РАЧУНАРСКА ТЕХНИКА И ИНФОРМАТИКА
/
COMPUTING AND INFORMATION ENGINEERING

(РТ/RTI)

Jedno rješenje posrednika u sistemu uslovnog pristupa digitalne televizije

Radenko Banović, Ilija Bašičević i Nemanja Lazukić

***Apstrakt*— Postoje dvije vrste TV (televizijske) usluge u domenu pretplate: javna (svima dostupan sadržaj za koji nije potrebna pretplata) i pretplatnička TV usluga (TV sadržaj je dostupan samo pretplaćenim korisnicima). Da bi pretplatnička TV usluga imala smisla potrebno je zaštititi TV sadržaj cijelim prenosnim putem. Postoji nekoliko modela zaštite pretplatničkog TV sadržaja, a jedan od njih je CAS (eng. Conditional Access System). Kompanija Widevine je kreirala rješenje sistema uslovnog pristupa (CAS) takvo da je besplatno za sve operatere. Da bi operateri mogli upravljati korisnicima i sadržajem, potrebno je implementirati korisničku upravljačku logiku sistema. U ovom radu je predstavljeno jedno rješenje softverskog posrednika (eng. Proxy) u kome je realizovana korisnička upravljačka logika sistema uslovnog pristupa u Widevine CAS sistemu.**

***Ključne reči*—Conditional Access System; Proxy; Digital Television;**

I. Uvod

Pretplatnička televizija je usluga koju nude satelitski, kablovski i drugi distributeri televizijskih kanala. Ključna tačka preduzetničkog modela u pretplatničkoj televiziji jeste zaštita televizijskog sadržaja cijelim prenosnim putem, od emitera do krajnjeg korisnika, čime se otklanja mogućnost pristupa sadržaju nepretplaćenim korisnicima[1]. Postoji nekoliko tehnologija zaštite televizijskog sadržaja, a najpoznatije su upravljanje digitalnim pravima (eng. Digital Rights Management) i sistem uslovnog pristupa (eng. Conditional Access System).

Sistem uslovnog pristupa predstavlja zaštitu prenosnog puta[2] (i on se najčešće koristi za televiziju uživo), dok je upravljanje digitalnim pravima zamišljeno kao mehanizam zaštite sadržaja (te se najčešće koristi za televiziju na zahtjev (eng. On Demand)). Za razliku od upravljanja digitalnim pravima, u sistemima uslovnog pristupa uobičajno je da se nakon određenog vremenskog intervala mijenjaju ključevi kojima je skremblovan sadržaj koji se dostavlja korisniku[3]. Kompanija Widevine je kreirala sopstveno rješenje CAS sistema za Android TV i Android STB (Set-Top Box) uređaje koje je napravilo veliki pomak u industriji digitalne televizije.

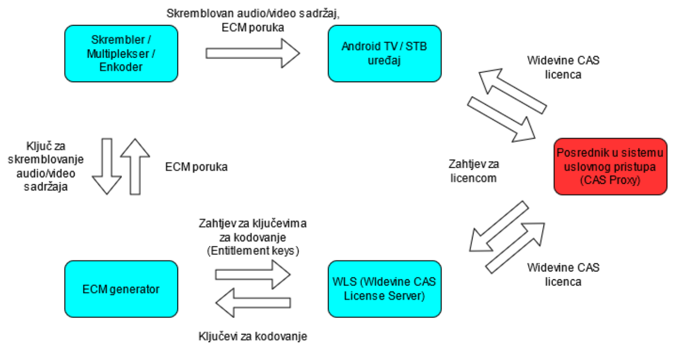
Radenko Banović – Fakultet Tehničkih Nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (e-mail: Radenko.Banovic@rt-rk.com).

Ilija Bašičević – Fakultet Tehničkih Nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (e-mail: ilibas@uns.ac.rs).

Nemanja Lazukić – Istraživačko-razvojni Institut RT-RK, Novi Sad, Srbija, (e-mail: Nemanja.Lazukic@rt-rk.com).

II. Widevine CAS sistem

Ključna prednost Widevine CAS rješenja u odnosu na konkurenciju jeste to što je kompletan CAS ekosistem dat operatorima na besplatno korištenje, pod uslovom da se izvršava na Android TV operativnom sistemu[5]. Komponente Widevine CAS sistema su : licencni poslužioc (eng. License Server), OEMCrypto (modul koji se integriše u Android TV), ECM (eng. Entitlement Control Message) generator, skrembler i posrednik u sistemu uslovnog pristupa. Sl. 1. prikazuje komponente Widevine CAS sistema na visokom nivou apstrakcije.


Skremblovan audio/video sadržaj, ECM poruka
Skrembler / Multiplekser / Enkoder
Android TV / STB uređaj
Widevine CAS licenca
Ključ za skremblovanje audio/video sadržaja
ECM poruka
Zahtjev za licencom
Posrednik u sistemu uslovnog pristupa (CAS Proxy)
Zahtjev za ključevima za kodovanje (Entitlement keys)
ECM generator
WLS (Widevine CAS License Server)
Widevine CAS licenca
Ključevi za kodovanje

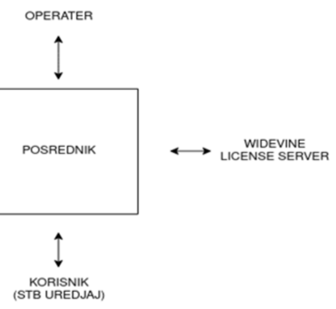
Sl. 1. Widevine CAS sistem

Neke komponente sistema su date tako da se ne mogu prilagođavati (OEMCrypto, licencni poslužioc), dok ECM generator i posrednik u sistemu uslovnog pristupa moraju da se implementiraju za svakog provajdera posebno, ali tako da se oslanjaju na Widevine SDK (eng. Software Development Kit). Licencni poslužioc je ključna tačka sistema u kojoj se sastaju predajna i prijemna strana. Korištenje licencnog poslužioca je moguće nakon što Widevine odobri zahtjev za korištenjem, i kreira posebne URL putanje prema zahtjevu provajdera.

Sa predajne strane se licencnom poslužiocu šalje zahtjev za dostavljanjem ključa (eng. Entitlement Key) kojim se enkriptuje ECM poruka u kojoj se nalaze ključevi kojima je skremblovan televizijski sadržaj. Sa prijemne strane se licencnom poslužiocu šalje zahtjev za dostavljanjem licence iz koje se izvlače ključevi kojima je moguće dekriptovati ECM poruku, te sa ključevima izvučenim iz ECM poruke deskremblovati televizijski sadržaj i prikazati ga korisniku.

III. Posrednik u sistemu uslovnog pristupa sa korisničkom upravljačkom logikom

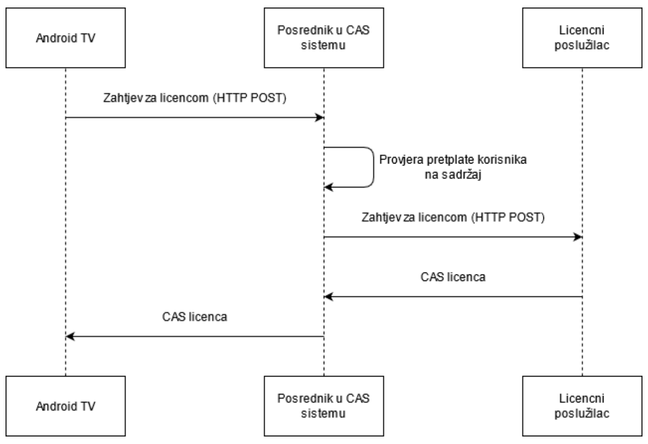
Posrednik kao jedan od elemenata CAS sistema komunicira sa Android TV / STB uređajem, kao i sa licencnim poslužiocem. Takođe, potrebno je kreirati korisnički interfejs preko kog je moguće unositi podatke vezane za korisnike, kanale, pakete na koje su korisnici pretplaćeni, što je ilustrovano u Sl. 2.



Sl 2. Interakcija posrednika sa okolinom

Posrednik je zamišljen kao mrežno orijentisan servis koji koristi REST (eng. Representational State Transfer) API (eng. Application Programming Interface) arhitekturu softvera[4]. Korištenje REST API arhitekture je omogućilo identifikovanje različitih resursa uz pomoć definisanja posebnih URI (eng. Uniform Resource Identifier) putanja, tako da i operater i korisnik (STB uređaj) mogu koristiti posrednik šaljući različite zahtjeve ka njemu. URI putanje namijenjene komunikaciji sa operaterom se odnose na upravljanje sadržajem baze podataka (dodavanje i brisanje korisnika, uređaja, paketa kanala, ažuriranje informacija o pretplatama korisnika).

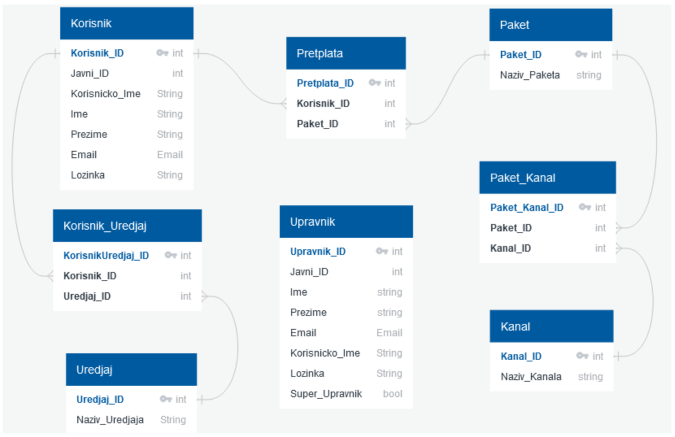
STB uređaj korisniku šalje zahtjev za licencom, zatim se nakon obrade zahtjeva provjerava da li je korisnik koji zahtjeva licencu za svoj STB uređaj pretplaćen na željeni sadržaj. Ukoliko jeste pretplaćen, zahtjev se proslijeđuje licencnom serveru, te se licenca dobijena od strane licencnog servera proslijeđuje korisniku koji je uputio zahtjev za licencom. Komunikacija između STB uređaja i posrednika, te posrednika i licencnog poslužioca je prikazana u Sl. 3



Sl. 4. Dijagram poziva posrednika

IV. Opis realizacije

Posrednik je ralizovan kao HTTP poslužilac u C++ programskom jeziku, jer je SDK na koji se on oslanja takođe realizovan u C++ programskom jeziku. Jezgro posrednika predstavlja baza podataka u kojoj se nalaze sve relevantne informacije na osnovu kojih je moguće odrediti da li je korisnik pretplaćen na odgovarajuće pakete kanala. Šema baze podataka je prikazana na Sl. 4.



Sl. 4. Šema baze podataka

A. *Alati korišteni za realizaciju*

Pošto je C++ izabran kao programski jezik, a posrednik treba da bude HTTP poslužilac izabrali smo Mongoose biblioteku[6] u kojoj je implementiran na događaj pobuđeni (eng. Event-driven) neblokirajući API za HTTP i uz koji je moguće kreirati REST API servise koji su neophodni za komunikaciju sa okolinom. Za upravljanje bazom podataka korištena je SQlite biblioteka[7].

B. *Implementacija obrade zahtjeva za licencom*

Nakon što posrednik zaprimi zahtjev na URI putanji *dobavi_licencu* u funkciji *handle_lic_req()* se uz pomoć poziva SDK funkcije *getDeviceInfo()* iz zahtjeva za licencu dobijaju informacije o uređaju i to : proizvođač, model, identifikacioni broj uređaja i serijski broj sertifikata uređaja. Iz zahtjeva za licencu se uz pomoć poziva SDK funkcije *getContentId()* dobavlja informacija o paketu kanala za koji se šalje zahtjev za licencu.

Na osnovu dobijenih informacija o uređaju iz baze podataka se dobavlja informacija o korisniku. Zatim se na osnovu informacije o korisniku i paketu kanala provjerava da li je korisnik pretplaćen na željeni paket kanala. Ukoliko je korisnik pretplaćen na paket kanala pozivom SDK funkcije *GenerateLicenseRequestAsJSON()* se na osnovu zahtjeva za licencu generiše zahtjev koji se preko HTTP Post metode korištenjem Curl biblioteke šalje licencnom serveru.

HTTP odgovor dobijen on licencnog poslužioca se proslijeđuje STB uređaju koji je poslao zahtjev pozivom funkcije mg_printf() biblioteke mongoose. Ukoliko korisnik nije pretplaćen na željeni sadržaj na STB uređaj se odmah šalje HTTP odgovor sa statusnim kodom 405 koji se odnosi na to da takav zahtjev nije dozvoljen.

Svaka akcija za popunjavanje baze podataka je kreirana sa posebnom uri putanjom i funkcijom koja obrađuje zahtjev. Akcije koje su obrađene su : dodaj korisnika, obriši korisnika, pretplati korisnika, ukini pretplatu korisnika, dodaj uređaj, obriši uređaj, dodaj kanal, obriši kanal, dodaj pretplatu, obriši pretplatu, dodaj kanal u paket i izbaci kanal iz paketa. U ovoj fazi razvoja nije predviđena realizacija prednjeg dijela (eng. Front-end) zbog čega su implementirane samo funkcije za popunjavanje baze podataka, i čitanja iz baze podataka neophodna za dobavljanje licence.

V. Testiranje

Predloženo rješenje je testirano na Synaptics BG5CT STB (Sl. 8.) uređajima sa operativnim sistemom Android Q. Korištena je Live Channels korisnička aplikacija koja se oslanja na Comedia DTV (eng. Digital Television) srednji sloj kompanije iWedia, u kom je integrisam OEMCrypto koji kreira zahtjev za licencom i koji služi za deskremblovanje televizijskog sadržaja.

Sl. 8. Synaptics BG5CT platforma

Sa predajne strane je korišten TSDuck set alata [8] koji se u ovom slucaju koristio za skremblovanje TS (eng. Transport Stream) toka podataka koji se nalazio u izvorišnoj datoteci, kao i za slanje skremblovanog toka podataka ka odredništnom STB uređaju korištenjem računarske mrežne infrastrukture i IPv6 (eng. Internet Protocol version 6) protokola. Takođe, sa predajne strane je korišten ECM generator koji je razvijan u paraleli sa posrednikom u sistemu uslovnog pristupa.

Funkcionalnost je testirana korištenjem više prijemnih uređaja pri čemu su mijenjane informacije o pretplaćenim korisnicima u bazi podataka. Kreirano je nekoliko testnih slučajeva u kojima su različiti korisnici u različitim testnim slučajevima bili pretplaćeni na željeni sadržaj. Jedan primjer testnog slučaja: korisnik A je pretplaćen na željeni sadržaj, korisnik B nije pretplaćen na željeni sadržaj, testiranjem je utvrđeno da korisnik A ima pristup sadržaju, dok korisnik B nema pristpu sadržaju.

Po završetku testiranja utvrđeno je da su STB uređaji pretplaćenih korisnika uspješno deskremblovali i reprodukovali sadržaj iz izvorišne datoteke koja je emitovana ka njima. U slučajevima nepretplaćenih korisnika STB uređaji su dobijali odgovor od posrednika da nisu pretplaćeni na željeni sadržaj, te nisu dobili licencu iz koje bi mogli izvuću ključeve kojima bi uspješno deskremblovali sadržaj.
Testiranjem je utvrđena funkcionalnost rješenja.

VI. Zaključak

U ovom radu je prikazano jedno rješenje posrednika u sistemu uslovnog pristupa sa korisničkom upravljačkom logikom. Opisan je Widevine CAS sistem u cjelini kao i uloga posrednika u njemu. Navedeni su svi alati korišteni u realizaciji rješenja, te je dat detaljan opis rješenja. Rješenje je testirano korištenjem nekoliko prijemnih uređaja i nakon uspješno završenih testova potvrđena je funkcionalnost rješenja. Doprinos ovog rada u odnosu na postojeća rješenja je u tome što je kompatibilan sa Widevine CAS ekosistemom. U budućnosti ovo rješenje može biti unaprijeđeno kreiranjem prednjeg dijela poslužioca čime bi se omogućio jednostavan vizuelni prikaz i lakše upravljanje pretplatom korisnika, te proširenjem zadnjeg dijela poslužioca.

Literatura

[1] I. Kaštelan, V. Peković, V. Zlokolica, J. Zloh, D. Trifunović, “Simultaneous automated verification of conditional access system on multiple TV sets,” Proc. IEEE International Conference on Consumer Electronics, Berlin, Germany, pp. 269-270, Sept. 2012.
[2] Fu-Kuan Tu, Chi-Sung Laih and Hsu-Hung Tung, "On key distribution management for conditional access system on pay-TV system," in *IEEE Transactions on Consumer Electronics*, vol. 45, no. 1, pp. 151-158, Feb. 1999, doi: 10.1109/30.754430.
[3] Milan Bjelica, Nikola Teslić, Velibor Mihić, “Softver u digitalnoj televiziji 1”, 2017.
[4] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine
[5] Widevine CAS, Jun 2021. [online]. https://www.widevine.com/solutions/widevine-cas
[6] Mongoose - Embedded Web Server, Jun 2021. [online]. https://github.com/cesanta/mongoose
[7] SQLite, Jun 2021. [online]. https://www.sqlite.org
[8] TSDuck, Jun 2021. [online]. https://tsduck.io/

Abstract

There are two types of TV (television) services in the domain of subscription: public (content available to all for which no subscription is required) and subscriber TV service (TV content is available only to subscribed users). In order for the subscriber TV service to make sense, it is necessary to protect the TV content throughout the transmission. There are several models of protection of subscriber TV content, and one of them is CAS (Conditional Access System). Widevine has created a conditional access system (CAS) solution that is free for all operators. In order for operators to be able to manage users and content, it is necessary to implement the user management logic of the system. This paper presents a solution of a proxy server in which the user control logic of the conditional access system in the Widevine CAS system is realized.

One solution of proxy server in the digital television conditional access system

Radenko Banović, Ilija Bašičević, Nemanja Lazukić

Jedno rješenje ECM generatora

Radenko Banović, Ilija Bašičević, Ksenija Popov i Milenko Maksić

***Apstrakt*—Zaštita televizijskog sadržaja predstavlja jedan od najvećih izazova u industriji digitalne televizije usljed sve manjeg broja televizijskih kanala čije se gledanje ne naplaćuje. Da bi omogućili naplaćivanje televizijskog sadržaja korisnicima, potrebno je zaštiti televizijski sadržaj cijelim prenosnim putem. Najkorišteniji model zaštite živog televizijskog sadržaja je CAS (eng. Conditional Access System). CAS model podrazumijeva postupak zaštite video i audio sadržaja skremblovanjem koje ima za cilj sprječavanje neovlaštene reporodukcije audio i video sadržaja. Kontrolne riječi kojima je izvršeno skremblovanje se prenose istim prenosnim kanalom kao i skremblovani sadržaj u okviru ECM (eng. Entitlement Control Message) poruke ali u enkriptovanom obliku. Kompanija Widevine je realizovala sopstveni CAS ekosistem potpuno besplatan za sve korisnike. U ovom radu je predstavljeno jedno rješenje ECM generatora u Widevine CAS sistemu.**

***Ključne reči*—ECM generator, Conditional Access System, Digital Television**

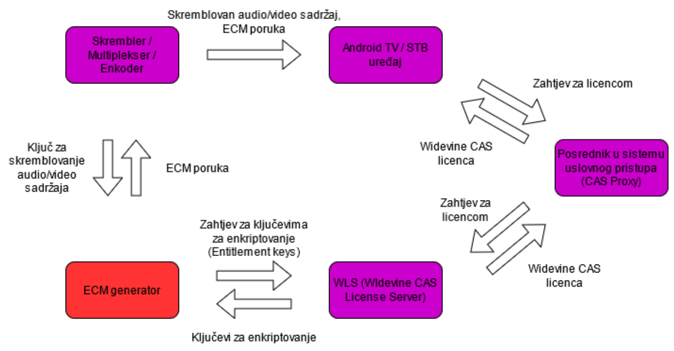
I. Uvod

U junu 2014. godine je prvi put prikazan Andoid TV operativni sistem, koji je prilagođena verzija Android operativnog sistema za televizore i STB (set-top box) uređaje[4]. Do danas je veliki broj proizvođača televizora i STB uređaja, kao i operatera televizijskih kanala integrisalo Android TV kao operativni sistem koji se izvršava na njihovim uređajima[5].

Kako bi privoljeli i preostale operatere televizijskih kanala i proizvođače televizora i STB uređaja da integrišu Android TV na svoje uređaje kreiran je Widevine CAS sistem uslovnog pristupa koji je na korištenje dat potpuno besplatno, ali može da se koristi samo uz Android TV operativni sistem. Da bi sistem postao funkcionalan, potrebno je implementirati ECM generator i posrednik u sistemu uslovnog pristupa, za šta je dat SDK (eng. Software Development Kit).

Komponente sistema koje je potrebno implementirati nisu implementirane da bi svaki operater televizijskih kanala prilagodio sistem svojim potrebama. Postoji nekoliko primjera implementacije ECM generatora [1, 2], ali oni nisu prilagođeni Widevine CAS ekosistemu. Widevine CAS sistem je prikazan u Sl. 1.

Radenko Banović – Fakultet Tehničkih Nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (e-mail: Radenko.Banovic@rt-rk.com).
Ilija Bašičević – Fakultet Tehničkih Nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (e-mail: ilibas@uns.ac.rs).
Ksenija Popov – Istraživačko-razvojni Institut RT-RK, Novi Sad, Srbija, (e-mail: Ksenija.Popov@rt-rk.com).
Milenko Maksić – Istraživačko-razvojni Institut RT-RK, Novi Sad, Srbija, (e-mail: Milenko.Maksic@rt-rk.com).


Skremblovan audio/video sadržaj, ECM poruka
Skrembler / Multiplekser / Enkoder
Android TV / STB uređaj
Zahtjev za licencom
Widevine CAS licenca
Posrednik u sistemu uslovnog pristupa (CAS Proxy)
Ključ za skremblovanje audio/video sadržaja
ECM poruka
Zahtjev za licencom
Widevine CAS licenca
Zahtjev za ključevima za enkriptovanje (Entitlement keys)
ECM generator
WLS (Widevine CAS License Server)
Ključevi za enkriptovanje

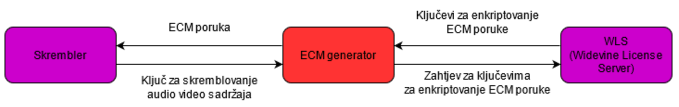
Sl. 1. Widevine CAS sistem

II. ECM generator

Da bi audio i video sadržaj bio zaštićen tokom kompletnog prenosnog toka vrši se postupak skremblovanja. Inverzni postupak u odnosu na skremblovanje se naziva deskremblovanje, njime se zaštićeni sadržaj prevodi u osnovni format razumljiv audio i video dekoderima[3].

Skremblovanje se vrši korištenjem kontrolne riječi (ključa za skremblovanje). Korištenje kontrolne riječi u procesu skremblovanja omogućuje promjenu kontrolne riječi u vremenu, a period između dve promjene se naziva periodom kriptovanja. Što je češća izmjena kontrolne riječi, to je proces skremblovanja bezbjedniji.

