µ vIMS: una arquitectura más escalable basada en Microservicios

Juan S. Orduz, Gabriel D. Orozco, Carlos H. Tobar-Arteaga y Oscar Mauricio Caicedo Rendon
Grupo de Ingeniería Telemática, Universidad del Cauca
Correo electrónico: juanorduz, gabrielorozco20, carlost, omcaicedo @ unicauca.edu.co

Resumen: los pasos hacia todo IP se han definido para IP Multimedia Subsystem (IMS) como la tecnología de facto para aprovisionamiento de servicios multimedia de extremo a extremo en 5G. sin embargo, el El crecimiento impredecible de usuarios en 5G requiere mejorar IMS escalabilidad para manejar el tráfico dinámico de usuarios. Varias obras tienen abordó este problema mediante la introducción de mecanismos de autoescalado en arquitecturas IMS virtualizadas (vIMS). Sin embargo, el vIMS actual las implementaciones usan diseños monolíticos que no permiten escalabilidad En este artículo, presentamos µvIMS, una arquitectura que utiliza microservicios para proporcionar una escalabilidad más fina y más efectiva uso de recursos que el diseño monolítico regular. Para probar nuestro arquitectura, evaluamos el prototipo µvIMS con respecto al uso de la CPU, Uso de RAM, tasa de llamadas exitosas (SCR) y métricas de latencia. Los resultados de nuestras pruebas revelan que µvIMS logra un SCR más alto, utilizando los recursos disponibles de manera efectiva con una latencia insignificante creciente. Por lo tanto, podemos afirmar que dividiendo los vIMS monolíticos La arquitectura en microservicios permite proporcionar una escalabilidad más fina.

I. Introducción

El subsistema multimedia IP (IMS) es una arquitectura de telecomunicaciones destinado a proporcionar servicios de voz y multimedia IP menos redes de acceso [1]. Algunas características relevantes de IMS son apertura, interoperabilidad y soporte de tecnología de tendencias gies, como VoLTE, VoWiFi y WebRTC [2], [3]. Consid- ering estas características, el ETSI (European Telecommunications Instituto de Normas) ha decidido continuar usando IMS en el Redes de 5ta generación (5G) para aprovisionamiento multimedia servicios [4]. Como en 5G el tráfico de usuarios será dinámico, IMS debe adaptarse a dicho dinamismo [5]. IMS actual las implementaciones tienen problemas de escalabilidad porque su arquitectura los componentes principales generalmente se implementan en dispositivos de red o monolíticamente Estos aparatos dificultan la escala las capacidades de arquitectura efectivamente porque Network Func- Las opciones (NF) están vinculadas a un hardware específico. A su vez, un monolítico IMS implica la asignación de recursos a un grupo de Funciones de red virtualizadas (VNF), evitando lograr una escalabilidad más fina (es decir, escalar una función específica para manejar tráfico de red particular).

Varios trabajos utilizan la virtualización de funciones de red (NFV) [6], [7] para tratar el problema de escalabilidad de IMS obteniendo virtual- IMS ized (vIMS) y apoyando su operación con adicional sistemas [5], [8]. En [5], los autores usan un marco que monitorea la calidad ofrecida por un vIMS y la escalan con respecto a las condiciones de la red (por ejemplo, tiempos de ejecución del servicio y ancho de banda disponible). En [8], los autores agregan un autoescalado mecanismo que considera varias métricas de vIMS (por ejemplo, recurso uso, número de solicitudes y latencia), lo que permite escalar dependiendo del tráfico de usuarios. Sin embargo, los autores usan un mono- arquitectura lítica que evita una escalabilidad más fina y falla al usar recursos disponibles de manera eficiente.

Para tratar con arquitecturas monolíticas vIMS, en el último años, se ha utilizado el concepto de microservicios [9], [10]. Los microservicios son un modelo arquitectónico que divide un aplicación monolítica en diferentes componentes, cada uno de ellos con funcionalidad específica [11]. Ya que estos com los componentes son más pequeños que la aplicación monolítica, es más fácil agregar y eliminar instancias de microservicios [12], lo que permite para lograr una escalabilidad más fina. En [9], los autores proporcionan IMS procesos de registro, autorización y autenticación como servicios que utilizan microservicios. En [10], los autores presentan un vIMS basado en microservicios para computación en la nube, en el que Los componentes principales de IMS están dentro de un único microservicio. Su arquitectura basada en microservicios utiliza un orquestador y un equilibrador de carga para escalado horizontal automático (es decir, adición y lanzamiento de instancias de microservicios para mejorar la arquitectura capacidades). En resumen, a lo mejor de nuestro conocimiento, el Soluciones vIMS basadas en microservicios propuestas en la literatura considerar un número limitado de procesos IMS o no dividir IMS funciona eficientemente, lo que deja de seguir al microservicio diseño, evitando lograr una escalabilidad más fina.

En este artículo, presentamos ivIMS, un microservicio basado Arquitectura vIMS que tiene como objetivo proporcionar una escalabilidad más fina en Redes 5G. Por microservicios, nuestra arquitectura ofrece mejores escalabilidad, asignando recursos solo en el IMS necesario funcionalidades En ivIMS, descomponemos el núcleo IMS en microservicios y elementos de uso de MicroService Archi- tecture (MSA) para garantizar la seguridad, fiabilidad y capacidad de gestión ity. Implementamos un prototipo ivIMS usando Clearwater y Kubernetes Analizamos el rendimiento del prototipo ivIMS comparándolo con un vIMS. En tal comparación, evaluamos la capacidad de ivIMS para manejar un mayor número de usuarios con la misma cantidad de recursos, midiendo el éxito Tasa de llamadas (SCR) para un aumento de llamadas por segundo (CPS) número que simula tráfico impredecible. La evaluación los resultados revelan que ivIMS alcanzó un SCR más alto usando un similar cantidad de recursos, y con un aumento de latencia aceptable ing, mostrando la viabilidad de proporcionar una escalabilidad más fina y asignar recursos de manera efectiva.

Las principales contribuciones presentadas en este documento son:

- Un diseño de vIMS basado en microservicios para enfrentar el escalado de IMS problemas de habilidad.
- Un prototipo vIMS basado en microservicios que sigue el

978-1-7281-2561-9 / 19 / $ 31.00 © 2019 IEEE

Página 2

Diseño descrito.

