UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE POLITEHNICA BUCUREŞTI

Şcoala Doctorală de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Decizie nr. 206 **din** 21-09-2024

REZUMAT TEZĂ DE DOCTORAT

Ing. Oana-Mihaela Ungureanu (Dumitru-Guzu)

METODE DE VIRTUALIZARE ŞI GESTIONARE A CONTAINERELOR ÎN MODELELE ARHITECTURALE BAZATE PE MICROSERVICII

PATTERNS OF VIRTUALIZATION AND CONTAINER MANAGEMENT IN MICROSERVICES-BASED ARCHITECTURE MODELS

COMISIA DE DOCTORAT

Prof. Dr. Ing. Ion MARGHESCU U.N.S.T Politehnica din Bucureşti	Preşedinte
Prof. Dr. Ing. Călin VLĂDEANU U.N.S.T Politehnica din Bucureşti	Conducător de doctorat
Prof. Dr. Ing. Romulus-Mircea TEREBEŞ Univ. Tehnică din Cluj-Napoca	Referent
Conf. Dr. Ing. Cristian-Iulian RÎNCU Academia Tehnică Militară Ferdinand I din Bucureşti	Referent
Conf. Dr. Ing. Şerban Georgică OBREJA U.N.S.T Politehnica din Bucureşti	Referent

BUCUREŞTI 2024

Cuprins

Listă de tabele **iv**

Listă de figuri **v**

1 Introducere **1**

1.1 Prezentarea domeniului tezei de doctorat 1

1.2 Scopul tezei de doctorat 1

1.3 Conţinutul tezei de doctorat 1

2 Tehnologii si sisteme folosite în arhitectura noii generații de core mobil 5G **4**

2.1 Arhitectura autonomă (SA) 4

2.2 Introducerea conceptelor de NFV și SDN 5

2.3 Scurtă taxonomie a soluțiilor existente ce folosesc SDN si NFV 5

2.4 Perspectiva de cloud nativ 6

3 Utilizarea unei abordări native în cloud pentru proiectarea funcțiilor de rețea în generațiile viitoare de core mobil 5G **8**

3.1 Descrierea soluției propuse 8

4 Analiza potențialelor vulnerabilități în implementarea noii generații de mobile core 5G **11**

4.1 Obiectivul soluției propuse 11

5 Comunicarea Cloud-Edge: Un model de orchestrare bazat pe API declarativ pentru Noua Generație 5G Core **13**

5.1 Descrierea contextului cloud-edge 13

5.2 Orchestratoare de containere pentru edge computing 13

5.3 Modelul de ClusterAPI 14

5.4 Configurare experimentală - implementarea nucleului de următoare generație 5G în clustere Kubernetes 14

6 O abordare inovativă pentru scalarea segmentării rețelei 5G Core rulate într-un cadru de cloud nativ pe mai multe site-uri **18**

6.1 Comparație între cadrul ETSI MANO și ochestratorul Kubernetes . . . 18
6.2 Scalarea orizontală vs. verticală în cloud 19
6.3 Scalarea în implementările Kubernetes multi-cluster și multi-cloud . . . 19
6.4 Descrierea setup-ului de testare și a celor două scenarii de peering . . . 20

7 Optimizarea clusterelor de tip Kubernetes utilizând metoda hibridă de planificare a stărilor partaje 23
7.1 Lucrări conexe și o scurtă taxonomie a mecanismelor de planificare existente 23
7.2 Probleme identificate în planificatorul Kubernetes 23
7.3 Descrierea metodei de planificare propuse: planificator hibrid cu stări partajate 24
7.4 Logica Funcției de Corecție (SC) 25
7.5 Analiza metodei de planificare cu stări partajate în comparație cu alte soluții de planificare existente 26

8 Concluzii 28
8.1 Rezultate obţinute 28
8.2 Contribuţii originale 29
8.3 Lista lucrărilor originale 29
8.4 Perspective de dezvoltare ulterioară 30

Bibliografie 31

Listă de tabele

2.1 Proiecte open-source pentru virtualizarea nucleului mobil de 4G and 5G 6

7.1 Taxonomia diferitelor cadre de planificare existente în literatura actuală 24
7.2 Comparația caracteristicilor în planificatoarele integrate de Kubernetes . 26

Listă de figuri

3.1 Scenariul propus instalat într-un cloud public 9
3.2 Graful de "service-mesh" cu distribuția și procentele de trafic 10
3.3 Throughputul de interogare și răspuns al funcției AMF măsurat pe interfața de ieșire 10

4.1 Throughputul de răspuns al funcției AMF în cele 2 scenarii de atac cu 5Greplay 12

5.1 Setup-ul folosit pentru 5G Core SBA implementat cu orchestrator Kubernetes la periferia rețelei (Figura 5.4 din teză) 15
5.2 Comparație între latența implementării Open5GS folosind CAPI vs K3s (Figura 5.5 din teză) 17

6.1 Cadrul propus pentru segmentarea rețelei în nucleul mobil de 5G nativ în cloud 19
6.2 Throughputul de interogare pentru funcția AMF măsurat la sursă pentru planul de control descărcat către clusterul de la distanță (Figura 6.13 din Teză) 20
6.3 Throughputul de răspuns pentru funcția AMF măsurat la destinație pentru planul de control descărcat către clusterul de la distanță (Figura 6.14 din Teză) 21
6.4 Throughputul de interogare pentru funcția SMF măsurat la destinație pentru planul utilizatorului descărcat către clusterul de la distanță (Figura 6.15 din Teză) 21
6.5 Throughputul de interogare pentru funcția SMF măsurat la destinație pentru planul de control descărcat către clusterul de la distanță (Figura 6.16 din Teză) 22
6.6 Latența în planul utilizatorului pentru scenariul de peering in-band (Figura 6.19 din Teză) 22

7.1 Arhitectura propusă de planificare hibridă cu stări partajate 25

Capitolul 1

Introducere

1.1 Prezentarea domeniului tezei de doctorat

Prezenta lucrare "Metode de virtualizare și gestionare a containerelor în modelele arhitecturale bazate pe microservicii" abordează un subiect de actualitate - integrarea tehnologiilor cloud native în actuala generație de 5G core mobil precum și în generațiile viitoare cu scopul de a îmbunătăți scalabilitatea rețelei, de a reduce costurile operaționale și de a aduce programabilitatea la ficare nivel al rețelei.

1.2 Scopul tezei de doctorat

În urma extinderii pieței în continuă creștere pentru diferite industrii, urmatoarea generație de nucleu mobil 5G este pe punctul de a evolua către o arhitectura bazată pe servicii. Prin urmare, această lucrare prezintă o abordare inovativă a rulării funcțiilor de rețea 5G pentru noua generație ca și aplicații native în cloud fiind scalată cu ușurință la cerere pentru a se alinia la constrângerile diferitelor sectoare.

1.3 Conţinutul tezei de doctorat

În Capitolul 1 este prezentat contextul de cloud computing și tehnologiile native de cloud precum containere, orchestratori, etc. în comparație cu noțiunile de virtualizare. De asemenea pentru a satisface cerințele actualei generații de 5G mobile core din punct de vedere al infrastructurii ce trebuie scalată pentru a agrega și procesa cantități masive de date ce vin din diferite industrii, tehnologiile native de cloud oferă multiple beneficii din punct de vedere al costului, scalabilității pe diferite site-uri și capacității de abstractizare la nivelul rețelei.

Capitolul 2 face o introducere în noua arhitectură 5G și descrie conceptele de NFV și SDN precum și architectura MANO. Este prezentată o taxanomie a proiectelor open-source pentru virtualizarea nucleului mobil de 4G și 5G care implementează virtualizarea.

În plus, se face o analiză a principalelor soluții de orchestrare a containerelor și o introducere în arhitectura 5G, evidențiind principalele diferențe dintre arhitectura non-autonomă 5G NSA (non-standalone) și cea de tip autonom SA (standalone). În continuare, este prezentată arhitectura orchestratorului Kubernetes ce face obiectul aceste lucrări precum și capabilitațile de provizionare în medii multi-cloud și multi-cluster.

Capitolul 3 adresează implementarea funcționalității de tip 5G SA într-un mediu virtualizat ce rulează într-un cloud public cu interfeţe programabile bazate pe servicii si orchestrat de Kubernetes. De asemenea, pentru a valida comunicarea între microservicii s-a folosit o soluție de "service mesh". Configurația propusă are scopul de a valida funcționalitatea 5G SA prin integrarea a două dintre simulatoarele open-source: Open5GS și UERANSIM folosit ca si emulator RAN. În plus, a fost analizat comportamentul în cazul mai multor solicitări concomitente de stabilire a sesiunii PDU.

În Capitolul 4 sunt prezentate riscurile de securitate pe principale interfețe AMF si SMF prin injectarea traficului cu un tool numit 5Greplay. Mai departe sunt abordate doua scenarii de atac pentru care este măsurat throughputul de răspuns al funcției AMF.

Capitolul 5 prezintă un model declarativ bazat pe API-uri care permite implementarea multi-cloud, totodată delegând logica de control la periferia rețelei. Modelul bazat pe implementarea ClusterAPI permite folosirea unor distribuții mai ușoare de Kubernetes, prin urmare sunt analizate două cele două integrări K3s și Kind cu ClusterAPI. De asemenea, au fost evaluate diferite modele de orchestrare a containerelor concepute pentru edge computing. Printr-o soluție de service mesh, numită Linkerd este comparată latența generată de cele două tool-uri în throughputul de răspuns al funcțiilor Open5GS.

Capitolul 6 abordează capabilitățile multi-site de 5G native în cloud solicitând capacitatea rețelei la cerere, precum și programabilitatea rețelei și configurația nucleului mobil prin valorificarea beneficiilor API-urilor. Mai departe, sunt analizate două scenarii de comunicare între site-uri denumite în continuare "in-band" și "out-of-band" peering din punct de vedere al descărcării planului de utilizator și de control către clusterul adiacent. De asemenea, se face o comparație între cadrul MANO și noul model propus de gestionare a funcțiilor containerizate care are la bază orchestrarea Kubernetes. Mai mult, sunt validate trei scenarii de segmentare a rețelei și măsurat traficul end-to-end pentru sesiuni concomitente ale utilizatorilor ce acceseaza cele trei slice-uri.

În Capitolul 7 se analizează un model hibrid de cadru de planificare bazat pe Kubernetes cu stări partajate care deleagă majoritatea sarcinilor agenților de planificare distribuiți și are o funcție de corecție a programării care prelucrează în principal sarcinile neprogramate și neprioritate. De asemenea, sunt prezentate principalele tipuri de planificatoare existente în literarura ce au la bază implentarea Kubernetes și sunt analizate capabilitățile noului planfifocator propus în comparație cu trei cadre de planificare Kubernetes.

