

Micro-services internal load balancing for Ultra Reliable Low Latency 5G Online charging system

Ngoc Tien Nguyen*, Thanh Son Pham*, Van Duong Nguyen*, Cong Dan Pham*, Duc Hai Nguyen*

*OCS Research Center, Viettel High Technology, Viettel Group, Hanoi, Vietnam

(tiennn18, sonpt26, duongnv21, danpc, haind13)@viettel.com.vn

Abstract—Connection mesh plays an important role in micro-service architecture and greatly influences system performance. 3GPP proposed a 1ms requirement for 1-way transmit activities on the 5G URLLC data plane. As a 5G component, our Online Charging System needs to manage sub-milliseconds operations. In this paper, we model the request flows in the mesh as a max flow problem with back-pressure. Inspired by the connection lib approach, we propose our connection library Microchassis. We select basic routing (front-pressure) methods including round-robin (RR), least concurrent (LCC), and proposed two more routing methods using *back-pressure flow control policy based on concurrent status* (BCC) and *back-pressure flow control policy based on throughput combined with concurrent status* (BTC) for the library. Then we evaluate the performance and latency among four methods of Microchassis and validate our modeled formulas. Finally, we simulate the OCS architecture of the main charging flow and qualify it against the URLLC latency standard. The result shows that back-pressure strategies can maintain great QoS under extreme conditions, superior to the front-pressure algorithm family. BTC performed well in all cases with the highest throughput in the incident. Besides, BCC only shows its advantages on high throughput systems. Front-pressure strategies are better choices for internal well-managed service layers, and LCC is outperformed on RR.

Keywords—Service mesh, Connection library, Internal load balancing, Benchmark, 5G, URLLC, Online-charging system

I. INTRODUCTION

Connection lib shows its important role in a software system when millions of customers from billions of devices demand high-quality network services. Especially in the telecom sector, the explosion of new services and end-user devices in the 5G era is required as eMBB, mMTC, and URLLC standards [1], widely known in Telecom, Medical, Industry, Banking, and Automotive industries [2]. Our OCS decides the availability of service to end users. This calls for the need for robust message forwarding in micro-service networks. Internal client routing algorithms need to be fast and efficient, so this paper evaluates different strategies.

In particular, we discuss the connection lib approach in micro-service mesh architecture with four routing Internal Load-balancing methods relevant to OCS, two popular and our two proposed: simple Round robin (RR), metric-based Least concurrent (LCC), and Back-pressure flow control methods, based on concurrent state (BCC), and based on flow speed and concurrent state (BTC). First, we introduce the mesh problem

in section I and the connection lib approach in section II. Then we model the max flow and back-pressure problem in section III. In section IV, we demonstrate our adoption of the Microchassis in OCS. In the same section, we propose our back-pressure flow control policies BCC and BTC. Then we analyse the performance of each method in section V with further discussion in section VI.

The main focus of the paper is the performance and latency of the four methods in various scenarios on the same input. Our three contributions are: the request flow as a max flow problem with back-pressure; a guide on implementing back-pressure strategies BCC and BTC, as well as benchmark results to compare strategies.

A. Background

1) *Overhead in service mesh*: Service mesh with an independent sidecar is favored for its flexible capability and cost-effectiveness in a variety of purposes with the ability to be transparent under network layers. Sidecar greatly influences the response time, hence the overall system's performance which was studied in previous research. Istio in their guide [3] stated factors of latency in the data plane and showed a positive estimate. Istio measured each network I/O event gain 1ms more. Envoy [4] is the most popular proxy sidecar, especially in Istio. While [5] assesses the overhead impact of using Envoy, Zhu et al. considered the mesh overhead as an inevitable aspect and predicted it by message size, network speed, business process, and zero-copy factor. Zero-copy shows a minor impact on latency but a big trade-off for the effort of managing the buffer programmatically.

2) *Routing strategy*: A survey [6] mentions some well-known LB methods in switching. Envoy [4], F5 load balancer [7], IPVS [8] and Nginx all support popular methods like Round-robin, Least connection, Destination or Source hashing, Shortest delay, and Maglev. NGINX uses P2R combined with the least connection or least time strategy [9] to reduce the complexity and probability of errors. However, HAProxy refuted the idea that P2R is universally efficient, contrary to theoretical reasoning [10]. This reinforced our motivation to compare complex back-pressure algorithms against simple algorithms such as least-X, and RR.

3) **Back-pressure:** Originally a concept in dynamic fluid theory, back-pressure is used in computing to describe the resistance or force opposing the flow of data through services. It exists in any resource utilization process. The need to handle back-pressure in the max flow problem was mentioned in [11]. Instead of optimizing for QoS by each specific root cause, the back-pressure approach senses the high pressure and then adjusts the flow based on output metrics. The back-pressure approach deals with the case where the input rate exceeds the system's threshold - detailed below as the saturation point. This approach aims to maintain the highest throughput of the best quality. Theoretically, back-pressure algorithms rely on at least one of the following strategies:

Buffering: handles temporary spikes and flattens the distribution of incoming requests, but is ineffectual when the incoming requests consistently overwhelm the system. Little's Law dictates that the required queue length would reach infinity.

Drop: dropping is the last resort in any routing strategy. Dropping needs to respect the sequence of message events, specified by the business flow. As proved in III.5.3, overflowing messages must be dropped. Discarding messages and responding promptly is conducive to better responsiveness perceived by the end users, compared to letting messages wait in queues.

Control speed: Flow control mitigates request waves by cutting off requests to attain a sustainable flow. The producer does this based on the consumer's status, which can be implemented in two ways.

- 1) Push: the producer listens to the connection throughput or the consumer's capacity information to decide. The push model is used in RxJS, RxJava, RSocket, etc
- 2) Pull: consumer signals that it can handle more requests. Pull model is used in Apache Flink v.15+ cho TCP-based và Credit-based back-pressure [12].

Circuit breaker (CB) is considered semi-auto flow control with the ability to terminate the flow and then raise the speed slowly, according to an FSM configured by the developer. CB is most suitable for front-pressure routing methods. However, it can perform flow control only after incidents have happened, resulting in a latency graph with cycling peaks that resemble shark teeth, as illustrated in [11].

