Caching and Containerization of IP Address Allocation Process in 5G Core Networks for Performance Improvements

Nguyen Anh Tuan
Switching Technology Research Center
Viettel High Technology Industries Corporation
Hanoi, Vietnam
tuanna137@viettel.com.vn

To Quoc Hung
Switching Technology Research Center
Viettel High Technology Industries Corporation
Hanoi, Vietnam
hungtq40@viettel.com.vn

Nguyen Tai Hung
School of Electrical & Electronic Engineering,
Hanoi University of Science and Technology
Hanoi, Vietnam
hung.nguyentai@hust.edu.vn

Nguyen Tien Dong
Switching Technology Research Center
Viettel High Technology Industries Corporation
Hanoi, Vietnam
dongnt18@viettel.com.vn

Dinh Viet Quan
Switching Technology Research Center
Viettel High Technology Industries Corporation
Hanoi, Vietnam
quandv@viettel.com.vn

2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM) | 978-1-6654-5348-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/IMCOM56909.2023.10035655

***Abstract*— We present the implementation of a caching system within the IP Allocation Service of the 5G Core Networks. The IP allocation service is one of many services in the Session Management Function (SMF) of the 5G Core that supports the allocation and management of User Equipment (UE) IP addresses. Normally, this service uses a database layer to manage the available IP address ranges. However, the current design of this function requires a database fetch for every IP address allocation request, which is called every time session establishment is called. This is costly in terms of computing and networking resources. Our proposed solution employs a caching system that fragments the available IP pool between pods (deployable computing units managed by Kubernetes that make up the service), saves the ranges to the pods' local memory resources from a shared database layer, and allows each pod to independently manage IP addresses within its range. Our testing results show that this architecture greatly improves the networking resources consumed while maintaining the consistency of IP address allocations across the network.**

Keywords—IP Allocation, 5G Core Network, Caching, Kubernetes, Pods

I. Introduction

5G systems support interworking with intranet networks or the Internet based on the Internet Protocol. To communicate and receive service on an IP network, a host must have an IP address. In the 5G network, the host in question is the User Equipment (UE), and the Session Management Function (SMF) is responsible for allocating IP addresses to the UE [1]. This can be either a static or a dynamic IP address.

The 5G architecture is split into control and user planes to better manage networking and computing resources. The control plane (CP) includes network functions that manage signaling, subscription management, authentication, and fee charging and does not have high bandwidth, low latency requirements. The SMF is one such function and manages the establishment, modification, and release of UE connectivity sessions, also called PDU (Protocol Data Unit) Sessions. The user plane (UP), on the other hand, handles user traffic and is deployed at the network edge to provide low latency and high bandwidth services. In traditional IP networks, the interworking with subnetworks or other IP networks is done with IP routers. From the perspective of the IP network interworking with a 5G network, the UP is seen as a normal IP router [2]. Therefore, the SMF must also allocate IP chunks–which are ranges of IP addresses–to the UP so that the UP may advertise and correctly route traffic between the network and the UE. The UE generally receives an IP address that falls into the IP chunk of the serving User Plane Function (UPF).

To adhere to the cloud native architecture of the 5G Core, the SMF is made up of loosely coupled micro-services that split functionality between each service. The IP Allocation service is one such micro-service that provides the function of allocating and managing IP addresses for UE. A high level decomposition of the SMF is as follows: Layer 1: Ingresses and Egresses, which act as gateways for other network functions and manage high availability and load balancing traffic between computing units; Layer 2: Application and Logic Core, which consists of a number of stateless software cores that handle the SMF's internal business logic; and Layer 3: Database, which keeps track of system state. The IP Allocation service resides in the Application layer and uses the Database layer to maintain consistency.

The IP Allocation service must be able to dynamically allocate IP addresses that belong to the correct chunks for the UE, allocate unique IP addresses for each UE on the network, and handle heavy traffic during peak hours. The IP service must also be robust, able to handle errors such as when the UE does not return the IP address allocated to the PDU session when the session is released. In this paper, we present two different strategies for the IP allocation service that adhere to these requirements and compare the performance of the presented strategies.

II. Previous Works and Problem Statement

IP address management has always been a core technology of IP networks. Early networks split the entire IP address space into classes, with larger network operators being allocated a fixed IP address space based on their network size [3]. This scheme proved to be wasteful, as the true size of networks

rarely matched the total number of allocatable addresses. The rapid growth of the Internet required a better IP management mechanism. A classless scheme to manage IP addresses, Classless Inter-Domain Routing, was introduced, which allowed arbitrary local network sizes [4]. The IANA is the current authority that oversees global IP address allocation. Private IP networks, along with NAT [5] [6], are another technology used to better manage IP addresses based on the observation that most IP hosts do not require a public IP address and may share a public IP address with other hosts in a local network. Dynamic IP addresses allow for even further utilization of IP addresses. When a host is no longer part of the network or temporarily disconnected, dynamic IP addresses allow other hosts to reuse the IP address. Most private networks use Dynamic Host Configuration Protocol, or DHCP, to support dynamic IP addresses [7].