Capitolul 8 sintetizează rezultatele experimentale obținute în cadrul cercetării împreună cu contribuțiile originale și trasează direcțiile viitoare de dezvoltare în vederea

aprofundării acestui domeniu de cloud computing și posibile aplicații în noile arhitecturi mobile.

Capitolul 2

Tehnologii si sisteme folosite în arhitectura noii generații de core mobil 5G

Acest capitol are ca obiectiv introducerea principalelor tehnologii folosite in arhitectura noii generații de core mobil 5G precum și diferențele intre arhitectura arhitectura non-autonomă (NSA) și arhitectura autonomă (SA).

2.1 Arhitectura autonomă (SA)

Arhitectura non-autonomă (NSA) permite compatibilitatea intre vechiul nucleul 4G și noul nucleu mobil 5G, iar arhitectura de sine stătătoare - standalone (SA) are ca scop acomodarea noilor servicii care provin din diferite sectoare si industrii (vehicule cu conducere autonomă, robotică de precizie, IoT etc.) și să ofere comunicații ultra-fiabile cu latență scăzută (URLCC). De asemenea, in acest capitol este explicată funcționalitatea principalelor funcții de retea: Funcția de management al accesului și al mobilității - Access and Mobility Management Function (**AMF**) asigură autentificarea, autorizarea și mobilitatea intre user și rețea, Funcția de management al sesiunii - Session Management Function (**SMF**) asigură autentificarea, autorizarea și mobilitatea intre user și rețea. Funcția de aplicație - Application Function (**AF**) oferă informații despre resurse, iar Funcția de plan al utilizatorului - User Plane Function **UPF**) are rol de rutare și transfer al pachetelor, menține sesiunea de transfer **PDU** (Protocol Data Unit) intre user și rețea.

Funcția de server de autentificare - Authentication Server Function (**AUSF**) - oferă posibilitea funcției AMF de a autentifica userul. Funcția Managementul unificat al datelor - Unified Data Management (**UDM**) este responsabilă cu stocarea datelor despre subscripția userului. Funcția depozit de date unificat - Unified Data Repository (**UDR**) stochează în principal datele de abonament și profilurile clienților, Funcția de selecție a segmentelor de rețea - Network Slice Selection Function (**NSSF**) menține o listă cu

segmentarea instanțelor de rețea definite de operator. Funcția de expunere în rețea - Network Exposure Function - expune servicii și resurse via API intern sau extern rețelei 5G. Funcția de depozit de rețea - Network Repository Function (**NRF**) menține o listă cu funcțiile de rețea disponibile și profilele asociate. Funcția de control - Control Function (**PCF**) - este responsabilă cu asigurarea politicilor și îndeplinirea calității serviciilor (QoS) pe baza subscripției.

O alta componentă introdusă in arhitectura SA este componenta MEC (Multi-access Edge Computing) ce se află în apropierea stațiilor de bază si are scopul de a minimiza timpul de răspuns. Ideea principală în arhitectura MEC este să asigure o latență scăzută, lărgime de bandă mai mare precum și putere de procesare masivă la periferia rețelelor mobile aproape de terminale.

2.2 Introducerea conceptelor de NFV și SDN

Grupul ETSI NFV este în procesul de a standardiza interfața de orchestrare cu funcțiile de management și orchestrare NFV, denumită și Management și orchestrare- Management and Orchestration (**MANO**).

Cadrul ETSI NFV Management and Orchestration (MANO) este format din trei blocuri funcționale: Manager de infrastructură virtualizată - Virtualized Infrastructure Manager (**VIM**), Orchestrator NFV - NFV Orchestrator (**NFVO**) și Manager VNF - VNF Manager (**VNFM**). Platforma vCloud NFV include un VIM integrat, care expune interfețe către nord la VNFM și NFVO.

2.3 Scurtă taxonomie a soluțiilor existente ce folosesc SDN si NFV

Literarura de specialitate este împărțită în principal între diferite soluții pentru a orchestra NFV-uri într-o arhitectură de cloud prezentată în lucrările [1], [2] și virtualizarea funcțiilor dezvoltate pentru nucleul mobil 4G și 5G. Tabelul 2.1 rezumă proiectele OSS existente dedicate în principal care vizează virtualizarea nucleului de pachete mobile conform specificațiilor 4G și prezintă puține inițiative pentru dezvoltarea de NF-uri pentru arhitectura 5G SBA.

Printre proiectele care suportă atât VNF-uri 4G, cât și 5G precum și CNF, sunt OpeNess [11] și Open5GS [7] pe care il vom lua în considerare în continuare în configurația noastră. Software-ul Open Network Edge Services (OpenNESS) este un instrument pentru a simula arhitectura MEC cu scopul de a furniza CNF-uri. Acest proiect a fost dezvoltat în colaborare cu Intel și rulează în întregime pe o arhitectură bazată pe microservicii ce oferă API-uri pentru comunitatea open-source.

Tabel 2.1 Proiecte open-source pentru virtualizarea nucleului mobil de 4G and 5G

Proiect open-source	Limbaj	Tip de licența	4G	5G	CNF	Contributori
OpenAirInterface [3]	C	Apache v2.0	da	da	wip*	OpenAir Software Alliance EURO-COM
NextEPC [4]	C	GNU AGPLv3	da	wip*	wip *	NextEPC
corenet [5]	Python	GPL-2.0 License	da	nu	nu	Corenet
openLTE [6]	C++	GNU AGPLv3	da	nu	nu	openLTE
open5GS [7]	C	GNU AGPLv3	da	da	da	Open5GS
OMEC [8]	C++	Apache v2.0	da	da	nu	ONF, Intel, Deutsche Telekom, Sprint, AT&T
free5GC [9]	Go, C	Apache v2.0	nu	da	wip*	Free5C
srsLTE [10]	C++	GNU AGPLv3	da	nu	nu	srsLTE
OpenNESS [11]	Go	Apache-2.0	da	da	da	Intel

* încă în lucru.

Open5GS [7] este un proiect open-source dezvoltat în limbajul C care implementează funcțiile de rețea conform standardului 3GPP Release 16 (AMF, SMF, PCF, UDM, AUSF, NRF, NSSF, UDR) și UPF) [12]. De asemenea, oferă o interfață grafică WebUI dezvoltat în Node.JS și React. Unele dintre aceste funcții au echivalentul lor din 4G Evolved Packet Core (EPC): AMF, SMF, PCF, UDM și AUSF. Accesul de bază și Funcția de management al mobilității (AMF) este responsabilă accesul utilizatorului și autorizarea la rețea, interacționează cu alte NF (adică SMF, AUSF) pentru a gestiona mobilitatea UE și corespunde Entității de management al mobilității (MME) în 4G.

2.4 Perspectiva de cloud nativ

În zilele noastre, Kubernetes a devenit un orchestrator popular de containere pentru a ajuta operatorii de telefonie mobilă cat și clienții să gestioneze serviciile 5G într-un cadru containerizat, indiferent de infrastructura lor.

O cale de migrare reușită dintr-o moștenire arhitecturală monolitică către una nativă de cloud nu adresează numai decuplarea hardware a VNF-urilor, dar și un design modular prin intermediul API-urilor și o nouă arhitectură distribuită bazată pe automatizare și autogestionare. Din moment ce cadrul MANO menționat mai sus nu abordează managementul CNF-urilor, un candidat de drept pentru orchestrarea containerelor este Kubernetes [13] pentru a ajuta MNO-urile să se modernizeze infrastructura lor costisitoare și să o transforme într-un sistem modular conceput pentru integrări multiple cu un Sistem de suport Operațional - Operational Support System (OSS) pentru o performanță și reziliență mai bune.

Un serviciu (sau micro-serviciu) este compus dintr-unul sau mai multe pod-uri și politici. Un job poate rula mai multe operații pentru a crea unul sau mai multe poduri. La nivelul planului de control există un API server responsabil cu actualizarea stării podurilor, un controler manager care monitorizează starea clusterului, un scheduler care dictează căror noduri (mașini virtuale) le sunt alocate podurile, un agent care rulează pe fiecare nod și un proxy responsabil politicile de trafic.

Principalele avantaje pentru rularea nucleului containerizat de următoarea generație mobilă în clustere multiple de Kubernetes implementate in diferite platforme de cloud sunt **izolarea completă** și **scalabilitatea**. De exemplu, o aplicație URLLC poate rula în segmentul de rețea „slice 1”, aplicația eMBB în „slice 2” și aplicația mMTC în „slice 3”. Pentru a urma regulile de conformitate a dezvoltării (adică, DevOps), este posibil sa separam mediile de dezvoltare și de organizare prin rulare acestor segmentul de rețea în diferite clustere izolate de producție.

De asemenea, sunt explicate conceptele de **multi-cloud** și **multi-regiune**. Hostarea multi-regiune acoperă problemele legate de disponibilitate și failover, precum și latența sau localizarea geografică în cazul protecției datelor și conformitatea GDPR, in timp ce hostarea multi-cloud asigură capabilități de recuperare în caz de dezastru și evită blocarea cu un anumit furnizor de cloud.

Capitolul 3

Utilizarea unei abordări native în cloud pentru proiectarea funcțiilor de rețea în generațiile viitoare de core mobil 5G

În acest capitol este propusă o arhitectură de cloud nativă si implementarea soluției Open5GS în containere care rulează într-un cloud public prin utilizarea unui manager de pachete dedicat orchestratorului Kubernetes, rezultatele fiind publicate in lucrarea [14].

Migrarea către funcții de rețea native în cloud (CNF) oferă o soluție pentru orchestrarea VNF-urilor deoarece microserviciile pot rula în containere și pot acomoda cerințe precum autoscalare, reziliență și monitorizare de rețea.

3.1 Descrierea soluției propuse

Configurația propusă (vezi Figura 3.1) folosește versiunea containerizată a tool-ului open-source Open5GS [7], versiunea 15 care introduce conceptul de “standalone” pentru nucleul 5G ce constă dintr-o rețea de bază complet virtualizată pentru a valorifica implementarea și gestionarea funcțiilor de rețea virtualizate (NFV) împreună cu capabilitățile de segregare și segmentare ale rețelei. Terminalele mobile și stațiile de bază în 5G sunt simulate în configurația propusă folosind UERANSIM care este un simulator open-source pentru implementarea UE și 5G gNB. UERANSIM este compatibil cu Open5GS și are două componente principale: gNodeB (gNB) care se conectează la AMF și se ocupă de traficul datelor în Internet prin legătura radio simulată, iar utilizatorii (UE) reprezintă abonaţii de telefonie mobilă care generează trafic în Internet. Ambele simulatoare Open5GS și UERANSIM sunt hostate într-un cloud public.

