has been one of the first large scale initiatives to integrate FPGA technology directly on the networking fabric in order to accelerate network functions, as a joint effort by academia and large FPGA manufacturers. While the focus has been infrastructure (board) development, a few parallel activities and independent projects, benefiting from the availability of the NetFPGA technology, have showcased individual dynamics control functions such as packet pacing [45], rate limiting [46], [47], [48] for isolated packet flows, as well as ECN-based [49] congestion control [50], [51], [14]. Most of these examples of dynamics control functions have been either implemented as discrete programs running on soft-CPU IP cores, or translations of discrete programs as FSMs in HDL.

Chimpp [2] raises the level of abstraction while programming network functions on a FPGA-based data-plane. Chimpp relies on the design-by-module-composition approach of the Click framework [52]. It uses a description of device capabilities and objectives written in a DSL, which upon compilation synthesises a complete RTL design and generates a bitstream from pre-developed HDL modules. The deployment setting is a hybrid environment combining host CPUs and NetFPGA devices configured at boot time.

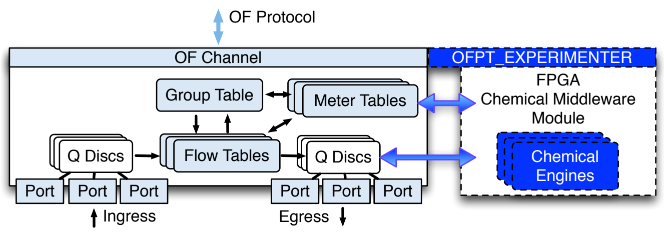


Fig. 14. Integration of CAs in the OpenFlow architecture

SDNet [3] is probably the most promising framework for delivering a hardware runtime-programmable data plane. Similar to Chimpp, it relies on a DSL for describing data-plane features and capabilities, which upon compilation however delivers a partial bitstream. PR is therefore used in SDNet to enable reconfiguration and replacement of functions at runtime.

Finally, the ServerSwitch hardware programmable platform [41] enables hardware programmability with an approach similar to that of Active Networks [53]. It relies on ASIC technology to perform basic tasks and on kernel-level processing on a host CPU to perform the more complex tasks.

Focussing on network dynamics control functions in SDN, authors in [40] effectively pinpoint a minimal set of primitives required to support *southbound* extensions for controlling the fast-path scheduling and queueing behaviour of an OpenFlow switch. In fact, the integration of our chemical engines in the SDN environment is subject to the availability of such an interface in order to allow monitoring of the internal state of CAs (northbound), and instantiation/modification of CAs on the FPGA fabric (southbound). We are currently working on a similar but less extensive set of primitives using the “experimenter extensions” of OpenFlow. The block diagram in Fig. 14 illustrates how this integration is effected conceptually, with chemical engines operating in parallel with current OpenFlow infrastructure.

As no I/O is involved in explicit managing or buffering packet state on the side of the chemical engines, there is no requirement to operate CAs in packet boundaries. Time synchronisation with events and ordering is also not critical because CAs operate statistically; (“macroscopically”) at the signal level, a few event misses are averaged out and do not impact the correctness of the system.

VII. Conclusion

We have introduced, implemented and evaluated a framework that enables runtime (re-)programmable algorithms for the control of network dynamics on FPGA hardware. The representation of these algorithms in programs is by means of *chemical reaction networks*. This provides an expressible and easy-to-use metaphor for designing and implementing functions to control arrival and departure processes at network queues. This simple and high-level representation

- has allowed the expression of accurate mathematical models directly on hardware without the need for low-level HDL programming,

- leads to modular parallel implementations where parts of an algorithm can be modified separately and independently of the rest of the program,
- has enabled fast instantiation and modification of functions on FPGA hardware at runtime, and without the need to re-synthesise RTL code or re-generate bitstreams.

Moreover, CAs can be studied with accurate mathematical models and designed using a robust analysis methodology. This helps to foresee CAs' effects and the artefacts of their customisation/tuning. Thanks to the simple *visual* primitives describing a CA and its parameters, the framework presented in this paper represents an expressive programmatic northbound interface for SDN-based management. Moreover, integrating CAs with existing SDN infrastructure is straightforward and requires hooks that are already available in most suites like OpenFlow.