Larger network operators, such as ISPs, may use more sophisticated IP address management (IPAM) systems [8] to efficiently use the allocated address space. ARPIM [9] is one such IP pool management system that takes advantage of SDN technology to dynamically allocate addresses and automate scheduling. Reference [10] describes another IP management model based on attributes that reduces IP packet traversal times based on topology in fixed broadband networks. Mobile networks present a unique problem. Whereas traditional IP networks assume fixed endpoints, mobile networks have to ensure session consistency even as endpoints move throughout the network. Support for mobile IP addresses, where IP datagrams are routed to mobile nodes on the Internet, is described in [11]. This is done by assigning a home address to the mobile node and a current address, which provides the real point of attachment of the node to the Internet.

However, 4G/5G networking systems that use the methods mentioned above for IP address allocation have certain drawbacks. In systems using the DHCP protocol, DHCP cannot identify the IP addresses in use by non-DHCP clients on the network, and when there is more than one DHCP server operating on one network, a DHCP server cannot verify the IP addresses already allocated by other servers. For these reasons, the DHCP protocol requests the client to use ARP (Address Resolution Protocol) to validate the allocated IP addresses. ARP is a protocol a host may use to discover which node on a network has a certain IP address. In multiple applications, the ARP response time definition exceeds 1 second, while DHCP requests the client to wait at least 10 seconds before initiating another application. During this time, the user equipment's in-use services (such as mobile calls over IP) have no allocated IP address and cannot be used [12]. Other IPAM systems suffer from drawbacks such as dependencies on external packages or systems, lack of scalability and data consistency, and heavy manual efforts from network operators. In the following sections, we introduce an IP address management solution that address these drawbacks.

III. IP Address Allocation on Mobile Networks

A. Curent Solution - Trivial Design

The following section describes an early implementation of the IP allocation service. We use the term "application" to refer to a single instance of the IP Address Allocation program. The IP Allocation Service is made up of one or more applications and can be scaled to meet demand by increasing the number of applications. The service uses a database layer to keep track of the following information:

- DP instance – Data processing module of the UPF,
- DNN – name of Data Network,
- IP Type – Type of IP address allocated, which is configured by the UE and the Network Core on PDU Session Establishment. This can be IPv4, IPv6 or IPv4v6,
- IP Range List – IP Ranges belonging to the operator network address space that the operator may choose to allocate to UEs. These IP chunks are usually /16 and /32 in CIDR notation.

1) Predefining IP Pool

A separate service fills the database with IP range tables based on the mobile network provider's needs. The tables are as follows:

- APN table: saves IP address chunks with the User Plane Function instance that uses this IP chunk and whether or not the chunk can be used to further allocate addresses.
- For each available IP range that can be used for IP address allocation, save 2 tables:
  - The first table stores the list of all IP addresses available for allocation in the IP range. The table name is set in the form of [POOL_IPType _DpName_IPRange]. The table has the IP address as the record key and the IP address's corresponding index as the value. Each index is a unique integer value generated for each IP address. This index is used to manage the IP addresses.
  - The second table stores the list of all indexes corresponding with all allocated IP addresses. The table name is set in the form of [POOL_IPType_DpName_IPRange_Index].
- For each configured DNN, save two tables with their names in the form of [IPv4_Allocation_DNN] and [IPv6_Allocation_DNN]. Each table stores all allocated IP addresses of the corresponding IP Type.

2) Allocating Dynamic IP addresses

When a PDU Session Establishment takes place, either requested by the UE or triggered by the network, the SMF looks at the Session Establishment Request and requests an IP address based on the data network and session type from the IP Allocation Service:

- Step 1: Select the IP range that will be used for IP address allocation based on DNN, IP Type value received from the IP address allocation request message, and the DpName value selected by the round-robin method.
- Step 2: Pop the first index in the indexes list corresponding to IPRange. (When the IP allocation application starts, for each IPRange, a list of all available

IP addresses' indexes is stored in the application's memory.)

- Step 3: From the [POOL_IPType_DpName_IPRange] table, get the IP address with the corresponding index obtained in step 2. Return the selected IP address value through an IP address allocation response message.
- Step 4: Update allocation information to the tables in the database:
  - Update the information of the allocated IP address to the corresponding [IPv4_Allocation_Dnn] or [IPv6_Allocation_Dnn] table.
  - Update the newly allocated IP address's index to the [POOL_IPType_DpName_IPRange_Index] table.