Atât pentru configurațiile UE, cât și pentru gNB sunt definite numărul de segmente de rețea denumite "network slices". Scopul utilizării de "network slices" în 5G este de a oferi izolare și QoS pentru diferitele cerințe de rețea end-to-end care partajează aceeași resurse fizice. De exemplu, două segmente de rețea pot partaja același SMF.

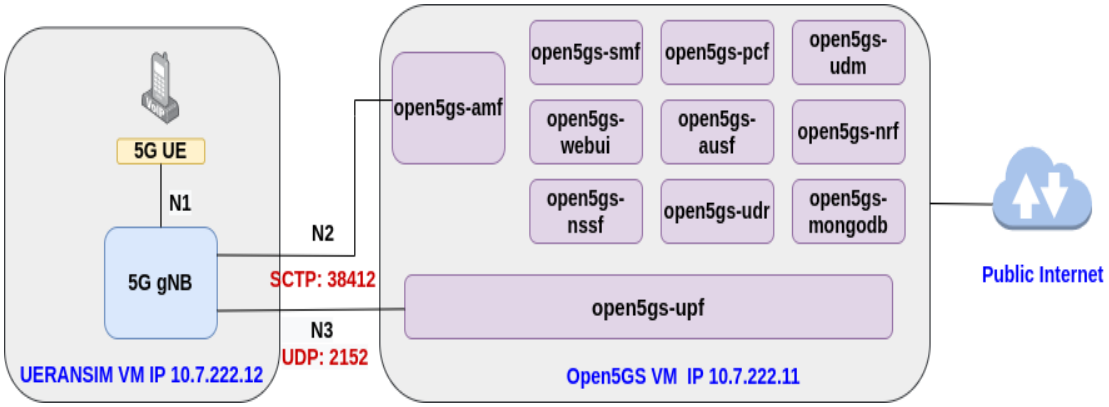


Figura 3.1 Scenariul propus instalat într-un cloud public

Mai departe, este validată funcționalitatea pe trei interfețe principale: interfața radio - între UE și RAN, interfața de control - între RAN și AMF și interfața utilizator - între RAN și UPF. Datorită limitarii impuse de simulatorul UERANSIM de a permite doar configurarea maximum 15 sesiuni concomitent, a fost rulat scriptul UE de trei ori pentru a testa conectarea a 45 de utilizatori, respectiv stabilirea a 45 sesiuni PDU in decurs de o oră.

De asemenea este implementată o soluție de “service-mesh” care ne arată comunicarea inter-servicii în timp real în termenii de distribuție a traficului, durata cererilor de interogare și throughput-ul pentru toate serviciile de rețea ale nucleului mobil 5G. Pentru a vizualiza graful de service mesh cu distribuția și procentele de trafic a fost folosit instrumentul Istio [15], peste care s-a adaugat un plugin numit "kiali" [16] (vezi Figura 3.2).

Din analiza experimentală reiese că throughputul cererii la interogarea SMF este mai mare comparativ cu celelalte cereri de servicii. Acest lucru este determinat de faptul că AMF este responsabilă de selecția Network Slice și de a determina functia SMF cea mai potrivită să gestioneze cererea de conectare interogand NRF (vezi Figura 3.3).

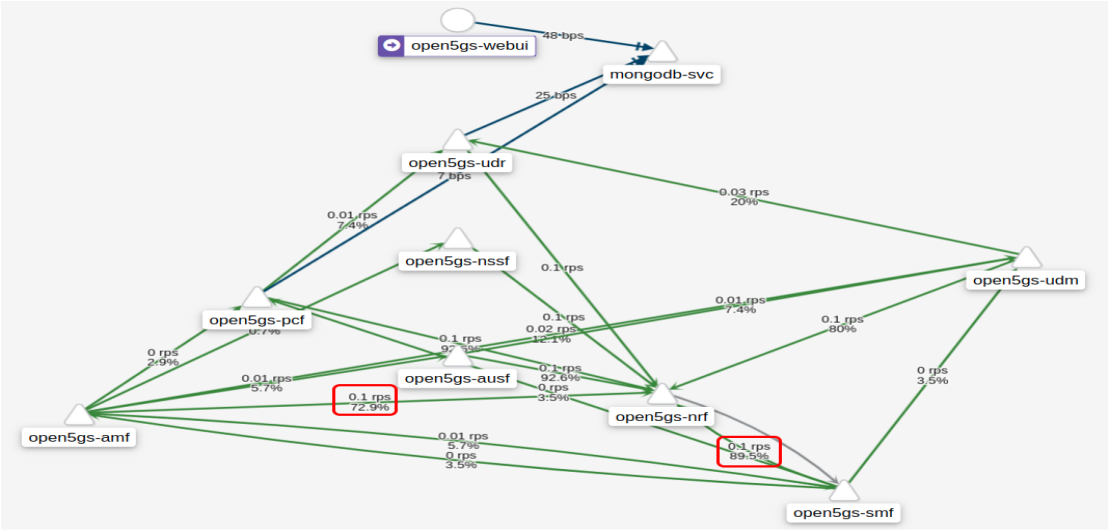
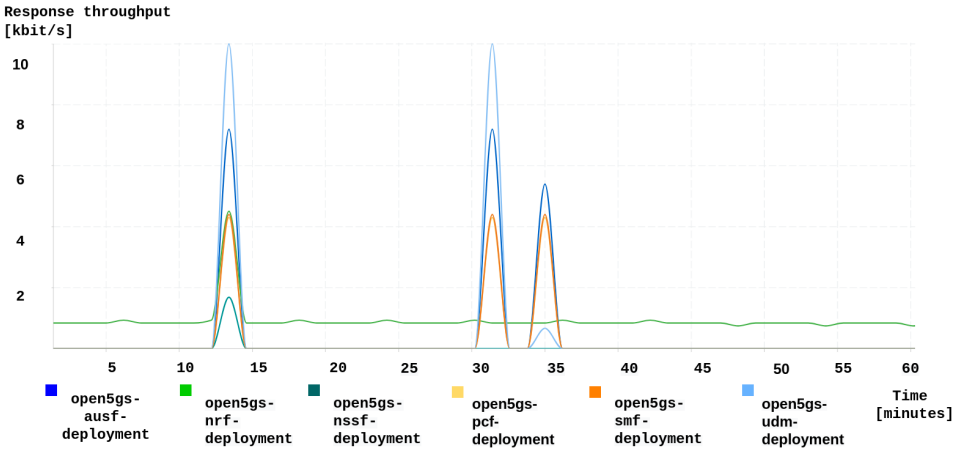
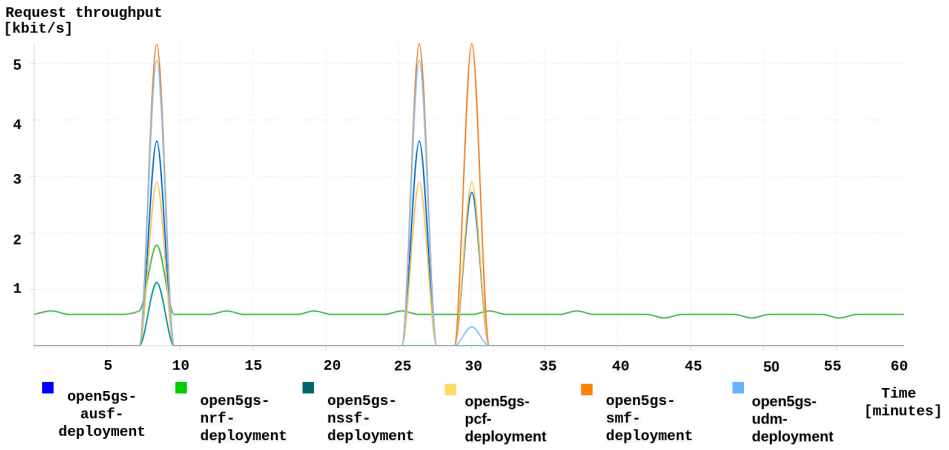


Figura 3.2 Graful de "service-mesh" cu distribuția și procentele de trafic



(a) Throughput de răspuns al funcției AMF măsurat la sursă



(b) Throughputul de interogare a funcției AMF măsurat la destinație

Figura 3.3 Throughputul de interogare și răspuns al funcției AMF măsurat pe interfața de ieșire

Capitolul 4

Analiza potențialelor vulnerabilități în implementarea noii generații de mobile core 5G

În acest capitol sunt prezentate două scenarii de atac pe interfețele principale ale planului de control și utilizator. Rezultatele au fost publicate in lucrarea [17].

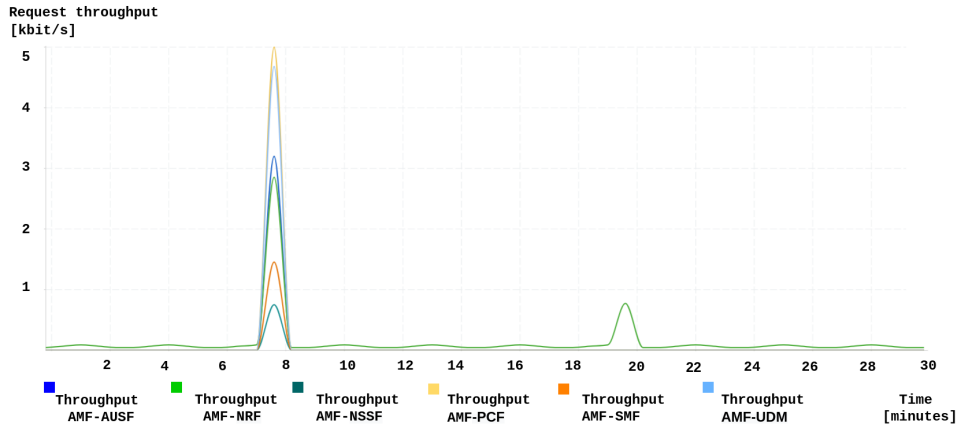
Interfețele care conectează 5G RAN cu AMF și respectiv, UPF sunt ținte pentru atacatori, deoarece transmit date sensibile din planul utilizatorului între rețeaua de acces și reteaua centrală.