Trenutno korištena kontrolna riječ prenosi se u okviru ECM poruke koja se generiše u ECM generatoru. PID (eng. Packet Identifier) vrijednost TS (eng. Transport Stream) paketa u kom se nalazi ECM poruka se nalazi u CA deskriptoru PMT tabele. ECM generator u Widevine CAS sistemu komunicira sa licencnim poslužiocem (eng. License Server) i skremblerom. Pozicija ECM generatora u Widevine CAS sistem je prikazan na Sl. 2.

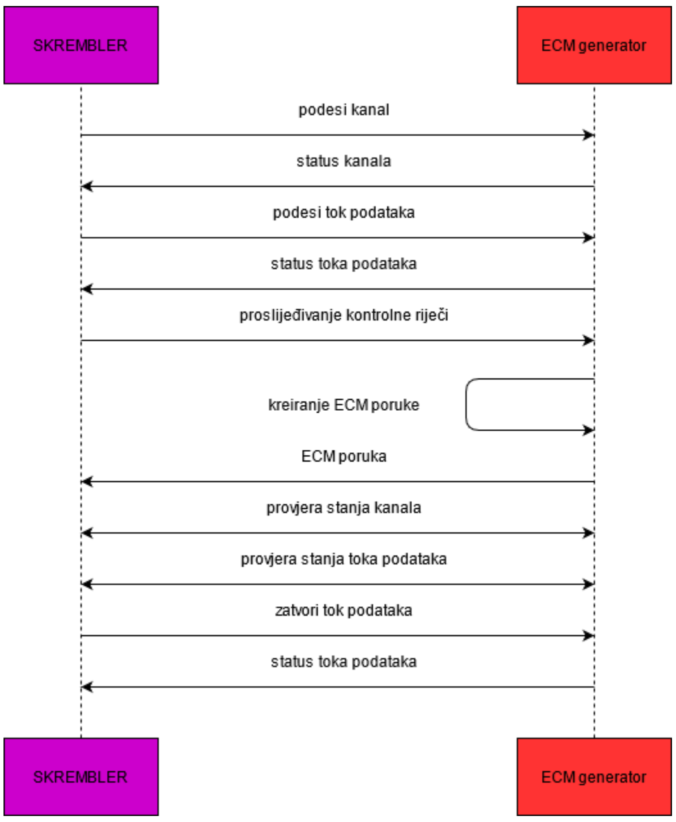

ECM poruka
Skrembler
Ključ za skremblovanje audio video sadržaja
ECM generator
Ključevi za enkriptovanje ECM poruke
WLS (Widevine License Server)
Zahtjev za ključevima za enkriptovanje ECM poruke

Sl. 2. Pozicija ECM generatora u widevine sistemu

U ovom radu je korišten skrembler implementiran u TS Duck programskoj podršci. TS Duck je set alata koji se koristi za manipulaciju MPEG prenosnim tokovima, a jedan od alata je i skrembler koji može da koristi i eksterni ECM generator za generisanje ECM poruka[6]. ECM generator i TS Duck komuniciraju po ECMG/SCS (eng. Simulcrypt Synchroniser) protokolu[7].

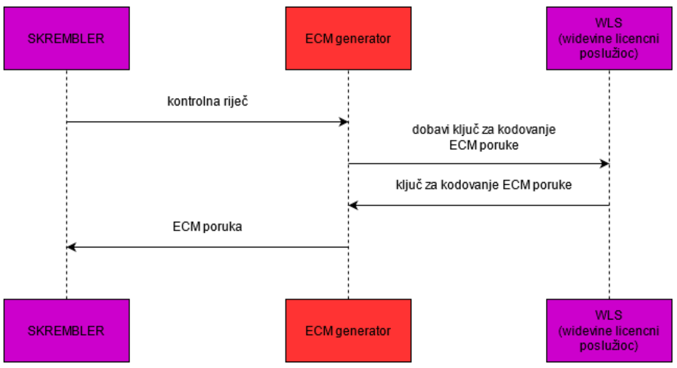
U komunikaciji između generatora i skremblera, skrembler je implementiran kao klijent, dok generator treba da bude implementiran kao poslužioc, te je ECM generator je u smislu komunikacije sa skremblerom implementiran kao TCP/IP poslužioc koja čeka zahtjeve od skremblera.

Po uspostavljanju veze generatora i skremblera kreira se sesija koja je zadužena za razmjenu poruka u okviru ECMG/SCS protokola. ECMG/SCS protokol je prikazan na Sl. 3.



Sl 3. Dijagram ECMG/SCS protokola

WLS (eng. Widevine License Server) je već gotovo rješenje sa kojim ECM generator komunicira putem HTTP protokola. ECM generator treba od WLS da dobavi (eng. Entitlement) ključeve kojima će enkriptovati ECM poruku u kojoj se nalaze ključevi kojima je skremblovan sadržaj, tako da u slučaju presretanja ECM poruke presretač ne može da dobije informaciju o ključevima kojima je sadržaj skremblovan. Dijagram komunikacije skremblera, ECM generatora i WLS je prikazan na Sl. 4.


SKREMBLER
ECM generator
WLS (widevine licencni poslužioc)
kontrolna riječ
dobavi ključ za kodovanje ECM poruke
ključ za kodovanje ECM poruke
ECM poruka
SKREMBLER
ECM generator
WLS (widevine licencni poslužioc)

Sl. 4. Dijagram komunikacije ECM generatora

III. OPIS REALIZACIJE

ECM generator je realizovan kao C++ CLI (eng. Command Line Interface) aplikacija. Prilikom pokretanja aplikacije potrebno je proslijediti broj porta na kom aplikacija osluškuje zahtjev klijenta (skrembler) za uspostavljanjem veze. Kompletno rješenje se oslanja na Widevine SDK (eng. Software Development Kit). Rješenje možemo podijeliti u tri logičke cjeline, i to: TCP/IP poslužilac, ECMG/SCS protokol, i ECM generator.

A. TCP/IP poslužilac

Ovaj modul sadrži dvije funkcije*: void TCPstart(int port, void (*onSessionEstablished)())* i *void TCPstop(int port).* Funkcija *TCPstart* kreira TCP/IP utičnice sa podrškom za IPv4 i IPv6 protokole, stavlja poslužioca u stanje čekanja zahtjeva za konekcijom klijenta, te uspostavlja vezu i kreira sesiju za korisnik / poslužilac komunikaciju. Funkcijom *TCPstop* se zatvara otvorena sesija.

B. ECMG/SCS protokol

U ovom modulu je implementiran ECMG/SCS protokol. Implementiran je tako da se izvršava u *while* petlji, poziva se funkcija *read()* koja je blokirajuća, i koja zaustavlja izvršavanje petlje dok se memorija za smještanje dolaznih podataka ne popuni. Iz pristiglih podataka se pozivom funkcije *int32_t msg_pars(const uint8_t* buff, uint32_t size, struct ecmgp_msg* msg)* popunjava sktruktura *ecmgp_msg*.

Jedno od polja strukture koja predstavlja poruku je tip poruke, na osnovu kog se korištenjem *swich* grananja bira grana u kojoj se priprema odgovor na poslatu poruku. Tip poruke može biti : *CHANNEL_SETUP*, *STREAM_SETUP*, *CW_PROVISION*, *STREAM_CLOSE_REQUEST*. Svaka od grana popunjava strukturu koja predstavlja poruku, te se poziva funkcija *int32_t msg_generator(uint8_t* buff, struct ecmgp_msg* msg)* koja od podataka iz strukture kreira poruku koja se šalje ka klijentu.

Poruka tipa *CW_PROVISION* nosi i vrijednost ključa za skremblovanje audio/video sadržaja koja kriptovana treba da se nađe u ECM poruci. U funkciji *int32_t gen_ecm_datagram(uint8_t* ecm_datagram, struct ecmgp_msg* msg)* je implementirano kreiranje ECM poruke, te se ona poziva u grani obrade poruke tipa *CW_PROVISION*. Nakon poziva ove funkcije ECM poruka se dodaje kao polje strukture *ecmgp_msg*, poziva se funkcija *msg_generator* nakon koje se kreirana ECM poruka šalje skrembleru.

C. ECM generator (u užem smislu)

Za generisanje ECM poruke i kreiranje TS paketa, te kreiranje zahtjeva za ključevima za enkriptovanje ECM poruke i parsiranjem odgovora dobijenog od WLS korištene su funkcije dobijene iz Widevine SDK paketa. Funkcija u kojoj se kreira ECM poruka *gen_ecm_datagram()* kroz parametar dobija poruku dobijenu od skremblera u kojoj se nalaze ključevi za skremblovanje audio/video podataka. Pored ključeva za skremblovanje, potrebno je dobaviti ključeve za enkriptovanje ECM poruke koji se dobijaju slanjem ispravnog HTTP zahtjeva ka WLS.

Pozivom funkcije *CreateEntitlementRequest()* koja je dio Widevine SDK kreira se zahtjev za ključevima za enkriptovanje ECM poruke. Da bi se kreirao ispravan zahtjev potrebno je funkciji proslijediti sledeće podatke: identifikator sadržaja za skremblovanje, naziv operatera, broj ključeva za skremblovanje (jedan, ili dva), rezolucija sadržaja, ime operatera koji potpisuje zahtjev za licencom, ključ za potpisivanje enkriptovanog zahtjeva i vektor za potpisivanje zahtjeva. Nakon što je zahtjev ispravno kreiran korištena je CURL biblioteka[8] kako bi se poslao HTTP zahtjev ka WLS, nakon čega se odgovor upisuje u željeni dio memorije.

Nakon dobijenog odgovora, poziva se funkcija *ParseEntitlementResponse()* koja iz sirovog odgovora izvlači dva ključa za enkriptovanje ECM poruke. Pozivom funkcije *GenerateEcm()* kojoj se kao parametri proslijeđuju ključevi za enkriptovanje ECM poruke kreira se ECM poruka, da bi se na kraju pozivom *GenerateTsPacket()* kreirao paket koji se šalje ka skrembleru.

IV. Testiranje

U paraleli sa izradom ECM generatora, kreirano je i rješenje posrednika u sistemu uslovnog pristupa (CAS Proxy), te je integrisana Widevine OEMCrypto biblioteka u Android STB uređaj. Nakon što na Android STB uređaj pristigne skremblovan audio/video sadržaj, on posredniku u sistemu uslovnog pristupa šalje zahtjev za licencom, sa informacijom o kom sadržaju je riječ. Ukoliko dobije odgovor od posrednika, licenca se proslijeđuje OEMCrypto biblioteci koja iz licence izvlači ključeve za dekriptovanje ECM poruke. Ukoliko se poruka uspješno dekriptuje, ključevima dobijenim iz ECM poruke se deskrembluje audio/video sadržaj, te je na ekranu moguće vidjeti audio/video sadržaj koji je poslat na STB uređaj. Predloženo rješenje je testnirano na Synaptics BG5CT STB (Sl. 5.) uređajima sa operativnim sistemom Android Q. Korištena je Live Channels korisnička aplikacija koja se oslanja na Comedia DTV (eng. Digital Television) srednji sloj kompanije iWedia.

Sl. 5. Synaptics BG5CT platforma

Pošto su STB uređaji uspješno deskremblovali i reprodukovali audio/video sadržaj skremblovan ključevima generisanim u TS Duck alatu, te ECM porukama enkriptovanim ključevima dobijenim od WLS, konstatovali smo da je testiranjem utvrđena funkcionalnost rješenja.

V. Zaključak

U ovom radu je prikazano jedno rješene ECM generatora u Widevine CAS sistemu. U uvodu je objašnjena uloga i značaj CAS sistema, kao i njegova komercijalna primjena. Bliže je opisan način zaštite televizijskog sadržaja u CAS sistemima, Prikazan je opis rješenja. Rješenje je testirano na nekoliko prijemnih uređaja, sa nekoliko ulaznih tokova podataka te je potvrđena funkcionalnost sistema. U budućnosti se ovo rješenje može unaprijediti podrškom za kreiranje ECM poruke za više različitih tokova podataka u paraleli.

Literatura

[1] Li Xi and Chen Xin, "Design of Digital Video Broadcasting Conditional Access System Headend Communication Interface," in *Computer and Modernization*, vol. 1, no. 3, pp. 118-121, 2012.

[2] In-Hee Jo and Byoung-Soo Koh, " Building a common encryption scrambler to protect paid broadcast services," in *International Journal of Internet Technology and Secured Transactions*, vol. 6, no. 3, Nov. 2016, doi: 10.1504/IJITST.2016.080391.

[3] Milan Bjelica, Nikola Teslić, Velibor Mihić, "Softver u digitalnoj televiziji 1", 2017.

[4] Google Unveils First Android TV Device, Jun 2021. [online]. https://www.nexttv.com/news/google-unveils-first-android-tv-device-384772

[5] Android TV OS reaches 80M monthly active devices, Jun 2021. [online]. https://techcrunch.com/2021/05/18/android-tv-os-reaches-80m-monthly-active-devices-adds-new-features/

[6] TSDuck, Jun 2021. [online]. https://tsduck.io/

[7] ECMG/SCS protokol, Oktobar 2008. [online]. https://www.etsi.org/deliver/etsi_ts/103100_103199/103197/01.05.01_60/ts_103197v010501p.pdf

[8] Curl library, Jun 2021. [online]. https://curl.se/

Abstract

The protection of television content is one of the biggest challenges in the digital television industry due to the declining number of free-to-air television channels. In order to enable the charging of television content to operators, it is necessary to protect television content throughout the transmission. The most widely used model for the protection of live television content is the CAS (Conditional Access System). The CAS model involves a process of protecting video and audio content by scrambling that aims to prevent unauthorized reproduction of audio and video content. The scrambled control words are transmitted via the same transmission channel as the scrambled content within the ECM (Entitlement Control Message) message but in encrypted form. Widevine has implemented its own CAS model completely free for all users. In this paper, one solution of ECM generator in Widevine CAS system is presented.

One solution of ECM generator

Radenko Banovic, Ilija Basicevic, Ksenija Popov, Milenko Maksic

Aplikacija za demonstraciju *XSS* sigurnosnih propusta

Katarina Simić, Žarko Stanisavljević

***Apstrakt* — XSS (eng. *Cross-site scripting*) je jedna od najčešćih ranjivosti veb aplikacija uprkos tome što postoji veliki broj različitih mehanizama zaštite. U ovom radu prikazana je implementacija jedne ranjive aplikacije u okviru koje je moguće demonstrirati različite tipove XSS sigurnosnih propusta, kao i načina njihove zloupotrebe, ali i eliminisanja. Aplikacija se može koristiti kao edukativno sredstvo za praktičnu obuku softverskih inženjera u zatvorenom i bezbednom okruženju.**

***Ključne reči* — XSS, OWASP top 10, sigurnosni propusti.**

I. Uvod

Važnost veb aplikacija posebno je došla do izražaja u trenutku pandemije koronavirusa koja je počela 2019. godine i još uvek traje. Od tada ljudi sve više završavaju svoje poslove i obavljaju određene aktivnosti, poput online kupovine ili korišćenja društvenih mreža, uz pomoć veb aplikacija. Na ovaj način korisnici na različitim mestima ostavljaju svoje poverljive podatke, verujući veb aplikacijama da oni neće pasti u pogrešne ruke. Iz ovog razloga je veoma bitno da svaka aplikacija u svakom trenutku bude zaštićena od različitih tipova napada i pokušaja krađe osetljivih podataka, i na taj način zadobije i zadrži poverenje svojih korisnika.

Važan deo bezbednosti veb aplikacija jeste koncept polise zajedničkog porekla (eng. *same-origin policy*, *SOP*) [1]. Zahvaljujući ovom mehanizmu, skripte jedne veb stranice mogu da pristupe podacima druge veb stranice samo ako su istog porekla. Dva *URL*-a su istog porekla ako su im protokol, host i port identični. Na ovaj način sprečeno je da napadači preko svojih zlonamernih veb aplikacija dođu u posed osetljivih podataka smeštenih na nekom drugom veb sajtu. Zbog toga su napadači morali da osmisle nove načine kako mogu doći do korisničkih podataka, a da pritom zaobiđu polisu zajedničkog porekla.

XSS [2][3] je jedan od bezbednosnih propusta koji zaobilazi polisu zajedničkog porekla. Jedan je od retkih napada koji se iznova nalaze na *OWASP*-ovoj godišnjoj listi top 10 bezbednosnih propusta [4] i gotovo da ne postoji veliki veb sajt koji u nekom trenutku nije bio ranjiv na ovaj napad. *XSS* podrazumeva umetanje klijentskih skripti u ranjivu aplikaciju, koje su kasnije dostupne korisniku nakon učitavanja određenih veb stranica te aplikacije. *XSS* bezbednosni propust je i dalje popularan i zastupljen na velikom broju veb aplikacija. Razlog tome često može biti neiskustvo, neupućenost i neobazrivost programera koji izrađuju veb aplikacije, kao i nedostatak testiranja aplikacija na propuste prilikom svake velike izmene ili nadogradnje aplikacije.

U ovom radu prikazana je implementacija i način korišćenja ranjive veb aplikacije, na kojoj je moguće na određenim mestima umetnuti zlonamerne skripte i izvršiti neku od zlonamernih akcija na štetu regularnog korisnika. Cilj ove aplikacije je jednostavna demonstracija nekih od najčešće primenjenih i praktičnih *XSS* napada, koja bi na taj način pomogla korisniku da bolje razume kada i kako ti napadi mogu da se dese, kao i na kojim mestima u aplikaciji. Nakon korišćenja aplikacije, korisnik bi trebao da razume osnovne koncepte *XSS* napada, kao i da bude u stanju da primeni odgovarajuće mere zaštite, koristeći naučeno, prilikom izrade sopstvene aplikacije.

U drugom poglavlju se opisuju detalji *XSS* sigurnosnog propusta. U trećem poglavlju je prikazan razvoj aplikacije koji se koristi u demonstrativne svrhe *XSS* napada, uz detaljan opis korišćenih tehnologija. U četvrtom poglavlju je opisan rad aplikacije i način korišćenja aplikacije. U petom poglavlju je dat zaključak.

II. *XSS*

Pojavom *JavaScript* programskog jezika sredinom devedesetih godina prošlog veka omogućen je veliki napredak u izradi veb aplikacija, koje su sada mogle biti i interaktivne. Ali, pored svih dobrih i interesantnih mogućnosti koje su sada bile dostupne, pojavile su se i one loše koje mogu uticati negativno po korisnika, poput *XSS* napada. Prvobitno se *XSS* napadu nije pridavalo mnogo pažnje, jer su serveri bili izazovnija i interesantnija meta napadačima. Ali tokom godina situacija se preokrenula. Serveri su vremenom postajali mnogo zaštićeniji nego ranije i bilo je sve teže probiti njihovu zaštitu. Uvidelo se i da serveri nisu bili neophodni za izvršavanje napada sa klijentske strane. Pojavljivali su se različiti pretraživači koji izvršavaju klijentski kod, svaki sa svojim propustima u zavisnosti od verzije, što je programerima dodatno otežavalo posao zaštite. Sa druge strane, programeri zbog manjka vremena ili budžeta, kao i manjka iskustva i znanja, ne posvećuju dovoljno pažnje bezbednosnim propustima, te ih je veoma lako i napraviti. Zbog svega ovoga se *XSS* danas smatra za jedan od najopasnijih i najučestalijih napada. Dve trećine svih veb aplikacija imaju *XSS* propuste u sebi, i svaka velika i popularna aplikacija je u nekom trenutku imala ovaj propust.

Primarni cilj napadača jesu korisnici ranjivih aplikacija. Napadi najčešće podrazumevaju krađu sesije, preuzimanje osetljivih podataka, izvršavanje nedozvoljenih akcija u ime korisnika, dostavljanje zlonamernih softvera korisniku (eng. *malware*), pa čak i narušavanje zaštite aplikacije od drugih napada. Osim što ovi napadi oštećuju same korisnike, mogu

Katarina Simić je student master studija na Elektrotehničkom fakultetu, Univerziteta u Beogradu, Bul. kralja Aleksandra 73, 11120 Beograd, Srbija (e-mail: sk193473m@etf.bg.ac.rs).

Žarko Stanisavljević radi na Elektrotehničkom fakultetu, Univerziteta u Beogradu, Bul. kralja Aleksandra 73, 11120 Beograd, Srbija (telefon: +381-11-3218-484; e-mail: zarko.stanisavljevic@etf.bg.ac.rs).

da imaju i destruktivne posledice po samu aplikaciju, ponajviše zbog gubitka poverenja od strane korisnika ako aplikacija zahteva visok nivo bezbednosti zbog veoma osetljivih i bitnih korisnikovih podataka podeljenih sa tom aplikacijom.

Postoji nekoliko varijacija *XSS* napada, koje se mogu podeliti u tri glavna tipa: reflektujući, snimljeni i *DOM* bazirani *XSS* napad.

A. Reflektujući XSS napad

Reflektujući *XSS* napad (eng. *Reflected XSS attack*) [5] je najzastupljeniji od sva tri tipa. Pod ovim napadom se podrazumeva da se umetnuta, zlonamerna skripta pošalje na server kao deo zahteva i zatim odmah reflektuje u korisnikovom pretraživaču u vidu odgovora koji sadrži tu skriptu. Dakle, podatak poslat serveru je vraćen i prikazan na stranici bez ikakve prethodne provere tog podatka. Da bi uspešno sproveo ovaj napad, napadač prvo mora da osmisli *URL* koji će sadržati zlonamernu skriptu. Zatim će taj *URL* proslediti korisnicima na lukav način. Ako neko od korisnika ništa ne posumnja, zahtevaće *URL* od aplikacije i server će vratiti odgovor koji će sadržati i napadačevu skriptu. Korisnikov pretraživač će sada, između ostalog, izvršiti i napadačevu skriptu i ukradeni podaci se šalju na napadačev server i postaju mu dostupni. Ovakva vrsta napada se izvršava samo ako korisnik otvori napadačev *URL*, te je tako ovaj napad često jednokratan. Da bi napadač naveo korisnika da nasedne i otvori sastavljen *URL*, mora da se posluži lukavim trikovima. Ako su mu meta individualne osobe, napadač će im zamaskirani *URL* upotpunjen uverljivom porukom poslati direkno (npr. preko *e-mail* poruke), tako da korisnik poželi da taj *URL* i otvori. Ako mu je meta veći broj ljudi, zamaskirani *URL* će postaviti na nekim drugim veb stranicama u vidu linka, te će čekati da neko taj link i otvori.

B. Snimljeni XSS napad

Snimljeni XSS napad (eng. *Stored XSS attack*) [6] je najopasniji tip *XSS* napada, zato što može da ima značajnije posledice po veći broj korisnika. Za razliku od reflektujućeg napada, zlonamerna skripta se čuva na serveru, tako da će se svakom korisniku, koji od servera bude zahtevao stranicu sa umetnutom skriptom, u pretraživaču ta skripta i izvršiti. Ovi napadi su najčešći na sajtovima gde korisnici imaju vid međusobne komunikacije (forumi, komentari, pitanja korisnika, itd.). Za izvršavanje ovog napada napadač ne mora da podmeće korisnicima direktno sastavljen *URL*, već je dovoljno da sam umetne skriptu u aplikaciji. Pošto će ona biti sačuvana na serveru, biće dostupna svakom korisniku te stranice. Ovaj napad će najverovatnije dati bolje rezultate u odnosu na reflektujući, jer da bi napadač ukrao korisnikove osetljive podatke, korisnik u najvećem broju slučajeva mora biti ulogovan, što će verovatno i biti slučaj prilikom izvršavanja snimljenog *XSS* napada, a manje verovatan slučaj prilikom reflektujućeg. Snimljenim napadom je veća i verovatnoća da napadačeva žrtva bude administrator napadnute veb aplikacije, što znači da cela aplikacija može biti komprimitovana i ugrožena.