3) Deallocation of IP Address

The IP allocation service collects allocated IP addresses only when it receives the IP address giveback request from SMF. The IP allocation service processes the request as follows:

- Specify the IP address value that SMF requests to give back.
- Check if the IP address is in a corresponding [IPv4_Allocation_Dnn] or [IPv6_Allocation_Dnn] table for the IP type and DNN. If not, return a de-allocation error message through the IP address giveback response and end the procedure.
- Delete the giveback IP address from the [IPv4_Allocation_Dnn] or [IPv6_Allocation_Dnn] table.
- Based on the IpType value received from the de-allocation request message, DpName obtained from the [IPv4_Allocation_Dnn] or [IPv6_Allocation_Dnn] tables, and IPRange specified from DNN, DpName, and IPPrefix information, add the IP Address information to the corresponding [POOL_IPType_DpName_IPRange] table.
- Return a successful de-allocation message through the IP address giveback response.

There are some notable drawbacks to this current solution:

- First, the reserved IP addresses for allocation are stored in a list type in the database. Although IP space is separated into smaller lists based on DNN, IP type, and DNN, each list still contains many IP addresses. Accessing such a large list costs significantly more time and resources (especially CPU resources), which affects IP address allocation response time and makes the application's resource usage over time greatly fluctuate.
- The application performs too many database operations when it processes a request from SMF. For example, when the application allocates an IP address, it performs one query operation (step 3 in "2) Allocating Dynamic IP Addresses" of Section A) and two write operations (step 4 in "2) Allocating Dynamic IP Addresses" of Section A). Too many database operations directly increase response time and resource usage.
- Features such as automatic recovery of unused IP addresses are difficult to implement.

Therefore, to improve the IP address management system and overcome these drawbacks, we propose a new solution with a cached and containerized design in the following section.

B. The Proposed Solution – Cached and Containeried Design

1) First procedure: Generating IP address space and information tables

- Create IP Address Space: The IP Address Space is stored as a table in the database. Each record in the table relates to one predefined IP address list for a DP (data plane) node instance that belongs to a DNN. This IP address space contains all the IP addresses that can be allocated. The record key is the IP address range name, which is set in the format [DNN_DpName_IpType] and is used to distinguish between different IP address lists. The record value is a list containing all IP addresses in the corresponding IP address list.
- Initialize the information tables: as the IP address allocation application processes incoming requests, it needs to know certain information regarding allocation state, for example, which IP addresses have been allocated and which IP addresses have been taken back and are ready to be added again to the IP address space. That information is stored in different tables (called information tables) as follows:
  - Allocated IP addresses: Stores all the IP addresses that have been allocated to UE. For easier management, tables are divided into the IPv4 table and the IPv6 table.
  - Deallocated IP addresses: Stores all the IP address that have been taken back, both from SMF requests and automatically taken back by the application. For easier management, tables are divided into the IPv4 table and the IPv6 table.
  - Cached IP addresses: Stores all the IP addresses that had been loaded and stored in application memory. For easier management, tables are divided into the IPv4 table and the IPv6 table.
- Save the created IP range lists to the database layer. To search for IP spaces by DNN, DP Instances, and IP Type, the database saves the valid IP address space as [DNN_DPName_IPType] e.g., suppose a UE wants to connect to the Internet, resides in the service area of the UPF instance dp0, and requests the core network for an IPv4 address, and dnn0 is the DNN that connects to the wider internet inside the core. The application will search the database for the appropriate IP address space using the key "dnn0_dp0_ipv4".
- The service is deployed in a master slave configuration, with a single master initializing the IP address space.

2) Second procedure: Initializing pods

- With the trivial design described in Subsection A, the IP addresses in IP space are stored in a list type with many elements. With the cached and containerized design, a

part of the IP space is loaded and stored in the application memory. This part is the IP cache list. The application uses this IP cache list to allocate IP addresses. When all IP addresses in the cache list are allocated, the application will load another cache list to continue allocating IP addresses.

- In order to ensure that the IP addresses are not lost when an application is restarted, a copy of the IP cache list is also stored in the database.
- The application will scan and check the database tables for IP addresses that can be deallocated.

3) Third procedure: Allocating IP Addresses

The IP address allocation procedure is triggered when the SMF sends an IP address allocation request to the IP address management system.

The call flow between the SMF, IP address management, and the database layer in the allocating IP addresses procedure is depicted in Fig. 1. For all IP address allocation requests received, the IP address management system reads the request parameter, gets the appropriate IP address or IP address range to allocate, and updates the database with changes to the allocated IP list and cached IP list.

