Controlling service-chains with end-to-end deadline constraints

Victor Millnert, Enrico Bini, Johan Eker

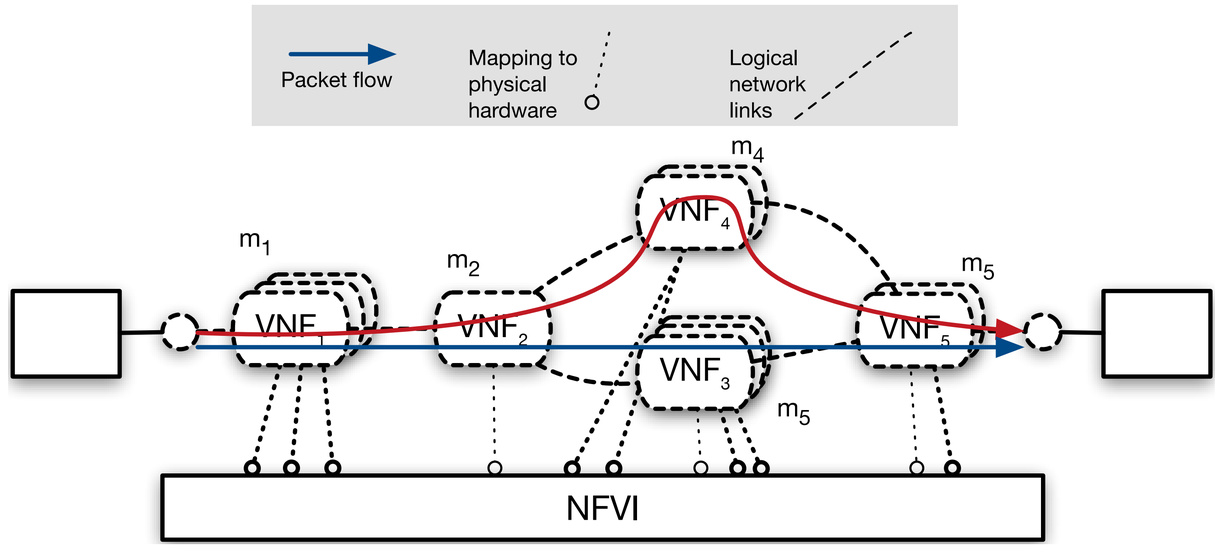
Lund University

WASP Wallenberg Autonomous Systems and Software Program

Introduction — telecom meets cloud

Over the last years, cloud computing has swiftly transformed the IT infrastructure landscape, leading to large cost-savings for deployment of a wide range of IT applications. Initially the cloud technology was mostly used for IT applications, e.g. web servers, databases, etc., but has now found its way into new domains. One such domain is packages processed by a chain of network functions such as firewalls, routers, etc.

Cloud technology make it possible to build network functions with virtual resources, such as virtual machines, forming virtual network functions (VNFs). These virtual resources are running on some infrastructure (NFVI). VNFs are connected forming a forwarding graph (e.g., the blue and red arrow in the figure). The blue FG consist of $\{\text{VNF}_1, \text{VNF}_2, \text{VNF}_3, \text{VNF}_5\}$.