C. DOM bazirani XSS napad

Za razliku od reflektujućeg i snimljenog *XSS* napada, za čije je izvršavanje neophodno vraćanje zlonamerne skripte sa servera, *DOM* (eng. *Document Object Model*) bazirani *XSS* napad [7] će se izvršiti bez da server vrati skriptu prosleđenu u *URL*-u. Ovo je moguće zato što *JavaScript* može pristupiti *DOM*-u i samim tim može dohvatiti i parametre *URL*-a. To znači da će zlonamerni kod biti preuzet sa *URL*-a i obrađen u *JavaScript* kodu. Ovakva vrsta napada ima više sličnosti sa reflektujućim *XSS* napadom nego snimljenim, jer zahteva od napadača da sastavljen *URL* na razne načine podmetne korisnicima, ali je zbog svoje prirode znatno opasniji. Razlika u odnosu na reflektujući *XSS* napad jeste što će *JavaScript* kod procesirati napadačev *URL*, pa samim tim i napadačevu skriptu umetnutu u *URL*, tako da bilo šta što će server vratiti kao odgovor nije važno prilikom ovog napada. Najveći problem kod ovog propusta jeste naći uzrok zbog kojeg nastaje, a pošto se taj uzrok može naći bilo gde u klijentskom kodu, programer bi morao dobro da poznaje projekat prilikom istrage.

D. Zaštita od XSS napada

Nakon upoznavanja sa XSS propustima i uviđanja njihovih mogućnosti i posledica po korisnike, naredni korak za programere bi bio da svoje aplikacije od istih i zaštite. S obzirom da je malo potrebno da se propusti naprave, neophodno je redovno testirati aplikacije na njih prilikom svake nadogradnje koda. Radi efikasnije zaštite aplikacija preporučljivo je primeniti metode poput validacije *input*-a [8], validacije *output*-a [9], konfiguracije aplikacije tako da vraća *Content-Security-Policy* zaglavlje [10], zabrane unosa korisničkih podataka na potencijalno opasnim mestima u okviru aplikacije [11], kao i korišćenja odgovarajućih zaglavlja odgovora koji bi mogli da detektuju *HTML* ili *JavaScript* kod u *HTTP* odgovorima, i samim tim spreče njihovo ubrizgavanje, poput *X-XSS-Protection* zaglavlja [12].

III. Implementacija aplikacije

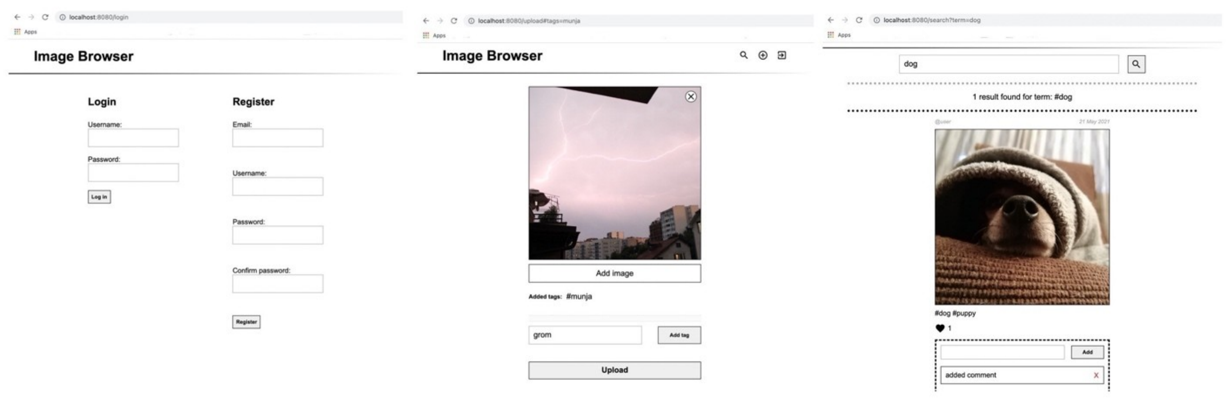
Implementirana aplikacija se sastoji iz dva dela. Prvi deo čini jednostavna, ali ranjiva veb aplikacija, koja služi za isprobavanje različitih *XSS* napada na različitim mestima u okviru te aplikacije. Drugi deo čini napadačev server, na koji pristižu ukradeni podaci nakon uspešno izvršenih napada.

A. Implementacija ranjive aplikacije

Prvi deo alata za učenje predstavlja jednostavnu veb aplikaciju za pretraživanje i dodavanje slika pod nazivom *ImageBrowser* (Sl. 1).

Na početku se od korisnika traži da se registruje ili uloguje na aplikaciju. Nakon što se korisnik uloguje, prikazuje mu se stranica sa porukom dobrodošlice. Nakon toga korisniku su dostupne dve različite stranice sa akcijama. Na prvoj stranici korisnik može da dodaje svoje slike uz koje ostavlja i određene tagove – reči koje služe za opisivanje slike. Na drugoj stranici korisnik može da, uz pomoć *input* polja, pretražuje slike na osnovu postojećih tagova, da pretražuje druge korisnike aplikacije dodajući simbol @ ispred korisničkog imena, kao i da prikazane slike komentariše i na njih reaguje.

Sama aplikacija ima minimalan skup potrebnih funkcionalnosti, ali i namerno napravljene *XSS* bezbednosne propuste na više mesta radi demonstracije nekih od *XSS* napada.

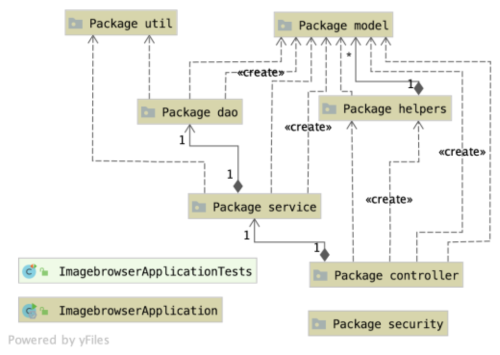


a) b) c)

Sl. 1. Izgled ranjive aplikacije u pretraživaču: a) početna stranica, b) stranica za dodavanje slika i c) stranica za pretraživanje slika

Da bi korisnik mogao da dodaje i pretražuje slike, kao i da ostavlja komentare i reakcije, određeni podaci moraju da se čuvaju u bazi podataka. Aplikacija koristi *H2 Java in-memory* bazu [13], koja omogućava da se prilikom svakog pokretanja aplikacije koristi inicijalno stanje podataka baze i da se sve dotadašnje izmene gube, što omogućava lakše testiranje aplikacije.

Za implementaciju servera je korišćen razvojni okvir *Spring* [14], kao i *SpringBoot* [15] koji koristi *Spring* kao podlogu. U projektu se koristi i *Apache Maven* [16], alat za izvršavanje `compile` i `build` naredbi *Java* koda. *Spring Initializer* [17] je projekat otvorenog koda (eng. *open source*), koji omogućava generisanje konfigurisanog *Spring* projekta uz odabir potrebnih zavisnosti (eng. *dependencies*). *Java* kod je grupisan po paketima, od kojih je okružujući *application.imagebrowser* u okviru koga se pored drugih nalazi i *ImagebrowserApplication,* glavna klasa aplikacije (Sl. 2).



Sl. 2. *UML* dijagram paketa unutar *application.imagebrowser* okružujućeg paketa

Unutar ovog paketa se nalazi nekoliko drugih paketa, koji sadrže sav potreban kod za uspostavljanje komunikacije i ispravan rad servera sa bazom podataka, obradu pristiglih zahteva na serveru i vraćanje odgovarajućih stranica i podataka koji se učitavaju na stranici sa kojima će korisnik interagovati.

Klijentska strana sadrži kod koji se izvršava u pretraživaču, i sadrži sve što korisnik može da vidi i sa čime može da interaguje. Najbitnije i najosnovnije tehnologije koje su korišćene prilikom izrade klijentskog dela aplikacije su *HTML* (eng. *Hyper Text Markup Language*), *CSS* (eng. *Cascading Style Sheets*) i *JavaScript*. U okviru posmatrane aplikacije se koriste *JSP* (eng. *Java Server Pages*) stranice [18] koje podržavaju dinamički sadržaj, što omogućava umetanje *Java* koda unutar *HTML* koda uz pomoć specijalnih *JSP* tagova. *JSP* komponenta predstavlja *servlet* koji ispunjava ulogu korisničkog interfejsa za *Java* veb aplikaciju. Radi olakšanog i preglednijeg fajla, uz *JSP* se koristi i biblioteka *JSTL* (eng. *The JSP Standard Template Library*) [19], koja omogućava da se *Java* kod zameni tagovima koji će raditi identičan posao. Fragmenti *JSP* koda koji se mogu koristiti na više mesta su smešteni u zasebne fajlove, koji se uz pomoć tagova učitavaju na određenoj *JSP* stranici. *JSP* fajlovi u aplikaciji koji čine stranice su:

- *login.jsp,* gde korisnik može da se uloguje ili registruje,
- *index.jsp*, koji predstavlja glavnu stranicu prikazanu korisniku nakon što se uloguje,
- *upload.jsp*, gde korisnik može da dodaje slike i tagove,
- *search.jsp*, gde korisnik može da pretražuje slike po tagovima ili drugim korisnicima, kao i
- *searchResults.jsp* i *noResults.jsp*, dve stranice koje server vraća kao rezultat *AJAX* [20] poziva, prva sa rezultatima, i druga sa informacijom da rezultata nema.

B. Implementacija napadačevog servera

Drugi deo aplikacije čini napadačev server, na kojem će se obrađivati pristigli zahtevi, tačnije prethodno dohvaćeni osetljivi podaci korisnika. Da bi zahtevi uopšte mogli da pristignu na server, napadač mora da sastavi *URL* koji će u sebi sadržati zlonamerni kod za dohvatanje podataka i za redirekciju. Nakon dohvatanja podataka, napadač može da ponovo izvrši redirekciju nazad ka napadnutoj aplikaciji tako da korisnik ni ne posumnja da je bio napadnut.

U okviru ove aplikacije je napadačev server implementiran u *Node.js* [21] platformi koja predstavlja asinhrono *runtime* okruženje za *JavaScript* jezik, i koja omogućava stvaranje skalabilnih veb aplikacija, samim tim i izvršavanje *JavaScript* koda van pretraživača.

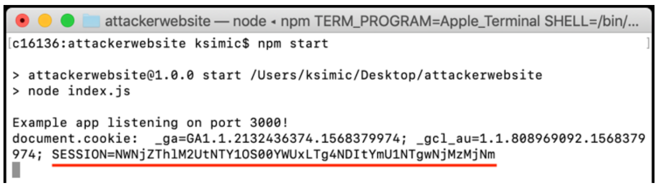
U slučaju napadačevog projekta je instaliran paket *Express* [22], koji predstavlja radni okvir za organizaciju aplikacije prema *MVC* arhitekturi. Pomoću *Express*-a se mogu na jednostavan način obrađivati pristigli zahtevi. Napravljen je jedan *JavaScript* fajl, *index.js*, u kojem se, prilikom izvršavanja koda, pokreće server koji osluškivanjem čeka na zahteve i obrađuje one koji su pristigli. Svaki zahtev se obrađuje tako da se u terminalu napadača gde je pokrenuta skripta ispišu pristigli podaci, a zatim po potrebi izvrši i redirekcija. Dokle god napadačev server radi, moći će da obrađuje pristigle zahteve.

IV. NAČIN KORIŠĆENJA APLIKACIJE

U ovom poglavlju je dat prikaz nekoliko tipičnih scenarija *XSS* napada, gde je prvo objašnjen cilj napada, uz priložene zlonamerne skripte za ispunjenje tog cilja, a zatim je na kraju svakog primera dat savet za sprečavanje tog napada. Bitno je napomenuti da su ovakve vrste napada kažnjive zakonom svuda u svetu (npr. u Srbiji prema Krivičnom zakoniku Republike Srbije (Članovi 298 do 304a)) ukoliko se sprovode prema aplikacijama fizičkih i pravnih lica koja nisu upoznata i saglasna sa aktivnostima na proveri ranjivosti.

A. Krađa korisnikove sesije

Nakon što se korisnik uspešno uloguje na aplikaciju, server će poslati kolačić sesije preko *Set-Cookie* zaglavlja. Taj kolačić će se sada slati ka serveru uz svaki korisnikov zahtev. Zbog toga je kolačić sesije izuzetno osetljiv podatak, jer ako napadač nekako uspe da dođe do njegove vrednosti moći će da šalje zahteve ka serveru u ime oštećenog korisnika. *HttpOnly* predstavlja deo *Set-Cookie* zaglavlja u vidu *flag*-a. Ako je taj *flag* postavljen, to će sprečiti klijentske skripte da pristupe vrednostima kolačića. U slučaju da taj *flag* nije postavljen, krađu sesije je moguće izvesti. U slučaju ove aplikacije *flag* nije postavljen, tako da je moguće pristupiti vrednosti kolačića sesije u *JavaScript*-u preko *document.cookie* atributa (Sl. 3).



attackerwebsite — node ◂ npm TERM_PROGRAM=Apple_Terminal SHELL=/bin/...

```
c16136:attackerwebsite ksimic$ npm start

> attackerwebsite@1.0.0 start /Users/ksimic/Desktop/attackerwebsite
> node index.js

Example app listening on port 3000!
document.cookie:  _ga=GA1.1.2132436374.1568379974; _gcl_au=1.1.808969092.1568379
974; SESSION=NWNjZThlM2UtNTY1OS00YWUxLTg4NDItYmU1NTgwNjMzMjNm
```

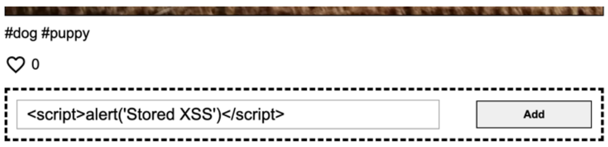
Sl. 3. Prikaz ukradenog kolačića sesije u terminalu napadača

1) Krađa kolačića sesije - reflektujući XSS napad

Na *search* stranici aplikacije postoji reflektujući *XSS* propust. Prilikom pretrage slika, uneti termin postaje *URL* parametar, i prilikom vraćanja rezultata od strane servera se vraća i pretražen termin koji se dodaje na stranicu. Validacija na tim osetljivim tačkama nije realizovana, samim tim korisnik umesto termina može da ukuca skriptu unutar `<script>` taga. Napadač takođe proverava na svojoj mašini da li *document.cookie* vraća njegovu tekuću sesiju. Nakon što utvrdi da vraća, napadač može da sastavi *URL* koji sadrži zlonamerni kod.

2) Krađa kolačića sesije - snimljeni XSS napad

U aplikaciji postoji i snimljeni *XSS* propust, tako da napadač može u komentare da ubacuje zlonamerni kod. Ako je situacija ista kao kod reflektujućeg *XSS* propusta, napadač sada može isti zlonamerni kod da doda kao komentar (Sl. 4). To znači da će svakom korisniku kojem se taj komentar bude učitao na stranici biti ukraden kolačić sesije. Ovo je suptilniji način za prevaru korisnika, te je veća verovatnoća da će korisnici biti prevareni ovom metodom nego da su kliknuli na *URL* primljen od napadača u slučaju reflektujućeg *XSS* napada.



#dog #puppy

♡ 0

<script>alert('Stored XSS')</script>

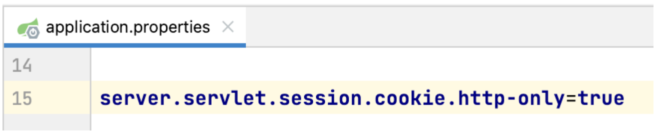
Sl. 4. Trenutak dodavanja napadačeve skripte u komentar

3) Krađa kolačića sesije - DOM bazirani XSS napad

Na *upload* stranici aplikacije postoji *DOM* bazirani *XSS* propust. Korisnik može da dodaje tagove koji se pridodaju slici, i kako se neki tag doda on postaje deo *URL*-a u vidu `href` parametra. Problem je što se taj deo *URL*-a nikada ne šalje na server i ne obrađuje, ali se prilikom učitavanja stranice sa takvim *URL*-om tagovi automatski dodaju. To znači da se taj deo *URL*-a obradio negde u *JavaScript* kodu.

4) Način sprečavanja napada

Najlakši i najefikasniji način sprečavanja ovog napada jeste jednostavno podesiti *HttpOnly flag*, koji u tom slučaju sprečava klijentske skripte da dohvate podatke o kolačićima (Sl. 5). Na ovaj način *document.cookie* će uvek vratiti praznu vrednost i kolačić sesije će ostati bezbedan. Iako ovaj mehanizam odbrane od krađe kolačića funcioniše, to ne znači da napadač na istom mestu ne može da izvrši druge zlonamerne akcije. Ovaj *flag* se setuje na različite načine, u zavisnosti od korišćenog programskog jezika.



application.properties

```
server.servlet.session.cookie.http-only=true
```

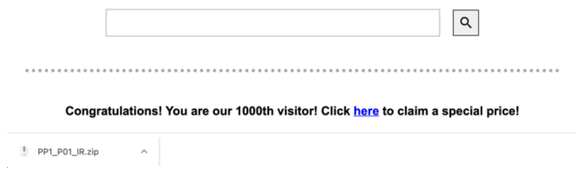
Sl. 5. Postavljanje *HTTPOnly flag*-a radi sprečavanja krađe kolačića sesije

B. Umetanje napadačevog koda na stranicu

U prethodnim primerima je data jednostavna skripta koja izvršava redirekciju i prosleđuje kolačić sesije napadaču, ali ako je primenjena navedena tehnika zaštite od tog napada, napadač mora da nađe drugi način da naškodi korisniku. Još jedna tehnika jeste umetanje *HTML* koda na stranicu, kojem se dodeljuju stilovi tako da izgleda kao da je zapravo deo stranice. Ako je kod dovoljno uverljiv, može da uveri korisnika da uradi određene akcije vođene tim kodom. U naredna dva primera se može videti kako umetanjem *HTML* koda napadač može da ukrade kredencijale korisnika, i kako može da navede korisnika da preuzme sumnjiv sadržaj na svoju mašinu.

1) Umetanje koda za preuzimanje sumnjivih fajlova

Ponekad napadaču nije samo cilj da ukrade korisnikove podatke, već i da ga navede da preuzme određeni fajl. To postiže umetanjem linka, na čiji klik se započinje preuzimanje nekog fajla. U zavisnosti od toga šta je cilj napadača, taj fajl može da ima nikakav ili razoran uticaj na korisnikovu mašinu. Dovoljno je samo sastaviti taj link dovoljno uverljivim da navede korisnika da klikne na njega, tako da će napadač i ovde uneti *inline* stilove, kao i uverljiv tekst (Sl. 6).

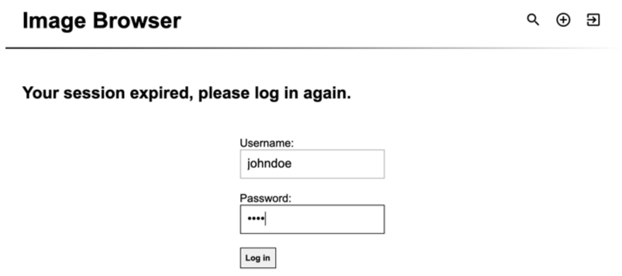


Congratulations! You are our 1000th visitor! Click here to claim a special price!

Sl. 6. Prikaz stranice za pretraživanje slike sa umetnutim linkom za preuzimanje sumnjivih fajlova

2) Phishing tehnika

Iako kolačići nakon primenjene zaštite ne mogu biti ukradeni, *XSS* propust i dalje postoji na istim mestima. Napadač uviđa da može da umetne skriptu koja sa stranice briše ceo *HTML* kod i zameni ga svojim. U ovom primeru je za brisanje i dodavanje koda korišćen *Jquery* (tačnije *.empty* i *.append* funkcije). Kod koji se dodaje predstavlja lažnu formu za unos korisničkog imena i šifre. Ta taktika umetanja ovakve vrste koda gde korisnik „dobrovoljno“ ostavlja lične podatke napadaču se zove *phishing*. Napadač dodaje tekst da uveri korisnika da treba tu formu da popuni, na primer saopšti korisniku da mu je istekla sesija i da mora ponovo da se uloguje (Sl. 7).



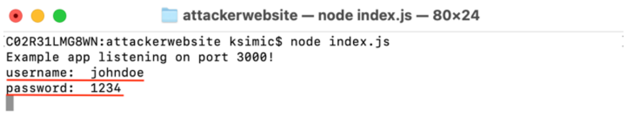
Your session expired, please log in again.

Username: johndoe

Password: ••••

Sl. 7. Prikaz stranice za pretraživanje slike sa umetnutom formom

Ako korisnik ništa ne posumnja, može uneti svoje lične podatke. Klikom na dugme za logovanje se zapravo izvršava redirekcija ka napadačevom serveru i štampaju se podaci u napadačevu konzolu i na kraju ponovo dešava redirekcija nazad ka aplikaciji. Napadač sad ima korisnikovo ime i šifru, što može da zloupotrebi na sličan način kao i prilikom krađe sesije (Sl. 8).



```
C02R31LMG8WN:attackerwebsite ksimic$ node index.js
Example app listening on port 3000!
username:  johndoe
password:  1234
```

Sl. 8. Prikaz ukradenih kredencijala u terminalu napadača pomoću *phishing* tehnike

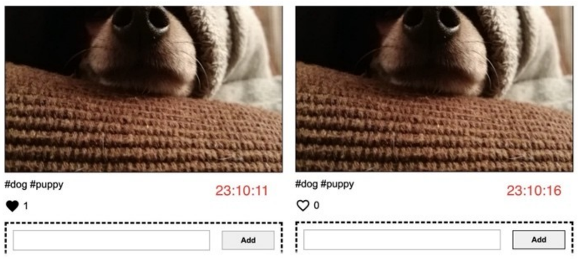
3) Način sprečavanja napada

U prethodnim primerima nije bilo potrebno raditi dodatnu validaciju zbog prirode napada. Ali u ovom slučaju bi bilo poželjno uraditi validaciju i *input*-a i *output*-a, na klijentskoj i na serverskoj strani. Prilikom pretrage se dešava *AJAX* poziv, pa je poželjno da se pre toga uradi sanitizacija *input*-a, najverovatnije uz pomoć regularnog izraza. U slučaju da *input* ne ispunjava zahteve, *AJAX* poziv se neće izvršiti i korisniku se ostavlja poruka da zna da je *input* polje bilo neispravno popunjeno. Sledeći korak je uraditi validaciju u kontroleru prilikom obrade zahteva i vraćanja odgovarajuće stranice. Tu bi najbolje rešenje bilo korišćenje gotovih metoda za zamenu (*escaping*) *HTML* koda. Ako je ipak potrebno dozvoliti određene tagove ili specijalne karaktere, metode moraju ručno da se pišu uz veliki oprez. Na primer, ako korisnik zameni `<script>` tag praznim stringom bez rekurzije, napadač može da sastavi skriptu sa *script* tagom `<scr<script>ipt>`. Takođe mora da se vodi računa da se obrade i *uppercase* i *lowercase* karakteri. Bitno je određene karaktere i stringove menjati u celoj skripti, a ne samo po prvom pojavljivanju. Zlonameran kod može biti prosleđen i kao atribut nekog dozvoljenog taga, pa je i za to potrebna validacija. Nakon što programer utvrdi sve šta mu je potrebno za validaciju na serveru, može da uradi i validaciju *output*-a. Najlakši način uraditi to u okviru ove aplikacije jeste koristeći *JSTL* tag `<c:out>`, jer ovaj tag omogućava *escaping* vraćenog koda. Na kraju se uz ove tehnike postiže visok nivo zaštite od ovakvog *XSS* napada.