4.1 Obiectivul soluției propuse

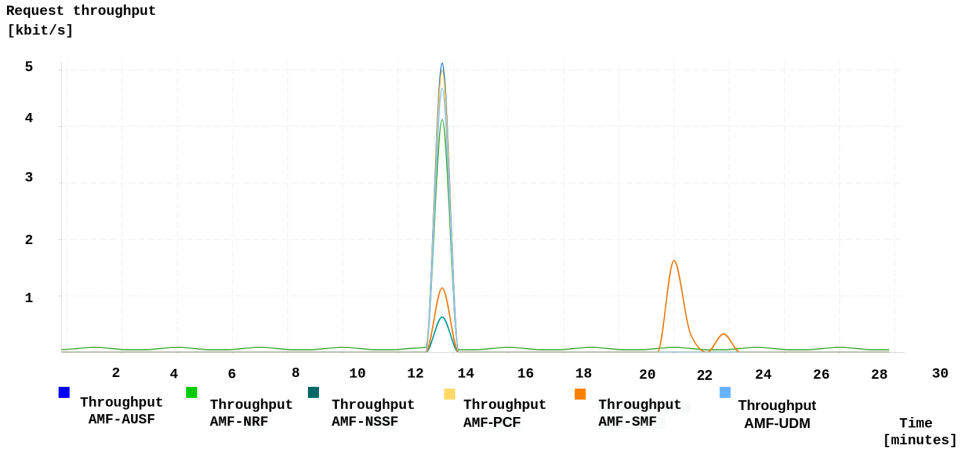
Pentru a simula cele doua scenarii de atac, a fost ulitizat, 5Greplay care este o soluție open-source ce poate fi configurată pentru a reda traficul și a transmite pachete de rețea modificate la placa de interfață de rețea (NIC) pe porturi predefinite. Traficul generat de instrument este conform protocoalelor standardizare în 3GPP Release 16 și este compatibil cu cadrele existente de 5G open-source, precum Open5GS.

În primul scenariu dupa ce este după ce cele 15 sesiuni PDU au fost stabilite cu succes, respectiv cele 15 conexiuni ale utilizatorilor, este specificat protocolul țintă, adică SCTP și adresa gazdei unde AMF rulează. S-a constat că interfața TUN (planul utilizatorului între UE și UPF) nu este afectată, conexiunea UE la Internet rămânand disponibilă. Cu toate acestea, este afișat un avertisment ”Unhandled SCTP connection received” și la pornirea unui noi sesiuni PDU, planul de utilizator nu mai este stabilit. Prin urmare, dacă AMF nu implementează un mecanism de protecție împotriva acest tip de atac, rețeaua nu va renunța la pachet. În plus, această vulnerabilitate ar putea fi exploatată de un actor rău intenționat care poate uzurpa identitatea utilizatorului.

În cel de al doilea scenariu este simulat un atac pe interfața N3 dintre gNB și UPF. Pentru aceasta simulare a fost rulat scriptul 5Greplay unde s-a modificat gazda țintă care



(a) Throughputul de răspuns al funcției AMF în urma atacului 5Greplay pe interfețele N1/N2



(b) Throughputul de răspuns al funcției AMF în urma atacului 5Greplay pe interfața N3

Figura 4.1 Throughputul de răspuns al funcției AMF în cele 2 scenarii de atac cu 5Greplay

este masina virtuală pe care rulează UERANSIM și portul țintă pentru UDP. Este injectat traficul PCAP de la gNB în nucleul 5G, care întrerupe sesiunea PDU. Mai departe este rulat scriptul gNB pentru a restabili conectivitatea planului de date.

Monitorizarea în cadrul clusterului Kubernetes a fost de asemenea realizată prin implementarea unei soluții de rețea de servicii (service mesh) de la Istio care servește ca proxy pentru fiecare dintre funcțiile de bază 5G.

S-a observat că atunci cand este injectat atacul 5Greplay, planul utilizatorului este întrerupt, deci vedem răspunsul AMF la SMF care este aproape jumătate din valorile inițiale (Figura 4.1 -b)). Pentru a solicita o nouă sesiune, UE și gNB utilizează protocolul NGAP pentru a transporta mesaje NAS. Funcția AMF primește aceste solicitări și este responsabilă să gestioneze gestionarea mobilității în timp ce redirecționează cerințele sesiunii de rețea către SMF. În acest fel, AMF determină care SMF este cel mai potrivit pentru a gestiona cererea de conexiune prin interogarea NRF.

Capitolul 5

Comunicarea Cloud-Edge: Un model de orchestrare bazat pe API declarativ pentru Noua Generație 5G Core

Prezentul capitol prezintă beneficiile de implementare a serviciilor și funcțiilor de rețea în clustere diferite pe o infrastructură asigurată de mai mulți provideri de cloud pentru îndeplini izolarea deplină între chiriași/tenanți. De asemenea, au fost evaluate diferite modele de orchestrare a containerelor concepute pentru "edge computing".

Unul dintre obiectivele principale a fost comparația dintre două modele de implementare din perspectiva latenței pentru funcțiile 5G rulate în containere și consumul de resurse de memorie pentru planul de control. Rezultatele a fost publicate in lucrarea [18].

5.1 Descrierea contextului cloud-edge

Conceptul de segmentare a rețelei sau "network slicing" a fost introdus în arhitectura 5G precum și o nouă componentă numită Multi-Acces Edge Computing (MEC) ce poate fi implementată la periferia rețelei mai aproape de utilizatorii finali.

5.2 Orchestratoare de containere pentru edge computing

În această secțiune sunt prezentate trei modele de orchestrare: **kind** [19], **K3s** [20] si **KubeEdge** [21]. Instrumentul open-source **Kind** a fost dezvoltat de Kubernetes Special Interest Groups (SIG-uri) și suportă implementări cu mai multe noduri și mai multe clustere, in plus poate fi integrat cu ușurință cu alte API-uri de Kubernetes, de exemplu ClusterAPI (CAPI) [22].

Kubedge este un alt proiect open-source dezvoltat sub CNCF (Cloud Native Computing Foundation) construit pe Kubernetes cu scopul de a orchestra Aplicații IoT. Arhitectura KubeEdge are două componente principale: cloud și edge cu modulele corespunzătoare „hub” și „margine”.

Un alt instrument dedicat rulării de clustere Kubernetes la marginea retelei este **K3s** care vine cu o distribuție de Kubernetes mai ușoară ce necesită mai puține resurse. Acest proiect a fost dezvoltat de Rancher Labs și este potrivit în principal pentru cazurile de utilizare IoT și tehnologiile de vârf. Implementarea este împărțită în servere K3s și mai mulți agenți care pot rula la margine rețelei și, spre deosebire de KubeEdge, nu necesită comunicare în partea cloud.

5.3 Modelul de ClusterAPI

Cadrul CAPI constă din trei concepte: clusterul de management care stochează informațiile de la toți furnizorii de cloud, furnizorii de "bootstrap" folosiți pentru a instala nodurile dedicate planului de control Kubernetes și clusterele de tip „workload” sau „tenant” create ca resurse obiect de cluster asociate cu furnizorii de cloud [23].

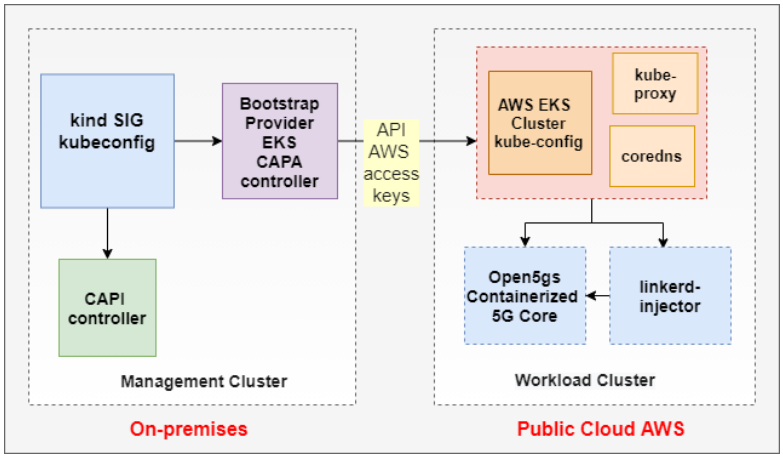
Nucleul rețelei mobile de pachete poate rula în tenanți diferiți de cloud dar într-un cadru diferit numit cloud privat virtual (VPC) corespunzând unui segment de rețea. Rolul principal al unui VPC este de a oferi segmentarea rețelei și securitate în întreaga configurare a politicii.

Fiecare cloud și mediul local au propriul provider de ClusterAPI dedicat care asigură furnizarea clusterului, furnizorul CAPI pentru AWS (CAPA), în timp ce CAPI pentru vSphere (CAPV) este dedicat furnizorului VMware. Furnizorul CAPI pentru AWS (CAPA) acoperă atât implementarea Elastic Compute (EC2) precum și Elastic Kubernetes Service (EKS), adică CAPA EKS. Un utilizator poate defini diferite tipuri de resurse într-o manieră declarativă utilizând sabloane de definire a resurselor, Custom Resource Definition (CRD) care diferă în funcție de infrastructura furnizorului de cloud.

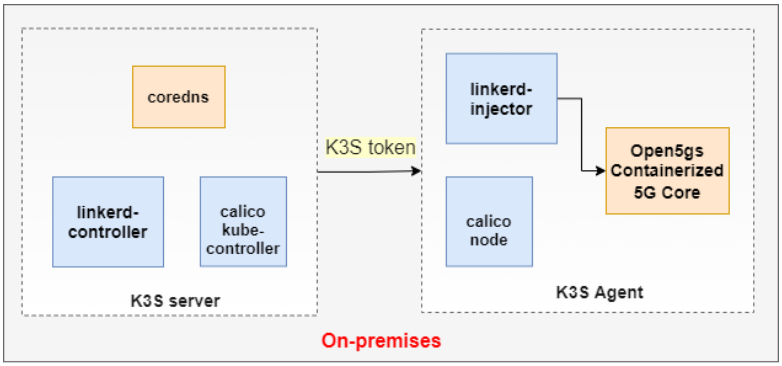
5.4 Configurare experimentală - implementarea nucleului de următoare generație 5G în clustere Kubernetes

În primul scenariu de test (Figura 5.1 (a)) am instalat două clustere Kubernetes cu configurații diferite: tipul *SIG kubeconfig* generat cu Kind care corespunde clusterului de management și o doua configurație de *kubeconfig* pentru clusterul EKS unde clusterul de tip workload rulează.

Al doilea scenariu de testare (a se vedea Figura 5.1 (b)) rulează într-un mediu local de VMware (on-premises) care constă dintr-un cluster K3s compus dintr-un server K3s care acționează ca un nod de tip master și un K3s agent ca și nod lucrător.