II. CONNECTION LIB APPROACH

Detached sidecar cannot obtain business KPIs to make routing decisions. This lack of capability may prove to be detrimental in a distressed system. Integrated Internal client-side Load-Balancing library (ILB) controls the flows on a per-hop per-flow basis, solves the latency problem by bypassing external proxies and sidecars, maintains quick, continuous, self-managed communications in complex misinformation cases, adapts to the local context without compounding the network delay. The ILB can run independently or interact with the centralized control plane. With the ILB lib, capacity planning and auto-scaling rules are temporary solutions that rely on a rate limiter. Those solutions only reduce the probability of a

cascading failure without full-time protection. The priority of a per-flow rate limiter is to maximize the throughput under a threshold. Furthermore, one of the minor problems of ILB is that it worsens the computing overhead for flow control.

Uber Hyperbahn [13], Java Maven library Netflix Ribbon [14], and Meta ServiceRouter Lib (SRLib) [15] also adopt this architecture.

A. Drawbacks of connection lib

The ILB connection lib handles all messages indiscriminately, without paying attention to the message semantics including:

- 1) Time dimensions: (a) ILB needs to handle any range of messages time (e.g. minutes, days, etc.) (b) Timeout is defined by the business process. Many related research leveraged time dimensions: (1) as metrics of service mesh routing [11], [16], (2) to minimize the latency [17], and to predict the latency [5].
- 2) Size
- 3) API Destination or path

In OCS, we usually trade system utilization (around 60%) for consistently low latency in sensitive and high-performance businesses (also shown in [2]). Since the need for an adaptive ILB to serve over 100 million profiles in OCS, we extended the problem of routing and monitoring beyond micro-services, in our connection lib to fulfill the 5G URLLC requirements. The lib routes continuously and flexibly while taking business KPIs into account. We have designed Microchassis - OCS ILB connection lib, implemented in Java, in place of the service mesh sidecar.

III. MODELING

A. Max flow problem modeling

We model the L7 max flow problem, as the following:

Definition III.1 (Service layer). Let $\mathcal{L}_i$ denote a service layer i th of a q service layers system $\mathcal{S}_{\gamma, \delta, q}$ or a subsystem since services p th to q th denoted by $\mathcal{S}_{\gamma, \delta, p, q}$, describes γ incoming gate and δ outgoing gate. Each layer $\mathcal{L}_i$ at time t contain n_i homologous instances ζ_i

$$|\mathcal{L}_i(t)| = \left| \{ \zeta_i^k | k \in [1, n] \} \right| = n_i \quad (1)$$

Iff tail layer $\mathcal{L}_j$ has its unneglectable pressure responses, then consider it as mid of the response flow (2). Else it should be considered as the tail of the flow (3) and separated into two, request (3a) and response (3b) flow.

$$\mathcal{L}_0 \xrightarrow[\leftarrow Y_{2j-1}]{\xrightarrow{X_1}} \mathcal{L}_1 \xrightarrow[\leftarrow Y_{2j-2}]{\xrightarrow{X_2}} \dots \xrightarrow[\leftarrow Y_{j+1}]{\xrightarrow{X_{j-1}}} \mathcal{L}_{j-1} \xrightarrow[\leftarrow Y_j]{\xrightarrow{X_j}} \mathcal{L}_j \quad (2)$$

$$\left\{ \begin{array}{l} \mathcal{L}_0 \xrightarrow{X_1} \mathcal{L}_1 \xrightarrow{X_2} \dots \xrightarrow{X_{j-1}} \mathcal{L}_{j-1} \xrightarrow{X_j} \mathcal{L}_j \quad (3a) \\ \mathcal{L}_0 \xleftarrow{Y_{2j-1}} \mathcal{L}_1 \xleftarrow{Y_{2j-2}} \dots \xleftarrow{Y_{j+1}} \mathcal{L}_{j-1} \xleftarrow{Y_j} \mathcal{L}_j \quad (3b) \end{array} \right.$$

Definition III.2 (Compound instance). A service instance is a combination of H handling functions $\mathcal{H}$ share the same set of resources with at least 1 of the following differences in signature:

- incoming processor
- one of a function of inner instance processing chain
- outgoing processor

, each chain of handling summarized a handle f_h

$$\mathcal{H}_i = \bigcup_{h:h \in [1,H]} f_h, \quad \forall h_1 \neq h_2 \in [1,H] \left\{ \begin{array}{l} f_{h_1} \cup f_{h_2} \neq f_{h_1} \\ f_{h_1} \cup f_{h_2} \neq f_{h_2} \end{array} \right. \quad (4)$$

Definition III.3 (Discrete time). On segment time $\mathcal{T}_m$, or window T_k is the amount of time which has absolute duration time Δ_{t_k} bounded with time t_k since beacon time t_0 . Notice that when mentioning segment time $\mathcal{T}_m$, it is a range where many inclusive requests fit in, and when mentioned window T_k , it illustrates ordered windows occur exactly $k+1$ inclusive requests. Minor time ϵ_{t_k} is the idle duration time between two adjacent requests. By the order of processing messages in the network buffer, messages come in order, there aren't any two requests coming at the same time.

$$\begin{aligned} \forall t_k > t_{k-1}; \quad t_k > t_0 \\ \Delta_{t_k} &= \Delta(t_0, t_k) = |t_k - t_0| \\ \epsilon_{t_k} &= \Delta(t_{k-1}, t_k) = |t_k - t_{k-1}| \\ t_k &= t_0 + \Delta_{t_k} = t_{k-1} + \epsilon_{t_k} \\ |T_k| &= |\mathcal{T}_k| = \Delta_{t_k}, \quad T_0 = t_0 \text{ and } |T_0| = 0 \end{aligned} \quad (5)$$

for a window slide from t_a to t_b , both count since t_0 , we have

$$\forall t_a < t_b, \quad |T_a^b| = t_b - t_a \quad (6)$$

Definition III.4 (Capacity varying). Representative utilizable $\mathcal{U}$ is used for forwarding request x_{t_0} , consumed demand $d(x_{t_0})$. Then, the utilizable resources amount is formalized as:

$$\mathcal{U} = \frac{d(X_{T_m})}{\Delta t_m} = \frac{\sum_{j:j \in [0,m]} d(x_j)}{\Delta t_m} \quad (7)$$

Definition III.5 (Messages). X_{T_m} denotes set of $m+1$ requests of same type appears within the window T_m bounded inclusive by t_0 and t_m requests. X_{T_0} is not a valid range, although it defines 1 request x_{t_0} but without a time range.