C. Izvršavanje nedozvoljenih radnji u ime korisnika

Osim krađe podataka, napadač može pomoću zlonamernog koda i da izvrši neku radnju na stranici u ime korisnika. U tom slučaju nema potrebe da išta radi na svom serveru, već samo da pripremi kod koji će se izvršiti. Ako je cilj napadača da što više korisnika ošteti, to bi najbolje postigao uz snimljen *XSS* napad.

Skripta za ovaj primer je sastavljena tako da se u određenom vremenskom intervalu daju ili sklanjaju korisnikove reakcije na slike (Sl. 9). Iako ovaj napad nema veće posledice po korisnika, sam napad može da izazove nelagodnost i zbunjenost. Ovo je samo jedan primer izvršavanja nedozvoljenih akcija u skladu sa datim alatom, ali i u ovom slučaju zlonamerne skripte mogu da izazovu i znatno veće posledice, posebno ako korisnik nije ni svestan da se nešto desilo.



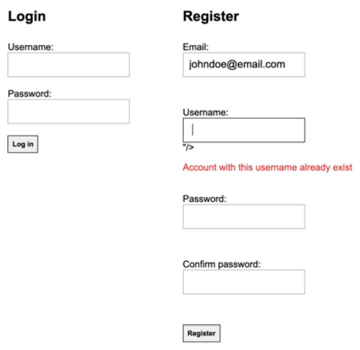
Sl. 9. Prikaz izvršavanja napadačeve skripte u ime korisnika u toku određenog vremenskog intervala

Iako je cilj napadača različit u odnosu na prethodni primer, metoda sprečavanja od *XSS* napada je ista. Potrebno je validirati *input* i *output*, sa klijentske i serverske strane. Poželjno je raditi validaciju na svim osetljivim mestima da bi se smanjio rizik od napada.

D. Keylogger

Još jedan način na koji napadač može da dođe do osetljivih korisnikovih podataka jeste da umetne skriptu koja napadačevom serveru prosleđuje karaktere koje korisnik unosi, karakter po karakter uz tačno vreme unosa. Ova tehnika nadgledanja pojedinačnih karaktera koje korisnik unosi se naziva *keylogger*, i najefikasnija je na stranicama na kojima se unose poverljivi podaci, poput broja kreditne kartice ili kredencijala. Skripta radi tako što pravi ograničen niz karaktera koji se šalje u određenom vremenskom intervalu samo ako je taj niz popunjen.

Na stranici za registraciju namerno je napravljen bezbednosni propust. Kada korisnik prilikom registracije unese već postojeće korisničko ime ili *e-mail*, prilikom osvežavanja stranice će ta informacija biti prisutna u *URL*-u i iz nje biti ispisana u odgovarajuće *input* polje (Sl. 10).



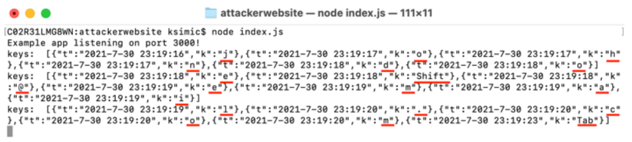
johndoe@email.com

"/>

Account with this username already exist

Sl. 10. Prikaz početne stranice u toku izvršavanja *keylogger XSS* napada

Napadač je otkrio da može svojim zlonamernim kodom da zatvori tag tog *input* polja, i dalje samo izvrši svoju skriptu. *Chrome*, *Safari* i *IE* pretraživači su se u ovom slučaju pokazali otpornim na napad zahvaljujući *X-XSS-Protection* zaglavlju odgovora. Svi korisnici koji se nalaze na ostalim pretraživačima će biti ranjivi. Napadač na ovaj način može da sazna kredencijale korisnika kroz individualne karaktere (Sl. 11).



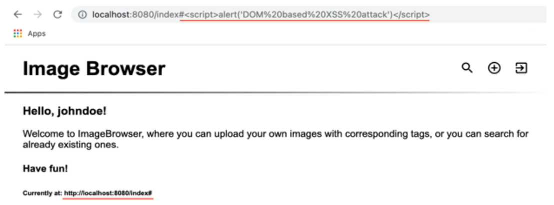
```
C02R31LMG8WN:attackerwebsite ksimic$ node index.js
Example app listening on port 3000!
keys:  [{"t":"2021-7-30 23:19:16","k":"j"},{"t":"2021-7-30 23:19:17","k":"o"},{"t":"2021-7-30 23:19:17","k":"h"
},{"t":"2021-7-30 23:19:17","k":"n"},{"t":"2021-7-30 23:19:18","k":"d"},{"t":"2021-7-30 23:19:18","k":"o"}]
keys:  [{"t":"2021-7-30 23:19:18","k":"e"},{"t":"2021-7-30 23:19:18","k":"Shift"},{"t":"2021-7-30 23:19:18","k"
:"@"},{"t":"2021-7-30 23:19:19","k":"e"},{"t":"2021-7-30 23:19:19","k":"m"},{"t":"2021-7-30 23:19:19","k":"a"},
{"t":"2021-7-30 23:19:19","k":"i"}]
keys:  [{"t":"2021-7-30 23:19:19","k":"l"},{"t":"2021-7-30 23:19:20","k":"."},{"t":"2021-7-30 23:19:20","k":"c"
},{"t":"2021-7-30 23:19:20","k":"o"},{"t":"2021-7-30 23:19:20","k":"m"},{"t":"2021-7-30 23:19:23","k":"Tab"}]
```

Sl. 11. Prikaz ukradenih informacija u terminalu napadača tokom izvršavanja *keylogger XSS* napada

E. Jednostavan primer DOM baziranog XSS propusta

Preporučljivo je zaobići manipulaciju *DOM*-a u klijentskom kodu koliko god je to moguće da bi se smanjile šanse za *DOM* bazirani *XSS* napad. U ovom primeru se namerno na dnu svake stranice nalazi dekodovani *URL* tekuće stranice, koji se i dohvata i dekoduje u *JavaScript* kodu (Sl. 12). Ovde napadač takođe može da umetne skriptu koja će potpuno zaobići server i manipulisati kod koji se izvršava u *JavaScript*-u.

Dekodovani *URL* tekuće stranice se dohvata prilikom učitavanja te stranice u *JavaScript* kodu uz pomoć svojstva *document.URL*. Na ovaj način se vrši manipulacija *DOM* podataka, što otvara mogućnost da stranica bude ranjiva na *DOM* bazirani *XSS* napad. Eliminacijom tog koda problem bi u potpunosti nestao, ali ako je ipak potrebno taj kod i izvršiti, neophodno je uraditi validaciju. Tokom izvršavanja atributa *document.URL*, on se i dekoduje, te je preporučljivo ukloniti tu funkcionalnost. Ako informacija treba da bude dekodovana, treba izvršiti zamenu (eng. *escaping*) nepoželjnih karaktera koji se mogu naći u skripti sa odgovarajućom interpretacijom tog karaktera. Na ovaj način je sprečeno izvršavanje zlonamerne skripte na svim stranicama aplikacije.


localhost:8080/index#<script>alert('DOM%20based%20XSS%20attack')</script>
Apps
Image Browser

Hello, johndoe!

Welcome to ImageBrowser, where you can upload your own images with corresponding tags, or you can search for already existing ones.

Have fun!

Currently at: http://localhost:8080/index#

Sl. 12. Prikaz ranjivosti stranice na kojoj se nalazi dekodovani *URL*

V. Zaključak

U ovom radu predstavljena je jedna ranjiva aplikacija pomoću koje se može izučavati *XSS* sigurnosni propust. Aplikacijom su pokriveni različiti tipovi *XSS* napada. Korišćenjem aplikacije moguće je demonstrirati neke interesantne zloupotrebe ovih propusta. Najvažnije je da korisnici mogu na siguran način u zatvorenom okruženju detektovati propuste, a zatim koristeći tehnike zaštite ispraviti propuste i uveriti se da njihova rešenja ispravno rade. Aplikaciju je takođe moguće nadograditi po potrebi u budućnosti, radi dodavanja novih primera i propusta koji bi korisnicima dodatno omogućili testiranje i učenje o XSS napadu. Aplikacija je korišćena za izvođenje laboratorijskih vežbi na predmetu Zaštita računarskih sistema i mreža na Elektrotehničkom fakultetu u Beogradu, ali efekti korišćenja nisu izmereni usled uslova izvođenja nastave izazvanih pandemijom koronavirusa.

Literatura

[1] Same origin policy, pristupano 21.05.2021, https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
[2] D. Stuttard, M. Pinto, "*The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws",* second edition, John Wiley & Sons Inc, 2011
[3] Cross-site Scripting, pristupano 21.05.2021, https://owasp.org/www-community/attacks/xss/
[4] OWASP Top Ten Project, pristupano 21.05.2021, https://owasp.org/www-project-top-ten/
[5] Reflected XSS Attacks, pristupano 21.05.2021, https://owasp.org/www-community/attacks/xss/#reflected-xss-attacks
[6] Stored XSS Attacks, pristupano 21.05.2021, https://owasp.org/www-community/attacks/xss/#stored-xss-attacks
[7] DOM Based XSS, pristupano 21.05.2021, https://owasp.org/www-community/attacks/DOM_Based_XSS
[8] Input Validation Cheat Sheet, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
[9] XSS Prevention Rules, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-1-html-encode-before-inserting-untrusted-data-into-html-element-content
[10] Content Security Policy (CSP), pristupano 21.05.2021, https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
[11] XSS Prevention Rules, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-0-never-insert-untrusted-data-except-in-allowed-locations
[12] X-XSS-Protection Header, pristupano 21.05.2021, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#x-xss-protection-header
[13] H2 Database, pristupano 21.05.2021, https://www.h2database.com/html/main.html
[14] Spring, pristupano 21.05.2021, https://spring.io/
[15] Spring Boot, pristupano 21.05.2021, https://spring.io/projects/spring-boot
[16] Apache Maven, pristupano 21.05.2021, https://maven.apache.org/
[17] Spring Initializr, pristupano 21.05.2021, https://start.spring.io/
[18] JSP Tutorial, pristupano 21.05.2021, https://www.javatpoint.com/jsp-tutorial
[19] JSTL (JSP Standard Tag Library), pristupano 21.05.2021, https://www.javatpoint.com/jstl
[20] AJAX, pristupano 21.05.2021, https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX
[21] Node.js, pristupano 21.05.2021, https://nodejs.org/en/
[22] Express, pristupano 21.05.2021, https://expressjs.com/

Abstract

XSS (Cross-site scripting) is one of the most common vulnerabilities in web applications, despite the fact that there are many defense mechanisms against it that are available. This paper presents the implementation of a vulnerable application in which different types of XSS vulnerability can be demonstrated, along with the ways they can be misused, but also the ways they can be eliminated. The application can be used as an educational tool for software developer practical training in a closed and safe environment.

AN APPLICATION FOR DEMONSTRATION OF XSS VULNERABILITY

Katarina Simic, Zarko Stanisavljevic

SQLiTrainer - sistem za učenje o SQLi sigurnosnim propustima u aplikacijama

Đorđe Madić, Žarko Stanisavljević

***Apstrakt* — Sigurnosni propusti u aplikacijama koji nastaju prilikom njihovog razvoja i ostaju nedetektovani u produkcionom okruženju mogu dovesti do narušavanja integriteta, poverljivosti i dostupnosti takvih aplikacija. *SQLiTrainer* predstavlja skup ranjivih aplikacija kojima se mogu demonstrirati različite vrste *SQLi (eng. SQL injection)* ranjivosti. U radu je opisan način implementacije *SQLiTrainer* sistema i dati su primeri na koji način se sistem može iskoristiti za praktičnu obuku programera. Sistem je uspešno korišćen za izvođenje laboratorijskih vežbi na predmetu Zaštita računarskih sistema i mreža na Elektrotehničkom fakultetu u Beogradu.**

***Ključne reči* — SQLi, sigurnosni propusti, razvoj bezbednog softvera.**

I. Uvod

Razvoj bezbednog softvera podrazumeva postojanje svesti kod programera o potencijalnim problemima, a zatim i primenu čitavog seta dobrih praksi, kao i automatizovanih alata tokom procesa razvoja softvera. Aplikacije kod kojih postoje sigurnosne ranjivosti koje se mogu zloupotrebiti mogu dovesti do štete kako za korisnike takvih aplikacija, tako i za njihove autore.

Jedan od sigurnosnih propusta koji je često zastupljen u veb aplikacijama je *SQL injection* [1], kod koga se na različite načine na nepredviđen način mogu umetnuti naredbe koje mogu narušiti integritet, poverljivost i dostupnost baza podataka. Ovaj propust se već duži niz godina nalazi na top listama najčešćih sigurnosnih propusta u aplikacijama koje objavljuju organizacije kao što je *OWASP (Open Web Application Security Project)* [2].

U opštem slučaju nije jednostavno omogućiti programerima da kroz praktičan rad unaprede svoje znanje o ovakvim problemima, jer to podrazumeva izučavanje različitih mehanizama kojima se narušava informaciona bezbednost aplikacija na kojima se primenjuju. Primena ovih mehanizama kada se izvršavaju prema sistemima fizičkih i pravnih lica koja nisu upoznata i saglasna sa aktivnostima na proveri ranjivosti i testiranju upada u njihove sisteme je kažnjiva svuda u svetu (npr. u Srbiji prema Krivičnom zakoniku Republike Srbije (Članovi 298 do 304a)). Iz tog razloga postoje različiti sistemi koji omogućavaju svojim korisnicima da u zatvorenom okruženju na praktičan način obave obuku, a da ne prekrše zakon [3-5]. U ovom radu prikazan je jedan novi sistem za učenje o *SQLi* sigurnosnim propustima u aplikacijama.

Đorđe Madić radi u kompaniji Zuehlke Engineering, Bul. Milutina Milankovića 1i, 11070 Novi Beograd, Srbija (e-mail: djordje.madic@zuehlke.com).

U trenutku rada na ovom istraživanju Đorđe Madić je bio student master studija na Elektrotehničkom fakultetu, Univerziteta u Beogradu.

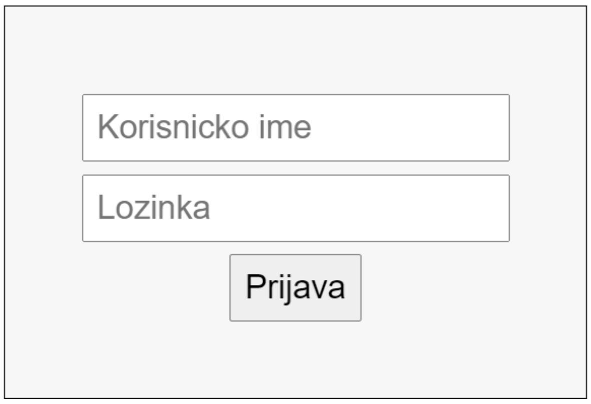
Žarko Stanisavljević radi na Elektrotehničkom fakultetu, Univerziteta u Beogradu, Bul. kralja Aleksandra 73, 11120 Beograd, Srbija (telefon: +381-11-3218-484; e-mail: zarko.stanisavljevic@etf.bg.ac.rs).

U drugom poglavlju su na primeru opisani uzrok i koraci kod izvođenja *SQLi* napada. U trećem poglavlju prikazan je način korišćenja realizovanog sistema na primeru jedne laboratorijske vežbe. U četvrtom poglavlju prikazana je implementacija laboratorijskih vežbi. U petom poglavlju dat je zaključak.

II. *SQL Injection*

OWASP definiše *injection* kao „slanje nepouzdanih podataka interpreteru kao deo komande ili zahteva“. *SQL injection* je tip napada koji se koristi za neovlašćeni pristup *SQL* bazi podataka koju aplikacija koristi. Napadač može čitati osetljive podatke, menjati njihovu strukturu, a u nekim slučajevima i izvršavati komande nad operativnim sistemom baze. Uzrok postojanja propusta je što aplikacija dozvoljava da korisnikov unos učestvuje u kreiranju *SQL* upita, omogućavajući mu da modifikuje originalni upit u svoju korist. U nastavku je na primeru *SQLi Login Bypass* napada objašnjen postupak izvršavanja napada.

SQLi Login Bypass počinje na stranici za prijavu korisnika, kao na Sl. 1. Cilj napada je, u bukvalnom prevodu sa engleskog, „zaobići prijavu“, odnosno prijaviti se ne znajući ni jedno korisničko ime ni lozinku.



Sl. 1. Stranica za prijavu korisnika

Prvi korak napada je analiza aplikacije. Ideja je pronaći ulazne podatke aplikacije koji se potencijalno koriste za kreiranje *SQL* upita. Ti podaci su kandidati da „nose“ napadački upit. U primeru prijave korisnika postoje dva ulazna podatka, korisničko ime i lozinka.

U narednom koraku potrebno je pretpostaviti kako izgleda *SQL* upit i odabrati ulazne podatke za napad. Na primer, upit za prijavu može biti sledeći:

```
SELECT * FROM korisnik
WHERE korisnicko_ime='$korisnicko_ime'
AND lozinka='$lozinka'
```

Delovi upita *$korisnicko_ime* i *$lozinka* su ulazni podaci sa korisničkog interfejsa, a napad se, na primer, može izvršiti kroz *$korisnicko_ime*.

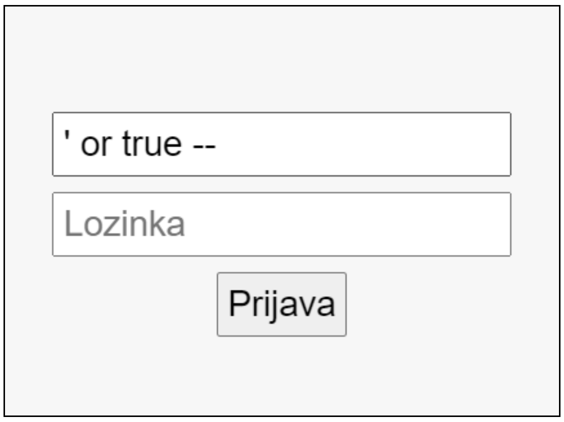
Napad se nastavlja na korisničkom interfejsu. U najčešćem slučaju tekstualni unos u polje za korisničko ime biće tretiran kao podatak, odnosno neće biti interpretiran kao komanda od strane baze podataka. Kako je korisničko ime tekstualnog tipa, u upitu se koristi apostrof za označavanje početka i kraja tekstualnog podatka. U slučaju da se u polje za korisničko ime unese apostrof, karakteri koji prethode tretiraće se kao podatak, dok će se oni koji slede tretirati kao komanda.

Koristeći ovu činjenicu napadač ima priliku da modifikuje originalni upit. Pod pretpostavkom da je prijava korisnika uspešna ukoliko upit vrati bar jedan rezultat, napad se može izvršiti kao na Sl. 2. što rezultuje sledećim upitom:

```
SELECT * FROM korisnik
WHERE korisnicko_ime='' or true --' and
password=''
```

Rezultat upita su svi redovi tabele „korisnik”. Simbol „--" predstavlja oznaku za komentar čime se ignoriše deo upita nakon korisničkog imena.

Uslov *„or true"* čini da svaki red bude deo rezultata, ignorišući korisničko ime.


' or true --
Lozinka
Prijava

Sl. 2. *SQLi* napad na slučaju korišćenja prijave korisnika

Osnovu prevencije *SQLi* sigurnosnog propusta čine parametrizovani upiti (eng. *Parametrized Statement*, *Prepared Statement*). Oni omogućavaju da se bazi prvo prosledi šablon upita koji će se koristiti, a zatim za svako izvršavanje šablona i konkretni podaci. Korišćenje parametrizovanih upita garantuje da konkretni podaci neće biti interpretirani, čime se eliminiše *SQLi* sigurnosni propust. U nastavku je primer korišćenja parametrizovanog upita za pronalazak korisnika sa određenim korisničkim imenom u programskom jeziku *Java*:

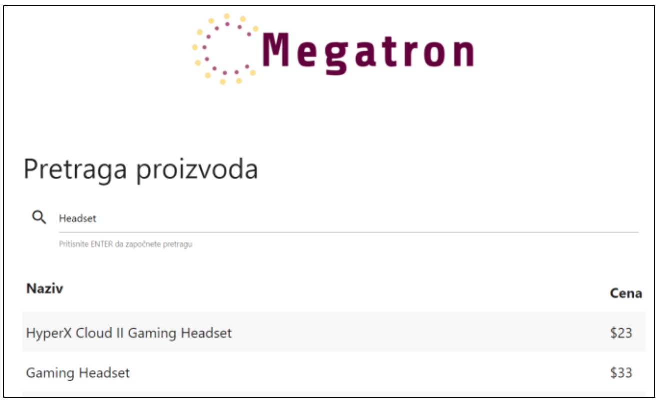
```
String query = "SELECT * FROM korisnik WHERE
korisnicko_ime = ?";
PreparedStatement preparedStatement =
connection.prepareStatement(query);
preparedStatement.setString(1, "petar");
ResultSet results =
preparedStatement.executeQuery();
```

III. Primer korišćenja *SQLiTrainer* sistema

Primer korišćenja realizovanog sistema biće dat kroz prikaz jedne od laboratorijskih vežbi, dok će način upotrebe sistema u nastavi biti prikazan na primeru predmeta Zaštita računarskih sistema i mreža (ZRM) [6] na Elektrotehničkom fakultetu u Beogradu (ETF).

A. *Laboratorijska vežba Megatron*

Laboratorijska vežba *Megatron* zasniva se na aplikaciji koja predstavlja veb prodavnicu kompjuterske opreme. Vežba počinje na stranici za pretragu proizvoda. Pretraga se vrši po nazivu, gde je moguće uneti i samo deo naziva proizvoda. Rezultat pretrage prikazuje se u vidu tabele, sa kolonama za naziv i cenu proizvoda, kao na Sl. 3.