- La demostración de que nuestra arquitectura es factible para Proporcionar una escalabilidad más fina para IMS y atender a más usuarios con los recursos disponibles

Este artículo está organizado de la siguiente forma: La sección II describe trabajo relacionado. La sección III presenta un escenario de motivación. Segundo-La sección IV describe la arquitectura ivIMS. La sección V presenta El prototipo ivIMS y su comparación con vIMS. Finalmente, La Sección VI proporciona conclusiones e implicaciones para el futuro. trabajos.

II R elacionadas W ork

Varios trabajos han abordado los problemas de escalabilidad de IMS teniendo en cuenta diferentes tecnologías y métodos. En el trabajo [8], los autores presentan una comparación entre dos auto-mecanismos de escala para vIMS. El primero está basado en VM (Máquina virtual) métricas (uso de CPU y RAM). El segundo El mecanismo se basa en métricas de VNF, como recursos y uso de red, conexiones TCP, número de caídas de llamadas y Solicitudes de protocolo de inicio de sesión (SIP). Como una comparación resultado, los autores determinan que el uso de métricas VNF es la mejor forma de escalar vIMS. En [5], los autores pro plantear una auditoría de calidad y corretaje de recursos que cumpla con NFV marco de referencia. Este marco permite escalar IMS de acuerdo con tráfico de usuarios y asignación de recursos en diferentes nubes plataformas Los autores analizan su propuesta desplegando un vIMS integrado con su marco en un banco de pruebas sobre un nube híbrida privada / pública real. Las propuestas anteriores comparten la misma deficiencia; tienen arquitectura monolítica IMS diseños, que no permiten una escalabilidad más fina e implica una asignación de recursos menos específica que conduce a la pérdida de recursos en funciones de escala no relacionadas con el tráfico.

En la literatura, pocos trabajos usan patrones de diseño de microservicios en arquitectura IMS. En [9], los autores proponen un enfoque para lograr un diseño VNF óptimo utilizando microservicios. Con este enfoque, los autores introducen IMS-as-a-Service que proporciona Registro, autorización y autenticación de IMS como servicio. Los autores probaron IMS-as-a-Service comparándolo con un microservicio vIMS. Como resultado, IMS-as-a-Service alcanzó un mayor número de llamadas exitosas y mejor rendimiento en utilización de recursos. Los autores se centran en el registro de IMS, procesos de autorización y autenticación, y no considere funcionalidades centrales completas de IMS. El trabajo [10] describe una arquitectura para la implementación elástica de vIMS basado en microservicios para computación en la nube. La AU-thors asigna la información del servidor de abonado doméstico (HSS) en bases de datos de microservicios. Además, comparan su im-plementación con un vIMS sin microservicios con respecto llame al retraso del establecimiento y concluya que su arquitectura mantiene resultados similares al IMS regular. Sin embargo, los autores. implementado las funciones de control de sesión de llamada (CSCF) en Un solo microservicio, evitando una escalabilidad más fina. Por lo tanto, no es posible asignar recursos en las funciones específicas que manejar el tráfico

En resumen, para mejorar el uso de los recursos, vIMS la arquitectura necesita una escalabilidad más fina y recursos específicos ubicación. Nuestro ivIMS aborda este problema distribuyendo el Principales funcionalidades de IMS en microservicios de manera eficiente. Esta la distribución permite escalar solo las funciones necesarias para para proporcionar servicios multimedia IP a un mayor número de usuarios con los recursos disponibles

III. M otivación

En redes 5G, IMS proporciona IP de voz y multimedia servicios con CSCF [4]. Estos CSCF son: Proxy-CSCF (P-CSCF), Serving-CSCF (S-CSCF), Interrogating-CSCF (I-CSCF). P-CSCF es el primer punto de contacto entre usuarios e IMS, que valida las solicitudes y las dirige a su destino. S-CSCF gestiona la sesión multimedia, registra usuarios y reenvía solicitudes al elemento IMS correcto. YO-CSCF verifica el perfil de los usuarios, asigna al usuario a un S-CSCF, y enruta solicitudes a otras redes IMS.

Ahora, consideremos el siguiente escenario: un operador de telecomunicaciones la empresa tiene un IMS que brindó servicio de voz en una región por muchos años. Después de analizar otros operadores de telecomunicaciones, el la empresa concluye que deben proporcionar servicios 5G (por ejemplo, telemedicina e Internet de las cosas) para seguir siendo competitivos. La implementación de estos servicios es exitosa, pero después un tiempo, los clientes del operador de telecomunicaciones se quejan de 5G Los servicios fallan continuamente. La empresa analiza el problema. y descubre que su IMS intenta sin éxito manejar el tráfico con picos impredecibles introducidos por estos servicios 5G. Por lo tanto, el operador de telecomunicaciones necesita mejorar sus capacidades de IMS.

En el escenario anterior, el operador de telecomunicaciones tiene un IMS que no puede manejar el tráfico impredecible porque no se escala eficientemente. Una solución para este escenario puede ser virtualizar su IMS y utiliza un sistema adicional para la escalabilidad automática [5], [8]. Sin embargo, esta solución utiliza un archivo monolítico de IMS tecture que implica una asignación de recursos menos específica cuando Está escalado. El operador de telecomunicaciones puede usar microservicios para tratar con esta arquitectura monolítica, pero los enfoques actuales [9], [10] no implementan las funciones de IMS necesarias para control de sesión multimedia o no proporciona recursos específicos asignación. En ivIMS, dividimos los CSCF en siete inde-microservicios pendientes que proporcionan la sesión multimedia completa controlar. Esta división permite escalar los microservicios con el funciones que manejan el tráfico entrante, utilizando mejor los recursos disponibles para manejar más usuarios.

IV. A i v IMS arquitectura

ivIMS es una arquitectura basada en microservicios y dispositivos firmado para mejorar la escalabilidad de IMS en 5G. Oferta de microservicios la capacidad de asignar recursos en las funciones necesarias para manejar el tráfico, lo que lleva a una escalabilidad más fina y reduce innecesariamente uso de recursos ensayarios. La Figura 1 muestra el ivIMS general vista que incluye dos tipos de componentes: CSCF Microser-Vice Cluster y elementos de mejora. Microservicio CSCF Cluster realiza el CSCF con microservicios. Realzando Los elementos mantienen el correcto funcionamiento de estos microservicios. realización de registro de microservicios, gestión de microservicios, e interoperabilidad con elementos externos (por ejemplo, redes 5G, Entidades de usuario, servidor de aplicaciones y HSS).