$$X_{T_m} = \{x_{t_0}, x_{t_1}, \dots, x_{t_m}\} \quad (8)$$

for an inclusive sliced window that contains requests in group X , we denote

$$X_{T_a^b} = \{x_{t_a}, x_{t_{a+1}}, \dots, x_{t_b}\} \quad (9)$$

Definition III.6 (Throughput state). Current throughput of layer i contain m_i messages

$$\omega_{i,\mathcal{T}_m} = \lim_{t \rightarrow t_m} |X_i| = \frac{dm}{dt} = \frac{m_i}{\Delta t_m} \quad (10)$$

Throughput over an instance a of layer i is number of message over window time T_m

$$\Theta(\zeta_i^a, X_{T_m}) = \frac{m+1}{\Delta t_m} \quad (11)$$

or segment time $\mathcal{T}_m$

$$\begin{aligned} \Theta(\zeta_i^a, X_{\mathcal{T}_m}) &= \Theta(\zeta_i^a, X_{T_a^b}) = \frac{|X_{\mathcal{T}_m}|}{|T_a^b|} \quad \forall x_j \in X \wedge x_j \in T_a^b \\ &= \frac{|X|}{|T_a^b|} = \frac{b-a+1}{t_b - t_a} \end{aligned} \quad (12)$$

Then throughput over layer $\mathcal{L}_i$ is

$$\Theta(\mathcal{L}_i, X_{\mathcal{T}_m}) = \sum_{a=1}^{n_i} \Theta(\zeta_i^a, X_{\mathcal{T}_m}) = \omega_{i,\mathcal{T}_m} \quad (13)$$

And we defined $\hat{\Theta}(X_{T_m})$ as configured throughput representing the route strategy allowed in window T_m . $\hat{\Theta} = 0$ is a disabled connection.

Definition III.7 (Throughput capacity). For each layer, throughput capacity $\phi_{i,\mathcal{T}_m}$ represents for max number of messages servable in duration $\mathcal{T}_m$.

$$\tilde{\Theta}(\zeta_i^a, X_{\mathcal{T}_m}) = \max_m \Theta(X_{\mathcal{T}_m}) \quad (14)$$

Throughput over a layer is the sum current throughput over all instances.

$$\tilde{\Theta}(\mathcal{L}_i, X_{\mathcal{T}_m}) = \sum_{a=1}^{n_i} \tilde{\Theta}(\zeta_i^a, X_{\mathcal{T}_m}) = \phi_{i,\mathcal{T}_m} \quad (15)$$

Definition III.8 (Concurrent in process state). Denote that client ILB ψ_i value represents in-flight messages of $\mathcal{L}_i$ client. In not jamming network and network, time assumed nearly 0, according to definition III.1, then minimum $\psi=1$ shown underlying server there was a single thread continuous processing requests. Ψ_i is the number of max allow to send, $\psi_i \in [0, \Psi_i]$. τ_i is constant number of threads in thread pool of $\mathcal{L}_i$ server, v is number of processing messages, and π_i is number of message in server queue. Active threads will handle the message, which means if the number of messages is less than the number of threads, the queue will be empty. Therefore number of message locate at instance k of layer i , $m_{\zeta_i^k}$ is:

$$m_{\zeta_i^k} = \min(v_{\zeta_i^k}, \pi_{\zeta_i^k} + \tau_{\zeta_i^k}) \quad (16)$$

$$\psi_i = m_{\zeta_i^k} + \psi_{i+1} \quad (17)$$

Definition III.9 (Saturation point). The system defines a performance rating ρ of current average flow demand within a segment time $\mathcal{T}_m$ on standard average demand on the ideal system.

$$\rho = \lim_{\Delta t_m \rightarrow \epsilon_1} \frac{\hat{d}(X_{T_m})}{d(X_{T_m})} = \frac{|\mathcal{T}_m|}{m+1} \quad (18)$$

If the performance rating drops under a predefined threshold or it makes the flow change the handle function, mark it as saturation point ϱ . Since $\rho < \varrho$, the number of messages over segment $\mathcal{T}_m$ decreased, so did the throughput.

The max flow problem is formalized as single data flow through service mesh, based on two Proposition III.1.1 and III.1.2. This proposition models the problem as a steady flow problem, which means the fluid of data is retained after layers.

Proposition III.1. *Single presentation in time:*

Proposition III.1.1 *Time representation: Any represent of the service layer is based on its path order on the processing flow, instead of its connected graph order. If the request traverse through the instance multiple time into different handle f_h that $f_{h_1} \neq f_{h_2}$, it is considered as two different steps to calculate back-pressure, as illustrated in fig 1.*

$$\forall t_0 < t_1 \Leftrightarrow, x_{t_0} \mapsto x_{t_1} \Rightarrow f_{h_1}(x_{t_0}, t_0) < f_{h_2}(x_{t_1}, t_1) \quad (19)$$

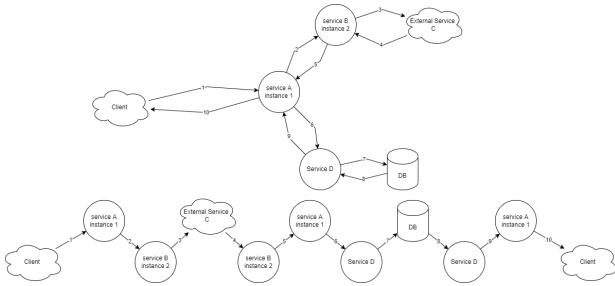


Figure 1. Flow stretching

Proposition III.1.2 *Split flow: (a) Any fan-out stage on the flow at the outbound of $\mathcal{L}_i$ which duplicates an output message is divided into different flows due to Definition III.2. (b) By the nature of sending time, if there were multiple different asynchronous outputs sent to multiple destinations, those are considered as multiple separated flows.*