Megatron

Pretraga proizvoda

Headset

Pritisnite ENTER da započnete pretragu

Naziv	Cena
HyperX Cloud II Gaming Headset	$23
Gaming Headset	$33

Sl. 3 Pretraga prozvoda

Cilj laboratorijske vežbe je pronaći korisničko ime i lozinku svih korisnika aplikacije. Boduju se i sledeće informacije:

- Naziv baze koju koristi aplikacija
- Verzija baze
- Nazivi tabela
- Nazivi kolona

Napad se izvršava kroz polje za pretragu proizvoda, dok se podaci koji su rezultat napada prikazuju u tabeli rezultata pretrage. Na primer, unosom sledećeg teksta u polje za pretragu dolazi se do naziva baze:

```
' and false union select 1, database() --
```

Sledećim unosom dolazi se do verzije baze:

```
' and false union select 1, h2version() --
```

Kako bi se došlo do korisničkog imena i lozinke svih korisnika aplikacije, prvo je potrebno pronaći naziv tabele koja sadrži korisnike. Sledećim unosom dolazi se do naziva svih tabela u bazi:

```
' and false union select 1, table_name from
information_schema.tables where
table_schema=database() --
```

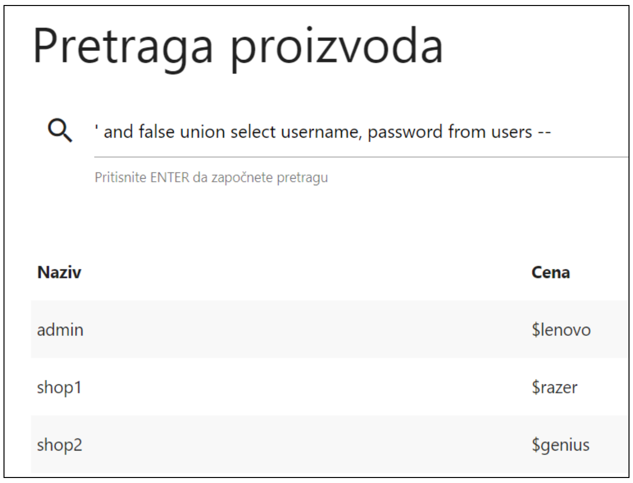
Iz prethodnog koraka saznaje se da je tabela sa korisnicima sačuvana pod imenom *users*. Sledećim unosom dolazi se do naziva svih kolona ove tabele:

```
' and false union select 1, column_name from
information_schema.columns where
table_name='users' --
```

Poslednji korak je definisanje upita koji prikazuje korisničko ime i lozinku svih korisnika:

```
' and false union select username, password
from users --
```

Tabela sa rezultatima sadržaće korisničko ime i lozinku svih korisnika, kao na Sl. 4.


Pretraga proizvoda

' and false union select username, password from users --

Pritisnite ENTER da započnete pretragu

Naziv	Cena
admin	$lenovo
shop1	$razer
shop2	$genius

Sl. 4 Korisničko ime i lozinka svih korisnika aplikacije

B. *Način upotrebe u nastavi*

ZRM je predmet master studija na Modulu za računarsku tehniku i informatiku ETF-a, koji je razvijen u okviru Erasmus+ KA2 projekta pod nazivom *Information Security Services Education in Serbia (ISSES)* [7]. Uzimajući u obzir probleme kod praktičnog izučavanja tema koje se obrađuju na predmetu, a koji su pomenuti u poglavlju I, za studente je napravljeno zatvoreno virtuelno laboratorijsko okruženje u okviru Laboratorije za informacionu bezbednost, koja je uspostavljena i opremljena u okviru istog (*ISSES*) projekta.

Svaki student ima sopstveno virtuelno laboratorijsko okruženje kome pristupa korišćenjem *VPN* veze. Za različite teme koje se obrađuju na predmetu koriste se različite konfiguracije virtuelnih laboratorijskih okruženja. U slučaju laboratorijskih vežbi u kojima se koristi *SQLiTrainer* sistem laboratorijsko okruženje se sastoji od jedne virtuelne mašine na kojoj je pokrenut *Ubuntu Linux* i na kojoj je pokrenut *SQLiTrainer* sistem. Studenti mogu da pristupe aplikacijama *SQLiTrainer* sistema iz svojih pretraživača korišćenjem *VPN* veze.

SQLiTrainer sistem se koristi za izvođenje laboratorijiskih vežbi, ali i kao deo finalnog praktičnog ispita na predmetu. Zahvaljujući načinu implementacije sistema, uz minimalne izmene u kodu, moguće je jednostavno izmeniti svaku od aplikacija tako da se dobiju drugačiji problemi sa istom tematikom, čime je omogućeno da se isti sistem iskoristi i prilikom obučavanja studenata, ali i prilikom provere njihovog znanja.

Još jedan važan aspekt, kada je u pitanju upotreba sistema, jeste i jednostavnost instalacije i konfiguracije. Prilikom konfigurisanja laboratorijskog okruženja za izvođenje laboratorijskih vežbi potrebno je pokrenuti više aplikacija istovremeno. Način implementacije *SQLiTrainer* sistema omogućava da se svaka aplikacija može pokrenuti na različitom portu, čime se prethodno postiže na jednostavan način. Kada je u pitanju instalacija, jedno rešenje je da se aplikacije iskopiraju na svaku virtuelnu mašinu i pokrenu odgovarajućim komandama. Ovo rešenje je vremenski zahtevno i nepraktično kada postoji veliki broj studenata na predmetu, a samim tim i veliki broj laboratorijskih okruženja koje je potrebno pripremiti. Način implementacije *SQLiTrainer* sistema dozvoljava da se iskoristi neki od alata za automatizaciju, kao što je na primer *Ansible* [8], čime se prethodni problem efikasno rešava na taj način što se napišu odgovarajuće skripte za ovaj alat kojima se prethodno manuelni posao kopiranja i pokretanja aplikacija u potpunosti automatizuje.

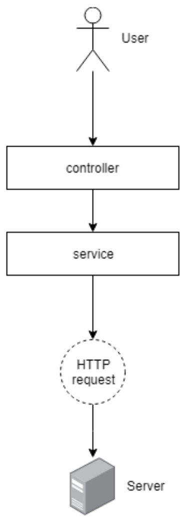
Opisani sistem je korišćen u nastavi u dve uzastopne školske godine 2019/2020 i 2020/2021. Prve školske godine finalni praktični ispit uspešno je savladalo 67% studenata (22/33), dok je u drugoj školskoj godini uspešno bilo 65% studenata (53/81).

IV. IMPLEMENTACIJA *SQLITRAINER* SISTEMA

Sistem čine četiri aplikacije identične strukture. Dve demonstriraju *Union-based SQLi*, pravolinijski i jednostavan napad gde se u kratkim iteracijama otkriva sve više podataka iz baze. Sledeća demonstrira *Blind SQLi* za koju je specifično da se podaci iz baze nikada ne prikazuju napadaču, i spada u teže napade za manuelno izvršavanje. Poslednja demonstrira *SQLi Login Bypass*, gde je cilj napadača da uspešno izvrši korisničku prijavu bez prethodnog poznavanja bilo kog korisničkog imena ili lozinke. U ovoj aplikaciji student se upoznaje sa upotrebom *HTTP Proxy* server, kao alata u izvršavanju *SQLi* napada.

Sistem je implementiran kao skup *Java* veb aplikacija koristeći *Spring Boot* [9] i *h2* [10] *in-memory* bazu podataka. Korisnički interfejs aplikacija kreiran je koristeći *HTML*, *CSS* i *JavaScript*. Za kreiranje lepšeg korisničkog interfejsa korišćen je *MaterializeCSS* [11], dok *AngularJS* [12] pojednostavljuje pisanje koda za interakciju sa korisnikom i *HTTP* (eng. *Hypertext Transfer Protocol*) komunikaciju sa serverom.

Korisnički interfejs i serverska aplikacija mogu se posmatrati kao dve odvojene aplikacije koje komuniciraju preko *HTTP*-a.



Sl. 5 Životni ciklus zahteva u aplikaciji korisničkog interfejsa

Korisnički interfejs realizovan je kao *Single Page Application* (*SPA*). Srž aplikacije čine *HTML* stranica *index.html* i *JavaScript* kod *app.js*. Resursi aplikacije organizovani su u posebne direktorijume:

- *css* – sadrži *CSS* biblioteke i definicije stilova specifičnih za aplikaciju,
- *images* – sadrži fotografije korišćene na korisničkom interfejsu i
- *js* – sadrži *JavaScript* biblioteke i kod aplikacije.

Organizacija koda je slojevita i definiše dva sloja:

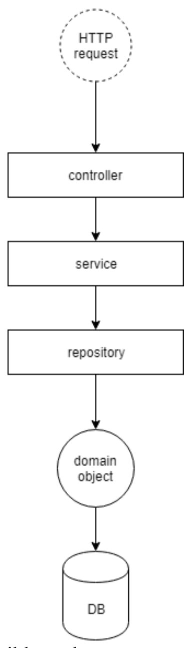
- *controller* – kod koji obrađuje korisničke akcije i
- *service* – kod za komunikaciju sa serverskom aplikacijom.

Sloj *controller* je viši sloj i zavisan od sloja *service*, a korisnički zahtevi prolaze kroz oba sloja aplikacije. Na Sl. 5 prikazano je kako korisnički zahtev putuje kroz aplikaciju korisničkog interfejsa i do serverske aplikacije.

Organizcija serverskog koda je takođe slojevita, gde su slojevi aplikacije predstavljeni sledećim *Java* paketima:

- *controller* – klase za razmenu podataka sa aplikacijom korisničkog interfejsa,
- *service* – klase koje izvršavaju poslovnu logiku aplikacije i
- *repository* – klase za čitanje i upis domenskih objekata u bazu podataka.

Pored navedenih paketa postoji i paket *domain* koji sadrži klase koje predstavljaju domenske entitete.



Sl. 6 Životni ciklus zahteva u serverskoj aplikaciji

Zahtevi koji dolaze sa korisničkog interfejsa prolaze kroz sve slojeve aplikacije. Način razmene podataka između slojeva prikazan je na Sl. 6 na primeru zahteva koji na kraju rezultuje upisom u bazu podataka. Svaki sloj aplikacije zavisan je od sledećeg (nižeg) sloja, a često su svi paketi aplikacije zavisni od domenskog. U nekim implementacijama izostavljen je servisni sloj, jer ne bi sadržao nikakvu logiku, već bi samo prosleđivao podatke sledećem sloju.

Svaka instanca aplikacije poseduje svoju instancu baze, koja se pokreće zajedno sa aplikacijom i čuva podatke u memoriji. Kod serverske aplikacije prate *SQL* skripte koje se izvšavaju nad bazom prilikom pokretanja aplikacije, kako bi pri svakom pokretanju aplikacije stanje baze bilo identično. Skripte se nalaze u sledećim fajlovima:

- <u>*schema.sql*</u> – izvršava se prva i njena uloga je da kreira relacionu šemu baze i
- <u>*data.sql*</u> – popunjava bazu podacima.

Prilikom zaustavljanja aplikacije zaustavlja se i baza podataka, a podaci iz nje trajno nestaju.

Varijacije problema laboratorijskih vežbi mogu se kreirati na više načina. Najjednostavniji je izmeniti *SQL* skripte čime se menja početno stanje baze podataka. Kompleksnije izmene, poput izmene imena kolona i tabela, zahtevaju i manje izmene u kodu aplikacije. Navedene skripte se mogu pokrenuti i nad nekom drugom bazom podaka, na primer *PostgreSQL* ili *MySQL*, za šta je dovoljno u konfiguracionom fajlu aplikacije navesti parametre za konekciju. Korišćenje različitih baza podataka povećava kompleksnost zadatka jer koriste različite dijalekte *SQL* jezika.

V. ZAKLJUČAK

U ovom radu predstavljen je jedan novi sistem za učenje o *SQLi* sigurnosnim propustima u aplikacijama. *SQLiTrainer* služi za praktičnu obuku programera u oblasti razvoja bezbednog softvera. Realizovan je kao skup ranjivih aplikacija kojima se mogu demonstrirati različiti tipovi *SQLi* propusta koji se mogu javiti u aplikacijama. Omogućava proveru postojanja propusta u sigurnom okruženju, kao i učenje tehnika kojima se mogu otkloniti uočeni propusti. U radu je prikazan opis *SQLiTrainer* sistema i način njegovog korišćenja. U budućnosti se planira dodavanje skupa vežbi u kojima će studenti biti u prilici da isprave propuste koji postoje u aplikacijama.

ZAHVALNICA

Autori žele da se zahvale master inž. Adrianu Milakoviću i prof. dr Pavlu Vuletiću na pomoći prilikom uvođenja sistema na laboratorijske vežbe na predmetu Zaštita računarskih sistema i mreža.

LITERATURA

[1] „*SQL injection*,“ Dostupno na: https://owasp.org/www-community/attacks/SQL_Injection, poslednji put pristupano: maj 2021.
[2] „*OWASP*,“ Dostupno na: https://owasp.org/, poslednji put pristupano: maj 2021.
[3] „*Avatao*,“ Dostupno na: https://avatao.com/, poslednji put pristupano: maj 2021.
[4] „*Juice Shop*,“ Dostupno na: https://owasp.org/www-project-juice-shop/, poslednji put pristupano: maj 2021.
[5] „*Security Idiots*,“ Dostupno na: http://www.securityidiots.com/, poslednji put pristupano: maj 2021.
[6] „Zaštita računarskih sistema i mreža,“ Dostupno na: https://www.etf.bg.ac.rs/fis/karton_predmeta/13M111ZRM-2019, poslednji put pristupano: maj 2021.
[7] „*Information Security Services Education in Serbia*,“ Dostupno na: https://isses.etf.bg.ac.rs/, poslednji put pristupano: maj 2021.
[8] „*Ansible*,“ Dostupno na: https://www.ansible.com/, poslednji put pristupano: maj 2021.
[9] „*Spring Boot*,“ Dostupno na: https://spring.io/projects/spring-boot, poslednji put pristupano: maj 2021.
[10] „*H2 database*,“ Dostupno na: https://www.h2database.com /html/main.html, poslednji put pristupano: maj 2021.
[11] „*Materialize CSS*,“ Dostupno na: https://materializecss.com/, poslednji put pristupano: maj 2021.
[12] „*AngularJS*,“ Dostupno na: https://angularjs.org/, poslednji put pristupano: maj 2021.

ABSTRACT

During the application development process security vulnerabilities can occur and remain in application in production environment. These vulnerabilities can cause confidentiality, integrity and availability breaches. SQLiTrainer represents a set of vulnerable applications that can be used to demonstrate different types of SQLi vulnerabilities. Implementation of the SQLiTrainer system is given in the paper and the examples on how to use the system for programmer practical training is proposed. The system was successfully used for laboratory exercises at the Advanced System and Network Security course at the University of Belgrade, School of Electrical Engineering.

SQLITRAINER - SYSTEM FOR LEARNING ABOUT SQLI VULNERABILITY IN APPLICATIONS

Djordje Madic, Zarko Stanisavljevic

Jedno rješenje analize i prikaza kontrolnih tačaka definisanih podešavanjem AUTOSAR nadzornog časovnika

Ivana Tešević, Branko Milošević, Dejan Bokan i Bogdan Pavković

Apstrakt— **Razvojem automobilske industrije i softvera unutar nje kao tehnička posljedica javila se potreba za obaveznom integracijom zaštitnih mehanizama u ugrađenim jezgrima operativnog sistema. Jedan od osnovnih mehanizama za zaštitu sistema jeste nadzorni časovnik (eng. watchdog, WDG). Ova komponenta ima za cilj da nadgleda sve ostale komponente pokrenute od strane raspoređivača i time omogući bezbjedan rad sistema. Kako je probleme koje nadzorni časovnik prijavljuje relativno teško ispratiti i analizirati u stvarnom sistemu, došlo se do ideje da se oponaša rad komponente nadzornog časovnika na računaru sa istim ulaznim parametrima kao u živom sistemu. U ovom radu je dato rješenje za simulaciju mehanizama nadgledanja sistema definisane AUTOSAR arhitekture. Simulacijom je omogućeno da se minimalizuju odstupanja, predvide greške u sistemu i olakša sama analiza. Rad može doprinijeti bržem razvoju sistema jer omogućava da se prije implementacije predvide greške koje će se desiti u sistemu.**

Ključne riječi*— *AUTOSAR, WDG, WdgM, WdgIf, nadgledanje krajnjih rokova, logički nadzor, nadgledanje u realnom vremenu, najduže vrijeme izvršenja(WCET).

I. UVOD

AUTOMOBILSKA industrija je grana industrije koja se sveobuhvatno razvija u posljednjoj deceniji. Dizajn vozila u automobilskoj industriji tradicionalno se oslanja na diskretne hardverske komponente (elektronske upravljačke jedinice - ECU), sa vrlo malo potrebnog softvera. Sa poboljšanjem automobilske industrije, softver za nju se razvijao. Danas je softverski dio prevladao hardver[1][2] . Softverske komponente postale su komplikovanije i zahtjevnije od hardverskih komponenti. Danas automobili nude mnogo više mogućnosti, uključujući i autonomne funkcije pri vožnji[1]. Postoji pet nivoa automatizacije vožnje, dok je industrija trenutno na trećem nivou, očekujući da će dostići nivo četiri i pet do 2025. godine[3]. Vozači će moći bezbjedno da skrenu pažnju sa vožnje, npr. gledati film ili čitati knjigu.

Činjenica je da je sve više dobavljača u ovoj grani industrije, pa se pojavila potreba za standardizacijom proizvodnje softvera. Da bi se udovoljilo ovom zahtjevu, stvorena je platforma AUTOSAR (eng. *Automotive Open System Architecture*) [1].

Kako se ova industrija sve više širi i kako rastu softverski zahtjevi povećava se i potreba za raznim alatima za održavanje bezbjednosti sistema. Ovakvi sistemi moraju podlijegati raznim testovima i konstantno se nadgledati kako bi se u potpunosti otklonila mogućnost greške, jer i najmanja greška može imati fatalne posljedice.

Nadzorni časovnik je jedna od komponenti koja za cilj ima nadzor cijelog sistema. Ova komponenta kao takva sama po sebi mora imati maksimalni kvalitet koda i podlijegati najvećim provjerama. Bilo kakva greška primijećena od strane WDG komponente biće ispraćena reakcijom gašenja cijelog sistema. Ovakav vid zaštite u industriji otežava testiranje, predviđanje ali i pronalaženje greške u toku rada. Zato se javila potreba da se prikaže jedno rješenje za oponašanje sistema kako bi se moglo predvidjeti i upoznati sa greškama i načinima na koji dolazi do njih.

Ovaj rad prikazuje jedno rješenje analize i prikaza kontrolnih tačaka definisanih podešavanjem *AUTOSAR* nadzornog časovnika. Prikazaće se simulacija poremećaja u sistemu koji će nadzorni časovnik prepoznati pomoću nadzornih mehanizama. Namjerno izazivanje poremećaja i reakcija nadzornog časovnika na te poremećaje doprinijeće lakšem testiranju i predviđanju u stvarnom sistemu. Pomoću ovog rješenja nudi se mogućnost korišćenja stvarnih ulaznih parametara i testiranje raznih poremećaja i lanca događaja nakon namjerno izazvane greške. Mogućnost predviđanja i vizualni prikaz sistema nakon poremećaja jesu glavni doprinos ovog rada. Postojeća literatura na temu nadzornog časovnika[5][6] skoncentrisana je na unaprjeđenju mehanizama zaštite ili na načinu testiranja sistema i ispravnosti sprege nadzornog časovnika.

Drugo poglavlje će dati teorijske osnove o načinu rada svih modula, samom *AUTOSAR* standardu i vertikali nadzornog časovnika sa definisanim modulima.

U trećem poglavlju biće opisan način rada, pristup rješenju i podloga za nastavak i samu implementaciju.

U četvrtom poglavlju su opisani moduli, dato je programsko rješenje i sami postupci implementacije.

U petom poglavlju su prikazani rezultati rada, način testiranja, kao i svrha samog rješenja.

II. AUTOSAR STANDARD

Osnovan 2003. godine, AUTOSAR predstavlja međunarodno razvojno partnerstvo stranaka iz automobilske industrije. Cilj ove saradnje bio je stvaranje i uspostavljanje otvorene i standardizovane softverske arhitekture za osnovne elektronske jedinice autonomnog vozila nazvane ECU.

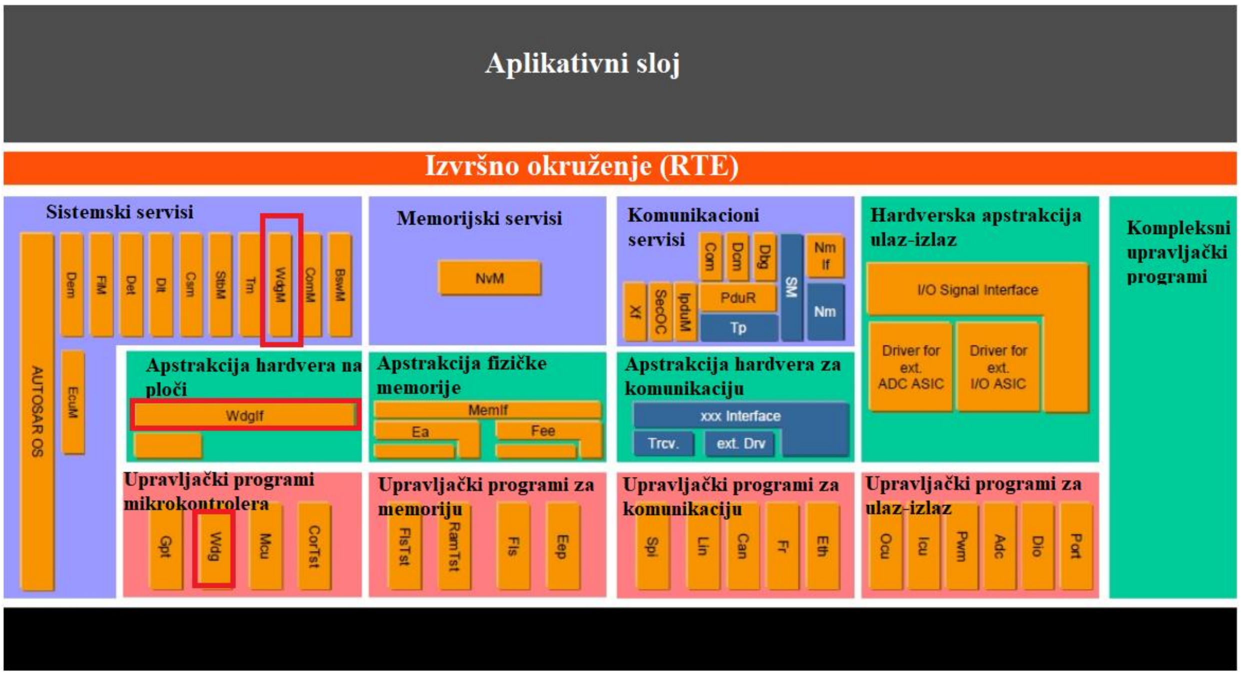
Ivana Tešević, RT-RK Institute for Computer Based Systems, Novi Sad, Srbija (e-mail: ivana.tesevic@rt-rk.com)

Branko Milošević, RT-RK Institute for Computer Based Systems, Novi Sad, Srbija (e-mail: branko.milosevic@rt-rk.com).

Bogdan Pavković, RT-RK Institute for Computer Based Systems, Novi Sad, Srbija (e-mail: bogdan.pavkovic@rt-rk.com)

Dejan Bokan, RT-RK Institute for Computer Based Systems, Novi Sad, Srbija (e-mail: dejan.bokan@rt-rk.com).

AUTOSAR standard daje set specifikacija koje opisuju funkcionalnosti softverskih modula i realizuje zajedničke metode daljeg razvoja na osnovu standardizovanog formata[1]. Arhitektura ovog standarda, odnosno AUTOSAR modela, na najvišem nivou apstrakcije prepoznaje tri različite softverske cjeline[4] (Slika 1.):



Sl. 1. AUTOSAR model [1]

• Osnovni softver (eng. *Basic software module, BSW*)- ovaj sloj se sastoji od modula koji su neophodni za funkcionisanje višeg softverskog sloja. Slojevi od kojih se sastoji osnovni softver su: sloj apstrakcije ECU, složeni upravljački programi, sloj apstrakcije mikrokontrolera (eng. MCAL).