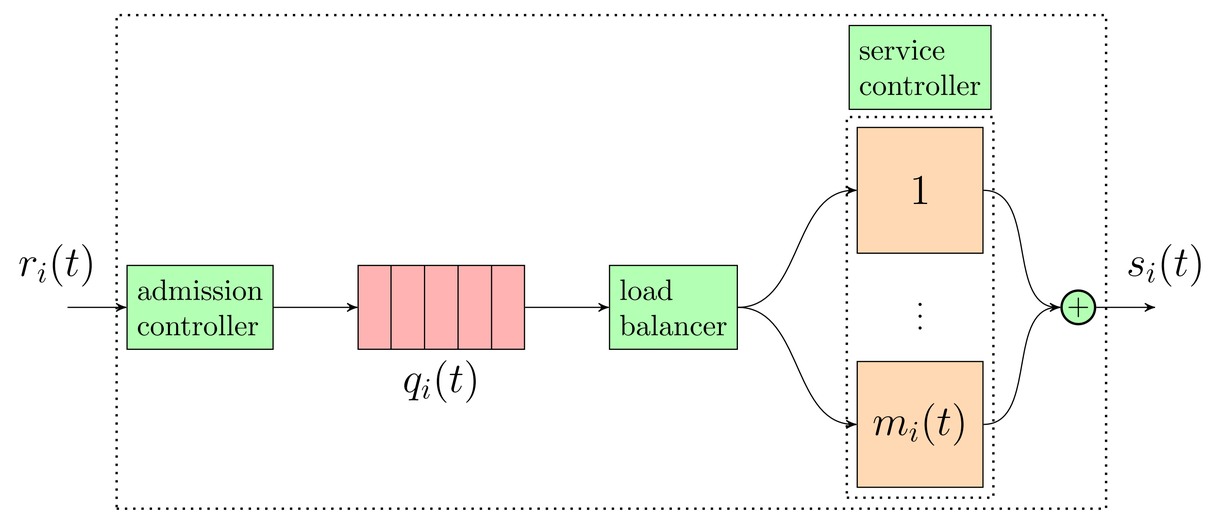


Goal

- Control the amount of resources allocated to the network functions such that the packets flowing through the service-chain meet their end-to-end dedaline.

Why is this hard?

- The packets must be processed by all functions in milliseconds.
- Starting new virtual machines take minutes.
- When you deploy a virtual machine you do not always get what you expect.
- There are large variations in the demand.



Expanded view of a function

How should we do it?

- **Use feedback from the queue.** Ensures that the admitted packets will meet their deadline.
- **Use feedforward between functions.** Gives the system faster reaction times to changes in the input load.
- **Use feedback from true performance.** Makes it robust against the machine uncertainties

Measuring the performance

Combine *availability* and *efficiency* to get the *utility*:

$$a_i(t) = \frac{\text{service}}{\text{demand}}, \quad e_i(t) = \frac{\text{service}}{\text{capacity}}$$

$$U(t) = \sum_{i=1}^{n} \frac{1}{t} \int_0^t a_i(x) \times e_i(x) \mathrm{d}x$$

Admission Controller

Compare the local deadline $D_i(t)$ with the worst-case delay $d_i^{\text{ub}}(t)$. Accept the incoming packet when $\alpha_i(t) = 1$.

$$\alpha_i(t) = \begin{cases} 1 & \text{if } D_i(t) \geq d_i^{\text{ub}}(t) \\ 0 & \text{if } D_i(t) < d_i^{\text{ub}}(t) \end{cases}$$

Service Controller

The control law for the number of machines in a network functions is given by:

$$m_i^{\text{ref}}(t) \approx \frac{\hat{r}_i(t)}{\bar{s}_i + \hat{\xi}_i(t)}$$

where $\hat{\xi}_i(t)$ is the *machine uncertainty* and $\hat{r}_i(t)$ is the *predicted input*:

$$\hat{r}_1(t) = r_1(t) + \Delta_1 \frac{\mathrm{d}r_1(t)}{\mathrm{d}t}$$

$$\hat{r}_i(t) \approx m_{i-1}^{\text{ref}}(t) \times (\bar{s}_{i-1} + \hat{\xi}_{i-1}(t)), \quad i \geq 2$$

Evaluation and results

Running a Monte Carlo simulation with a real-world trace we show that our method (AutoSAC) performs 30–40% better than state of the art in industry; dynamic autoscaling (DAS) and dynamic over provisioning (DOP). The main reason is the admission controller used in AutoSAC. When augmenting DAS and DOP with this admission controller, their performance is increased by more than 20%. However, AutoSAC still outperforms the augmented methods by 5–10% since it uses feedforward, making it faster to react to input changes, as well as feedback making it more robust to machine uncertainties.

Method	Mean$(U(t))$
AutoSAC	0.99
DAS	0.67
DOP	0.75
DAS with AC	0.94
DOP with AC	0.91