Corollary III.0.1. *Forward once: Based on proposition III.1.2, there are not multiple outputs from any $\mathcal{L}_i$ at any point of time. Then outbound of the upstream layer is the direct inbound of the downstream layer.*

$$Y_i \equiv X_{i+1} \quad (20)$$

Definition III.10 (Request type). *At any time t that system is not in pressure, set of requests x that go through same $\mathcal{H}_i$ handle, or any two different output formats of same handle $\mathcal{H}_{i-1}$ to the same next layer handle $\mathcal{H}_i$ is considered as the same message type $\tilde{X}$.*

$$\tilde{X} = \{x | \forall t \forall x_1, x_2, \mathcal{H}_i(x_1, t) = \mathcal{H}_i(x_2, t)\} \quad (21)$$

$$\mathcal{L}_i(\tilde{X}_i, T_m) = \tilde{Y}_i \quad (22)$$

Lemma III.1. *Discrete counter: The request counter of request flow at any point of time for whole layer $\mathcal{L}_i$ is a discrete non-negative integer, hence 3 following components share the same features:*

- request sliding-windows counter ($|X_{T_m}|$)
- concurrent value (ψ and Ψ)
- speed counter and rate limiter (Θ and $\hat{\Theta}$)

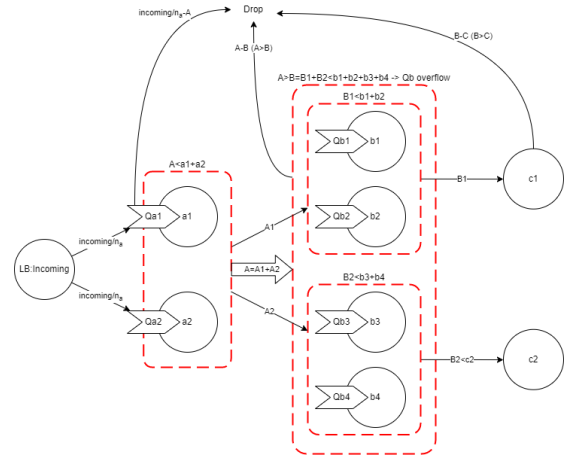


Figure 2. Internal queue analysis

Proof. By the definition III.3, III.5, a sliding windows count request represents a discrete value, then any sliding window-based counter and rate limiter display discrete non-negative integer value. $\square$

Theorem III.2. *Average resource demand: From equation 24 and definition III.4, we have*

$$\hat{d}(\tilde{X}_{T_m}) = \frac{\mathcal{U}\Delta t_m}{|\tilde{X}_{T_m}|} = \frac{\mathcal{U}\Delta t_m}{m+1} = \frac{\mathcal{U}}{\Theta(\tilde{X}, T_m)} \quad (23)$$

For combined load, have:

$$\hat{d}(X_{T_m}) = \frac{\mathcal{U}\Delta t_m}{|X_{T_m}|} = \frac{\mathcal{U}}{\Theta(X, T_m)} \quad (24)$$

Utilizable load $\mathcal{U}$ dynamically changed. Therefore $\hat{d}(X)$ is a function of t .

$$\hat{d}(X) = f(t) \quad (25)$$

Corollary III.2.1. *If max concurrent in process message reduced, then $\hat{d}(X_{T_m})$ reduce, hence reduce throughput $\Theta(X, T_m)$ due to the deterministic of $\mathcal{U}$*

Theorem III.3. *Drop throughput: $\bar{\Theta}(\mathcal{S}, X_{T_m}) = \omega_{\mathcal{S}, T_m} - \Theta(\mathcal{S}, X_{T_m})$*

Definition III.11 (Front-pressure algorithms). *We classify ILB routing algorithms are **Front-pressure** if they route incoming messages without the ability to slow down the total forwarding function throughput of the producer instance.*

By drop theorem III.3, Front-pressure must be controlled by capacity planning carefully and strictly work with auto-scaling, unless an infinite amount of drop and timeout will occur.

B. Back-pressure modeling

Theorem III.4 (Cascading concurrent condition). *The number of threads in the pool is advised by the business and can be overridden in run-time, therefore dynamic flow control Ψ_i*

parameter can be greater than τ_{i+1} . This denotes concurrent cascading. At any time we have:

$$\forall i \leq j, \psi_i \leq \Psi_j \quad (26)$$

$$\sup \Psi_{i+1} = \Psi_i \quad (27)$$

Proof.

$$\begin{aligned} \psi_i &= m_{\zeta_i^k} + \psi_{i+1} \leq \Psi_i \\ \Rightarrow \text{if } \Psi_{i+1} > \Psi_i, \text{ then } \psi_{i+1} &\leq \Psi_i \end{aligned} \quad (28)$$

On flow with throughput, generalizing we have

$$\Psi_i = \begin{cases} \psi_{i+1} = v_{i+1} & \Psi_i < \tau_{i+1} \\ \psi_{i+1} + \pi_{i+1} + \tau_{i+1} & \Psi_i \geq \tau_{i+1} \end{cases} \quad (29)$$

Expand to all layers within the flow, we have Cascading concurrent condition $\square$

Theorem III.5 (Cascading throughput condition). *From corollary III.0.1 have cascading throughput formula:*

$$\Theta(\mathcal{L}_i, X_{T_m}) = \Theta(\mathcal{L}_{i+1}, X_{T'_m}) \quad (30)$$

with T'_m is the time window $\mathcal{L}_i$ finished transforming load X_i into Y_i without any other load, then

$$\Rightarrow \Theta(\mathcal{L}_i, X_{T_m}) = \begin{cases} \Theta(\mathcal{L}_{i+1}, Y_{T_m^s}) & = \frac{m_i}{\Delta t} \text{ if } m_i < \phi_i, T_m \\ \phi_i, T_m & = \frac{m_r}{\Delta t} \text{ if } m_i \geq \phi_i, T_m \end{cases} \quad (31)$$

$$\Leftrightarrow \Theta(\mathcal{L}_i, X_{T_m}) = \frac{\min(m_i, m_r)}{\Delta t} = \Omega_{i, T_m} \quad (32)$$

that Ω_{i, T_m} is response throughput of layer i . From equation 30 and corollary III.0.1, have explicit and recurrent relation formula of response throughput:

$$\Rightarrow \Theta(\mathcal{L}_{i+1}, X_{T_m}) = \min(\Omega_{i, T_m}, \Omega_{i+1, T_m}) \quad (33)$$

Therefore layer throughput if formalized as

$$\Rightarrow \phi_{\mathcal{L}_i, T_m} = \min(\Theta(\mathcal{L}_i, T_{m_1}), \phi_{\mathcal{L}_{i+1}, T_{m_1}}) \quad (34)$$