(a) Open5GS provizionat in clusterul EKS folosind kind și CAPI

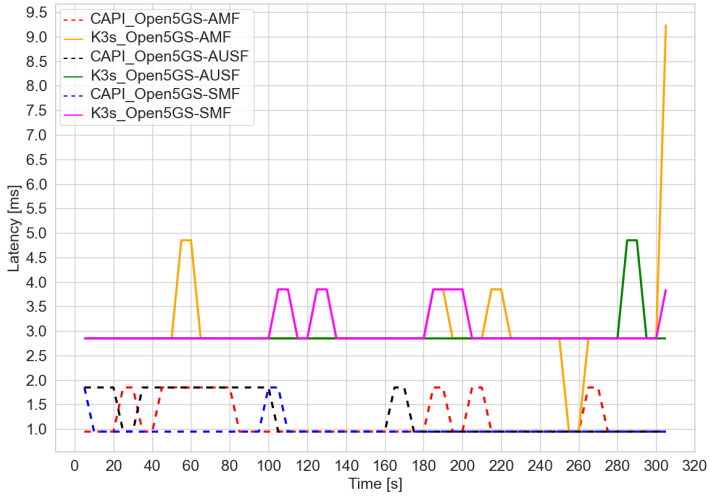


(b) Open5GS provizionat in clusterul K3S

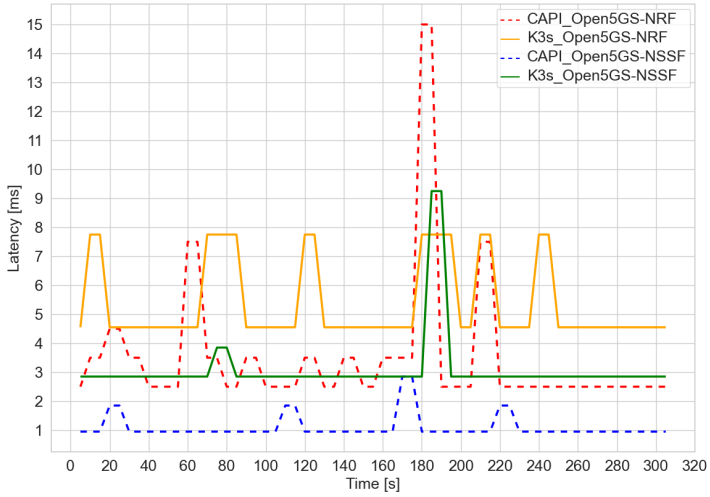
Figura 5.1 Setup-ul folosit pentru 5G Core SBA implementat cu orchestrator Kubernetes la periferia rețelei (Figura 5.4 din teză)

Ambele scenarii au în comun o componentă numită linkerd-injector. Aceasta face parte din Linkerd [24], un serviciu popular instrument mesh implementat pe cluster Kubernetes care permite și monitorizează comunicarea între servicii și direcționează traficul și apelurile API între servicii/puncte finale. Nivelul de „service mesh" implementat deasupra rolului de infrastructură Kubernetes este de a oferi abstractizarea logicii aplicației și asigura în principal monitorizarea și observabilitatea serviciului.

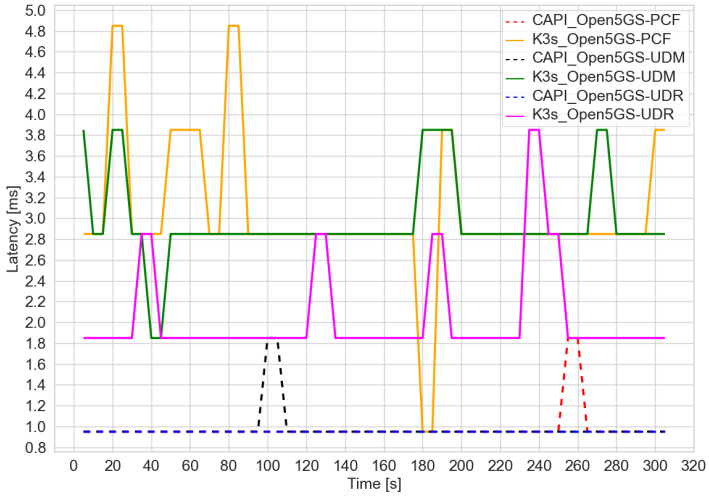
Principalul obiectivul acestei cercetări a fost de a evalua performanța obținută in urma comunicării între procesele de aplicații containerizate Open5GS folosind interfețe de programare a aplicațiilor (API) pentru ambele soluții de implementare propuse, CAPI și K3-urile. Rezultatele obținute arată valori mai bune ale timpului de răspuns pentru configurarea CAPI în comparație cu K3s (vezi Figura 5.2)care validează ipoteza noastră în ceea ce privește beneficiul de a rula Open5GS într-un cloud public deoarece atunci când am folosit la cerere resursele nu numai că scalabilitatea a crescut, dar au condus și la o comunicare mai rapidă între servicii.



(a) Funcțiile de autentificare, mobilitate și management al sesiunii - AMF/AUSF



(b) Funcțiile de gestionare a resurselor de rețea - NRF/NSSF



(c) Funcțiile de control al politicilor și de gestionare a datelor - PCF/UDM/UDR

Figura 5.2 Comparație între latența implementării Open5GS folosind CAPI vs K3s (Figura 5.5 din teză)

Capitolul 6

O abordare inovativă pentru scalarea segmentării rețelei 5G Core rulate într-un cadru de cloud nativ pe mai multe site-uri

În acest capitol este prezentată o nouă abordare bazată pe paradigma de "liquid computing", care are avantajul adaptării la mediul în schimbare. În acest mod nucleul mobil 5G poate fi gestionat ca o singură entitate de cluster Kubernetes care rulează într-un cloud public urmând modelele native din cloud pentru configurația declarativă bazată pe API-urile Kubernetes și alocarea de resurse la cerere. Mai mult, sunt analizate două scenarii de descărcare a funcțiilor utilizatorului Open5GS, precum și a planului de control către clusterul la distanța.

De asemenea, sunt validate trei cazuri de utilizare a rețelei end-to-end, prezentând automatizarea completă a nucleului 5G și valorificând capacitățile implementărilor cu mai multe clustere Kubernetes și monitorizării inter-serviciilor prin soluția de "service mesh" aplicată.

6.1 Comparație între cadrul ETSI MANO și ochestratorul Kubernetes

În Figura 6.1 este prezentată soluția propusă pentru containerizarea nucleului mobil de 5G in comparație cu cadrul ETSI MANO unde funcțiile de rețea sunt virtualizate.

Cadrul ETSI MANO diferă de modelul Kubernetes deoarece Managerul VNF deține o vedere detaliată a VNF-urilor implementate și o expunere spre nord către NFVO. În Kubernetes, aceste informații nu sunt exportate în nivelurile superioare, deoarece Kubernetes oferă o mai bună modalitate de a controla VNF-urile prin definiții de obiect

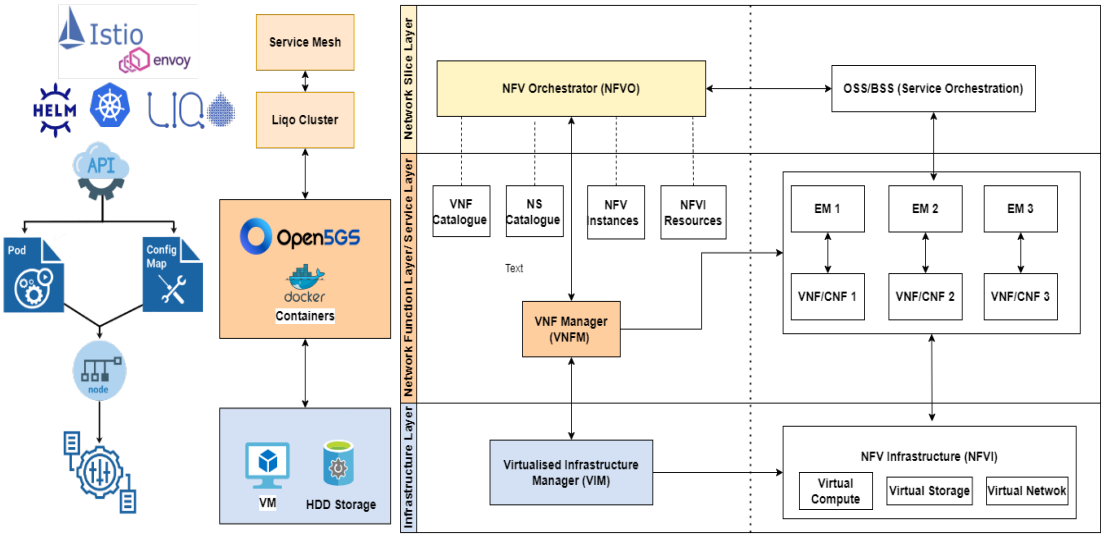


Figura 6.1 Cadrul propus pentru segmentarea rețelei în nucleul mobil de 5G nativ în cloud

(cum ar fi etichete, etichete, selectoare,etc.). În Kubernetes, conceptul de Pods există pentru a defini unde rezidă containerizarea aplicațiilor.

6.2 Scalarea orizontală vs. verticală în cloud

În Kubernetes, scalarea orizontală înseamnă extinderea numărului de poduri ca răspuns la volumul de lucru crescut. Caracteristica HorizontalPodAutoscaler (HPA) actualizează in mod automat o resursă de sarcină de lucru (cum ar fi o implementare de tip "Deployment" sau cu stări persistente "StatefulSet"), cu scopul de a scala automat volumul de lucru pentru a se potrivi cererilor de înregistrare a capacității [25].

În ceea ce privește scalarea verticală, planificatorul Kubernetes indică mai multe resurse (de exemplu, memorie sau CPU) podurilor care rulează deja ca parte a volumului de lucru. Rolul cheie al Vertical Pod Autoscaler (VPA) [26] este de a seta automat solicitările pentru containere pe baza utilizării podului printr-un mecanism optim de programare pentru a aloca la cerere resursele de calcul necesare pentru rularea fiecărui pod.

6.3 Scalarea în implementările Kubernetes multi-cluster și multi-cloud

Instrumentul open-source Liqo [27] folosește abstractizarea nodului virtual ca o extensie a proiectului Virtual Kubelet [28]. În Kubernetes, kubelet este agentul nodului principal, responsabil pentru înregistrarea nodului cu planul de control si manipularea programării podurilor. Virtual Kubelet înlocuiește un kubelet tradițional pentru un nod fizic prin

API-urile standard de Kubernetes atât pentru clusterele locale cât și pentru cele la distanță [29].

De asemenea, se transmite automat configurația negociată (Servicii, ConfigMaps, Secrete, Stocare) în capacitatea necesară pentru executarea corectă a sarcinilor de lucru descărcate printr-un mecanism care se numește "reflecție a resurselor". Toate resursele disponibile care există într-un anumit spațiu de nume sau „namespace” și sunt selectate pentru descărcare sunt propagate automat în spațiile de nume replicate corespunzător și create în clusterul selectat.