• Izvršno okruženje(eng. Runtime environment, RTE)- realizuje komunikaciju između softverskih komponenti i osnovnog softvera.

•Aplikativni sloj(eng. Application Layer)-funkcionalnost elektronskih kontrolnih jedinica je implementirana u obliku pojedinačnih softverskih komponenti.

A. *Nadzorni časovnik*

Za automobilske sigurnosne sisteme kritično je pitanje zadovoljavanja zahtjeva u realnom vremenu na deterministički način. Da bi se udovoljilo vremenskim ograničenjima, razvijeni su različiti mehanizmi praćenja, kao što su nadzorni hardver ECU jedinice[7], nadgledanje krajnjih rokova[8][9], nadgledanje vremena izvršenja 0. Ovakav vid nadzora kreiran je kako bi se osigurao tačan raspored zadataka0 .

Vertikalu nadzornog časovnika u AUTOSAR slojevitoj arhitekturi čine rukovodilac nadzornog časovnika(nalazi se u servisnom sloju eng. *Service Layer*), sprega nadzornog časovnika(smještena u ECU sloju apstrakcije) i upravljač nadzornog časovnika(smješten u sloju apstrakcije mikrokontrolera)[12] Ovi moduli pružaju usluge za praćenje vremena i ispravnosti izvršenja entiteta u aplikaciji i osnovnom softveru.

B. *Rukovodilac nadzornog časovnika*

Rukovodilac nadzornog časovnika (eng. *watchdog manager*, *WdgM*) je osnovni softverski modul u servisnom nivou koji nadgleda tok programa[13]. Kada se otkrije narušavanje unaprijed definisanih vremenskih ili logičkih ograničenja u programskom toku, potrebno je evidentirati grešku i preći u bezbjedno stanje nakon vremenskog kašnjenja. Sigurno stanje se postiže ponovnim pokretanjem ili izostavljanjem aktiviranja modula nadzornog časovnika.

Po *AUTOSAR* definiciji, tačke u kontroli toka nadgledanog entiteta gdje se aktivnost prijavljuje rukovodiocu nadzornog časovnika su kontrolne tačke.

Polja koja opisuju kontrolnu tačku su:

• *ID* kontrolne tačke

• Lokalni početak, lokalni kraj

• Globalni početak, globalni kraj

Lokalni prelazi predstavljaju prelaze između dvije kontrolne tačke unutar istog nadgledanog entiteta.

Globalni prelazi su prelazi između dvije kontrolne tačke koje pripadaju različitim entitetima.

Nadgledani entitet predstavljen je kontrolnim tačkama kojih može biti jedna ili više. Svaki nadgledani entitet može imati jedno ime i jedno stanje.

Kada se govori o mehanizmima nadgledanja u *WdgM* modulu pominju se tri tipa nadgledanja[14]:

• Nadgledanje u realnom vremenu (eng. *Alive Supervision*) – prati frekvenciju izvršavanja određenog softverskog dijela. To znači da rukovodilac provjerava da li se nadgledani entitet javlja suviše često ili suviše rijetko.

• Nadgledanje krajnjih rokova (eng. *Deadline Supervision*) – nadgleda vrijeme potrebno za izvršavanje nadgledanog entiteta. Glavna svrha je provjera vremenskog, dinamičkog ponašanja entiteta.

• Logički nadzor (eng. *Logical Supervision/Program Flow check*) – nadgleda tok izvršavanja u programu.

Dva su ključna pojma koja treba pomenuti kada je u pitanju nadgledanje i reakcija na greške, a to su vrijeme otkrivanja greške i vrijeme reakcije na grešku.

Vrijeme otkrivanja greške (eng. *Fault Detection*) traje od pojave greške do trenutka kada je ta greška otkrivena i prijavljena sistemu.

Vrijeme reakcije na grešku (eng. *Fault Reaction*) traje od trenutka otkrivanja greške do ponovnog pokretanja sistema. *WdgM* reakcija na grešku:

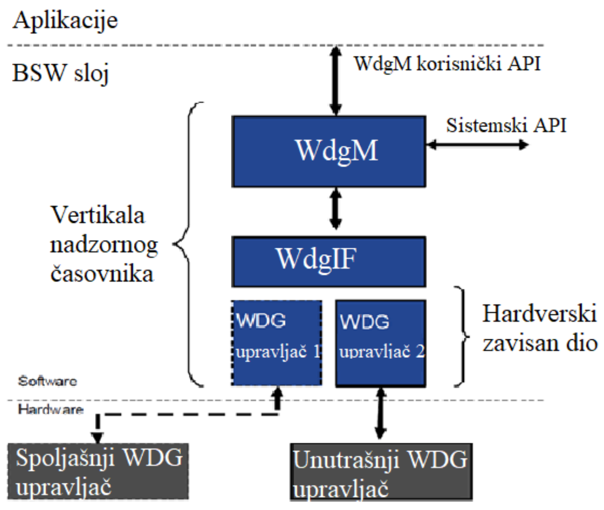
• Obavještenje iz funkcije povratnog poziva
• Ponovno pokretanje sistema
• Stopiranje okidanja nadzornog časovnika

C. *Sprega nadzornog časovnika*

Sprega nadzornog časovnika (*WdgIf*) je dio ECU apstraktnog sloja. Uvijek se nalazi ispod rukovodioca i iznad upravljača nadzornog časovnika. Sprega komunicira sa upravljačkim programima ispod. Implementacija sprege zavisi od broja upravljača[14].

D. *Upravljač nadzornog časovnika*

Upravljač je zadužen za pristup samoj periferiji direktno[15] (unutrašnjem i spoljašnjem nadzornom časovniku) i nalazi se u sloju apstrakcije mikrokontrolera. Modul za spoljašnji nadzorni časovnik koristi druge module za pristup spoljnom uređaju.



Sl. 2 Slojevita struktura WDG 1

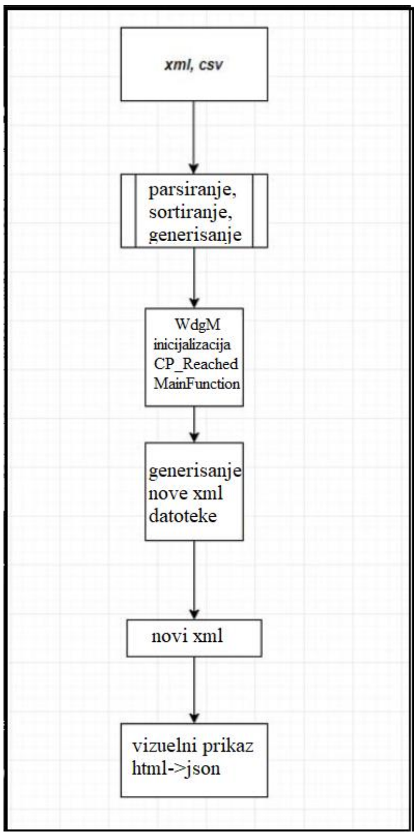
III. KONCEPT RJEŠENJA

Uključen nadzorni časovnik u stvarnom sistemu može stvarati velike probleme prilikom analiziranja nekog problema. Stalno gašenje i ponovno pokretanje jedan su od pokazatelja zašto je to tako. Kako bi se nastavila analiza neke greške na projektima se obično podliježe gašenju nadzornog časovnika i nakon toga se nastavlja sa analiziranjem. Ovaj rad predstavlja jedan pomoćni alat prilikom te analize koji je omogućio da pomoću komandne linije unese željeni poremećaj i isprati lanac događaja koji slijede nakon njega.

Na osnovu već generisanog operativnog sistema odrađeno je parsiranje redoslijeda zadataka i AUTOSAR generisane konfiguracije za stek modul nadzornog časovnika. Parsirani podaci bili su neophodni za grafički prikaz raspoređivača. Grafička predstava kontrolnih tačaka odrađena je tako da se vodilo računa o redoslijedu, kao i koja je kontrolna tačka dodijeljena kom zadatku. Pomenuti grafički prikaz omogućava da se jednostavno uoči poremećaj, tako da je grafički prikaz jedan od osnovnih alata koji su korišteni prilikom analize i testiranja. Ulazni parametri za parsiranje su definisani u csv i arxml formatu (*SE WCET*, *SE period*, *WCET neto*, *WCET abs*).

Sledeći korak jeste simuliranje nadgledanja sistema definisane AUTOSAR arhitekture. Osnovni cilj ovog koraka jeste da prikaže što približnije slijed događaja i grešaka na računaru, kao što je očekivano i u stvarnom sistemu. Glavna razlika je ta što greške koje vidimo na računaru nemaju nikakvu bezbjednosnu posljedicu po izvršenje, već služe isključivo u svrhu analize i rezultiraće ispisima i informativnim porukama, umjesto gašenjem sistema.

Nakon implementacije grafičkog prikaza i simulacije mehanizama, pristupa se analizi sistema, praćenju ponašanja sistema pod uticajima raznih poremećaja koji su namjerno izazvani i koji se odnose na mehanizme nadgledanja. Namjernim izazivanjem grešaka olakšava se analiziranje istih, očekivanih u procesu rada na stvarnom projektu. Data je mogućnost da se predvide različiti lanci događaja, kao i da se smanji vrijeme koje bi bilo potrošeno na pokušaje analize problema uslijed stalnog gašenja sistema.

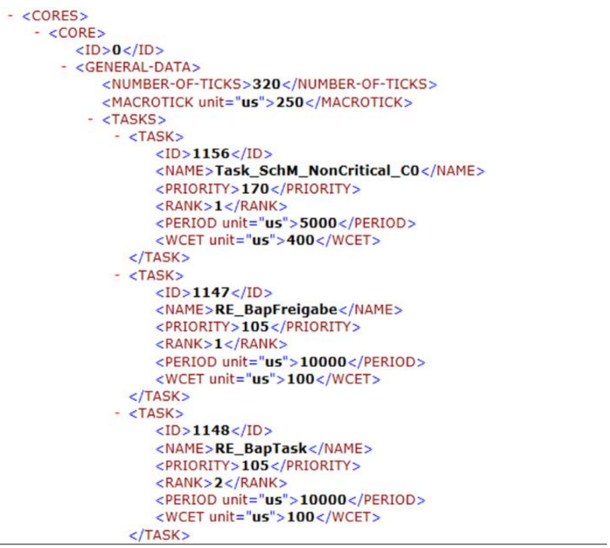

xml, csv
parsiranje, sortiranje, generisanje
WdgM inicijalizacija CP_Reached MainFunction
generisanje nove xml datoteke
novi xml
vizuelni prikaz html->json

Sl. 3 Dijagram rješenja

A. *Rukovodilac nadzornog sistema u višejezgarnom sistemu*

Rukovodilac može biti korišten u jednojezgarnim i višejezgarnim sistemima. U ovom radu obrađen je rukovodilac u višejezgarnom sistemu.

Svaka instanca treba da bude nezavisna jedna od druge i mora biti inicijalizovana njenom sopstvenom konfiguracijom. Poziv *Main* funkcije je odvojen. U stvarnom sistemu se softverske komponente izvršavaju paralelno i vremenski nezavisno. Svako jezgro ima svoje sopstveno vrijeme.



```
- <CORES>
  - <CORE>
      <ID>0</ID>
    - <GENERAL-DATA>
        <NUMBER-OF-TICKS>320</NUMBER-OF-TICKS>
        <MACROTICK unit="us">250</MACROTICK>
      - <TASKS>
        - <TASK>
            <ID>1156</ID>
            <NAME>Task_SchM_NonCritical_C0</NAME>
            <PRIORITY>170</PRIORITY>
            <RANK>1</RANK>
            <PERIOD unit="us">5000</PERIOD>
            <WCET unit="us">400</WCET>
          </TASK>
        - <TASK>
            <ID>1147</ID>
            <NAME>RE_BapFreigabe</NAME>
            <PRIORITY>105</PRIORITY>
            <RANK>1</RANK>
            <PERIOD unit="us">10000</PERIOD>
            <WCET unit="us">100</WCET>
          </TASK>
        - <TASK>
            <ID>1148</ID>
            <NAME>RE_BapTask</NAME>
            <PRIORITY>105</PRIORITY>
            <RANK>2</RANK>
            <PERIOD unit="us">10000</PERIOD>
            <WCET unit="us">100</WCET>
          </TASK>
```

Sl 4. Isječak iz arxml fajla

B. *Sortiranje kontrolnih tačaka prema rasporedu*

Parametri koji su značajni i koji opisuju izvršenje sistema u vremenu su zadati u xml datoteci, a potrebni entiteti su u csv datoteci. Ove dvije ulazne datoteke moraju biti međusobno povezane i predstavljaju jednu cjelinu. Parsiranjem ovih datoteka dobijeni su svi podaci potrebni za simulaciju nadgledanja. Ti parametri su: naziv, perioda, trajanje, vrijeme početka, prioritet i ID. Isječak iz arxml fajla je prikazan na Sl 4.

Nakon izvlačenja pomenutih podataka sve kontrolne tačke sortirane su po vremenu i po prioritetu. Kontrolna tačka koja ima manji prioritet će biti prekinuta ako se u toku njenog trajanja javi neka druga tačka većeg prioriteta.

Nakon sortiranja sve kontrolne tačke kreću sa izvršenjem, baš kao u stvarnom sistemu, istim redoslijedom kako je zahtijevano u ulaznoj datoteci i po prioritetu. Ono što je takođe bilo bitno prikazati jeste vrijeme trajanja koje je oponašano na osnovu ulaznih informacija.

C. *Simulacija nadgledanja*

Nakon uspješno obavljene inicijalizacije, sortiranja i prozivanja kontrolnih tačaka potrebno je simulirati rad prethodno opisanih načina nadgledanja.

Rukovodilac *Main* je sastavni dio izvršenja i on se prozove po zadatom intervalu od 10 milisekundi i tada se vrši provjera nadgledanje u realnom vremenu, nadgledanje krajnjih rokova, logički nadzor.

Vrijeme u sistemu nadzornog časovnika je predstavljeno u tikovima. Potrebno je simulirati vrijeme tako da odgovara vremenu iz stvarnog sistema.

Na osnovu tog vremena se provjeravaju nadgledanja. Provjeru nadgledanja u realnom vremenu treba obaviti tako da se u slučaju da se kontrolna tačka ne javi u očekivanom vremenskom intervalu na konzoli dobijemo ispis o grešci koja se desila.

IV. PROGRAMSKO RJEŠENJE

A. *Parsiranje rasporeda i sortiranje*

Za ulazne podatke iskorišćene su xml i csv datoteke iz stvarnog sistema. Parsiranje je rađeno u programskom jeziku *Python,* svi podaci koji su izvučeni iz tih datoteka su generisani i urađeno je prozivanje funkcije *WdgM_CheckpointReached().*

Osnovni problem koji se javio prije prozivanja ove funkcije bio je sortiranje kontrolnih tačaka prema rasporedu. Sortiranje je takođe odrađeno u programskom jeziku *Python* i korištene su funkcije:

•*expiry_points()* - funkcija koja sortira podatke koji su izvučeni iz ulaznih datoteka parsiranjem. Sortira kontrolne tačke po vremenu njihovog javljanja i po jezgrima.

•*priority_sort()* - Prethodno sortirana lista po vremenu javljanja se sortira i po prioritetima . Ako se desi da dvije kontrolne tačke počinju istovremeno prednost će imati tačka sa većim prioritetom, tačka manjeg prioriteta ostaje da čeka svoje red. Entitet može biti prekinut i u toku izvršenja, ako se desi da je došlo do javljanja izvršioca sa većim prioritetom, trenutni entitet ostaje u stanju čekanja sve dok mu se ne signalizira da je prioritetniji entitet završio sa radom.

Nakon sortiranja je generisano kojim se redoslijedom vrši pozivanje *WdgM_CheckpointReached()* funkcije.

B. *Implementacija rukovodioca*

Prvi korak koji je odrađen jeste postupak inicijalizacije. Svaki zadatak inicijalizovan je pomoću funkcije *WdgM_Init().*

U stvarnom sistemu WDG zadatak ponavlja se kružno na svakih 10 milisekundi. To znači da se poziv funkcije *WdgM_MainFunction()* ponavlja svakih 10 milisekundi.
Ova funkcija ima ključnu ulogu jer se u njoj vrše provjere ispravnosti. Trajanje jednog ciklusa naziva se hiper period, Na osnovu trenutnih ulaznih parametara koji su obrađeni u ovom primjeru koji će biti opisan hiper period je 80 milisekundi i nakon toga se završava jedan ciklus nadzora.
Nakon izvršenja *WdgM_MainFunction()* očekuje se neka od reakcija rukovodioca.

- •Ako dođe do greške u nadgledanju u realnom vremenu greška će biti detektovana na kraju nadgledanog referentnog ciklusa (eng. *Alive supervision reference cycle*).
- •U slučaju nadgledanja programskog toka ako dođe do greške ona će biti detektovana na kraju svakog nadgledanog ciklusa.
- •Ako je greška u nadgledanju krajnjih rokova ona će biti detektovana na kraju svakog nadgledanog ciklusa, nastavak kršenja ovog vida nadgledanja detektuje se na kraju svakog krajnji rok nadgledanog entiteta.

Ponašanje sistema nakon uočavanja neke od pomenutih grešaka zavisi od konfiguracije i tipa poremećaja.

V. PROVJERA ISPRAVNOSTI

A. *Opis testiranja*

U svrhu testiranja korišteni su ulazni parametri sa stvarnog sistema. Ovakav pristup omogućio je poređenje sa stvarnim sistemskim greškama i utvrditi ispravnost samog rada.

Kako je stvarni sistem čiji si ulazni parametri iskorišćeni sadržao 3 jezgra, a ona u sistemu rade u paraleli. U ovom rade sva tri jezgra su testirana istovremeno i softverski spojena u jednu cjelinu, prikazan je njihov paralelizam. Na računaru se pomoću komandne linije prati ispis rezultata, kao rezultat testiranja dobijaju se poruke o prekršajima.

Prilikom pokretanja radi se inicijalizacija sistema.

Korisnik treba da odabere jezgro na kome će nanijeti poremećaj kao i vrstu poremećaja koju želi, promjena *WCET* vremena ili greška u nadgledanju u realnom vremenu.

Ako je u sistemu sve prošlo bez greške prilikom izvršavanja rukovodioca prozvaće se *TriggerWindow* funkcija na osnovu čega je testirana ispravnost. Vizualni prikaz sistema bez greške dat je na Sl.5

B. Poremećaj nastao promjenom WCET vremena

Najgore vrijeme izvršenja predstavlja ukupno vrijeme koje je dato jednom zadatku da se izvrši. Vrijednost ovog parametra predstavljena je sa dva termina *bruto* i *neto WCET*. Kada je riječ o ukupnom *bruto* vremenu možemo reći da je to vrijeme koje protekne od početka do kraja zadatka sa uračunatim svim prekidima od strane prioritetnih zadataka. Dok je *neto WCET* vrijeme od početka do kraja ali predstavlja samo sabrane vremenske trenutke u kojima je aktivan dati zadatak.

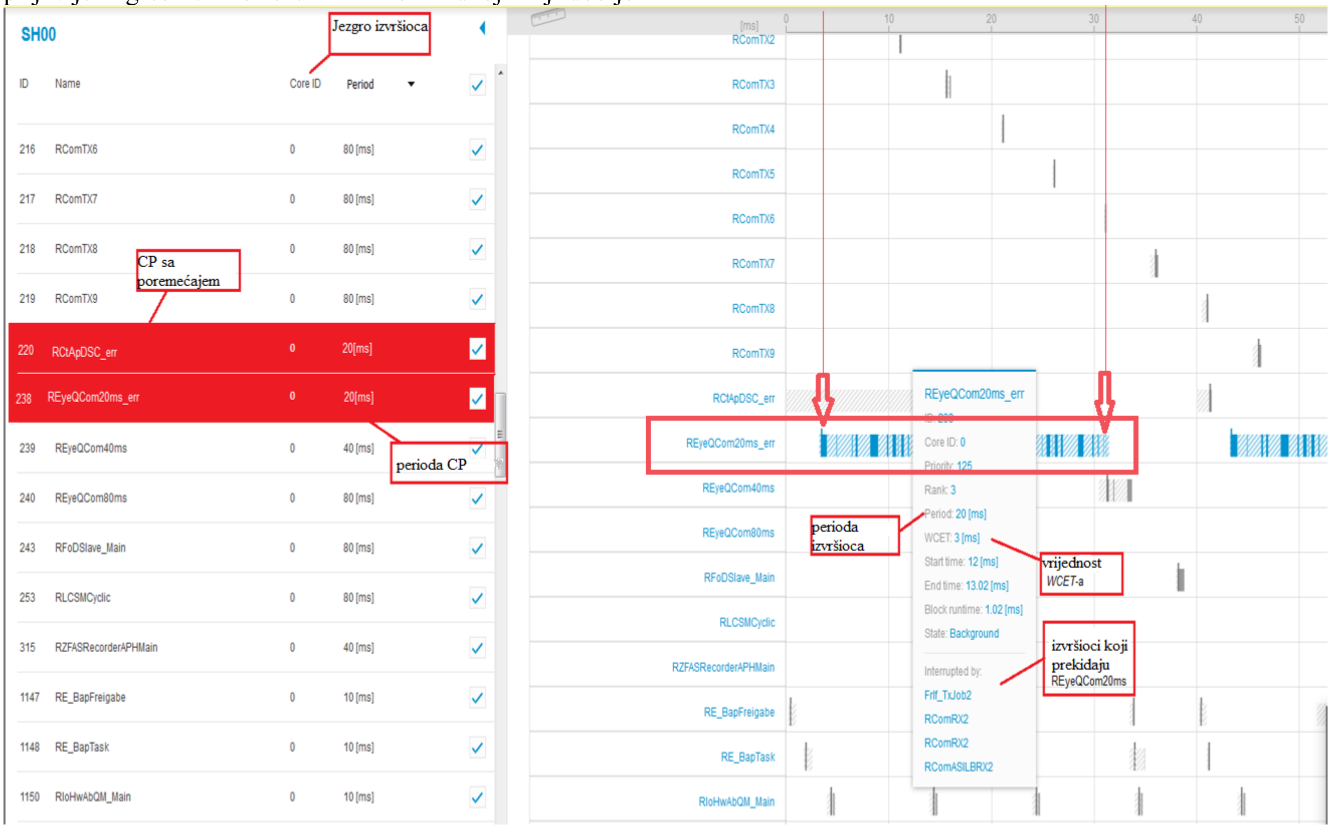
Vrijednost *WCET* vremena koja je konfigurisana za određeni nadgledani entitet i data u rasporedu očekivana je vrijednost u sistemu sa kojom svi nadzori rade ispravno. Nakon što se napravi neka promjena *WCET* vremena i ispisivanja poruka o ispravnosti sistema generiše se nova datoteka *arxml* koja predstavlja ulazni parametar za vizualizaciju sistema. Ako je neka promjena unesena na grafičkom prikazu promijenjena kontrolna tačka mijenja boju sto se vidi na Sl 5. Greška u sistemu vidljiva je na slici gdje su crvenom bojom markirani izvršioci kod kojih je prijavljena greška. Kao rezultat na komandnoj liniji dobije se poruka o svim prekršajima koje je izazvala promjena *WCET* vremena na željenom entitetu.