References

[1] "The NetFPGA project," http://netfpga.org.

[2] E. Rubow, R. McGeer, J. Mogul, and A. Vahdat, "Chimpp: A Click-based programming and simulation environment for reconfigurable networking hardware," in *Architectures for Networking and Communications Systems (ANCS), ACM/IEEE Symp. on*, La Jolla (CA), USA, Oct 2010, pp. 1–10.

[3] Xilinx Inc., "Software defined specification environment for networking (SDNet)," White Paper, 2014.

[4] M. Monti, T. Meyer, C. F. Tschudin, and M. Luise, "Stability and sensitivity analysis of traffic-shaping algorithms inspired by chemical engineering," in *IEEE Journal on Selected Areas of Communications (JSAC)*, vol. 31, no. 6, Jun 2013, pp. 1–11.

[5] M. Monti, M. Sifalakis, T. Meyer, C. F. Tschudin, and M. Luise, "A chemical-inspired approach to design distributed rate controllers for packet networks," in *Proc. of the IFIP/IEEE-IM Workshop on Distributed Autonomous Network Management Systems (DANMS)*, Ghent, Belgium, May 2013.

[6] A. Barkalov, L. Titarenko, and J. Bieganowski, "Synthesis of compositional microprogram control unit with extended microinstruction format," in *Mixed Design of Integrated Circuits Systems (MIXDES), Int. Conf. on*, Lodz, Poland, Jun 2009, pp. 328–331.

[7] P. Jääskeläinen, C. de la Lama, P. Huerta, and J. Takala, "OpenCL-based design methodology for application-specific processors," in *Proc. of the IEEE Int. Conf. Embedded Computer Systems (SAMOS)*, Samos, Greece, Jul 2010, pp. 223–230.

[8] N. McKeown, G. Parulkar, T. Anderson, L. Peterson, H. Balakrishnan, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," ONF White Paper, Mar 2008.

[9] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of Open vSwitch," in *Proc. of USENIX Symp. on Networked Systems Design and Implementation*, Oakland (CA), USA, May 2015, pp. 117–130.

[10] "Netfilter, firewalling, NAT, and packet mangling for Linux," http://www.netfilter.org/.

[11] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. on Networking*, vol. 1, no. 4, pp. 397–413, Aug 1993.

[12] K. Nichols and V. Jacobson, "Controlling queue delay," in *Magazine Communications of the ACM*, vol. 55, no. 7, May 2012, pp. 42–50.

[13] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," Draft Standard 00 draft-pan-aqm-pie, Internet Engineering Task Force (IETF), Dec 2012.

[14] M. Kuhlewind, D. Wagner, J. M. R. Espinosa, and B. Briscoe, "Immediate ECN," IETF-88 TSVAREA, Nov 2013.

[15] M. Monti, L. Sanguinetti, C. F. Tschudin, and M. Luise, "A chemistry-inspired framework for achieving consensus in wireless sensor networks," in *IEEE Sensors Journal*, vol. 14, no. 2, Feb 2014, pp. 371–382.

[16] T. Meyer, "On chemical and self-healing networking protocols," Ph.D. Dissertation, University of Basel, Switzerland, 2011.

[17] P. Dittrich, "The bio-chemical information processing metaphor as a programming paradigm for organic computing," in *Proc. of the Workshop Self-Organization and Emergence, Conf. on Architecture of Computing Systems (ARCS)*, Innsbruck, Austria, Mar 2005, pp. 95–100.

[18] J. Banâtre, P. Fradet, and Y. Radenac, "Principles of chemical programming," in *Electronic Notes in Theoretical Computer Science, Elsevir*, vol. 124, no. 1, Mar 2005, pp. 133–147.

[19] G. Paŭn, "Computing with membranes," in *Journal of Computer and System Sciences*, vol. 61, no. 1, 2000, pp. 108–143.

[20] W. Banzhaf, P. Dittrich, and H. Rauhe, "Emergent computation by catalytic reactions," in *Nanotechnology 7 (1996) 307–314*, vol. 7, no. 4, Dec 1996, pp. 307–314.