The IP address allocation is processed as follows:

- Step 1: The IP address allocation application chooses a DP instance to allocate the IP address from using a round robin method based on the DNN and IP Type from the IP allocation request. From DNN, IP type, and DP name information, the application defines the IP address range's name to be used.
- Step 2: The application takes one IP address from the IP cache corresponding to the name defined in step 1. This IP address will be deleted from the IP cache in the application memory and the IP cache table in the database. This IP address will be added to the table of allocated IP addresses in the database.
- Step 3: The application responds with the allocated IP address's information to the SMF application.

4) Fourth procedure: Deallocating IP Addresses

There are two methods for the IP allocation service to deallocate an IP address: either triggered by a PDU session release from the UE or the network, or automatically by periodically scanning the IP pool for IP addresses that have expired their lease. Our design supports both methodologies, the processing of which is depicted in Fig. 2 and Fig. 3.

IV. Analysis of Trivial and Cached Design Under High Load Condition

To evaluate the performance of the new system, we have setup the test-bed with the necessary components as depicted in Fig. 4 and conducted four different performance benchmarking tests to compare resource usage (CPU, memory) and processing speed (transaction latencies) between the trivial and cached and containerized implementation.

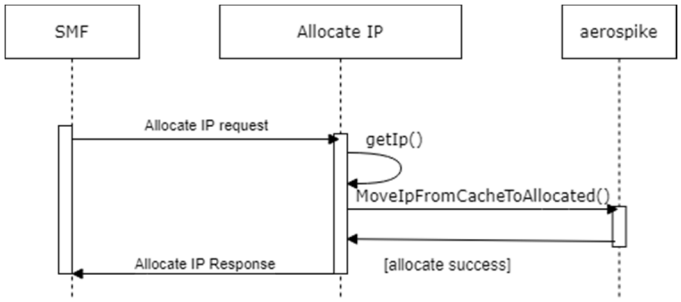


Fig. 1: The UML sequence diagram above depicts the IP address allocation procedure between the SMF service, the IP address allocation service with cache containerized design, and the database layer.

The test-bed consists of an IPAM client and a Kubernetes [13] cluster containing the IP address management service and an Aerospike [14] database, both deployed as services in the same cluster. The test consists of sending requests from the client to the IP address management application.

The first two benchmarking tests measure the application's performance when processing high rates of IP address allocation requests. The rates are, sequentially, 1200 transactions per second (TPS) and 3000 TPS. A transaction is the sequence of actions from when the client sends a request message to when the client receives the response message from the IP address allocation. The following two tests measure the application's performance when processing IP address deallocation requests at the same rates as above.

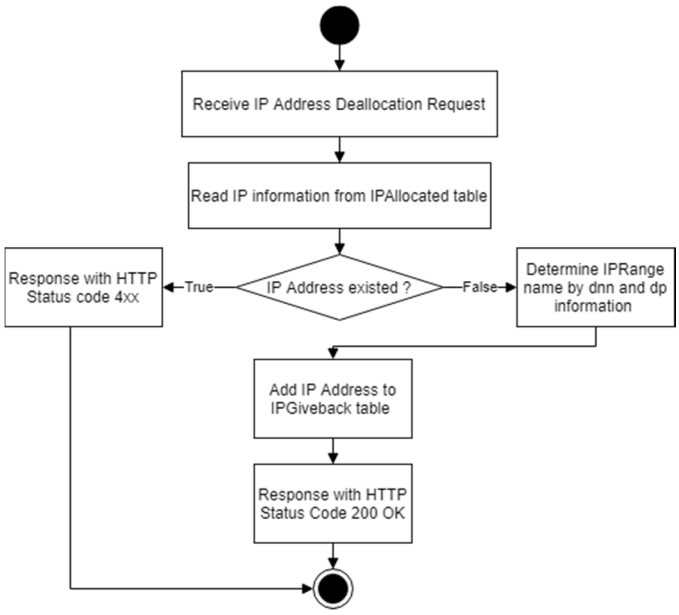


Fig. 2: IP deallocation in the case of PDU Session Release

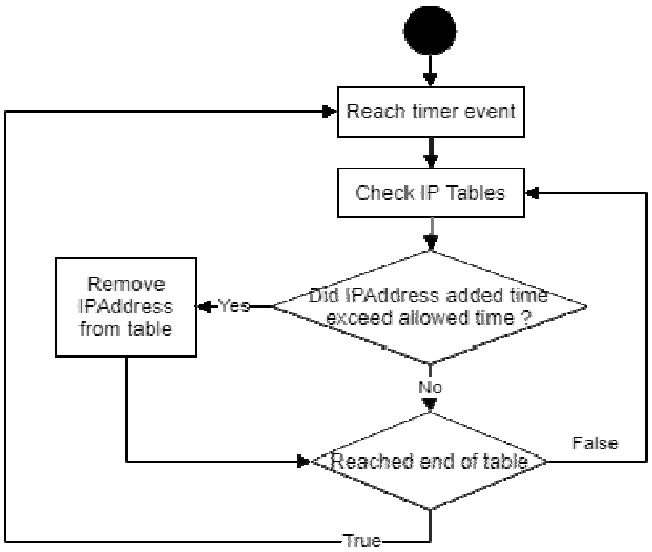


Fig. 3: IP address deallocation by automatically scanning for unused IP addresses. For each table a thread is run.