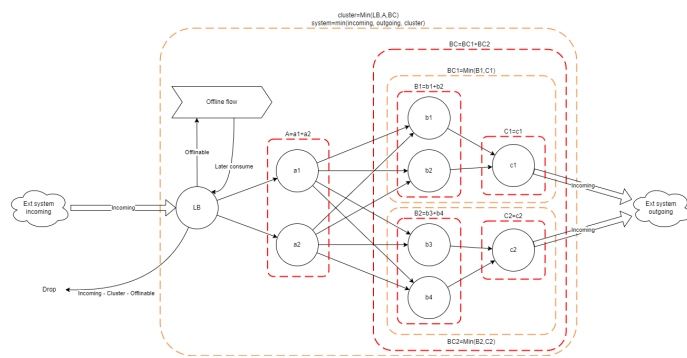


Figure 3. System throughput cascading

Corollary III.5.1. *Expand equation 34 to multiple load system, throughput over a back-pressure controlled system*

equivalent to current throughput to slowest layer. We have the cascading capacity formula:

$$\phi_{S, \dots, p, q, T_m} = \min_{i \in [p, q]} \phi_{\mathcal{L}_i, X_{T_m^i}} \quad \forall 0 < m_i < m_j \leq m \quad (35)$$

This proved the cascading characteristic of back-pressure controlled flow that propagates the pressure (the $\min(\Omega_i)$ function) along the services chain.

Corollary III.5.2. *Congestion only occurs at the weakest layer which is the bottleneck of the current system state.*

Definition III.12. Define pressure $\mathcal{P}_i^w$ of service flow w at a layer i is the rating between the overhead ξ of resource time actual demand over natural time on a single load. Pressure is based on a single load so it is identified by location of service $\mathcal{L}_i$, more specific is ζ_i^w . Then denote Pressure function $\mathcal{P}$ of a requests handled by function w at time t as:

$$\xi(x_i^w, t) = d(x_i^w, t) - \hat{d}(x_i^w, t) \Rightarrow \mathcal{P}(x_i^w, t) = \frac{\xi(x_i^w, t)}{\hat{d}(x_i^w, t)} \quad (36)$$

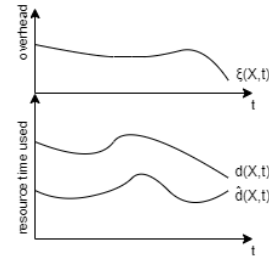


Figure 4. Pressure

From formula 24 and 36, we always have $d(x_i^w, t) \leq \hat{d}(x_i^w, t)$, therefore any pressure is non-negative. Then the pressure of the whole flow is integral to the pressure function.

$$\mathcal{P}(X_{T_m}) = \int_{T_m} \frac{\xi(x, t)}{\hat{d}(x, t)} dt = \int_{T_m} \frac{d(x, t)}{\hat{d}(x, t)} dt - t \quad (37)$$

Corollary III.5.3. *Performance rating is a function of time, inversely proportional to pressure.*

$$\begin{aligned} \rho(X) &= \frac{1}{\frac{\partial \mathcal{P}(X)}{\partial t} + 1} \in (0, 1] \\ \rho &\rightarrow 1 \Leftrightarrow \mathcal{P}(X) \rightarrow 0 \end{aligned} \quad (38)$$

When $\rho \geq \varrho$, throughput decreases, hence maximum throughput performs on saturation point.

Corollary III.5.4. *When back-pressure is locked by reaching a configured limit, it propagates upward and makes HoL locking, nearly immediately reducing the throughput of the whole system.*

$$\begin{aligned} \left[\begin{aligned} \Theta(\mathcal{L}_i) &= \min_{j \in (i, q)} \omega_j \\ \psi_i &\leq \Psi_j = 0, \forall i < j \end{aligned} \right] \rightarrow \begin{cases} \Theta(\mathcal{L}_1) = 0 \\ \psi_1 = 0 \end{cases} \rightarrow \begin{cases} \Theta(\mathcal{S}) = 0 \\ \psi_S = 0 \end{cases} \quad (39) \end{aligned}$$

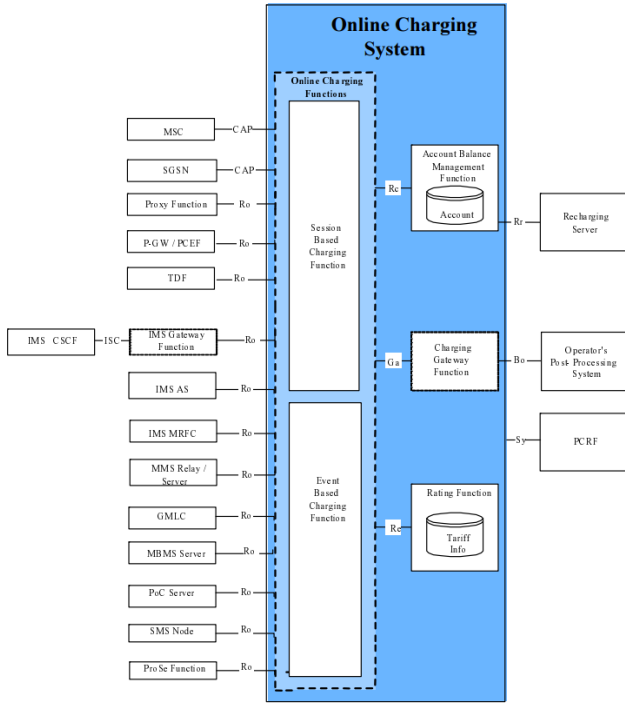


Figure 5. OCS architecture

Corollary III.5.5. *Back pressure is propagated directly when connections use synchronous or blocking actions. Asynchronous with enqueue and drop behavior or non-blocking actions lose pressure information, hence needed Cascading concurrent or throughput condition to form up an algorithm.*

Corollary III.5.6. *Back-pressure features: According to definition III.1, Back-pressure presents in both way of request and response. Back-pressure has a delay cascading effect due to queued layers. Losing cascading aid messages increases delay gaps. Back-pressure controller remove the need for sub-instance group divisions all can be connected in full mesh schema.*

Corollary III.5.7. *BTC control throughput policy: due to equation 33, the throughput of a layer is the minimum throughput of all under layers. The maximum of configuration throughput $\hat{\Theta}$ is:*

$$\max_{\mathcal{L}_i} \hat{\Theta} \leq \min_{j \in [i+1, q]} \phi(\mathcal{L}_j) \quad (40)$$

However on dynamic run-time, we cannot determine a strong $\phi(\mathcal{L}_j)$ for any layers, then we should use JIT value of this formula:

$$\Rightarrow \max_{\mathcal{L}_i} \hat{\Theta} \leq \min_{j \in [i+1, q]} \Theta(\mathcal{L}_j) \quad (41)$$

IV. VOCS MICROCHASSIS ILB

A. OCS features

OCS is a 24/7 service that receives billing orders from external systems. Messages incoming to the OCS are structured according to the OCS interface specification [18]. Knowing

that network components are high-performance, HA, and transparent with applications. OCS needs to fastest handling charging requests in any conditions at the application layer. With the characteristic of reserving business flow, OCS shows clear large rising waves, instead of peaks.