Página 3

Elementos de mejora

Infra-

tipo de registros de usuarios. Este microservicio necesita una base de datos

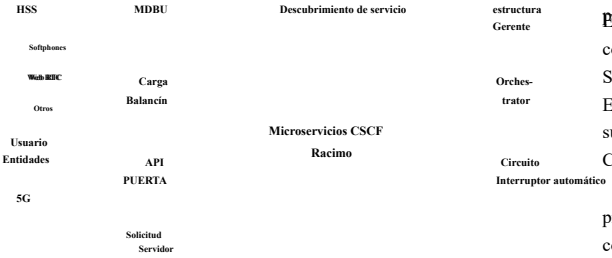


Fig. 1: Arquitectura general de ivIMS

En las siguientes subsecciones, describimos la división funcional. de los CSCF en microservicios. Después, presentamos el Mejora de elementos. Usamos los elementos de mejora que encontrado en la literatura [13], [14], [15], y proponemos nuevos unos para proporcionar interoperabilidad y seguridad.

A. Clúster de microservicios CSCF

Este clúster utiliza microservicios para construir los CSCF responsable del control de sesión multimedia de usuarios y aplicaciones Servidores de cationes (AS). Estos microservicios ofrecen a ivIMS el siguientes características. Primero, la división funcional CSCF Mantiene un tamaño adecuado. Tenga en cuenta que los microservicios de gran tamaño agrupa muchas funciones, volviendo al diseño monolítico. Pequeña- microservicios de tamaño desacoplan funciones dependientes pesadas de entre sí, lo que resulta en una complejidad de gestión adicional [9], [16]. Segundo, la división funcional CSCF mantiene datos independencia porque cada microservicio tiene su propia base de datos, una base de datos centralizada implica perder la independencia [10].

La Figura 2 describe la descomposición de CSCF en microservicios En esta división, brindamos en cada microservicio Una funcionalidad completa e independiente basada en CSCF especificado por el Proyecto de asociación de tercera generación (3GPP) [4]. Dividimos el P-CSCF en dos microservicios: reenvío Mensajes SIP y garantizar políticas de acceso. SIP de reenvío El microservicio de mensajes es responsable de enrutar el mensaje SIP. sabios generados por elementos externos (por ejemplo, teléfono tradicional, softphone y WebRTC), asegurando que los mensajes SIP tener el formato correcto, así como detectar y enrutar solicitudes de sesiones de emergencia. Las políticas de garantizar el acceso microservice gestiona las políticas del operador en el acceso a la arquitectura. Este microservicio almacena las políticas del operador en su base de datos y los proporciona para reenviar microservicios de mensajes SIP a Asegure el enrutamiento de mensajes SIP de acuerdo con las políticas del operador.

Dividimos el S-CSCF en cuatro microservicios: User Registry, garantizar políticas de sesión multimedia, sesión multimedia Control de siones y gestión del servicio de prioridad multimedia IMS (MPS) El microservicio de registro de usuarios es responsable de autorizando el registro de usuarios en la arquitectura, agrupando Funciones de registro I / S-CSCF. Además, traduce el Dirección E.164 (es decir, número único global para cada dispositivo en la red telefónica pública conmutada) requerido para algunos para almacenar información del usuario. La sesión Multimedia de Asegurar El microservicio de políticas administra las políticas del operador en multimedia control de sesión. Este microservicio certifica la identidad del suscriptor si Se configura mediante políticas de operador. Garantizar multimedia El microservicio de políticas de sesión almacena las políticas del operador en su base de datos y proporciona estas políticas a la sesión multimedia Control de microservicios.

El microservicio de Control de sesión multimedia proporciona el principales funcionalidades del S-CSCF relacionadas con la multimedia control de sesión. Estas funcionalidades incluyen inicio, mantenimiento nance y finalice las sesiones multimedia. Agrupamos el multimedia control de sesión de usuario con usuario y usuario con AS en un solo microservicio porque dependen mucho de cada uno otro. Por ejemplo, una llamada entre dos usuarios podría incluir AS interacción para redirigir, grabar o bloquear mensajes de voz usuarios no deseados El uso de dos microservicios para cada tipo de sesión multimedia implica tráfico de red adicional y lógica para comunicarlos y proporcionar control de sesión. Con el propósito de que cada microservicio proporcione funcionalidad, el microservicio de Control de sesión multimedia maneja cualquier tipo de sesión. Además, este microservicio tiene su propia base de datos para manejar la información del usuario.

Gestionar el microservicio IMS MPS maneja IMS MPS por Proporcionar tratamiento de acceso preferencial a usuarios prioritarios cuando la congestión en la red está bloqueando el establecimiento de la sesión. En En este sentido, este microservicio valida si un usuario está autorizado para servicio prioritario por AS, incluye nivel de prioridad en el usuario solicitar y reenviarlo. El microservicio Manage CDRs gestiona registros detallados de llamadas (CDR), agrupando P / I / S-CSCF lógica asociada con CDR. Este microservicio recuerda CDR de otros microservicios y genera CDR estandarizados que se almacena en su base de datos y se envía a una facturación externa entidad.