6.4 Descrierea setup-ului de testare și a celor două scenarii de peering

Liqo extinde spațiile de nume create peste limitele clusterului prin crearea de replici ale acestora în subsetul de clustere la distanță ori de câte ori este selectat un spațiu de nume pentru descărcare. Spațiile de nume găzduiesc pod-urile reale descărcate în clusterul corespunzător, precum și resursele suplimentare propagate de procesul de reflectare a resurselor.

În primul scenariu de "out-of-band" peering, inițiem peering-ul între două clustere K3S care sunt izolate unul de celalălt pentru că sunt implementate în tenanti diferiți din centre de date diferite ale cloud-ului public. Managerul de rețea Liqo reprezintă planul de control al rețelei Liqo.

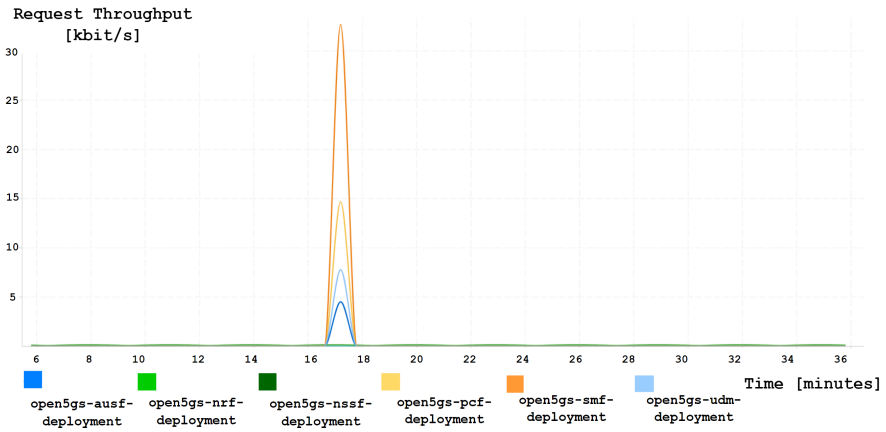


Figura 6.2 Throughputul de interogare pentru funcția AMF măsurat la sursă pentru planul de control descărcat către clusterul de la distanță (Figura 6.13 din Teză)

S-a observat că măsurăm un throughput mai mare pentru throughputul de interogare AMF în planul utilizatorului descărcat în comparație cu planul de control Open5GS prezentat în Figura 6.2, in timp ce throughputul de răspuns AMF măsurat la destinație pentru planul de control descărcat către clusterul la distanță este aproximativ jumatate

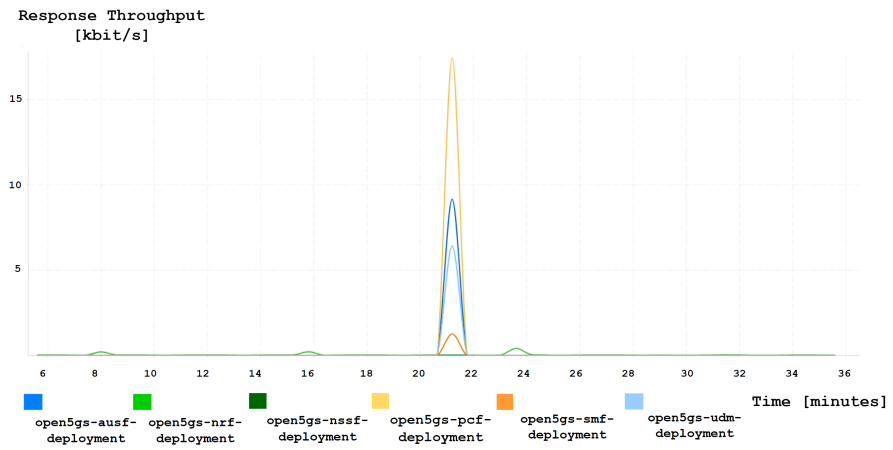


Figura 6.3 Throughputul de răspuns pentru funcția AMF măsurat la destinație pentru planul de control descărcat către clusterul de la distanță (Figura 6.14 din Teză)

din thoughputul de interogare AMF măsurat la sursă pentru planul de control descărcat către clusterul la distanță (Figura 6.3).

Motivul se datorează în principal traficului de stabilire a sesiunii PDU generat împreună cu crearea de interfețe in planul utiliatorului care se face între tenanți pentru peering-ul *out-of-band*.

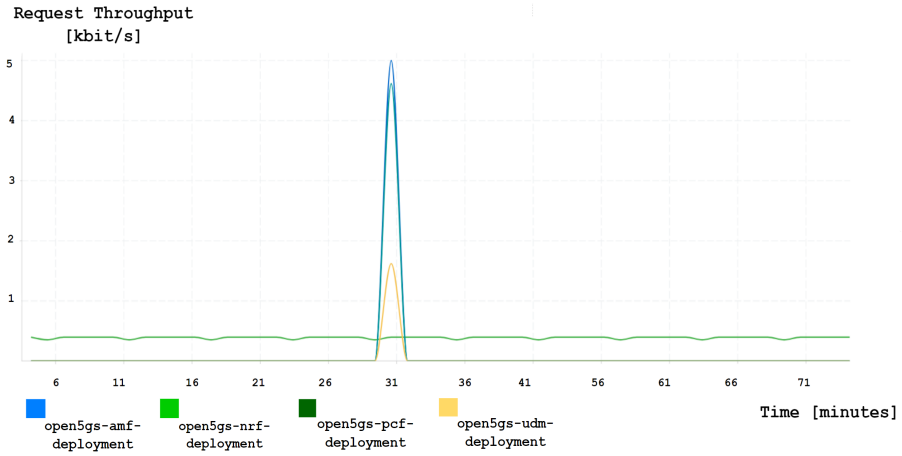


Figura 6.4 Throughputul de interogare pentru funcția SMF măsurat la destinație pentru planul utilizatorului descărcat către clusterul de la distanță (Figura 6.15 din Teză)

Pe de altă parte, **throughput-ul de interogare** pentru serviciul SMF pe destinația de ieșire pentru planul de control descărcat către clusterul la distanță este de trei ori mai mare decât în cazul planului utilizatorului descărcat Figura 6.4) și (Figura 6.5. Pe de altă parte, pentru **throughput-ul de răspuns** la sursă în cazul unui plan de utilizator descărcat pe clusterul la distanță, înregistrăm jumătate din valoarea **throughput-ului de interogare** măsurat la destinație pentru același scenariu.

În al doilea scenariu de "in-band" peering, analizăm serviciile planului de control Open5GS descărcate către clusterul la distanță. Clusterul remote rezidă într-un tenant și regiune diferite.

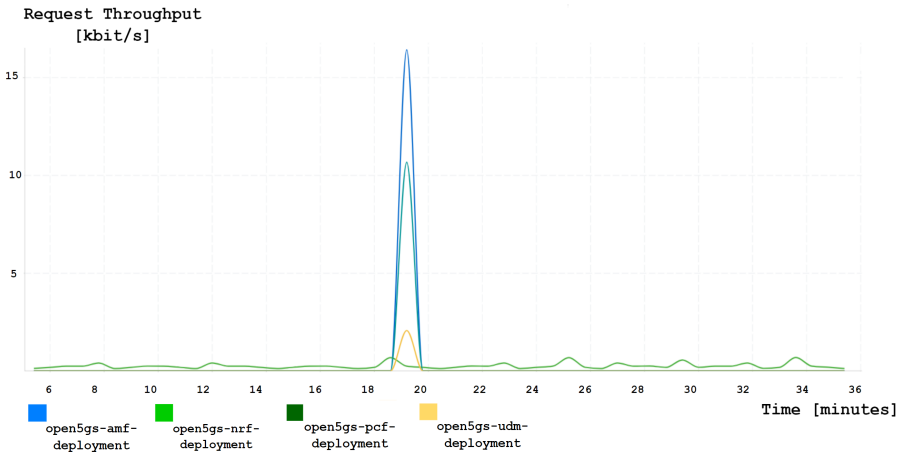


Figura 6.5 Throughputul de interogare pentru funcția SMF măsurat la destinație pentru planul de control descărcat către clusterul de la distanță (Figura 6.16 din Teză)

Particularitatea peering-ului în bandă sau "in-band" peering pentru cele două clustere este că întregul trafic din planul de control (inclusiv serviciile de autentificare) trece prin tunelul VPN. În acest scenariu, serviciul API Kubernetes nu este expus în afara clusterului. Peering-ul în bandă implică mai mulți pași pentru autentificare și stabilirea tunelului VPN folosind clientul WireGuard [30].

În cazul peering-ului "in-band", valorile în termeni de latență pentru cele trei funcții UPF sunt similare, deoarece traficul din planul utilizatorului dintre cele două clustere comunică prin tunelul VPN (Figura 6.6).

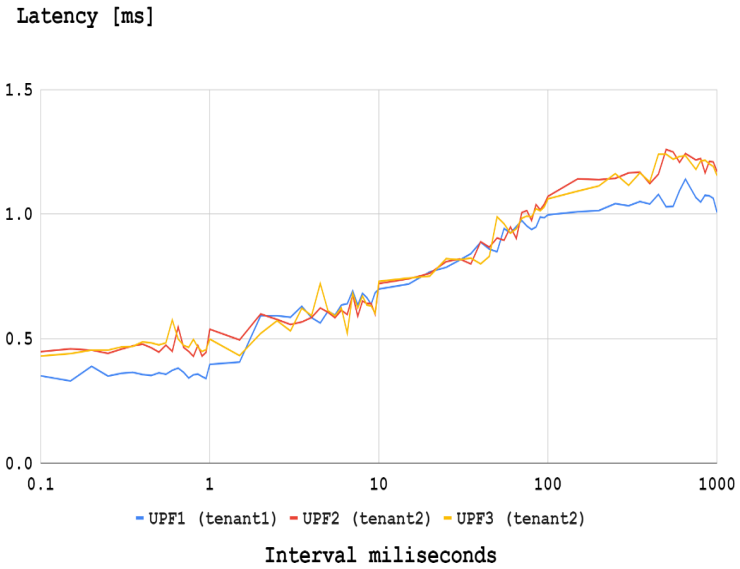


Figura 6.6 Latența în planul utilizatorului pentru scenariul de peering in-band (Figura 6.19 din Teză)

Capitolul 7

Optimizarea clusterelor de tip Kubernetes utilizând metoda hibridă de planificare a stărilor partaje

În acest capitol este analizat un nou model de planificator bazat pe Kubernetes care folosește o metodă hibridă de planificare cu o funcție de corecție a stărilor partajate. De asemnenea, este prezentată comparația intre acest tip de planificator și principale tipuri de planificatoare existente în literatura de specialitate.