Na osnovu vizualizacije omogućeno je lakše praćenje dešavanja u sistemu, način na koji se nakon bilo koje promjene izmiješaju kontrolne tačke. Simulirana je vremenska osa i kontrolne tačke u vremenu.

C. Poremećaj u nadgledanju realnog vremena

Kada se nanese ovaj vid poremećaja u sistemu dolazi do situacije u kojoj se određena kontrolna tačka ne prozove. Tada sistem detektuje grešku u nadgledanju u realnom vremenu. Primjer iz tabele takođe pokazuje grešku nad *RCtApDSC* koja ima period 20 milisekundi. Možemo uočiti da je došlo do problema kada je rukovodilac prepoznao da se u prvih 20 milisekundi nije javila ova kontrolna tačka, a mehanizam nadzora u realnom vremenu očekivao je da će doći do njenog javljanja. Nakon otkrivanja problema brojač se nije uvećao i kao status vraćena je vrijednost "nije uspjelo"(eng. FAILED).

Kada se poveća WCET u ovom slučaju nad nadgledanim entitetom pod nazivom *ReyeQCom20ms* koji je prikazan u tabeli (T 1.) desi se poremećaj u prvih 10 milisekundi. Očekivana vrijednost je 2.7 milisekundi jer je ovo nadgledani entitet koji je niskog prioriteta i isprekidan je od strane ostalih koji imaju veći prioritet. Prilikom testiranja promijenili smo vrijednost na 3 milisekunde. Zbog poremećaja na jednoj kontrolnoj tački i greške u nadgledanju krajnjeg roka u prvih 10 milisekundi očita se greška, ali se prozove i funkcija *TriggerWindow*.

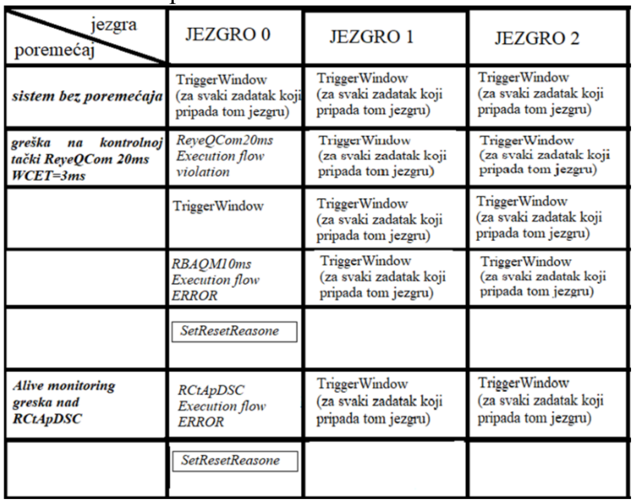

SH00

ID	Name	Core ID	Period	✓
216	RComTX6	0	80 [ms]	✓
217	RComTX7	0	80 [ms]	✓
218	RComTX8	0	80 [ms]	✓
219	RComTX9	0	80 [ms]	✓
220	RCtApDSC_err	0	20[ms]	✓
238	REyeQCom20ms_err	0	20[ms]	✓
239	REyeQCom40ms	0	40 [ms]	✓
240	REyeQCom80ms	0	80 [ms]	✓
243	RFoDSlave_Main	0	80 [ms]	✓
253	RLCSMCyclic	0	80 [ms]	✓
315	RZFASRecorderAPHMain	0	40 [ms]	✓
1147	RE_BapFreigabe	0	10 [ms]	✓
1148	RE_BapTask	0	10 [ms]	✓
1150	RIoHwAbQM_Main	0	10 [ms]	✓

REyeQCom20ms_err
ID: 238
Core ID: 0
Priority: 125
Rank: 3
Period: 20 [ms]
WCET: 3 [ms]
Start time: 12 [ms]
End time: 13.02 [ms]
Block runtime: 1.02 [ms]
State: Background
Interrupted by:
FrIf_TxJob2
RComRX2
RComRX2
RComASILBRX2

Sl. 5 Greške na jezgru 0 nakon promjene *WCET* vremena nadgledanog entiteta *ReyeQCom20ms*

Na Sl 5 može se uočiti kako će *ReyeQCom20ms* zadatak imati uticaj na sistem kada se njemu nanese vremenski poremećaj. Takođe primjetna je lista izvršilaca koji prekidaju pomenuti entitet zbog većeg prioriteta, pri čemu će svaki od pomenutih prekinuti izvršenje. trenutnog. Puna linija predstavlja izvršenje *ReyeQCom20ms* dok je isprekidanim poljima predstavljen period kada je posmatrani entitet u pozadini i čeka na izvršenje zadatka sa većim prioritetom.



jezgra / poremećaj	JEZGRO 0	JEZGRO 1	JEZGRO 2
sistem bez poremećaja	TriggerWindow (za svaki zadatak koji pripada tom jezgru)	TriggerWindow (za svaki zadatak koji pripada tom jezgru)	TriggerWindow (za svaki zadatak koji pripada tom jezgru)
greška na kontrolnoj tački ReyeQCom 20ms WCET=3ms	*ReyeQCom20ms Execution flow violation*	TriggerWindow (za svaki zadatak koji pripada tom jezgru)	TriggerWindow (za svaki zadatak koji pripada tom jezgru)
	TriggerWindow	TriggerWindow (za svaki zadatak koji pripada tom jezgru)	TriggerWindow (za svaki zadatak koji pripada tom jezgru)
	RBAQM10ms Execution flow ERROR	TriggerWindow (za svaki zadatak koji pripada tom jezgru)	TriggerWindow (za svaki zadatak koji pripada tom jezgru)
	SetResetReasone		
Alive monitoring greska nad RCtApDSC	*RCtApDSC Execution flow ERROR*	TriggerWindow (za svaki zadatak koji pripada tom jezgru)	TriggerWindow (za svaki zadatak koji pripada tom jezgru)
	SetResetReasone		

T 1. Ponašanje sistema nakon poremećaja 1

VI. ZAKLJUČAK

U okviru ovog rada prikazano je rješenje i način praćenja poremećaja prijavljenih od strane nadzornog časovnika. Česte su situacije da se u radu na realnoj platformi nailazi na poteškoće po pitanju očekivanog ponašanja hardvera na određene zahtjeve iz softvera. Kada govorimo o samom nadzornom časovniku i reakciji fizičkog upravljača nekada sa sigurnošću ne možemo da tvrdimo šta je uzrok okidanja greške i gašenja sistema. Sigurnosno gašenje može značajno usporiti proces analiziranja nekog problema koji sam po sebi ne mora biti vezan isključivo za nadzorni časovnik. Takav vid poteškoća je moguće pratiti samo isključenjem upravljača. Simulacijom je omogućeno da se minimalizuju ta odstupanja, predvide greške u sistemu i olakša analiza sistema. Greška koja se desi na jednoj kontrolnoj tački može da prijavi grešku tek na sledećoj kontrolnoj tački koja je u redu. Rad omogućava da se isprati i predvidi takav vid prekršaja.

Kao što je prethodno pomenuto za ovaj rad je korišten već postojeći raspored zadataka koji sam po sebi predstavlja preduslov za početak simulacije. Budući rad obuhvatiće unaprjeđenje postojećeg rješenja simulacijom operativnog sistema, gdje će se voditi računa i o simulaciji raspoređivanja zadataka kako bi se poremećaj mogao nanijeti direktno u raspoređivanju i ispratiti cijeli proces. Ovaj vid unaprjeđenja značajno bi mogao poboljšati analizu i predviđanje grešaka.

LITERATURA

[1] AUTOSAR, "*Layered Software Architecture*," *[Online]*, https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf., 2017-12-08 [Accessed July 2021]

[2] B. Catalin-Virgil, F. Ioan and F. Heininger, "*A new trend in automotive software: AUTOSAR concept*" SACI, IEEE 8th International Symposium, Timisoara, Romania, May 2013.

[3] Techemergence, "*The Self-Driving Car Timeline –Predictions from the Top 11 Global Automakers*", 2018.

[4] H. Fennel et al. "*Achievements and exploitation of the AUTOSAR development partnership*" SAE Technical Paper Series 2006-21-0019, SAE International, October 2006,[Accessed July 2021]

[5] Mazen Ahmed, Mona Safar, "*Formal Verification of AUTOSAR Watchdog Manager Module Using Symbolic Execution*", IEEE 30th International Conference on Microelectronics, 2018

[6] Mazen Ahmed, Mona Safar , *"Symbolic Execution based Verification of Compliance with the ISO 26262 Functional Safety Standard"*, IEEE 14th International Conference on Design & Technology of Integrated Systems In Nanoscale Era, 2019

[7] J. Ganssle, "*Watching the Watchdog*", Embedded World, 2003.

[8] *AUTOSAR* , "*Specification of Communication*" *[Online]*, https://www.autosar.org/fileadmin/user_upload/standards/classic/3-2/AUTOSAR_SWS_COM.pdf [Accessed July 2021]

[9] Michael Kunz, "*OSEK OS"*, March 18, 2009, http://www.uni-obuda.hu/users/schuster.gyorgy/rtos/OSEK.pdf [Accessed July 2021]

[10] The AUTOSAR Consortium, "*AUTOSAR Specification of Operating System"*, pp. 33-35, 2006.

[11] Nahmsuk Oh, P. Shirvani, E. McCluskey, "C*ontrol-Flow Checking by Software Signatures*", IEEE Transaction on Reliability, vol. 51, Mar-2002

[12] Texas Instruments MCUSW, "*Wdg Design Document " [Online]*, *http://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/latest/exports/docs/mcusw/mcal_drv/docs/drv_docs/design_wdg_top.html* [Accessed July 2021]

[13] *AUTOSAR*, "*AUTOSAR SWS Watchdog Manager*" *[Online]*, https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_WatchdogManager.pdf , *2017-12-08* [Accessed July 2021]

[14] *AUTOSAR*, " *Specification of Watchdog Interface*" https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_WatchdogInterface.pdf, [Accessed July 2021]

[15] *AUTOSAR*, "*Specification of Watchdog Driver* "*[Online]* https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_WatchdogDriver.pdf , [Accessed July 2021]

ABSTRACT

With the development of the automotive industry and software within it, as a technical consequence, there was a need for mandatory integration of protection mechanisms in the embedded cores of the operating system. One of the basic mechanisms for system protection is the watchdog. This component aims to monitor all other components initiated by the scheduler and thus enable the safe operation of the system. As the problems reported by Watchdog are relatively difficult to track and analyze in a real system, the idea came up to simulate the operation of the Watchdog component on a computer with the same input parameters as in a living system. This paper provides a solution for simulating the system monitoring mechanisms of the defined AUTOSAR architecture. The simulation makes it possible to minimize deviations, predict errors in the system and facilitate the analysis itself. Work can contribute to faster system development because it allows to predict errors that will occur in the system before implementation.

ONE SOLUTION FOR ANALYZING AND DISPLYING CHECKPOINTS IN THE AUTOSAR WATCHDOG CONFIGURATION

Ivana Tešević, Branko Milošević Dejan Bokan, Bogdan Pavković

Implementation of Smooth Streaming protocol through a generalized software framework

Miroslav Suša , Ilija Bašičević, *Senior Member, IEEE*

Abstract—Adaptive streaming is a technology for transmitting multimedia content over a network such as the Internet. This way the content is available at any time which has brought big changes. One of the many streaming technologies is Smooth Streaming. In addition to the transmission of content via one of the protocols, it is necessary to ensure its reproduction. In this paper, the implementation of the Smooth Streaming protocol within a single media player is presented. The implementation was performed through a generalized software framework, which will also be discussed. The role of the framework is to facilitate the integration of the remaining adaptive streaming protocols into the media player.

***Index Term*—adaptive streaming; content playback; Smooth Streaming;**

I. Introduction

The fourth industrial revolution brought many changes in terms of consuming multimedia content. When it comes to content transfer, the most important innovation is Streaming technology. Streaming makes it easier for users to access content whenever and at any time they want, which previous technologies could not provide. In the field of television, new technology is completely taking over the market from the traditional way of content broadcasting[1].

As a consequence of the development of a new way of data transfer, standards have emerged according to which this transfer will be performed. Some of the best known and most prevalent today are the MPEG-DASH and Smooth Streaming standards. In addition to the transmission of the content itself, it is necessary to ensure its reproduction.

This paper presents an implementation of the Smooth Streaming protocol, the creation of a generalized software framework for managing adaptive streaming protocols, as well as the integration of the software framework with the Smooth Streaming protocol and a media player for content playback.

The rest of the paper is organized in the following order: Section II discusses streaming technology and its types. Section III shows the architecture of the media player on which the work solution is implemented. Section IV defines a programmatic framework for managing adaptive streaming protocols. Section V discusses creating a library for the Smooth Streaming protocol. Section VI shows the architecture of the media player after applying the solution. Section VII deals with the validation of the solution and the results. Section VIII provides a conclusion on the work.

Miroslav Suša – RT-RK Institute for Computer Based Systems, Novi Sad, Serbia (e-mail: miroslav.susa@ rt-rk.com).

Ilija Bašičević – Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (e-mail: ilibas@uns.ac.rs).

II. Streaming

Streaming is a technique of continuous transmission of video and audio material via wired or wireless internet connection. Before the advent of streaming, playback of content from the Internet was possible in two ways. The first way is to upload the complete file to the device and only then is playback possible. Another way is to use a progressive download.

A. *Progressive streaming*

Progressive download allows you to play content while downloading it to your device [2]. Downloading is done regularly, which means that not only the selected part of the file can be downloaded, but the complete one. Any part of the downloaded content can be played as desired. The content transmitted in this way is of fixed quality and resolution. In other words, only one video file can be uploaded. Since different content resolutions require better or worse internet traffic, in a situation where the flow is poor, progressive downloads will often lead to transmission interruptions. In addition, the content will be displayed differently on different devices.

Higher resolution files take up more memory space, and the file transfer speed depends on the internet flow which tells us how much data the user can receive in a unit of time. If the flow is poor and the video has a higher resolution, part of it will not be able to be transmitted in its entirety and playback will be delayed. In addition to downtime, progressive downloads also cause the problem of presenting videos on devices with different screen resolutions. For example, when playing a video that is 720p resolution, on a screen with 1080p resolution, the image will be stretched and pixelated.

B. *Adaptive streaming*

Adaptive streaming is a streaming technology based on the HTTP (HyperText Transfer Protocol) protocol. The benefit of using HTTP technology is the unhindered passage through the firewall and NAT (Network Address Translation) devices that remap IP (Internet Protocol) addresses. In addition, the complete implementation of HTTP logic is on the side of the content seeker, which reduces the need for a continuous connection between the provider and the service provider.

Adaptive streaming, instead of a complete file, transmits and plays its parts for a few seconds [3]. We call such parts of a file its segments. Since the content is divided into segments, any part of it can be added as desired. Just before the segment expires, the next one to be played is delivered. After moving on to the next segment, the previous one is deleted, and the process takes place until the complete content expires. This gives the impression of continuous playback of content. Information about all video and audio files and their segments, as well as details of the need for their transfer and playback can be found in the manifest file.

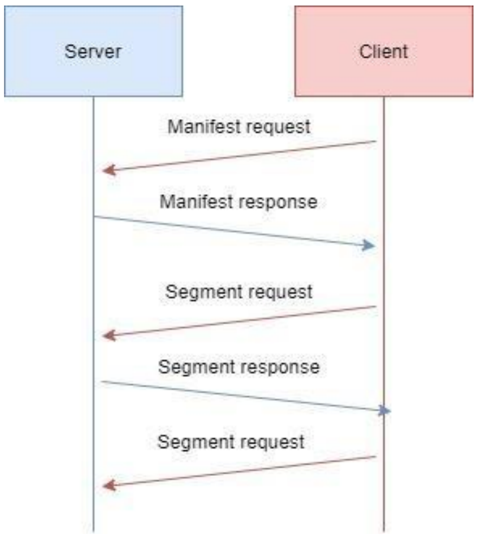


Fig. 1. File exchange between client and server during adaptive streaming

Adaptive streaming solves the problems of progressive download [4]. Files of different resolutions are created for the same content. Depending on the size of the client's screen, a file segment of the appropriate resolution is provided.

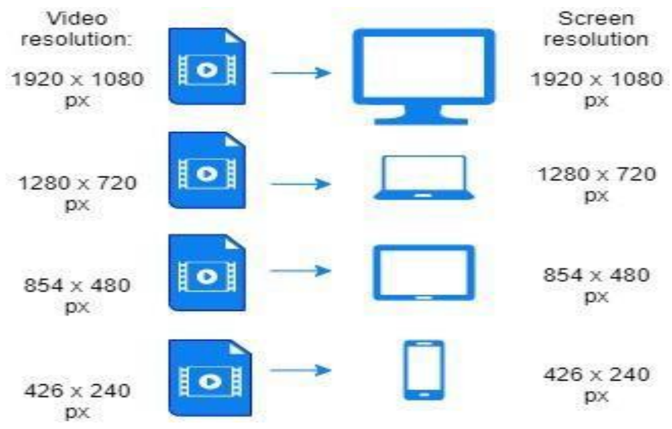


Fig. 2. Display video resolution selection for different screen resolutions.

In the typical communication scenario between a client and a server using adaptive streaming the client first sends a request for a manifest and receives it in response from the server. The client then sends requests for fragments which the server delivers.

III. Media player arhitecture before protocol implementation

IWedia Player (IWP) is a library written in the C ++ programming language that aims to provide a high-level player interface. The player allows you to play audio and video content. It is used as a part of an application written for the Android platform and provides it with a user interface.

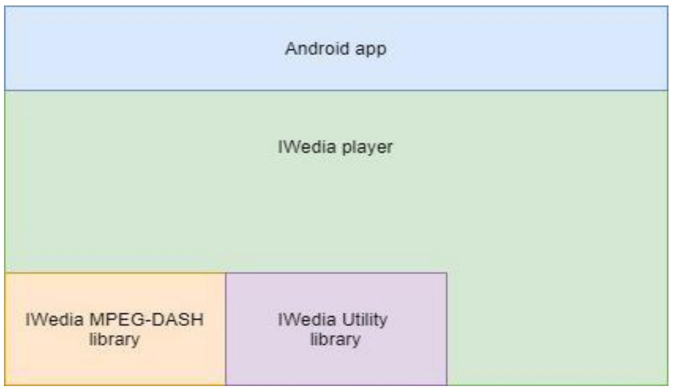


Fig. 3. Display of player architecture before implementing the solution.

In order for the player to enable the MPEG-DASH streaming protocol, an IWP-DASH library was created. In addition to defining the elements of this streaming protocol within the dash library, the logic for adaptive streaming has been implemented, which is closely related to the dash protocol. Components shared between other IWedia libraries such as the player and dash are housed in the IWedia utils library.

IV. Software framework for managing adaptive streaming protocols

When it became necessary for the media player to support, in addition to the MPEG-DASH protocol other adaptive streaming protocols as well, the implementation logic had to be generalized and displaced from the dash library.

The goal of the adaptive streaming protocol management framework is to provide the media player with an interface through which to obtain content for playback. All protocols aim to transfer content and regardless of their complexity, we can see numerous similarities between them. Since MPEG-DASH is an official international standard, it is the most developed and provides the most opportunities during implementation [5]. All other standards can be viewed as its subset.

The software framework can be divided into four logical units that have a role in downloading manifests, segments, adapting to network conditions and creating an entrance to the library.

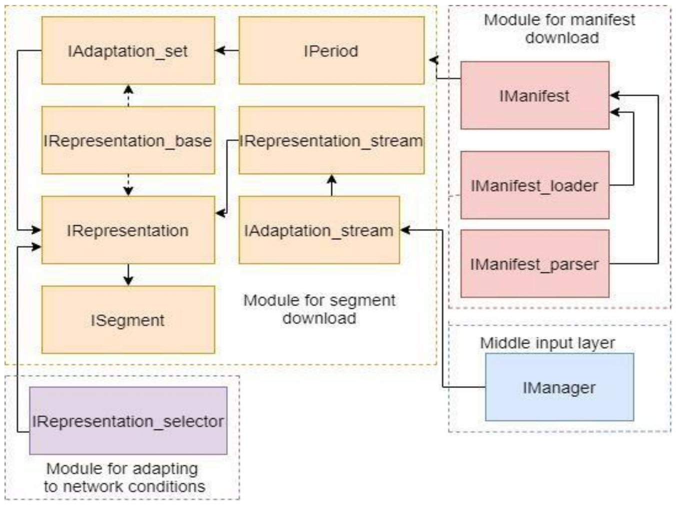


Fig. 4. Generalized software framework solution architecture.

A. Module for manifest download

Manifest, as mentioned, is the central document for gathering information on the content to be transmitted. It is available on the server along with the provided content. The type of manifest can be dynamic, if we broadcast live content, or static, if the content is downloaded on demand. To stream live content, the manifest needs to be delivered periodically because the content is constantly changing.

To provide a manifest, URI (Uniform Resource Identifier) from which it can be downloaded is required. The download is performed with the help of a previously implemented download class, which needs to be provided with data about the speed, number of attempts and download time of the manifest. As a result of successful delivery, a string with the contents of the manifest is obtained. After it is downloaded, the manifest is parsed.

At the level of libraries that implement the standard, it is necessary to implement interfaces that represent the manifest and the factory for creating the manifest.

B. Module for segment download

Within period elements of the manifest, that contain the initial time and duration of the content, there are adaptation elements. Their primary purpose is to provide information about the type of stream being transmitted. Inside the adaptation element are elements that represent the stream. They contain data on the flow rate required to download the stream in a certain quality as well as information on the segments that need to be downloaded. At the stream type level, a structure is created that will download the segments.

Libraries of specific protocols that implement the created interfaces define the form of stream representation as well as the form of segments. The representation form creates segments based on a given start time, carries information about the number of available segments, as well as the broadcast time period provided by the representation.

Segment download control is defined in this software framework. The time period in which the download will be performed is determined, the representation within which the segments will be downloaded is selected, the ordinal number of the next segment to be created is calculated and the creation of the segment is initiated.

C. Module for adapting to network conditions

The network adaptation module is key to performing adaptive streaming. Depending on the speed of the user's Internet flow, it is necessary to correct the representation of the stream being downloaded. The factor that influences the choice of representation, in addition to the flow rate, is the type of content that is downloaded. For the purpose of selecting a representation, a selector is created that stores all available representations and, based on the current flow and type of content, selects one of them to be played. After the download, the number of bits downloaded, as well as the time period required for the download, are forwarded to the flow rate meter. Based on the obtained parameters, the meter calculates the current flow rate that is available when initializing the next download.

Middle input layer

As part of the software framework, a manager has been created to manage the processes. Its presence is necessary in the media player that uses the library. The task of the manager is to initiate the loading of the manifest when it comes to streaming on demand or perform periodic loading of the manifest if a live broadcast is performed, as well as the interruption of these operations. In addition, the manager creates a program representation of the adaptation stream that has the ability to further manipulate stream representations and segments.

D. Software framework integration with the media player

In order to enable the reproduction of content by adaptive streaming protocols, it is necessary to integrate the software framework with media player. The integration is done by adding a component that has access to the software framework manager. In this way, the processes realized by the software framework are initiated, such as taking over the manifest and adding segments. The ultimate goal of process initiation is to obtain segments and prepare them for reproduction.