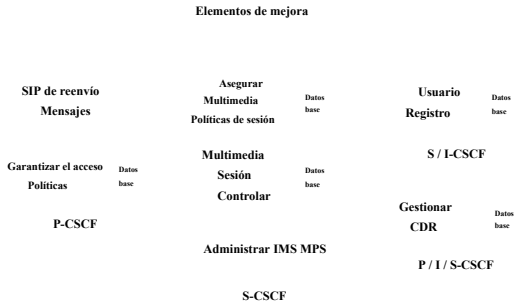


Fig. 2: Grupo de microservicios CSCF

B. Mejora de elementos

La Figura 3 presenta los elementos de mejora que proporcionan seguridad, fiabilidad y gestión de microservicios, a saber, Service Discovery, Orquestador, Gerente de Infraestructura, Circuit Breaker, API Gateway, Microservice DataBase Updater

Página 4

(MDBU) y Load Balancer. El servicio de descubrimiento es un elemento de registro que admite comunicación de microservicios. Incluye el registro y el descubrimiento y el autenticador ponentes El componente de registro y descubrimiento almacena y provide las direcciones de microservicios. Estas direcciones son URI que señalar una dirección IP necesaria para el reenvío de tráfico a los microservicios. Garantizar la seguridad en microservicios. Provisión de direcciones, proponemos un nuevo Servicio de Descubrimiento componente llamado Autenticador. Este nuevo componente verifica que el elemento que solicita una dirección de microservicio es confiable.

indica al orquestador qué instancias de microservicios Reiniciar. Si la falla persiste, el disyuntor concluye que la falla no se pudo manejar simplemente reiniciando instancia de microservicio. Por lo tanto, este componente indica a la puerta de enlace API para detener el tráfico externo e informa al administrador de red sobre la falla del microservicio.

API Gateway, Load Balacer y MDBU son interoperables Elementos que permiten una comunicación segura de microservicios con elementos externos como entidades de usuario, 5G y otros Redes. API Gateway es un elemento que impide

Autenticador podría implementarse con varias opciones. por ejemplo, un sistema básico de verificación de clave-contraseña.

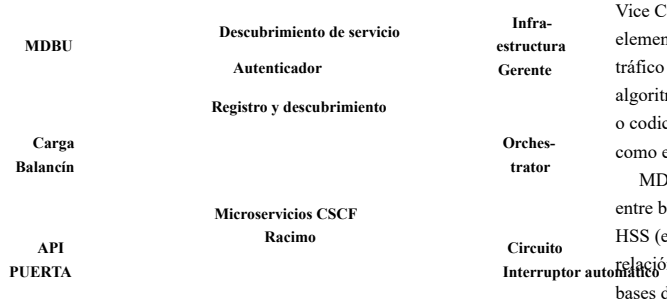


Fig. 3: Mejora de elementos

Los elementos de gestión son Orquestador, Infraestructura. Gerente y disyuntor. Orquestador e Infraestructura Manager se adaptan las funcionalidades de tres entidades de NFV-MANO [17]: Administrador de Infraestructura Virtualizada (VIM), VNF Manager (VNFM) y NFV Orchestrator (NFVO). O-chestrator adapta las funcionalidades de VNFM y NFVO. En este sentido, Orchestrator gestiona la vida de los microservicios. ciclo independiente de la infraestructura mediante la replicación, la migración, iniciar, pausar y eliminar instancias de microservicios. En En este sentido, el administrador de red puede gestionar fácilmente croservicios usando Orchestrator sin infraestructura manual Interacción.

El administrador de infraestructura adapta las funcionalidades de VIM. Por lo tanto, interactúa directamente con los recursos de infraestructura (es decir, computación, almacenamiento y redes) de cada máquina donde implementamos el clúster de microservicio CSCF. Infraestructura El gerente recibe órdenes de gestión del orquestador y los realiza en el clúster de microservicios CSCF. Por lo tanto, la El administrador de infraestructura decide dónde asignar un microser-vicios según los recursos disponibles. Además, cuando una máquina falla, Infrastructure Manager implementa microservicios que se ejecutó dentro de otra máquina disponible.

El disyuntor proporciona confiabilidad al CSCF Microser-vice Cluster manejando las fallas de microservicios. Cuando se produce un fallo de microservicio, el disyuntor lo registra. Después del número de fallas registradas por el disyuntor excede un umbral configurado por la administración de red tor, el disyuntor determina que es necesario reiniciar la instancia de microservicio de falla. Entonces, el disyuntor acceso no autorizado al CSCF Microservice Cluster. Cuando API Gateway garantiza un acceso confiable con una autenticación mecanismo, distribuye el tráfico entre CSCF Microser-Vice Cluster y arquitectura exterior [18]. Load Balancer es un elemento que mejora las capacidades de arquitectura al distribuir tráfico entre instancias de microservicio. Load Balancer utiliza un algoritmo para distribuir tráfico, como aleatorizado, round-robin o codicioso El tráfico distribuido proviene de elementos externos. como entidades de usuario o elementos internos como otros microservicios.

MDBU es un nuevo elemento de mejora que comparte información entre bases de datos de microservicios y las actualiza con el HSS (es decir, el repositorio principal para almacenar una información de recuperación relación de usuarios). HSS no se puede dividir en microservicios bases de datos porque 5G y otras redes lo usan. Con este elemento, nuestra arquitectura no afecta a otras redes por modificando el HSS. La MDBU realiza la sincronización HSS por dos pasos: (i) monitorea el HSS buscando cambios, (ii) actualiza las bases de datos de microservicios cuando el HSS es actualizado.

V. E valoración

Para evaluar nuestra arquitectura, primero, implementamos un ivIMS prototipo. En segundo lugar, creamos un entorno de prueba para comparar Nuestra arquitectura con vIMS sin microservicios. En tercer lugar, nosotros realizó pruebas con respecto a SCR, uso de recursos y latencia.

A. Prototipo

Utilizamos Clearwater para implementar nuestro microservicio CSCF Cluster modificando su arquitectura de acuerdo a nuestro diseño (las modificaciones se explican en el siguiente párrafo). Claro-el agua [19] es un núcleo IMS de código abierto ampliamente utilizado diseñado para entornos de nube que siguen el estándar IMS estandarizado interfaces Clearwater tiene dos tipos de despliegue, uno sobre Máquinas virtuales y otras sobre contenedores Docker. Clearwater-over-VM el despliegue sigue un enfoque monolítico, donde un solo el componente asigna varios componentes Clearwater-over-Docker nents. La tabla I muestra los componentes de ambas implementaciones y sus funcionalidades. En el despliegue de Clearwater-over-Docker-Además, cada componente se implementa en un solo contenedor. Además, estos componentes son similares a los microservicios de CSCF Cluster de microservicios. Por lo tanto, adaptamos Clearwater-over-Docker componentes usándolos o dividiéndolos para obtener ivIMS prototipo.