[21] J. Giavitto and O. Michel, "MGS: a rule-based programming language for complex objects and collections," in *Electronic Notes in Theoretical Computer Science, Elsevir*, vol. 59, no. 4, Nov 2001, pp. 286–304.

[22] N. Matsumaru, P. Kreyssig, and P. Dittrich, "Organisation-oriented chemical programming," in *Organic Computing – A Paradigm Shift for Complex Systems Autonomic Systems*, vol. 1, 2011, pp. 207–220.

[23] F. Horn and R. Jackson, "General mass action kinetics," *Archive for Rational Mechanics and Analysis*, vol. 47, no. 2, pp. 81–116, 1972.

[24] M. Monti and M. Sifalakis, "Extending the artificial chemistry to design networking algorithms with controllable dynamics," Technical Report CS-2012-003, Univ. of Basel, Switzerland, http://cn.cs.unibas.ch/pub/doc/cs-2012-003.pdf, Jul 2012.

[25] R. Polig, K. Atasu, H. Giefers, and L. Chiticariu, "Compiling text analytics queries to FPGAs," in *Proc. of Int. Conf. on Field Programmable Logic and Applications (FPL)*, Munich, German, Sep 2014, pp. 1–6.

[26] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Throughput oriented FPGA overlays using DSP blocks," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, Mar 14-18 2016, pp. 1628–1633.

[27] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Exploiting partial runtime reconfiguration for high-performance reconfigurable computing," *ACM Trans. Reconfigurable Techn. Syst.*, vol. 1, no. 4, pp. 1–23, Jan. 2009.

[28] L. Wirbel, "SDAccel - a unified development environment for tomorrow's data center software defined spec," Xilinx Inc., White Paper, 2014.

[29] P. Dittrich, J. Ziegler, and W. Banzhaf, "Artificial chemistries – a review," in *Artificial Life*, vol. 7, 2001, pp. 225–275.

[30] Xilinx Inc., "Spartan-6 family overview," Product Specifications DS160 (v2.0), Oct 2011.

[31] ——, "LogiCORE IP floating-point operator v6.0," Application Note DS816, Jan 2012.

[32] IEEE Computer Society, "IEEE standard VHDL language reference manual," IEEE Standard 1076-2008, Jan 2009.

[33] ——, "IEEE standard for Verilog hardware description language," IEEE Standard 1364-2005, Apr 2006.

[34] J. Auerbach, D. F. Bacon, I. Burcea, P. Cheng, S. J. Fink, R. Rabbah, and S. Shukla, "A compiler and runtime for heterogeneous computing," in *Proc. of ACM/EDAC/IEEE Design Automation Conf. (DAC)*, Jun 2012, pp. 271–276.

[35] A. Bourd, "The OpenCL Specification Version: 2.2 Document Revision: 06," Khronos OpenCL Working Group, April 2016.

[36] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 18–25, July 2009.

[37] D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons, and F. Morgan, "Xilinx Vivado high level synthesis: Case studies," in *Proc. of Irish Signals Systems Conf. and China-Ireland Int. Conf. on Information and Communications Technologies (ISSC/CIICT)*, Limerick, Irland, Jun 2014, pp. 352–356.

[38] J. Travis and J. Kring, *LabVIEW for Everyone: Graphical Programming Made Easy and Fun, 3rd Edition*. Prentice Hall, 2006.

[39] V. Kale, "MicroBlaze embedded processor," White Paper, Xilinx Inc., September 2015.

[40] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, "No silver bullet: Extending SDN to the data plane," in *Proc. of the ACM Workshop on Hot Topics in Networks (HotNets)*, New York (NY) USA, Nov 2013, pp. 1–7.

[41] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "ServerSwitch: A programmable and high performance platform for data center networks," in *Proc. of the USENIX Conf. on Networked Systems Design and Implementation*, Berkeley (CA), USA, Jun 2011, pp. 15–28.

[42] C.-L. Hsieh and N. Weng, "Virtual network functions instantiation on SDN switches for policy-aware traffic steering," in *Proc. of the ACM Symp. on Architectures for Networking and Communications Systems (ANCS)*, Santa Clara (CA), USA., Mar 2016, pp. 119–120.