The benchmarking tests were performed as follows:

- Step 1: Initiate the IP address allocation application with predefined configurations. To avoid context switching overhead caused by the Host, the application is pinned to a single CPU.
- Step 2: Initiate the client application with predefined configurations. The client is pinned to a single CPU.
- Step 3: The client application starts sending requests to the IP address allocation application at a preconfigured rate.
- Step 4: The average TPS and average latency are calculated in the client application. A transaction is considered complete when the client receives the response message from the IP address allocation application. The latency of a transaction is the time interval between when the request is sent by the client application and when the client application receives the corresponding response.
- Step 5: The IP address allocation application host monitors and reports resource usage (CPU and memory).

Each benchmark test includes one IP address allocation application acting as the server to handle requests and one benchmarking tool acting as the client. The server can set the maximum number of CPUs and memory available. Similarly, the client may limit the number of CPUs used, the number of concurrent requests sent to the IP address allocation application, and the rate of requests sent. Both the IP address allocation application and the client are implemented using Golang to utilize its goroutines feature for building high-concurrency applications. The IP address allocation application and the database are deployed in a Kubernetes cluster.

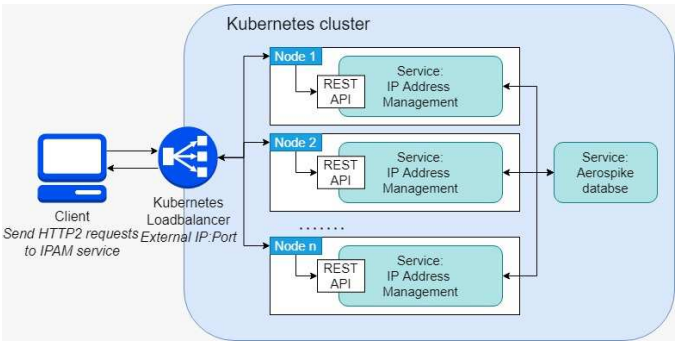


Fig. 4: Deployment map of the IP address allocation service on a Kubernetes cluster. Each IP address allocation application is deployed in a pod, with a single pod per node for quick scaling and high availability should any node fail. Client requests are load balanced via Kubernetes' LoadBalancer service, which presents a single REST API to the client. For the benchmark tests, to better evaluate performance, only one IP allocation instance was used.

A. First benchmarking test

1) Configurations

TABLE I. SERVER'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz
Limit number of CPUs	1
Limit memory used	2 GBs

TABLE II. CLIENT'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Number of CPUs	8
Number of concurrent worker	10
Tps	1200

2) Results

Fig. 5 compares resource usage in the cached implementation and trivial implementation of the IP address management system when benchmarking with 1200 TPS of IP allocation requests. As shown in Fig. 6, the new implementation has decreased the application's CPU usage by 25%, from 40% to 15%. Memory usage increases by a small amount.

Fig. 6 shows the latencies of the service when testing with 1200 TPS. The service latencies of the new implementation decreased, with the percentage of latencies under 10 ms being approximately 5% smaller compared to that of the trivial design. This difference is not really significant when testing with 1200 TPS.