Implementamos el microservicio de reenvío de mensajes SIP us-ing Bono. A su vez, usamos Ralf para implementar Administrar CDR microservicio Clearwater-over-Docker usa Sprout, Cassandra

Página 5

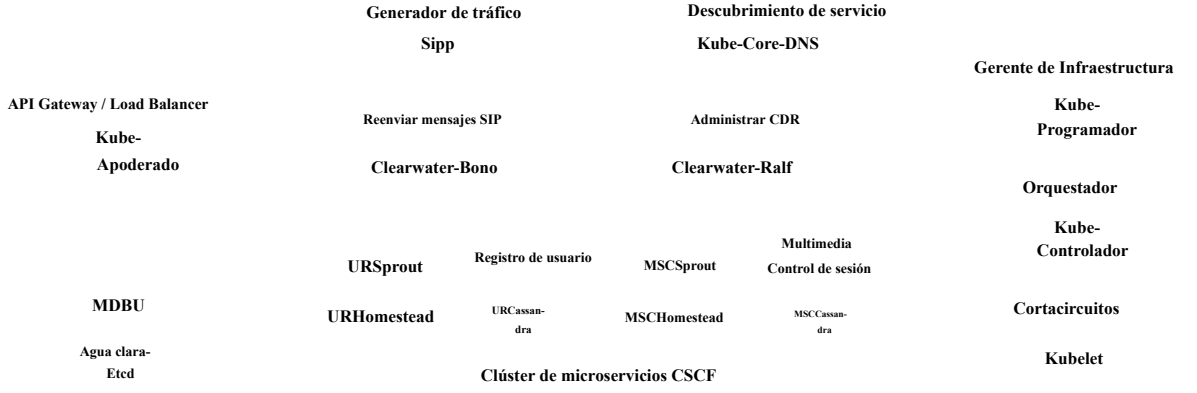


Fig. 4: prototipo ivIMS

CUADRO I: división Clearwater

Clearwater-over-VM	Clearwater-over-Docker	Funcionalidad
Ellis	Ellis	Portal de suministro basado en la web para el registro de usuarios, la administración de contraseñas y el control de la configuración del servicio.
Bono	Bono	Proxy de borde SIP que realiza funcionalidades P-CSCF.
Brote	Brote	Registro SIP y proxy de enrutamiento de autorización que realizan las funcionalidades I-CSCF y S-CSCF.
Homero	Homero	Servidor de gestión de documentos XML (XDMS) que almacena la configuración de los servicios del usuario.
	Ralf	API HTTP para recordar eventos facturables de Bono y Sprout.
Moneda de diez centavos	Granja	Interfaz para proporcionar comunicación entre Sprout y Cassandra.
	Homestead-prov	Interfaz para proporcionar comunicación entre Ellis y Cassandra.
	Astaire	Servicio de Memcached para almacenar estados de registro / sesión de usuarios.
Vitela	Cassandra	Base de datos para almacenar y proporcionar información de usuario a Sprout a través de Homestead.
	Chronos	Servicio de temporizador para permitir la gestión de eventos de larga duración de Sprout.
	Etcd	Servicio de valor clave distribuido para compartir información entre dos o más elementos del clúster.

base de datos y Homestead para el control de sesiones multimedia y registro de usuarios. Para mantener nuestra independencia entre el usuario Microservicios de Registro y Control de Sesiones Multimedia, nosotros divide los componentes Sprout, Homestead y Cassandra. En en particular, implementamos microservicios de registro de usuarios con UR-Sprout, URHomestead y URCassandra. Además, utilizamos MSC-Sprout, MSCHomestead y MSCCassandra para implementar Microservicio de control de sesión multimedia. Es esencial para destacar que URCassandra y MSCCassandra sincronizan su información porque MSCSprout gestiona multimedia sesión de usuarios registrados por URSprout en URCassandra. A abordar esta sincronización, implementamos MDBU con Etcd para compartir información de los usuarios, formando un grupo de Cassandras.

Usamos Kubernetes [20] para implementar Elementos destinados a proporcionar seguridad, confiabilidad y microser-Vicegerencia (Figura 4). Kubernetes es un ampliamente utilizado orquestador de contenedores de código abierto que gestiona nuestro CSCF Cluster de microservicios. Además, Kubernetes trabaja sobre uno o más máquinas que dan forma a un grupo de Kubernetes. El ku El grupo de Bernetes consta de un maestro y varios trabajadores, donde el maestro está a cargo de la gestión de recursos (CPU y RAM) proporcionados por los trabajadores para implementar microservicios. Para la implementación de Service Discovery, utilizamos un DNS servidor llamado Core-DNS que realiza Registro y Descubrimiento funcionalidad Implementamos funcionalidades de orquestador con Componente controlador Kube, que garantiza que los microservicios sean atropellar a un trabajador. Luego, usamos Kube-Scheduler componente para implementar funcionalidades de Infrastructure Manager vínculos, este elemento rastrea los recursos y las asignaciones de los trabajadores disponibles Cates microservicios en ellos. Implementamos el disyuntor desplegando Kubelet sobre cada trabajador. Asegura microservicios salud comprobando su estado. Finalmente, usamos Kube-proxy como API Gateway y Load Balancer, Kube-proxy está en cada trabajador que proporciona acceso externo y distribuye el microser-Vice carga de trabajo.

B. Entorno de prueba

En esta sección, presentamos el entorno de prueba utilizado para evaluar el prototipo ivIMS. Lo construimos sobre el Centro de datos de la Universidad del Cauca llamado Telco 2.0. Nosotros asignamos la misma cantidad de máquinas implementadas sobre VMWare 6.0 con la misma cantidad de recursos para dos implementaciones: el ivIMS prototipo, y el vIMS. La Tabla II muestra ambas implementaciones asignación de recursos y sus roles en cada implementación. La Figura 5 presenta el despliegue de ivIMS que consistió en siete máquinas virtuales (VM) que dan forma a un clúster de Kubernetes. Este grupo está compuesto por un maestro de Kubernetes y seis Trabajadores de Kubernetes, y una octava VM para generar tráfico. Sobre los seis trabajadores de Kubernetes, desplegamos el CSCF Cluster de microservicios. El maestro Kubernetes decide en qué

Página 6

TABLA II: Pruebe los recursos del entorno

Máquina	Recursos	vIMS	µvIMS
Máquina1		Ellis	Kubernetes Maestro
Máquina 2	4 CPU Intel (R) Xeon (R)	Bono	
Máquina 3	E5-2670 2.30 GHz y	Brote	Trabajadores Kubernetes (Microservicio CSCF Racimo)
Máquina 4	16 GB de RAM	Homero	
Máquina 5		Moneda de diez centavos	
Máquina 6		Vitela	
Máquina 7	4 CPU Intel (R) Xeon (R) E5-2670 2.30 GHz y 6 GB de RAM	Bind9	
Máquina 8	4 CPU Intel (R) Xeon (R) E5-2670 2.30 GHz y 16 GB de RAM		SIPp

El trabajador de Kubernetes asigna cada microservicio.