7.1 Lucrări conexe și o scurtă taxonomie a mecanismelor de planificare existente

Orchestratorul Kubernetes reprezintă un **mecanism de planificare centralizat** utilizat pe scară largă ce implementează un mecanism de stocare persistent partajat, expus prin intermediul API-urilor. Construirea de API-uri de management în jurul containerelor este considerată mai orientată spre dezvoltatori și schimbă preocuparea principală de distribuire a datelor de la nivelul mașinii la nivelul aplicației. Tabelul 7.1 oferă o clasificare a diferitelor modele de planificatoare existente în literatura curentă.

7.2 Probleme identificate în planificatorul Kubernetes

Au fost identificate principalele probleme ce pot determina necesitatea mutării pod-urile care rulează deja în alte noduri din diverse motive: nodurile sunt sub sau suprautilizate, precum și adăugarea sau înlăturarea de pe noduri a etichetelor sau cerințele de afinitate pentru pod/nod nu mai sunt îndeplinite. De asemenea, nodurile pot afișa erori și podurile lor au fost mutate în alte noduri sau atunci cand noduri noi sunt adăugate la clustere.

Tabel 7.1 Taxonomia diferitelor cadre de planificare existente în literatura actuală

Tip de Planificator	Centralizat	Două nivele	Stare partajată	Hibrid	Distribuite
Borg	✓				
Omega			✓		
Apache Hadoop YARN	✓				
Apollo			✓		
Kubernetes	✓				
Hadoop-on-Demand		✓			
Apache Mesos		✓			
Docker Swarm	✓				
Firmament-Poseidon	✓				
Nomad			✓		
Sparrow					✓
Tarcil			✓	✓	

In urma scenariului de test propus, s-a constat ca într-un cluster de Kubernetes care rulează pe o perioadă de lungă durată, după ce un nod eșuează, volumul de lucru nu este echilibrat în mod egal între noduri.

7.3 Descrierea metodei de planificare propuse: planificator hibrid cu stări partajate

Metoda propusă folosește o arhitectură hibridă cu stare partajată, în care interferența sarcinilor colocate este gestionată de o funcție de corecție a programării. Aceasta funcție de corecție a programării compară sarcina de timp de așteptare transmisă de nodurile de resurse cu estimarea acesteia, calculată pe baza numărului de tranzacții concurente efectuate în timp de fiecare dintre agenții de planificare.

Arhitectura din Figura 7.1 constă din mai mulți *Agenți de planificare - Scheduling Agents* (SA) configurați ca sclavi, un master care are o vedere de ansamblu asupra stării clusterului, *Master-State Agent* (MSA) și o *Funcție de corecție - Scheduling Correction* (SC) care estimează timpul de așteptare pentru joburile rămase neprogramate și neprioritate. *Nodul Resursă - Resource Node* (RN) stochează informații cu privire la utilizarea resurselor (de exemplu, utilizarea memoriei, CPU, debitul și timpul de execuție) și partajează aceste informații cu SA. RNs gestionează durata de viață a fiecărei sarcini, în timp ce SA rulează ciclul de viață al unui job. Fiecare SA prelucrează aceste informații împreună cu prioritățile jobului și *Fezabilitatea Nodului - Node Feasibility* (NF), astfel, după ce ia o decizie de plasare a unui job, trimite starea actualizată către MSA care o trimite în continuare către SC și tuturor agenților de planificare.

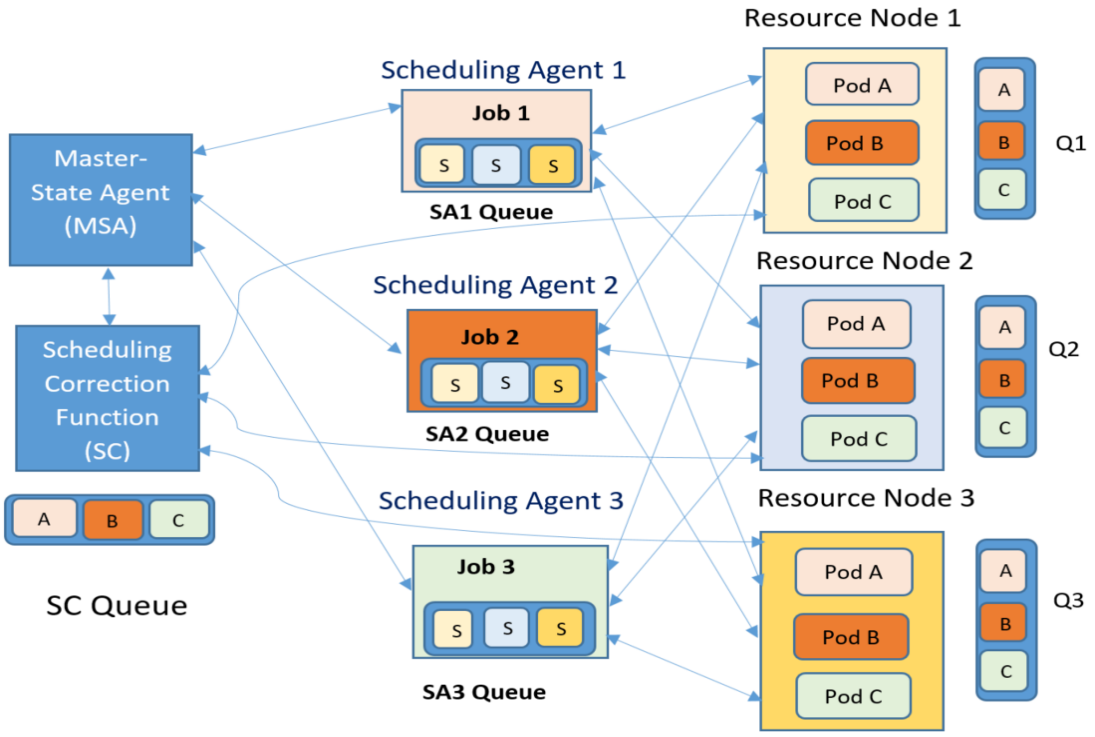


Figura 7.1 Arhitectura propusă de planificare hibridă cu stări partajate

7.4 Logica Funcției de Corecție (SC)

Considerăm timpul total de execuție al unui job ajuns în coada Funcției de Corecție (SC) $S = [S_1, S_2, \ldots, S_n]$. Definim timpul total de execuție al unei sarcini (S_i) ca suma timpului de așteptare estimat al sarcinii (T_{w_i}) plus timpul de execuție estimat al sarcinii (R_i), unde $i \in \{1, ..., n\}$. Timpul de interferență pentru un job primit este $I = [I_1, I_2, \ldots, I_n]$.

Pentru a calcula timpul total de execuție pentru o sarcină neprogramată, S'_i, funcția SC atribuie o pondere timpului derivat din latența interferenței (vezi ecuația 7.1).

$$w_i = 1 - \frac{T_a}{T_{wi}} \tag{7.1}$$

Unde T_a reprezintă durata medie de rulare estimată a sarcinii (T_a) normalizată la media timpului total de execuție pentru un job neprogramat ($\overline{S'_n} = \frac{Ta}{T_{exec}}$).

Ulterior, coada SC prioritizată conține noile valori pentru timpul total de execuție al unui job neprogramat $S' = [S'_1, S'_2, \ldots, S'_n]$ la care se adaugă timpul minim de interferență de colocare (vezi ecuația 7.2). Prioritizarea se face apoi pe baza sarcinii cu cel mai mare timp total de execuție, astfel încât să poată fi executată pe RN-ul ales.

$$S'_i = S_i + w_i \times I_i \tag{7.2}$$

Tabel 7.2 Comparația caracteristicilor în planificatoarele integrate de Kubernetes

Caracteristică	Planificator Default	Descheduler	Poseidon-Firmament	Planificator propus
`Afinitate/Anti-Afinitate nodului`	da	da	da	da
`Afinitate /Anti-afinitate inter-poduri`	da	parțial	da	da
`Impurități/Toleranțe`	da	nu	da	da
`Programare de bază/Programarea optimă`	nu	da	parțial	da
`Evitarea interferențelor de colocare`	nu	nu	parțial	da
`High-availability`	nu	da	nu	da
`Preempțiune prioritară`	da	nu	parțial	da
`Reprogramare inerentă`	nu	nu	da	da

În acest mod determinăm cazurile de colocare: dacă P_C =1, alocarea resurselor este optimă și rezultatul este trimis către SA, dacă $P_c < \overline{S'_n}$ avem o programare suboptimală, în caz contrar funcția SC determină întreaga reprogramare a cozii (Ecuația 7.3):

$$P_C = \begin{cases} 1 - F_n, & \text{if } F_n \geq \overline{S'_n}. \\ F_n & F_n < \overline{S'_n}. \end{cases} \tag{7.3}$$

7.5 Analiza metodei de planificare cu stări partajate în comparație cu alte soluții de planificare existente

Tabelul 7.2 prezintă o comparație a caracteristicilor comune, precum și a diferențelor dintre cei trei candidați de planificare integrati cu cadrul de planificare Kubernetes (**Descheduler, Firmament-Poseidon** și soluția propusă, planificatorul **hibrid cu stări partajată**).

Regula *Afinitate/Anti-Afinitate nodului* - Definește modul în care nodurile sunt selectate de către planificator folosind etichete personalizate pe noduri și selectoare de pe poduri, în timp ce regula *Afinitate /Anti-afinitate inter-pod* reprezintă o constrângere împotriva etichetelor podului, mai degrabă decât a etichetelor nodurilor.

Impurități/Toleranțe - Sunt opusul afinității nodului, deoarece permit unui nod să respingă un set de poduri. *Evitarea interferențelor de colocare* are loc între podurile care concurează pentru aceleași resurse de nod.

Programarea de bază/Programarea optimă - Pentru a lua decizii de planificare în conformitate cu cerințele de resurse, planificatorul folosește predicate și priorități.

Disponibilitate ridicată sau high-availability - Scenariul de disponibilitate ridicată a serviciilor în care pot fi programate mai multe replici pod pe același nod fără a lua în considerare disponibilitatea serviciului arată că programatorul implicit Kubernetes nu are un mecanism adaptat dinamicii clusterului.

Preempțiune prioritară - În cazul în care un pod nu poate fi programat, programatorul încearcă să elimine podurile cu prioritate inferioară pentru a reprograma podurile disponibile.

Reprogramare inerentă - Când un nod este terminat, podurile care rulau anterior pe acel nod vor fi reprogramate pe nodurile disponibile.

Capitolul 8

Concluzii

8.1 Rezultate obţinute

În Capitolul 1 sunt introduse principalele noțiuni și tehnologii din spectrul 5G precum și prezentarea ecosistemului de cloud cu tehnologiile aferente, iar Capitolul 2 prezintă noua arhitectură 5G și descrierea conceptelor de NFV și SDN precum și architectura MANO.