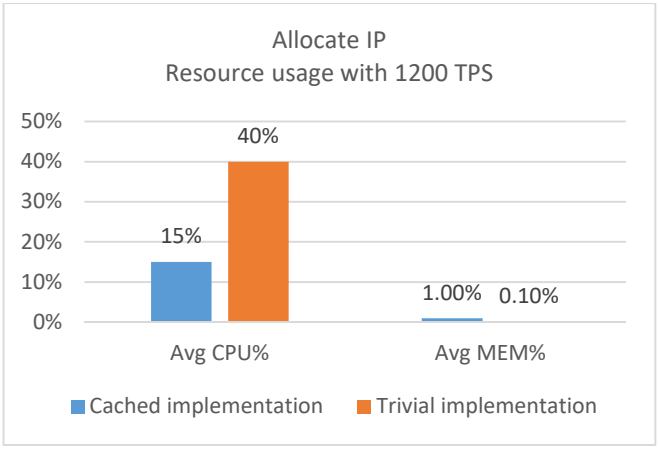


Fig. 5. Average CPU and memory usage during 1200 TPS benchmark test

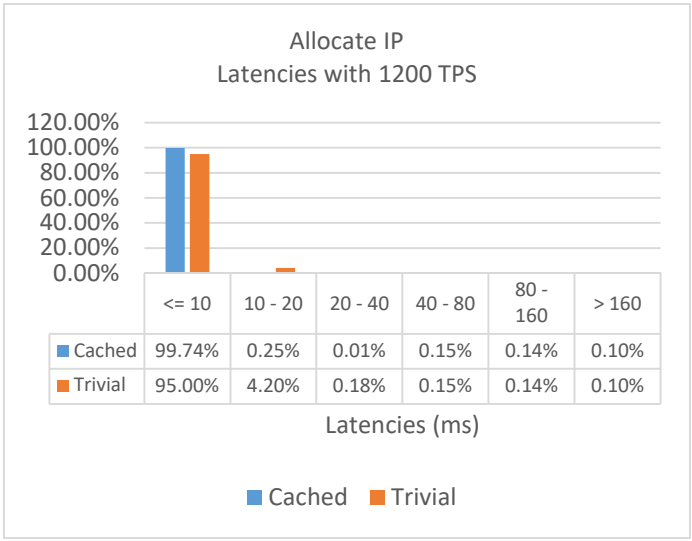


Fig. 6. Percentage of request latencies during 1200 TPS benchmark test

B. *Second benchmarking test*

1) *Configurations*

TABLE III. SERVER'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz
Limit number of CPUs	1
Limit memory used	2 GBs

TABLE IV. CLIENT'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Number of CPUs	8
Number of concurrent worker	10
Tps	3000

2) *Results*

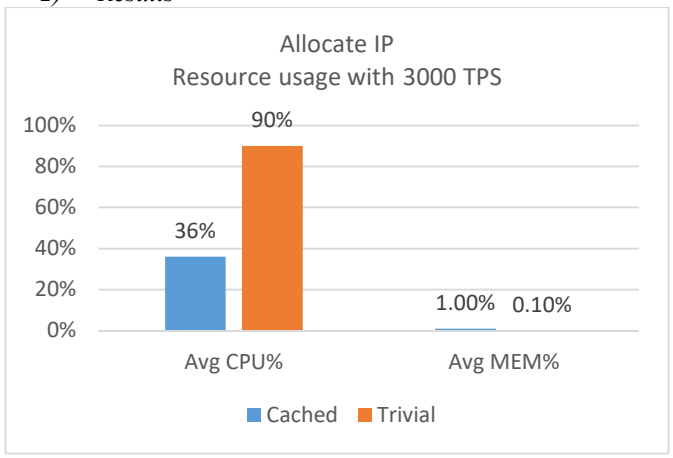


Fig. 7. Average CPU and memory usage during 3000 TPS benchmark test

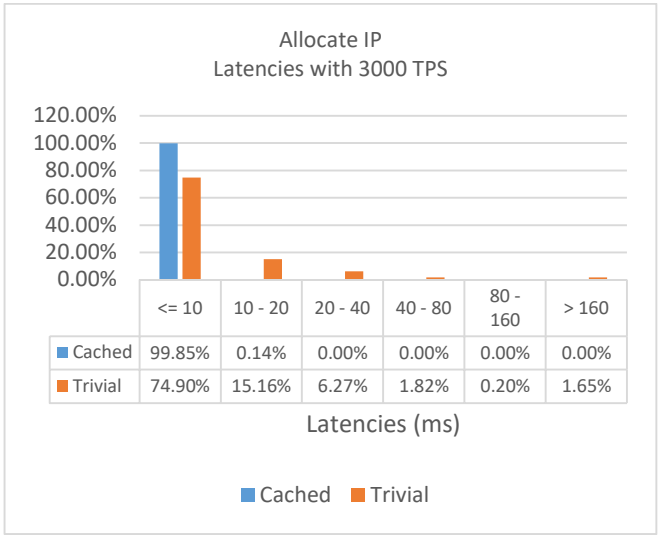


Fig. 8. Percentage of request latencies during 3000 TPS benchmark test

Fig. 7 compares the resource usage in the IP address management system with the new design to the IP address management system with the cached design when benchmarking with 3000 TPS of IP address allocation requests. As shown in figure 10, CPU usage is 36% for the cached design and 90% for the trivial design. Average memory usage in the new system is slightly higher than in the trivial system: 1% compared to 0.1%.

Fig. 8 shows the latencies at 3000 TPS. The percentage of requests that have latencies smaller than 10 ms in cached design remains nearly 100%. The percentage of requests that have latencies smaller than 10 ms in the trivial design reduces to nearly 75% when benchmarking at 3000 TPS. The percentage of latencies between 10 ms and 20 ms and between 20 ms and 40 ms is notable, at 15.16% and 6.27%, respectively.