La Figura 6 presenta la implementación de vIMS que incluye seis VM. Estas máquinas virtuales implementan cada una de Clearwater-over-VM componentes: Ellis, Bono, Sprout, Dime, Homer y Vellum. Estos componentes necesitan un DNS para comunicarse entre ellos. Por lo tanto, lo implementamos en una séptima VM usando Bind9 [21]. Finalmente, vIMS tenía ocho máquinas virtuales para generar tráfico. En ambos implementaciones, las ocho máquinas virtuales utilizaron el generador de tráfico SIPp [22] como herramienta de prueba.

establecimiento y llamada terminaron entre ellos. Este escenario es repetido para varios pares de usuarios al mismo tiempo, dependiendo en las llamadas por segundo (CPS) definidas para probar las implementaciones. Aumentamos el CPS de 25 a 200 en pasos de 25 CPS. Cada la prueba duró 60 segundos, y la repetimos 32 veces para promediar Los resultados. Emulamos este proceso en ambas implementaciones y las métricas que utilizamos para compararlas son SCR, recurso uso (es decir, CPU y memoria de las siete máquinas virtuales que conformaron los despliegues) y la latencia.

C. Resultados y análisis

Probamos la arquitectura de escalabilidad más fina comparando ivIMS con vIMS. Comenzamos desde una implementación de ivIMS con Una sola instancia para cada componente. Entonces, escalamos el componentes que consumen más recursos (es decir, Bono, URSprout, MSCSprout) para determinar el componente en el que asignar fuentes para lograr un mejor rendimiento. Luego, para determinar el siguientes combinaciones, agregamos nuevas instancias a la combinación que logró el mejor rendimiento. Otras combinaciones fueron no incluido en este análisis porque el rendimiento no mejorar. La Tabla III muestra el número de instancias de componentes desplegado en cada combinación. Corrimos el escenario de prueba para cada combinación y despliegue de vIMS, y medimos SCR. Después, medimos el recurso de implementación ivIMS uso (CPU y RAM) de cada combinación y compararla con vIMS. También realizamos una evaluación de latencia de ivIMS

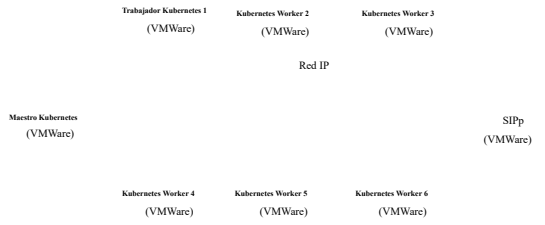


Fig.5: Clearwater modificado sobre Kubernetes (ivIMS)

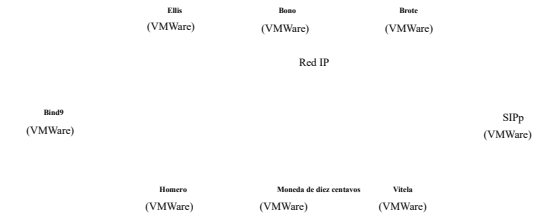


Fig.6: Clearwater sobre VM (vIMS)

Para comparar ambas implementaciones, definimos un escenario de prueba usando SIPp con tres fases: registro de dos usuarios, llamada

combinaciones de implementación y vIMS, teniendo en cuenta la latencia umbral para la señalización de IMS (es decir, 100 ms) [23]. Apuntamos para encontrar una combinación que logre un mejor SCR sin un aumento excesivo en el consumo de recursos y la latencia, corroborando la viabilidad de ivIMS finner-escalabilidad.

TABLA III: combinaciones de prototipos ivIMS

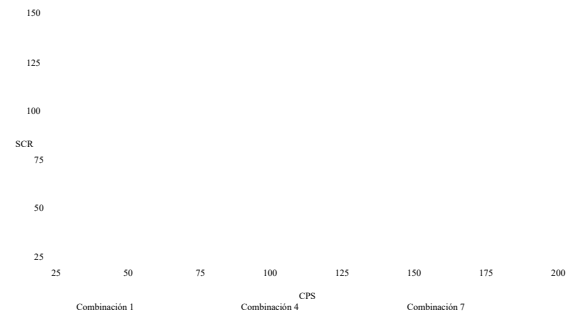
Despliegue	Bono (Instancias)	URSprout (Instancias)	MSCSprout (Instancias)
Combinación 1	1	1	1
Combinación 2	2	1	1
Combinación 3	1	2	1
Combinación 4	1	1	2
Combinación 5	3	1	1
Combinación 6	2	2	1
Combinación 7	2	1	2

La Figura 7 muestra los resultados de la evaluación SCR de vIMS y Las siete combinaciones de ivIMS. Estos resultados revelan diferentes hechos. Primero, sin componentes de escala (Combinación 1), ivIMS presenta un SCR ligeramente mejor que vIMS. Segundo, con otra instancia de Bono (Combinación 2), los resultados de SCR son mejores que la combinación 1. Así como, con la combinación 2, ivIMS alcanza un mayor número de SCR que vIMS. Tercero, Agregar otra instancia de URSprout (Combinación 3) u otra Instancia MSCSprout (Combinación 4), los resultados son peores que vIMS. Según esos resultados, la combinación 2 tiene mejor SCR. Por lo tanto, en las siguientes combinaciones, agregamos otros instancias componentes de la combinación 2. Cuarto, con tres Instancias Bono (Combinación 5), los resultados son peores que Combinación 2, pero son mejores que vIMS. Quinto, con

Página 7

dos instancias de Bono y dos URSprout (combinación 6), los resultados son mejores que vIMS y similares a la combinación 2. Finalmente, con dos instancias de Bono y dos de MSCSprout (Combinación 7), los resultados empeoran que la Combinación 2 y 6, y en CPS superiores, son similares a vIMS. Concluimos que la combinación 2 y 6 tienen un SCR más alto que vIMS y el Otras combinaciones de ivIMS. Esto corrobora la viabilidad de ivIMS para atender más llamadas con éxito con los disponibles recursos (es decir, la misma cantidad de máquinas con el mismo recursos), usándolos de manera efectiva.