În Capitolul 3 a fost testată funcționalitatea de tip 5G SA într-un cloud public în care rulează Open5GS adaptat unui mediu orchestrat de Kubernetes. Pentru a valida comunicarea intre microservicii s-a folosit o soluție de "service mesh" de la Istio ce monitorizează throughput-ul în cazul stabilizirii sesiunii a 45 de utilizatori înregistrați. Mai departe, în Capitolul 4 au fost abordate două scenarii de atac pe principalele interfețe 5G SA pentru a analiza comportamentul în planul de control și cel al utilizatorului.

În Capitolul 5 a fost prezentat un model declarativ bazat pe API-uri care permite implementarea multi-cloud, modelul bazat pe implentarea ClusterAPI. Mai departe sunt analizate două cele două integrări K3s și Kind cu ClusterAPI. Printr-o nouă soluție de service mesh, de aceasta dată utilizând instrumentul Linkerd este comparată latența generată de cele două tool-uri în debitul de răspuns al funcțiilor Open5GS.

În Capitolul 6 a fost abordat un nou model de scalare a segmentelor de rețea bazat pe Liqo care respectă paradigma de "liquid computing". În plus, acest instrument îndeplinește criteriile de scalare atât orizontală cât și verticală în implementările de Kubernetes multi-cluster. De asemenea, au fost validate trei scenarii de segmentare a rețelei și a fost măsurat traficul end-to-end pentru sesiuni concomitente ale utilizatorilor.

Capitolul 7 a avut ca obiectiv propunerea un nou model de planificator bazat pe Kubernetes care folosește o metodă hibridă de planificare cu o funcție de corecție a stărilor partajate, fiind comparat cu principalele tipuri de planificatoare ce au la bază implentarea Kubernetes.

8.2 Contribuţii originale

Această secțiune prezintă principale contribuții originale care au atins obiectivul prezentei lucrări împreună cu articolele publicate și listate în ordinea publicațiilor în Secțiunea 8.3.

(1) Propunerea unui model de planificator cu stări partajate și o funcție de corecție pentru utilizarea eficientă a resurselor într-un cluster de Kubernetes. Analiza funcționalității în comparație cu alte tipuri de planificatoare.

(2) Orchestrarea nucleului mobil 5G care rulează într-o infrastructură multi-cloud și aducerea logicii de alocare a resurselor la marginea rețelei printr-un model declarativ de comunicare intre cloud și edge.

(3) Implementarea unei arhitecturi de tip 5G SBA care rulează în microservicii într-un cloud public precum și validarea conexiunii utilizatorilor prin folosirea unui emulator pentru rețeaua radio de acces.

(4) Analiza principalelor vulnerabilități care pot fi exploatate într-o architectura 5G ce rulează în cloud și validarea unor scenarii de atac pe principale interfețe ale planului de control și utilizatorului.

(5) Scalarea multi-cloud și multi-regiune a noii generații de nucleu mobil 5G printr-un nou model denumit "liquid computing". Analiza unor scenarii de comunicare între diferite clustere de Kubernetes și testarea end-to-end a conexiunii utilizatorilor care accesează mai multe segmente de rețea concomitent.

8.3 Lista lucrărilor originale

1. O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, “Kubernetes cluster optimization using hybrid shared-state scheduling framework”, pp. 1–12, doi 10.1145/3341325.3341992, *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, **ICFNDS, Paris, France, Best Paper Award, iulie 2019**

2. O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, “Collaborative cloud - edge: A declarative API orchestration model for the NextGen 5G core,” pp. 124– 133, doi 10.1109/SOSE52839.2021.00019, *IEEE International Conference on Service-Oriented System Engineering*, **SOSE, Oxford, United Kingdom, august 2021**

3. O.-M. Ungureanu and C. Vlădeanu, “Leveraging the cloud-native approach for the design of 5G NextGen core functions,” IEEE Press, p. 1–7, doi 10.1109/COMM54429.2022.9817268, *14th International Conference on Communications*, **COMM, Bucharest, Romania, iunie 2022**

4. O.-M. Dumitru-Guzu and C. Vlădeanu, “Analysis of potential threats in NextGen 5G core,”, pp. 1–4,
   doi 10.1109/ISETC56213.2022.10010163,
   International Symposium on Electronics and Telecommunications, **ISETC, Timișoara, Romania, noiembrie 2022**

5. O.-M. Dumitru-Guzu and C. Vlădeanu and R. Kooij, “A novel framework for cross-cluster scaling in cloud-native 5G NextGenCore”, pp. 1-22
   doi.org/10.3390/fi16090325,
   MDPI Journal, Future Internet, **MDPI, Switzerland, selectat ca și copertă de revistă pentru Vol. 16, Iss. 9, septembrie 2024**

8.4 Perspective de dezvoltare ulterioară

O dezvoltare ulterioară o poate constitui reproducerea implementării scenariilor multi-cloud pentru diferiți furnizori de cloud public care să adreseaze scenariile de testare propuse într-o configurație diferită utilizând dispozitive fizice pentru soluția de radio access. O altă directie ar putea fi integrarea cadrului MANO sau altor solutii open-source de virtualizare precum Open5GS într-un mediu orchestrat de Kubernetes.

Ca și perspectivă ulterioară, planificatorul propus în Capitolul 7 ar putea fi integrat cu soluția testată și validată în capitolul 6 pentru a îmbunătăți modul în care sunt rulate serviciile pe mai multe clustere de Kubernetes.

De asemenea, se pot adresa si testa noi scenarii de network slicing cum ar fi streaming, IoT și comunicarea vehiculelor sau abordarea unor noi cazuri de utilizare pentru 6G pentru a testa performanța rețelei si a valida diferite cerințe de QoS.

Bibliografie

[1] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, vol. 182, p. 107516, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620311786

[2] S. Imadali and A. Bousselmi, "Cloud Native 5G Virtual Network Functions: Design Principles and Use Cases," 11 2018, pp. 91–96.

[3] OpenAir Software Alliance EUROCOMcorene t. OpenAirInterface, Accessed: Mar. 2021. [Online]. Available: https://openairinterface.org

[4] NextEPC. NextEPC, Accessed: Mar. 2021. [Online]. Available: https://nextepc.org/

[5] Cellular Network Infrastructure. Corenet, Accessed: Mar. 2021. [Online]. Available: https://github.com/mitshell/corenet

[6] openLTE. openLTE, Accessed: Mar. 2021. [Online]. Available: http://openlte.sourceforge.net

[7] Open5GS. Open source project of 5GC and EPC (Open5GS) (Release-16), Accessed: Mar. 2021. [Online]. Available: https://open5gs.org

[8] Omec. "Lightweight Kubernetes", Accessed: Feb. 2024. [Online]. Available: https://opennetworking.org/omec/

[9] free5GC. free5GC, Accessed: Mar. 2021. [Online]. Available: https://www.free5gc.org

[10] srsLTE. srsLTE, Accessed: Mar. 2021. [Online]. Available: https://www.srslte.com

[11] Intel. Open Network Edge Services Software (OpenNESS), Accessed: Mar. 2021. [Online]. Available: https://www.openness.org

[12] Open5Gs. "Open source project of 5GC and EPC (Release 16)", Accessed: Feb. 2022. [Online]. Available: https://open5gs.org,Accessed:Feb.2022

[13] Kubernetes. Kubernetes K8S, Accessed: Mar. 2024. [Online]. Available: https://kubernetes.io

[14] O.-M. Ungureanu and C. Vlădeanu, "Leveraging the cloud-native approach for the design of 5g nextgen core functions," in *2022 14th International Conference on Communications (COMM)*. IEEE Press, 2022, p. 1–7. [Online]. Available: https://doi.org/10.1109/COMM54429.2022.9817268

[15] Rancher. "The Istio service mesh", Accessed: Feb. 2024. [Online]. Available: https://istio.io

[16] Kiali. "Kiali", Accessed: Feb. 2024. [Online]. Available: https://kiali.io

[17] O.-M. Dumitru-Guzu and C. Vlădeanu, “Analysis of potential threats in nextgen 5g core,” in *2022 International Symposium on Electronics and Telecommunications (ISETC)*, 2022, pp. 1–4.

[18] O.-M. Ungureanu, C. Vladeanu, and R. Kooij, “Collaborative cloud - edge: A declarative api orchestration model for the nextgen 5g core,” 08 2021, pp. 124–133.

[19] Kubernetes SIGs. Kind, Accessed: Mar. 2024. [Online]. Available: https://kind.sigs.k8s.io

[20] k3s. "Lightweight Kubernetes", Accessed: Feb. 2024. [Online]. Available: https://k3s.io

[21] CNCF. KubeEdge, Accessed: Mar. 2024. [Online]. Available: https://kubeedge.io/en

[22] Kubernetes SIGs. ClusterAPI, Accessed: Mar. 2024. [Online]. Available: https://cluster-api.sigs.k8s.io

[23] G. Chandra, “Multi-Cloud and Multi-Cluster Declarative Kubernetes Cluster Creation and Management with Cluster API (CAPI — v1alpha3),” ITNEXT, Jul 24 2020. [Online]. [Online]. Available: https://itnext.io/

[24] Linkerd. Linkerd, Accessed: Mar. 2023. [Online]. Available: https://linkerd.io

[25] Kubernetes. "Horizontal Pod Autoscaling", accessed: Sep. 2023. [Online]. Available: https://docs.sd-core.opennetworking.org

[26] kubernetes. Vertical Pod Autoscaler, accessed: Sep. 2023. [Online]. Available: https://github.com/kubernetes/autoscaler/blob/master/vertical-pod-autoscaler/README.md

[27] Liqo. "Enable Dynamic and seamless Kubernetes Multi-cluster Topologies", Accessed: Sep. 2023. [Online]. Available: https://liqo.io/

[28] Virtual Kubelet. "Virtual Kubelet", Accessed: Sep. 2023. [Online]. Available: https://virtual-kubelet.io/docs/

[29] Kubernetes. "Kubernetes components", accessed: Sep. 2023. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/

[30] WireGuard. "WireGuard", Accessed: Apr. 2023. [Online]. Available: https://www.wireguard.com/

[31] ETSI. "Open Source MANO (OSM)", accessed: Sep. 2023. [Online]. Available: https://etsi.org/technologies/open-source-mano

[32] ETSI GS. "Network Function Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework, Dec., 2015. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_nfv-eve005v010101p.pdf

[33] ETSI. "Network Functions Virtualisation (NFV); Management and Orchestration, Dec., 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-MAN/001_099/001/01.02.01_60/gr_NFV-MAN001v010201p.pdf

[34] O.-M. Ungureanu, C. Vladeanu, and R. Kooij, “Kubernetes cluster optimization using hybrid shared-state scheduling framework,” 07 2019, pp. 1–12.