B. ILB implementation

Microchassis is a wrapped layer of Netty connections, control per-flow with local pressure information. Applications embedded lib Microchassis communicate with each other by TCP/IP protocol and HTTP1.1. Microchassis manages sending, monitoring, service discovery, establishing connections, flow control, and Front-pressure CB, flow/packet tracing, interacting with VNF vMANO as the control plane of vOCS systems. Even if an instance makes multiple connections to a specific destination, each connection decides isolated. This brings the mesh ability to localize errors and isolate jammed instances. With cross-service (cascading) pressure, although we enable auto-read of Netty inbound handler chain, however on a robust physical network, servers still represent congestions, The client side needs to check channel buffer isWritable() continuously before writtenAndFlush().

At the scope of ILB at the application layer, some threshold metrics that could be used are (1) the number of requests/objects and (2) the size of something: total allocated buffer, some active queue size, CPU time, etc. As mentioned in 1, we do not implement time-based algorithms like shortest delay, response time, or any statistical strategy based on it.

According to corollary III.5.5, we implemented two methods BCC and BTC along with two basic RR and LCC for dynamic switching strategies on ILB:

- Round-robin: based on current destinations, the pointer shifts over the list each time gets a new target. The complexity of operating is $O(1)$ and complexity of preparation is $o(n)$. When the destination list changes, we have to re-classify active/writable connection as a projectile snapshot and then shift over this list. RR only combined with speed increase-adjusting when the instance boots up or re-opens after the circuit broke without slow-down control. By accepting any incoming and not slowing down actively, RR is classified as a front-pressure strategy. The weighted extension of RR faces the same problems.
- Least concurrent: based on current concurrent on all connections. LCC maintains the same balanced amount of concurrent among connections. A high value of concurrent reflex global jamming situation but not identify it exactly. The complexity of selection is $o(n)$ and preparation is $o(n)$. The result of LCC can illustrate for P2R¹ and any least-X strategy.
- Back-pressure flow control based on concurrent state (BCC): is a flow control that senses for connection's concurrent state and decides that the connection should increase or reduce the concurrent. Not directly reducing

¹P2R only chooses better but does not prevent any pressure however adds unneglectable complexity to the implementation.

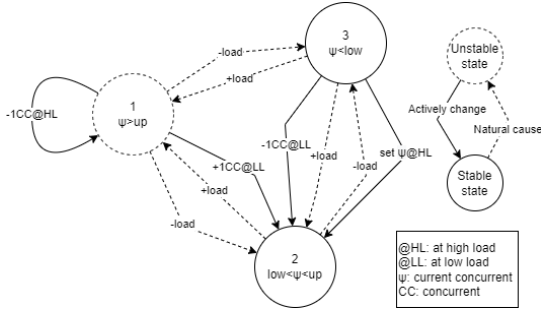


Figure 6. FSM of BCC

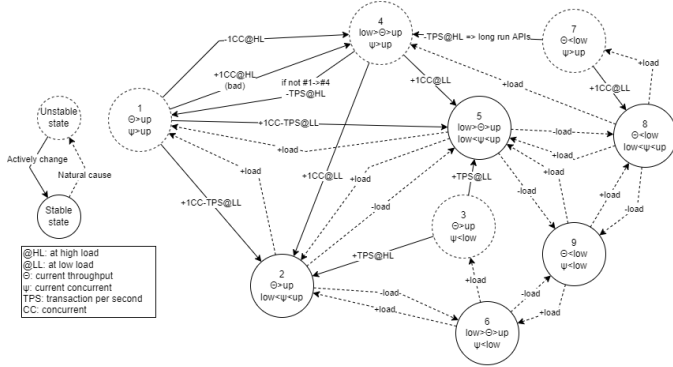


Figure 7. FSM of BTC

the speed but as shown in corollary III.2.1, BCC can control speed to be slow down by a decrease in configured max concurrent in-flight messages. The weakness of BCC is the chance to lock HoL. Transitions of BCC are illustrated in fig 6.

- Back-pressure flow control based on flow speed and concurrent state (BTC): a flow control utilized throughput and concurrent state as its score. Transitions of BTC are illustrated in fig 7. BTC shares the same complexity with BCC however direct control allows outcome throughput.

C. Back-pressure base on throughput and concurrent states

By two nature of a flow is unit speed and bandwidth, BTC propagated throughput and concurrent pressure to the HoL by sensing on each connection, on each cachable routing decision. BTC calculates scores of each flow based on an average of these two utilizations and selects a group of selectable destinations. To be sent, the message must wait through all conditions including concurrent, throughput, and itself computing time: $wait = \max(wait_{\Psi_i}, wait_{\hat{\Theta}(\mathcal{L}_i)}, wait_{compute})$. Expand to uncountable request flow, if we raise Ψ_i and $\hat{\Theta}(\mathcal{L}_i)$ up on well affordable instance, the total wait for each request only be computed time. We use 20% Ψ gap to minimize the HoL blocking effect of BCC and BTC.

Resolve loop configure and state locking The following loops may occur in the mesh:

- State 7 was short-lived. If state 7 increases Ψ when the load is low, it transitions to state 8. If there were a high

load then it returned to state 7. The 7-8-7 loop reduces throughput hence having a change to go to state 4. This case is self-solvable.