Es importante resaltar que la combinación de arquitectura muestra un comportamiento estable antes de alcanzar un SCR máximo. Después de que una combinación alcanza su máximo, las curvas se vuelven saturado, ya que las capacidades de ivIMS para esta combinación son excedidos Además, los resultados de SCR muestran que cada El rendimiento de bination está relacionado con las ventajas de un nuevo instancia y las desventajas de administrar esta nueva instancia. Por ejemplo, los resultados de la combinación 3 y 4 muestran ideas que agregar MSCSprout y URSprout a la Combinación 1 sí no mejora el rendimiento porque se agrega tráfico de control y la arquitectura no necesita más instancias de esas componentes pero otra instancia de Bono. Por esa razón, esos combinaciones presentan SCR más bajo que vIMS.



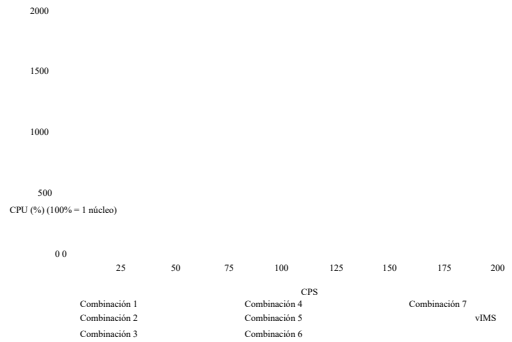
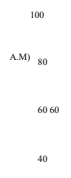


Fig.8: CPS vs CPU

4 que tienen una instancia adicional, usan más RAM que Combinación 1. Esto se debe a que cada réplica necesita más RAM para el rendimiento general. Otro aspecto importante es que el uso de RAM no aumenta significativamente con el cantidad de CPS en la misma combinación, lo que significa que RAM No es importante atender a más usuarios al mismo tiempo. En Además, la implementación regular de vIMS usa menos RAM que ivIMS; Esto se debe a los elementos de mejora. los Los resultados generales del uso de recursos revelan que la mejora Los elementos necesitan mucha RAM pero poca CPU para funcionar. Finalmente, en el caso de la arquitectura a escala, la adición de nuevas instancias aumenta notablemente el uso de RAM pero no el uso de la CPU.



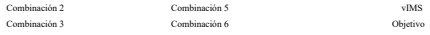


Fig.7: CPS vs SCR

La Figura 8 presenta los resultados de evaluación de uso de CPU de la siete combinaciones de ivIMS y vIMS. Estos resultados revelan que la diferencia entre combinaciones escaladas de nuestros ivIMS El despliegue es insignificante. El uso de CPU de vIMS para un pequeño El número de CPS es menor que ivIMS. Sin embargo, la evaluación muestra que para una gran cantidad de CPS, el uso de CPU de vIMS aumentar a un nivel similar a ivIMS. Además, el SCR y los resultados de la CPU muestran que un mayor número de CPS implica mayor uso de CPU, incluso si el SCR no mejora. Esto es porque las implementaciones intentan procesar más solicitudes pero son No puedo hacerlo con los recursos disponibles. Por consiguiente, la solicitud llega a un tiempo de espera. En cuanto a los resultados de SCR y uso de CPU, la combinación 2 y 6 logran un SCR más alto sin uso excesivo de la CPU.

La Figura 9 muestra los resultados de RAM. Estos resultados revelan que la adición de una nueva instancia de componente implica un crecimiento en el uso de RAM. Por ejemplo, combinaciones 2, 3 y

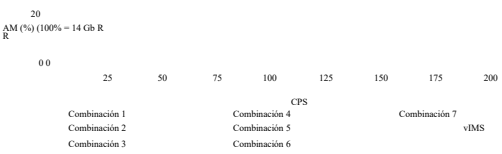


Fig. 9: CPS vs RAM

Finalmente, la Figura 10 presenta los resultados de la evaluación de latencia utilizando una función de distribución acumulativa (CDF) que agrupa la latencia obtenida de 25 CPS a 200 CPS para vIMS y cada combinación de ivIMS. La combinación 5, 6 y 7 presente una degradación de latencia visible, y superan los 100 ms umbral de latencia Las combinaciones 1, 2, 3, 4 y vIMS tienen una diferencia de latencia insignificante y mantienen bajo El umbral de latencia. Por lo tanto, con respecto a los resultados generales, nosotros concluir que la Combinación 2 presenta una mejora SCR visible Mente con los recursos disponibles sin afectar la latencia como combinación 6. En consecuencia, concluimos que ivIMS alcanza un SCR más alto que vIMS sin superar la latencia

Página 8

umbral para IMS.

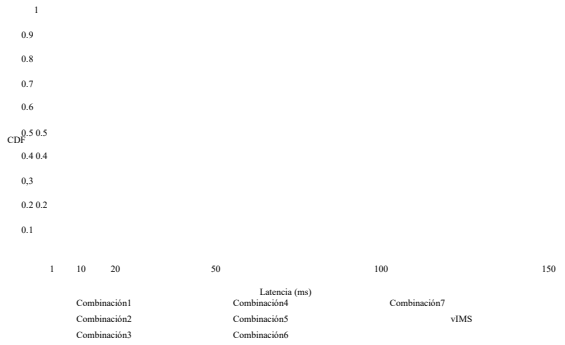


Fig. 10: Función de distribución acumulativa de latencia

Finalmente, haciendo una comparación cualitativa, podemos establecer el siguientes aseveraciones. Primero, nuestro diseño de arquitectura utiliza microservicios, diferentes a vIMS, cuya arquitectura es monolítico. En segundo lugar, ivIMS permite asignar recursos en el microservicios relacionados con el tráfico, logrando una escalabilidad más fina. Tercero, con los recursos disponibles, ivIMS logra un mayor SCR que vIMS, lo que demuestra la eficiencia del uso de recursos. Finalmente, nuestra arquitectura utiliza los recursos de manera eficiente sin exceso pasar el umbral de latencia para la señalización de IMS (es decir, 100 ms).