[43] S. Meier, "Software defined active queue management," October 2014.

[44] S. Gu, J. Kim, Y. Kim, C. Moon, and I. Yeom, “Controlled queue management in software-defined networks,” in *Proc. of IT Convergence and Security (ICITCS)*, Kuala Lumpur, Malaysia, Aug 24-27 2015, pp. 1–3.

[45] A. Dwaraki, “Hardware implementation of queue length based pacing on NetFPGA,” M.Sc. Thesis, University of Massachusetts, Amherst (MA),US, Feb 2014.

[46] N. Malangadan and G. Raina, “Rate based feedback: some experimental evaluation with NetFPGA,” in *Proc. of the IEEE Int. Conf. on Communication (ICC)*, Kyoto, Japan, Jun 2011, pp. 1–6.

[47] M. Anwer, M. Motiwala, M. Tariq, and N.Feamster, “SwitchBlade: A platform for rapid deployment of network protocols on programmable hardware,” in *Proc. of the ACM SIGCOMM*, Jun 2010, pp. 1212 – 1221.

[48] A. Lombardo, D. Reforgiato, V. Riccobene, and G. Schembra, “Netfpga hardware modules for input, output and ewma bit-rate computation,” *Int. Journal of Future Generation Communication and Networking*, vol. 5, no. 2, pp. 116–123, june 2012.

[49] K. Ramakrishnan, S. Floyd, and D. Black, “The addition of explicit congestion notification (ECN) to IP,” RFC 3168, IETF, September 2001.

[50] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center TCP (DCTCP),” in *ACM SIGCOMM Computer Comm. Review*, vol. 40, no. 4, Oct 2010, pp. 63–74.

[51] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, “AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers,” in *Proc. of the IEEE Hot Interconnects*, Mountain View (CA), USA, Aug 2010, pp. 58–65.

[52] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The click modular router,” *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[53] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: An intellectual history of programmable networks,” *ACM SIGCOMM Computer Comm. Review*, vol. 44, no. 2, pp. 87–98, Apr 2014.

Massimo Monti obtained the M.E. degree in Telecommunication Engineering in 2010 from the University of Pisa (IT). In 2014, he got the PhD degree with highest honours from the University of Basel, Mathematics and Computer Science, CH, and the Ph.D. degree in Information Engineering from Doctoral School ”Leonardo da Vinci” of the University of Pisa (IT). He did post-doctoral research at the University of Basel (CH) on applying chemistry-inspired algorithms for traffic shaping and controlling network dynamics. He is currently working in Elettronica Monti (IT) as a consultant in developing measurement and monitoring instruments for electromagnetic interferences (EMI).

Manolis Sifalakis holds a bachelor degree in Computing Systems Engineering from the Piraeus University of Applied Sciences (GR), a MSc degree in Computer Science from the University of Edinburgh (UK), and a PhD Degree on Active and Programmable Networks from Lancaster University (UK). After post-doctoral research at Lancaster University (UK) and the University of Basel (CH), he joined IBM Research (CH) where he currently works on algorithm acceleration based on FPGA, GPU and multicore technologies. His past work focused on programmable network technologies, network dynamics, network measurements, and content-centric networking.

Christian F. Tschudin is a Full Professor at the University of Basel and lead the Computer Networks Group. Before joining the University of Basel, he was at Uppsala University as well as ICSI in Berkeley, and did his Ph.D. at the University of Geneva. He is interested in mobile code, artificial chemistries, wireless networks and security.

Marco Luise is a Full Professor of Telecommunications at the University of Pisa, Italy. He’s authored more than 250 international publications and led a number of international research programs, including the recent European Network of Excellence in Wireless Communications NEWCOM#. He was the co-general-chair of IEEE’s ICASSP, Florence 2014. Formerly an Associate Editor of IEEE Trans. Commun., he is the co-founder of the Intl. J. of Navigation and Observation, and a Division Editor of the J. of Commun. and Networks. His main research interests lie in the broad area of signal processing for communications and of wireless communications and positioning.