C. *Third benchmarking test*

1) *Configurations*

TABLE VII. SERVER'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz
Limit number of CPUs	1
Limit memory used	2 GBs

TABLE VIII. CLIENT'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Number of CPUs	8
Number of concurrent worker	10
Tps	1200

2) Results

Fig. 9 compares resource usages in cached and trivial IP address management system implementations when benchmarking with 1200 TPS of IP de-allocation requests. As shown in Fig. 10, CPU usage in the cached system and the trivial system is 17% and 30%, respectively. Average memory usage in the cached system is slightly higher than in the trivial system: 0.5% compared to 0.1%.

Fig. 10 shows the latencies at 1200 TPS. No requests are shown to have a latency of more than 10 ms when using the cached design, compared to 2% with the trivial design.

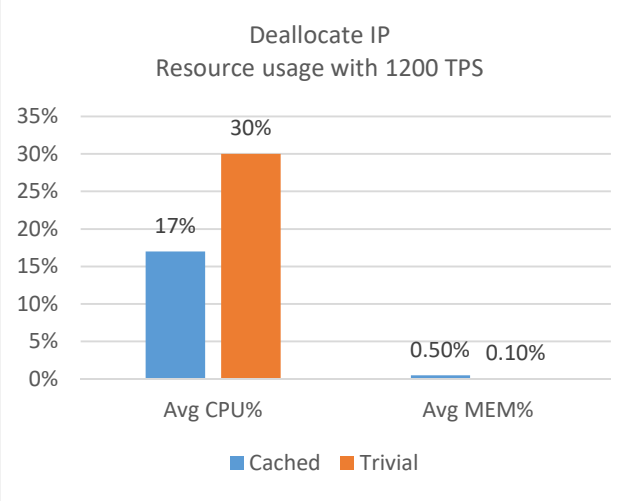


Fig. 9. Average CPU and memory usage during 1200 TPS benchmark test

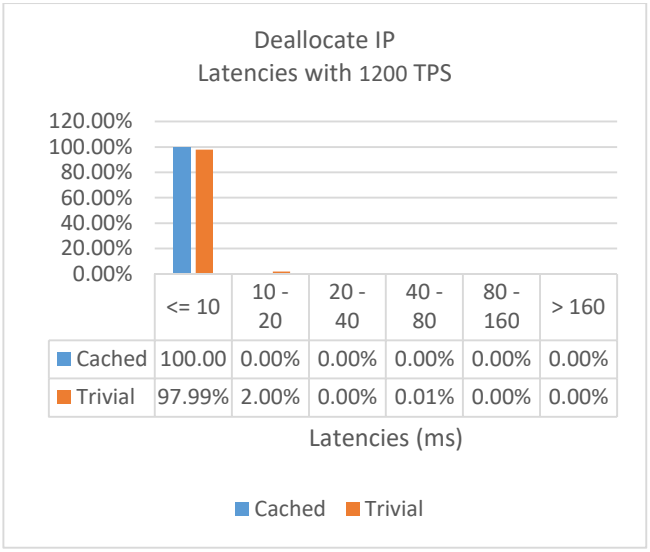


Fig. 10. Percentage of request latencies during 1200 TPS benchmark test

D. Fourth benchmarking test

1) Configurations

TABLE VII. SERVER'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz
Limit number of CPUs	1
Limit memory used	2 GBs

TABLE VIII. CLIENT'S CONFIGURATIONS

Parameter	Value
Type of CPU	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Number of CPUs	8
Number of concurrent worker	10
Tps	3000

2) Results

Fig. 11 compares resource usages in the cached implementation and trivial implementation of the IP address management system when benchmarking with 3000 TPS of IP de-allocation requests. As shown in Fig. 16, the average CPU usage in the new system is 35%, while that of the trivial system is 90%. Average memory usage in the new system is slightly higher than in the trivial system, 0.5% compared to 0.1%.

Fig. 12 shows the latencies with 3000 TPS of de-allocation requests. Similar to the 1200 TPS benchmark test, no requests were recorded to have a latency greater than 10 ms when using the cached design. The percentage of transactions with a latency of greater than 10 ms when using the trivial design is 14.98%.