- State 1 when increasing Ψ leads to state 4 or state 5. State 5 is a stop state. When state 1 switches back and forth with state 4: we need to break to not lead to the configure adjustment loop.
 - The chain 1-4-1 results in increasing Ψ and reduced throughput that finally comes to state 2 self-end the loop with a suitable Ψ value. However in an incident, this loop increases Ψ while the consumer is overloading, this can lead to collapsing the whole service layer. So we remove increasing Ψ behave at state 1 in a high load situation.
 - The chain 1-4-1 reduces Ψ and $\hat{\Theta}$, which reduces configuration value continuously, cause: (1) throughput bottleneck then get to stop state 2, find out suitable throughput for instance based on preset Ψ or (2) bottleneck on Ψ then go to state 4, Ψ jamming won't trigger reduce throughput to make a race loop, but increase Ψ again to convergent to stop state 2 or 5.

V. EXPERIMENTATION AND DISCUSSION

A. Experiment

We benched performance for four methods: RR, LCC, BCC, and BTC. RR and LCC are the most popular methods in the group of front pressure methods, with RR being the naive, simplest, and most effective option for low-load systems. LCC is a method of grouping "least-X" over a metric X. BCC is the most basic for back pressure over a single metric method. In contrast, BTC is a testing method that combines back-pressure over multiple metrics and can be the base for tuning and learning flow control algorithms.

In this simulation, the database is simulated by an echo server. Inner applications are homogeneous single-flow instance layers. API gateway (LB) is a modified application that has generator threads, that generate random fields 4KB messages in ByteBuf format into LB incoming queue. The result at LB gives us a view of front-line queue status and send cycle time statistics (over client component of LB) as the latency a physical LB must wait: $RTT = InSQ:avgTime + SendCy:avgTime$. We divided our scenarios into 2 dimensions:

- Sizing dimation: 2x 3x 4x 6x 8x with 1x = 1 vCore CPU + 1.5GB physical memory
- Load dimation: underload/overload.

With each scale, application configures like handleThread, Xmx, CPU factor, ... scaled proportionally. Each service layer is deployed into 1 specific non-overlapping physical node to force all requests must travel over the physical network rather than some travel internal by localhost. Table 1 lists the configurations of 9 deployed servers in this study. To archive 99% of the confident level at 5% margin error of unlimited population, we collect 666 data samples with 1s windows since the first 100 skipped, matched timestamps from data points

TABLE 1. HARDWARE FOR BENCHMARK

	Servers
CPU	Dual Intel (R) Platinum 8260 @ 2.40GHZ
RAM	256GB 2933 MT/s
Network	BCM57414 NetXtreme-E 10Gb/25Gb RDMA
Disk	Dell Express Flash NVME P4610 1.6TB SFF
OS	RHEL 7.8
Virtualization	Kubernetes with Calico network

spanned among layers instances. Log4j logging is enabled for statistics and errors. Statistics is accounting per interval (1s) by Windows.

B. Results and evaluations

2 cores based multiple sizing underload test: Underload at 9kTPS in total 27x of 9 instances with each type spawn 3 pods. The result in table 2 shows that all methods are qualified with minor differences in underload test.

TABLE 2. THROUGHPUT TEST 9KTPS 27X=(2X3+3X3+4X3) 1 CONNECTIONS (*AVGTIME: μ s)

Strategy	RR	LCC	BCC	BTC
algo*	5.04	12.9	9.06	9.99
str*	7.54	15.66	12.18	14.57
InSQ*	6.64	6.96	6.99	6.72
SendCy*	1020	1095	1090	1067
InSQ/RTT	0.6%	0.6%	0.6%	0.6%

1) 4 cores based multiple sizing overload test:: Overload with 172kTPS on 36x total size for 6 instances in multiple types of base 4 cores show results in table 3

TABLE 3. THROUGHPUT TEST 172KTPS 36X=(4X2+6X2+8X2) 3 CONNECTIONS (*AVGTIME μ s, **AVGTIME MS)

Strategy	RR	LCC	BCC	BTC
algo*	5.02	15.34	8.08	10.68
str*	62.36	19.46	729.63	22.44
InSQ*	2820	58	21,237	518
SendCy*	29,966	5,977	10,664	5,088
RTT**	32.79	6.04	31.9	5.61
InSQ/RTT	8.6%	1%	66.6%	9.2 %

2) 4 cores based same sizing overload test:: We test again overload test with the same total sizing but divided into 6 identical 6x instances. LCC and BTC deliver outperforming latency, BTC is the same as we tested at minimum throughput (500TPS as a base comparison). This shows an undeniable ability to maintain the best quality at high throughput to obtain the vision of corollary III.5.3. Although resource utilization and most metrics are no different from BCC, BTC sacrifices about 5% throughput in exchange for a few times latency reduction showing superior performance in congestion control.

C. Discussion

URLLC defines 1-way latency for data plane less than 1ms. Within 5 layers, we have 4 inner connected layers, with 2-way connections to an external (source) system. Therefore,

TABLE 4. THROUGHPUT TEST 172KTPS 36X=(6X6) 3 CONNECTIONS (*AVGTIME μ s, **AVGTIME MS)

Strategy	RR	LCC	BCC	BTC
algo*	4.97	15.52	7.58	10.95
str*	33.68	19.65	862.59	22.08
InSQ*	1,644	51	66,510	1,055
SendCy*	21,334	4,070	33,412	5,121
RTT**	22.98	4.12	99.92	6.18
InSQ/RTT	7.2%	1.2%	66.6%	17.1 %

10ms is the upper bound of the transmit latency. As we tested, the proposed back-pressure algorithm BTC and the basic front-pressure LCC fulfill URLLC requirements. The front-pressure algorithms outperformed back-pressure ones. Latency in the front-pressure setups absorbed incoming pressure, which was reflected in the unacceptable RTT, avgTime:SendCy, and avgTime:InSQ to tens of milliseconds. Although LCC tries to find the best target, in the cases of sending to degraded layers, the LCC strategy does not solve the problem adequately. In contrast, back-pressure strategies detect stronger instances in the underlying layers. BTC balances the quality among individual targets, thanks to the throughput-throttling mechanism instead of concurrent-throttling on Ψ value in BCC. BCC struggles with small streams when $\min \Psi_i = 2$. Among front-pressure results, LCC proved itself to be a great competitor to RR. If the load is mostly lower than the capacity, utilizing back-pressure anywhere in the mesh leads to resources wasted. Front-pressure strategies are thus the most effective. As shown in figure 8, there was a cross between increasing performance efficiency due to more flow control policy, and decreasing resource usage pre-sized by the front-pressure layers. Front-pressure layers rely on pre-allocating and auto-scaling instances to prevent incidents caused by any throughput changes. Besides, back-pressure can maintain the throughput near the instance's maximum capacity and guarantee QoS during scaling events, by mitigating message overflow into a manageable early drop reply stream, until more capacity is provisioned.