VI. C onclusiones y F uturo W ork

En este artículo, presentamos ivIMS, una arquitectura dirigida para proporcionar IMS escalabilidad más fina en redes 5G que sufren Variaciones dinámicas de tráfico. Para manejar este dinamismo, ivIMS descompone el CSCF en microservicios y utiliza Mejorando elementos, mejorando el uso de recursos disponibles y lograr una escalabilidad más fina. Las principales contribuciones son: a Diseño de IMS basado en microservicio llamado ivIMS, un prototipo de ivIMS tipo para la provisión de CSCF y comparación de rendimiento entre ivIMS y vIMS usando la misma cantidad de máquinas virtuales con la misma cantidad de recursos. Esta comparación revela que nuestro ivIMS alcanzó un SCR más alto que vIMS, con similar uso de recursos, y sin latencia excesiva aumentando. Nosotros corroboró que el uso de microservicios ivIMS proporciona un mejor escalabilidad porque los recursos se asignan en el específico

R eferencias

[1] Kang Wang, Guiqing Gao, Yuanli Qin y Xiangyong He. Construcción de sistema de comunicación para la eliminación de emergencias por accidentes nucleares basado en el subsistema multimedia ip. En las actas de la conferencia AIP. AIP, 2018.
[2] INTEL. vims para proveedores de servicios de comunicaciones. Intel Builders, 2016
[3] VG Ozianyi y N. Ventura. Diseño e implementación de escalables ims sistemas de carga. En conferencia sobre redes informáticas locales. IEEE, 2009.
[4] ETSI. Sistema digital de telecomunicaciones celulares (fase 2+) (gsm); sistema universal de telecomunicaciones móviles (umts); lte; ip multimedia subsistema (ims); Etapa 2. ETSI TS 123 228 V15.2.0, 2018.
[5] G. Carella, L. Foschini, A. Pernafini, P. Bellavista, A. Corradi, M. Corici, F. Schreiner y T. Magedanz. Auditoría de calidad y corretaje de recursos para Orquestación de virtualización de funciones de red (nfv) en nubes híbridas. En Conferencia de Comunicaciones Globales, páginas 1–6. IEEE, 2015.
[6] J. Lai y Q. Fu. Anycast Man-in-the-middle (mima): usuario de CDN la asignación del servidor se vuelve flexible. En conferencia sobre computadora local Redes. IEEE, 2016.
[7] CHT Arteaga, F. Rissoi y OMC Rendon. Una escala adaptativa mecanismo para gestionar las variaciones de rendimiento en las funciones de red virtualización: un estudio de caso en una epc basada en nfv. En internacional Conferencia sobre gestión de redes y servicios, páginas 1–7, 2017.
[8] AB Alvi, T. Masood y U. Mehboob. Escala automática basada en carga en subsistema multimedia virtual basado en ip. En comunicaciones al consumidor Conferencia sobre redes, páginas 665–670. IEEE, 2017.
[9] A. Boubendir, E. Bertin y N. Simoni. Un diseño vnf-as-a-service a través de microservicios desmontando los ims. En conferencia sobre Innovaciones en nubes, Internet y redes, páginas 203–210. IEEE 2017
[10] Pascal Potvin, Mahdy Nabaee, Fabrice Labeau, Kim Khoa Nguyen y Mohamed Cheriet. Micro servicio patrón de computación en la nube para el próximo redes de generacion. Repositorio de investigación informática, 2015.
[11] D. Gallipeau y S. Kudrle. Microservicios: bloques de construcción para Nuevos flujos de trabajo y virtualización. SMPTE Motion Imaging Journal, 127 (4): 21–31, mayo de 2018.
[12] F. Moradi, C. Flinta, A. Johnsson y C. Meirosu. Conmon: un sistema automatizado de monitoreo de rendimiento de red basado en contenedores. En Simposio sobre gestión integrada de redes y servicios, páginas 54–62. IFIP / IEEE, 2017.
[13] X. Larrucea, I. Santamaría, R. Colomo-Palacios y C. Ebert. Microservicios Software IEEE, 35 (3): 96–100, mayo de 2018.
[14] H. Knoche y W. Hasselbring. Uso de microservicios para software heredado modernización. Software IEEE, 35 (3): 44-49, 2018.
[15] Fabrizio Montesi y Janine Weber. Disyuntores, descubrimiento y API gateways en microservicios. CDR, 2016.
[16] S. Klock, JMEMVD Werf, JP Guelen y S. Jansen. Carga de trabajo agrupación basada en conjuntos de características coherentes en arquitecturas de microservicios. En Conferencia Internacional sobre Arquitectura de Software. IEEE, 2017.
[17] ETSI. Virtualización de funciones de red (nfv); gestión y orquesta- Tration. ETSI GS NFV-MAN 001 V1.1.1, 2014.
[18] D. Taibi y V. Lenarduzzi. Sobre la definición de microservicios malos olores. Software IEEE, 35 (3): 56–62, 2018.
[19] Clearwater. http://www.projectclearwater.org/.
[20] Kubernetes. https://kubernetes.io/.
[21] Bind9. https://www.isc.org/downloads/bind/.

Función IMS necesaria para manejar el tráfico.

Como trabajos futuros, estamos interesados en analizar el rendimiento ing de ivIMS con un número variable de trabajadores de Kubernetes, y el impacto del tráfico adicional introducido por Enhancing Elementos. Además, planeamos agregar una escalabilidad automática mecanismo que permite que ivIMS se adapte al tráfico de usuarios y analizar el rendimiento, la latencia y el uso de recursos.

Un reconocimiento

Los autores desean agradecer al equipo de trabajo de la Universidad del Cauca Telco 2.0 por su apoyo esencial para Esta investigación. Las implementaciones presentadas en este documento Todos se implementaron en el centro de datos Telco 2.0.

[22] Sipp. http://sipp.sourceforge.net/
[23] M. Taqi Raza, S. Lu, M. Gerla y X. Li. Refactorizando funciones de red módulos para reducir latencias y mejorar la tolerancia a fallas en nfv. IEEE Revista sobre áreas seleccionadas en comunicaciones, páginas 2275–2287, 2018.