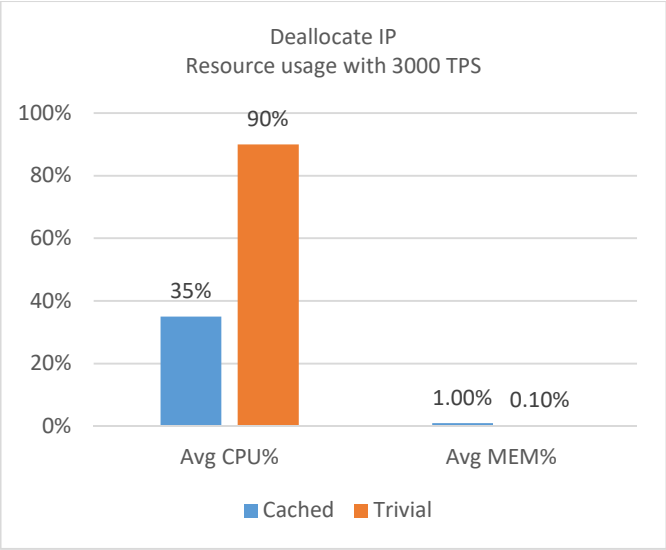
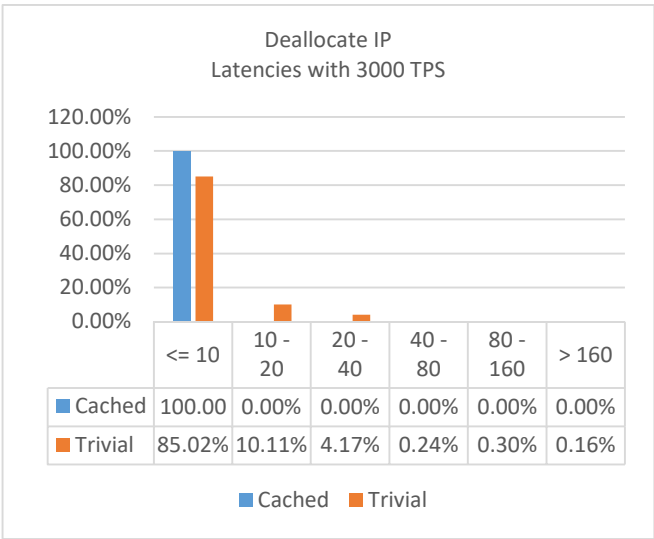


Fig. 11. Average CPU and memory usage during 3000 TPS benchmark test



	<= 10	10 - 20	20 - 40	40 - 80	80 - 160	> 160
Cached	100.00	0.00%	0.00%	0.00%	0.00%	0.00%
Trivial	85.02%	10.11%	4.17%	0.24%	0.30%	0.16%

Fig. 12. Percentage of request latencies during 3000 TPS benchmark test

Based on the six benchmarking test results above, the cached and containerized design of the IP address management system has been proven to greatly improve the system's performance. In terms of resource usage, the cached design has reduced CPU resource usage by approximately 40%. Regarding request latencies, the new design also decreases the amount of time needed by the IP address management system to respond to an incoming request. In the scope of the six benchmarking tests above, the cached and containerized IP address management system responded to 99% to 100% of the incoming requests in less than 10 milliseconds. The cached design uses more memory resources. However, this is a small amount and is considered acceptable.

V. Conclusion

In this paper, we have presented our proposed solution for the design of the IP address management service for the SMF, a core component of the 5G Core Network, and compared its performance with the current solution. Our experiments show that the proposed solution—a cached and containerized design—better utilizes the available resources compared to the trivial design. This new proposed system is being deployed and is undergoing a field trial in the real network environment of Viettel, one of the biggest mobile operators in Vietnam.

For future work, we plan to apply this system to large-scale networks and research further improvements to performance and consistency.

References

[1] 3GPP, *TS 23.501: System architecture for the 5G System (5GS)*, version 16.6.0 Release 16, October 2020.

[2] 3GPP, *TS 29.561: Interworking between 5G Network and external Data Networks*, version 16.4.0 Release 16, August 2020.

[3] K. Falls and W. Stevens, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.

[4] Y. Rekhter and T. Li, "An architecture for IP address allocation with CIDR", IETF RFC 1518, 1993.

[5] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. Jan de Groot, and E. Lear, "Address allocation for private Internets", RFC 1918, 1996.

[6] K, Egevang and P. Francis, "The IP network address translator (NAT)", RFC 1631, 1994.

[7] R. Droms, "Dynamic host configuration protocol", RFC 2131, 1997.

[8] T. Rooney, *IP Address Management: Principles and Practice*, John Wiley & Sons, 2011.

[9] C. Xie, J. Bi, H. Yu, C. Li, C. Sun, Q. Liu, Z. Zheng, and S. Liu. "ARPIM: IP address resource pooling and intelligent management system for broadband IP networks", *IEEE Communications Magazine,* vol. 55, no. 6, pp. 55-61, 2017

[10] K. Duran, B. Karanlik, and B. Canberk. "Graph theoretical approach for automated IP lifecycle management in telco networks", *International Journal of Network Management,* vol. 31, no. 4, pp. 18, 2021

[11] C. Perkins, "IP mobility support for IPv4", RFC 3344, 2002.

[12] L.Chen, "IP address allocation method", U.S. Patent 12 087 286, 2 Jan., 2007

[13] Google, Cloud Native Computing Foundation, *Kubernetes*, 2014.

[14] Aerospike, *Aerospike,* 2010