Expansion in the next OCS generation will require more robust methods. In light of this, LCC is currently the best internal strategy for the mesh and BTC deserves to be placed at the head of flow.

Some limitations in the scope of the study are:

- Research has not considered the effect of queue size.
- Microchassis limitation: Currently ILB in Microchassis only supports HTTP and TCP messages for single requests, not yet support RPC-based connection and persistent HTTP stream.

VI. CONCLUSION

To obtain a smooth 5G real-time charging service, we use the most latency-effective approach - connection lib - as the integrated chassis for the mesh. In the paper, we evaluated connection lib with four routing strategies RR, LCC, BCC, and BTC and determined that the lib satisfied URLLC requirements under various conditions. Among tests, LCC is

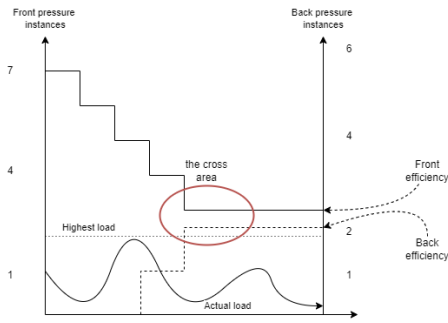


Figure 8. Effective trade-off cross

the most suitable for internal routing with higher resource efficiency, while BTC should be used for border gateways to maintain overall QoS. In the future, we will continue to enhance our strategies, along with tuning algorithms to better adapt edge cases. In particular, we will expand the problem to include geo-location and the ability to work with centralized solutions.

REFERENCES

- [1] I. T. Union. Minimum requirements related to technical performance for imt-2020 radio interface(s). [Online]. Available: <https://www.itu.int/pub/R-REP-M.2410>
- [2] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018.
- [3] Istio. Performance and scalability. [Online]. Available: <https://istio.io/latest/docs/ops/deployment/performance-and-scalability/#data-plane-performance>
- [4] Envoy. Supported load balancers. [Online]. Available: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/load_balancers#arch-overview-load-balancing-types
- [5] X. Zhu, G. She, B. Xue, Y. Zhang, Y. Zhang, X. K. Zou, X. Duan, P. He, A. Krishnamurthy, M. Lentz *et al.*, "Dissecting service mesh overheads," *arXiv preprint arXiv:2207.00592*, 2022.
- [6] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019, pp. 1–3.
- [7] F5. K42275060: There are several load balancing methods. which one is best for your environment? [Online]. Available: <https://my.f5.com/manage/s/article/K42275060>
- [8] I. Alexandre Cassen. Ipv6 scheduling algorithms. [Online]. Available: https://keepalived-pqa.readthedocs.io/en/latest/scheduling_algorithms.html
- [9] N. Owen Garrett, F5. Nginx and the "power of two choices" load-balancing algorithm. [Online]. Available: <https://www.nginx.com/blog/nginx-power-of-two-choices-load-balancing-algorithm/>
- [10] H. Willy Tarreau. Test driving "power of two random choices" load balancing. [Online]. Available: <https://www.haproxy.com/blog/power-of-two-load-balancing>
- [11] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *4th USENIX Symposium on Internet Technologies and Systems (USITS 03)*, 2003.
- [12] N. K. Apache Flink. A deep-dive into flink's network stack. [Online]. Available: <https://flink.apache.org/2019/06/05/a-deep-dive-into-flinks-network-stack/#credit-based-flow-control>
- [13] Uber. Hyperbahn. [Online]. Available: <https://github.com/uber-archive/hyperbahn>
- [14] Netflix. Ribbon. [Online]. Available: <https://github.com/Netflix/ribbon>
- [15] H. Saokar, S. Demetriou, N. Magerko, M. Kontorovich, J. Kirstein, M. Leibold, D. Skarlatos, H. Khandelwal, and C. Tang, "{ServiceRouter}: Hyperscale and minimal cost service mesh at meta," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 969–985.

- [16] M. R. S. Sedghpour, C. Klein, and J. Tordsson, "Service mesh circuit breaker: From panic button to performance management tool," in *Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems*, 2021, pp. 4–10.
- [17] F. S. Samani and R. Stadler, "Dynamically meeting performance objectives for multiple services on a service mesh," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 219–225.
- [18] 3GPP. Ts 32.296: Telecommunication management; charging management; online charging system (ocs): Applications and interfaces. [Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1913>



Ngoc Tien Nguyen received his B.Sc. degree in Information System in 2020 from University of Engineering and Technology - Vietnam National University, Hanoi. He is currently a software engineer in Viettel High Technology, Viettel Group. He has industry experience with databases and high-performance computing, especially for telecommunications.



Thanh Son Pham received his B.Sc. degree in Mathematics and Informatics Engineering from Hanoi University of Science and Technology, Vietnam in 2018. He is currently a software engineer in Viettel High Technology, Viettel Group. He has industry experience with micro-service and high-performance computing, especially for telecommunications.



Van Duong Nguyen received his B.Sc. degree in Mathematics and Informatics Engineering from Hanoi University of Science and Technology, Vietnam in 2015. He is currently manage a database team of the Research Center for OCS (Online Charging Platform) of Viettel High-Tech Industry Corporation (VHT), Viettel Group. He has many years of experience in research and development of 5G core systems such as online charging system. His research interests include telecommunications core networks and database technologies.



Cong Dan Pham received his PhD. Degree in Probability and Statistics from Aix-Marseille University, France in Jun 2014. He has academic experience in universities like Aix-Marseille University, Hanoi University of Education, and industry experience companies like Viettel Group. He has published various journal and conference papers in IEEE and ScienceDirect. His research interests include probability and statistics, data science and machine learning.



Duc Hai Nguyen received his B.Sc. in Mathematics and Informatics Engineering from Hanoi University of Science and Technology, Vietnam in 2009. He is currently director of the Research Center for OCS (Online Charging Platform) of Viettel High-Tech Industry Corporation (VHT), Viettel Group. He has many years of experience in research and development of 5G core systems such as online charging system. His research interests include telecommunications core networks, Cloud Computing.