Within the media player, there is also logic for determining the adaptive streaming protocol that will be used, as well as factories that will, depending on the selected protocol, create a component that has access to the software framework.

V. Smooth Streaming library

The Smooth Streaming library is a C ++ implementation of the Microsoft Smooth Streaming protocol used by the IWedia player to play content. The library consists of "*ismc*" and *"abr"* modules. The *Ismc* part of the library was named after the extension of the Smooth Streaming protocol manifest client. Within this part, the manifest is parsed and elements and attributes representing the data collected by parsing are realized. *Abr* part of the library represents the implementation of a software framework for managing adaptive streaming protocols.

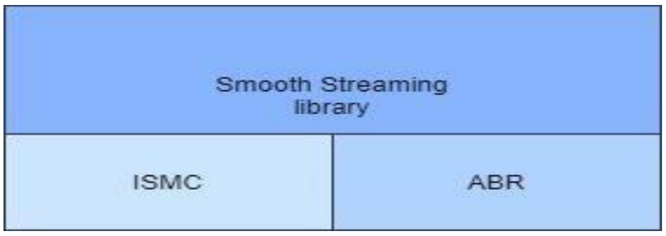


Fig. 5. Smooth Streaming Library Components.

A. Library creation

In order to implement the protocol, it is necessary to parse the protocol manifest and present its elements within the library. In addition, it is necessary to provide the types of content exchange messages defined by this transport protocol, which are: manifest request, manifest response, segment request and segment response.

1) Manifest request

A manifest request is sent to obtain a manifest containing all the necessary information to reproduce the content. In order to send this request, a URI to the manifest is required as well as information on which extension of the manifest file is expected. Manifest extensions differ in whether it is a server manifest that has an *ism* extension or a client manifest whose extension is *ismc*.

2) *Manifest response*

The manifest response is obtained in the form of an *ismc* file with metadata related to the playback of the content. The file is a well-formed XML (Extensible Markup Language) and consists of the following elements: SmoothStreamingMedia, Protection, StreamIndex, QualityLevel and StreamFragment. All of the above elements are presented within the library as classes, and their correlations are clearly visible and described below.

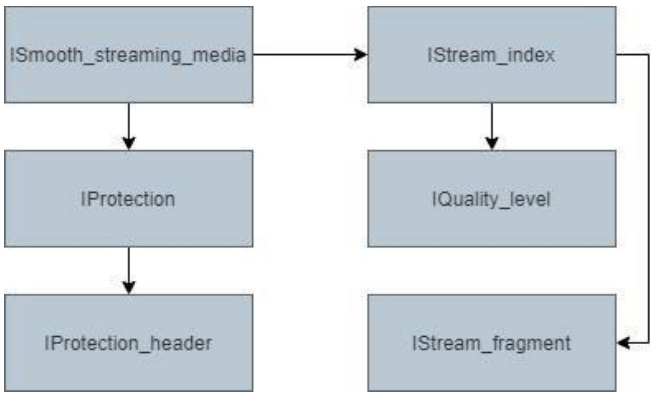


Fig. 6. Smooth Streaming library solution architecture.

a) SmoothStreamingMedia

SmoothStreamingMedia is a root element that contains all the other elements of the manifest. The direct descendants of this element are the StreamIndex and Protection elements. Its attributes carry information about the main and secondary versions of the manifest as well as whether the manifest describes live or on-demand content. Within the attribute, the duration of the content described in the manifest is also defined.

SmoothStreamingMedia is implemented within the library so that its creation requires URI of manifest as well as xml files in string format. The string is then parsed using the sub-element names and attributes listed as constants.

a) Protection

Protection is an xml element that includes the metadata needed to play protected content. It contains information on the unique identification of the security system used on the given content, as well as the encoded data that the system uses to enable the reproduction of the content to authorized users.

b) StreamIndex

StreamIndex is the most important element within a manifest because it contains metadata for playing a specific stream. This means that the element provides information about the type of content that is transmitted by a particular stream, that is, whether it is an audio, video or text stream. Based on the stream type, the availability of attributes within an element changes. Only in the case of video, there is information about the maximum available content resolution that is available, as well as the recommended playback resolution. The number of qualities, segments, as well as the duration of the stream are available within this element.

b) QualityLevel

The QualityLevel element carries metadata about the playback of a specific track within the stream. Depending on the type of stream in which it is located, its attributes differ. For video within the video stream, the required resolution attributes as well as parameters specific to a particular media format are required. When it comes to audio recording within the audio stream, in addition to the previously mentioned attributes, we also have data on the number of channels of the audio tape, sampling rate, sample size, limits for optimizing audio decoding and identification of media format used. The attributes that each record contains are those that carry information about the unique identification of each record and the download speed required to retrieve a particular record.

c) StreamFragment

The StreamFragment element contains metadata about a set of related segments in the stream. Its attributes carry information about the start time of the segment, its duration, the order in a series of segments as well as the possibility of repetition. For a segment to be valid, it must contain either a duration attribute or a start time attribute. A series of segments is called adjacent if the start time of any segment, with the exception of the first, is equal to the sum of the start time and the duration of its predecessor.

3) *Segment request*

A segment request is created to retrieve the desired segment from the server. To create it, it takes URI to the desired segment, its bitrate, the name of the stream within which the fragment is located, the start time of the desired stream, as well as the type of response that the client expects from the server.

4) *Segment response*

A segment response is a response that is received after sending a request to obtain a segment. The answer can be complete or partial. If the answer is complete it contains media and segment metadata, while partial responses contain only media or metadata.

B. *Framework implementation*

In the Smooth streaming library it is necessary to implement two of the four modules of the framework and they are: Module for manifest download and Module for segment download.

2) *Implementation of the module for manifest download*

As mentioned earlier, it is necessary to implement the manifest factory interface as well as the manifest interface.

a) IManifest interface

IManifest methods gather the necessary information that each manifest should have, namely: whether the manifest is live or on-demand, the duration of the manifest, the minimum time required to load the manifest, as well as adding the manifesto period. All data can only be obtained by parsing the manifest.

b) IManifest_factory

The manifest creation factory contains only one method that instantiates a class that implements the IManifest interface. It forwards manifest uri and the contents of the manifest that is necessary to parse.

3) *Implementation of the module for segment download*

Module for segment download defines the necessary logic to supply the parsed element data, as well as the logic for creating segments.

a) IRepresentation

Interface methods obtain, from the QualityLevel element, data described in the part of the paper with the same name. All data is present in the node and is very easy to obtain.

b) IAdaptation_set

The IAdaptation_set interface is composed from set of methods that retrieve data from the Stream_index element of the library. All methods return the present attributes or sub-elements of a given element.

c) ISegment

This interface is defined by a set of methods for retrieving the Stream_fragment element attribute with the exception of the get_uri method. The get uri method calculates the uri to a given segment that is different from the manifest uri

d) IRepresentation.

The role of the IRepresentation interface is to create segments, add the total number of segments, add the duration of all segments and find the segment with a given index

The number of segments is obtained when initializing the class of this interface by going through all segments and taking into account their repeat attribute which tells how many times a given segment is repeated.

During the process of calculating segments, the total duration of all segments can be easily obtained. The timestamp of the first and last segment is taken, or their length and repeat tag if the timestamp is not available.

A segment with a given index is supplied by going through all available segments, taking their duration and repeat attribute, calculating the index of each segment and returning the resulting one. The limitation of this method is to pass an index that is not less than zero and that is not greater than the total number of segments.

VI. Media player arhitecture after protocol implementation

By removing the definition of adaptive streaming protocol from the dash library and generalizing it, a software framework for managing adaptive streaming protocols is obtained. This makes it easier to use and add new adaptive streaming protocols such as the Smooth Streaming protocol. In addition, their integration with the player is facilitated.

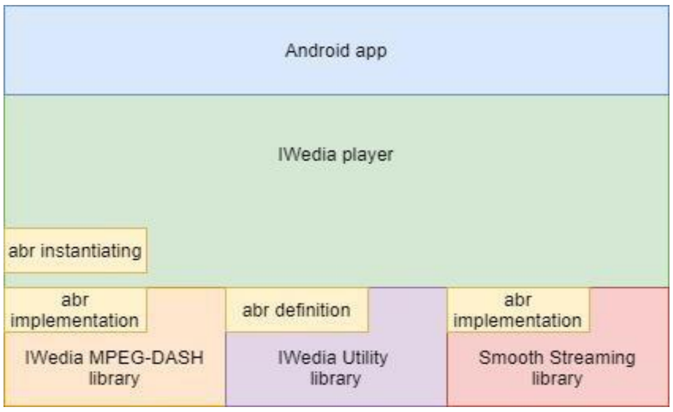


Fig. 7. Display of player architecture after solution implementation.

VII. Testing

A. Description of the test environment

The environment for testing of this solution comprises Smooth Streaming content that can be accessed via the network, an application for playing content, as well as an Android P development board on which the application will be launched.

The content playback application is written in the Java programming language for the Android platform. It provides a simple user interface from which content playback can be controlled, and uses the IWedia player library interface for playback itself.

Content preparation, implementation of the Smooth Streaming library and software framework, as well as their integration with the media player are described in the previous chapters. The Android P development board connects to the same network from which the prepared content is available to it, and the Android application is installed and launched on it. With this step, the test environment is ready and testing can begin.

B. Testing procedure

After installing the Android application on the board and launching it, you get access to the list of all available streams. Clicking on the desired stream starts playback. Playback can be interrupted, paused or restarted at any time.

C. Test results

As stated, by clicking on the desired stream, in this case on the stream belonging to the group of Smooth Streaming streams, playback starts. The start of content playback always takes place at a lower resolution until the user's internet flow is determined. After that, if it is determined that the conditions are met, playback continues at higher resolutions, which tells us that the SmoothStreaming protocol has been successfully implemented.

Selecting one of the available streams that are transmitted by other adaptive streaming protocols results in content playback. Successful playback start shows that the generalized software framework has been correctly implemented.

VIII. Conclusion

Within this paper, a solution for integration of Smooth Streaming standards for broadcasting content is described. Also, a generalized program framework has been implemented, which enables easier integration of the remaining standards.

The protocol library and software framework are written in C++. In this way, speed and flexibility are achieved. Like the media player in which they are implemented, they can be used on both Android and Linux platforms.

The integration framework in the media player enables easier integration of existing, as well as the protocols that may arise in the future which can be seen as a subset of the MPEG-DASH protocol.

Acknowledgment

On this occasion, I would like to thank my colleague Nikola Špirić on the provided support.

References

[1] Nenad Lovcevic, Jelena Simic, Miroslav Dimitraskovic and Ilija Basicevic “Modul za prijem i obradu JSON komandi u programskoj podršci digitalnog TV prijemnika” 61st IcEtran conference in Kladovo, Serbia, June 5 – 8, 2017.
[2] Stefan Lederer, Christopher Mueller,Christian Timmerer, Hermann Hellwagner “Adaptive Multimedia Streaming in Information-Centric Networks” IEEE Network, November 2014.
[3] Ilija Bašičević, Nenad Lovčević, Nenad Šoškić, Milan Ačanski “Internet as Infrastructure for Digital Television” 62st IcEtran conference in Palić, Serbia, June 11 – 14, 2018.
[4] Rubem Pereira, Ella Pereira “Dynamic Adaptive Streaming over HTTP and Progressive Download: Comparative Considerations”, IEEE 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA) - Victoria, Canada, May 1 2014
[5] Sunho Seo; Younghwan Shin; Jusik Yun; Wonsik Yang; Jong-Moon Chung “Adaptive high-resolution image transmission method using MPEG-DASH” International Conference on Information and Communication Technology Convergence (ICTC) Jeju, Korea (South), October 18-20, 2017.

Implementation of the GDPR Compliant Data Handling for Smart Home Solution

Sandra Bugarin, Sandra Ivanović, Marija Antić

Abstract—**The amount of personal data collected and shared in the Internet of Things (IoT) is causing increasing concerns regarding the user privacy in IoT. The recently introduced General Data Protection Regulation (GDPR) is a legal framework that sets guidelines for the collection and processing of personal information and aims to strengthen user rights. In order to comply with the GDPR requirements, the existing smart home system is extended with the cloud service, responsible for user consent management and appropriate data handling. The architecture of the solution, as well as the results of functional and performance testing are presented in this paper.**

***Index Terms*— GDPR; smart home automation; IoT**

I. Introduction

The users surfing the web are under a risk of privacy violation, as the websites are collecting data about them and may be sharing it with third party services. Recently, the General Data Protection Regulation (GDPR) has entered into force, with the aim to protect the user privacy, and allow them better control over the collected data and the scenarios it is used in [1]. According to GDPR, the services are required to inform the users about the types of data collected and the purpose of this action, so the users can choose to engage only with websites and services that do not violate their privacy, or opt out of the use of their information for particular purposes.

While the data collected by the websites usually serves only marketing purposes, and is not necessary for the normal operation of the website, the problem of GDPR compliance in Internet of Things (IoT) solutions is of a more complex nature [2]. Namely, IoT systems typically connect multiple devices owned by a single user, and allow them to perform a certain function together. Therefore, the exchange of data is in the essence of IoT. On the other hand, there exists a tendency in the IoT solutions to collect more data than actually needed for the normal system operation, as it may become useful in the future scenarios [3], [4]. This data should be carefully stored and protected, as well as anonymized [5], and the users should be provided with the mechanisms to inspect or delete the collected data at any time [6]. Also, it is necessary to be transparent about the ways data is processed, to inform the users timely when the privacy policies change, and to allow them to opt out of the service if they do not agree to the changes.

Studies have been conducted that show that the user attitude towards data collection depends on multiple factors, such as the environment the data is related to (home, office, traffic), types of data collected (video, photo, sensory data, voice), who has access to it (government, businesses), as well as the purpose of data collection (safety, convenience, marketing) [7]. Smart home users are willing to allow data collection as long as it is used only within the system, for the purpose of connectivity and convenience [8], but seem not aware of the possible privacy issues associated with machine learning and potentially sensitive information that can be revealed by data analytics [9]. This information should be communicated through the privacy policy and terms of use, in a manner that is transparent and clear to the user, and explains why certain types of data are needed for the normal operation of the system [10].

In this paper, we extend the existing smart home solution with the cloud service responsible for GDPR-compliant data handling. This service allows administrators to handle privacy policy updates, and the users to request the export or deletion of personal data, as well as the deletion of the user account. First, we introduce the smart home solution architecture in Section II. Then, in Section III the operation of the GDPR service is explained, while the results of functional and performance testing are presented in Section IV and Section V.

II. Smart Home System Architecture

The smart home solution we extend is comprised of a gateway, client applications (Android, iOS and web) and cloud services.

The gateway is a key component in the smart home because it acts as a bridge between clients and smart devices in the smart home system. Gateway's main purpose is to pull together all compatible devices into a universal platform. This allows applying control scenarios to all of them while being agnostic of the actual communication interface – ZigBee, ZWave, and IP nodes are seamlessly integrated into one unified device/node network. On top of this core functionality gateway implements network API's for client applications, mechanism to define and execute rules, advanced control over the home zones, firmware upgrade, backup/restore, etc.

Sandra Bugarin is with OBLO Living, Narodnog fronta 21a, Novi Sad, Serbia (e-mail: sandra.bugarin@ obloliving.com).

Sandra Ivanović is with the Faculty of Technical Sciences, University of Novi Sad, Serbia (e-mail: sandra.ivanovic@rt-rk.uns.ac.rs).

Marija Antić is with the Faculty of Technical Sciences, University of Novi Sad, Serbia (e-mail: marija.antic@rt-rk.uns.ac.rs).

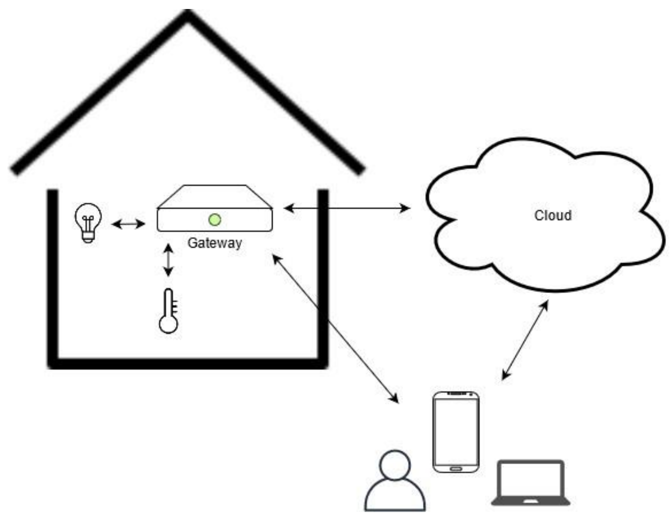


Fig. 1. Smart home system architecture.

Client applications provide the user interface for system provisioning, configuration and management. They enable users to access their home gateway in local network or remotely over the cloud. On the other hand, the cloud is responsible for user and gateway identity management, mirroring the smart home gateway configuration and data, allowing remote access for the client applications, historical data collection and analytics.

For certain functionalities of the system to be enabled, the user needs to provide the address of the household, i.e. their geolocation [11]. Also, phone number and email are needed for the purpose of smart notifications. Additionally, the system collects and stores the state changes of all devices, in order to provide the users with the possibility to inspect the way certain device types have been used in the previous period [12]. Also, the information about the local IoT network is stored for diagnostics purposes. All of these entries represent the data that should be treated according to GDPR.

III. GDPR-Compliant Data Handling

To comply with the GDPR requirements, the microservice is created within the smart home solution cloud, which enhances the system with the following functionalities:

- Update of Privacy Policy and Terms of Service
- Export of Personal Data,
- Deletion of User Account

In this section, the details about the service implementation will be presented. All of the cloud services are highly available, and GDPR service is no exception. The simplified architecture is presented in Fig. 2. Multiple instances of the service implemented in Node.JS are running on the environment. They share the long-term MongoDB data storage, as well as the temporary Redis storage. Also, the shared cloud storage disk is available to all services that need to store large files, not suitable for MongoDB database. To control the load and orchestrate tasks within the environment, RabbitMQ is used.

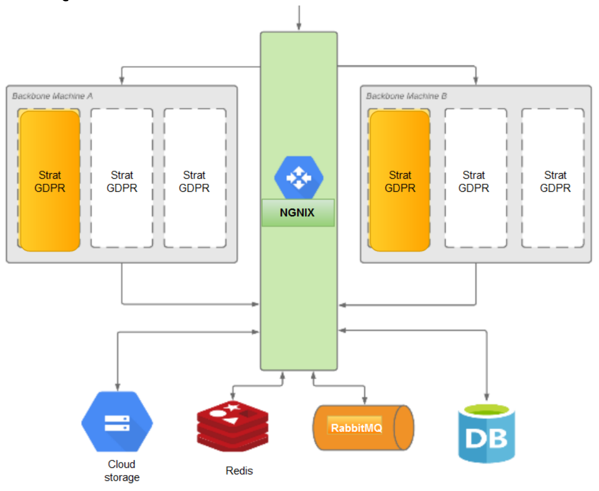


Fig.2. Highly available GDPR service.

A. Update of Privacy Policy and Terms of Service

The Privacy Policy should help users to understand what information is collected, for which purpose, and how users can update, export, and delete their information. Information about privacy policy and terms of service is the part of registration process, so all new users have to read it, and agree in order to register their smart home account.

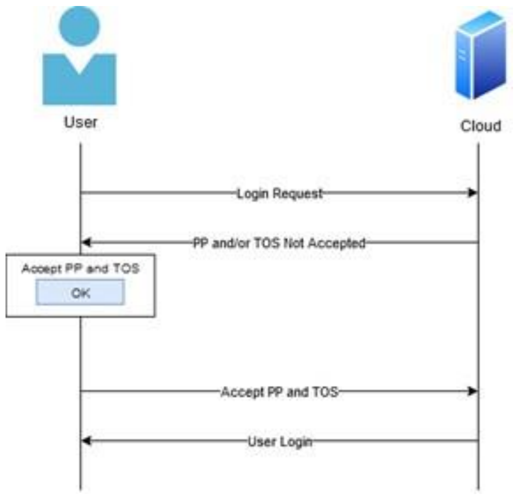


Fig. 3. Privacy policy acceptance upon login.

The existing users that have not read the new privacy policy well be prompted to read it and agree to it after logging onto web or mobile applications, as presented in Fig. 3. Until they agree to new privacy policy, users will not be able to use the applications. Gateways connected to user's account will be deactivated and prevented from sending any new sensory information to the smart home cloud.

Users can reactivate their account and gateways if they accept new privacy policy on login, or if they click on the link to new privacy policy that has been emailed to them.

B. Personal Data Export

As already said, personal data consists of user's personal profile information, such as name, email, phone contact, geolocation and address of the household. It also includes the state history of the end devices in the system, gateway backups and local IoT network history logs, which are stored for the purpose of diagnostics. Therefore, the exported data contains three groups of JSON files. The first group contains the data from the user's profile, the second one represents the snapshot of the gateway's current state, while the third one represents the usage history of all devices that have been connected to user's gateway(s).

The data export service will run on demand, under control of administrator. It performs the following tasks:

- Database crawl for personal data,
- Compression of this data to a ZIP archive, which is temporarily stored in the cloud, until the user downloads it,
- Deletion of outdated personal data

The collection and deletion of all data for an individual user can be started by the administrator, upon a request from the user (Fig. 4). Administrators can start or stop data export task that have not been completed, and delete completed export tasks and data file associated with them if they are older than 15 days. Also, they can monitor the progress of currently running data collection tasks. Administrators are not able to view the contents of the exported data files or to remove data export tasks that still haven't completed.

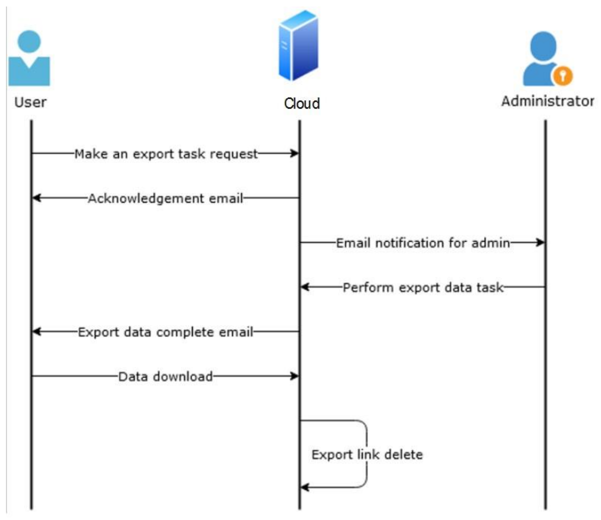


Fig. 4. Export data flow

When the user makes a personal data export request, they will be notified via email that their request has been acknowledged. A similar notification will be sent to the administrator, with the link that allow to monitor that export task. When the export task has been completed, another notification will be sent, this time with a HTML link to data export file. Download link will be available for the next 15 days. After this period, the export task and the associated export data file will be removed.

By default, only single process per backbone instance is allowed to execute data collection and compression tasks. Reason for this is intense I/O and CPU utilization (for DB crawl and data compression, respectively). Every process will be given a certain amount of time to complete it (e.g. 5 minutes) by placing the key-value pair in redis with the same expiration time. Given that database and redis are the only shared state between backbone instances, they can be used for tracking of task progression: if one of the instances that is running collection task crashes or restarts, time for task completion will expire and this task will fail.

C. User Account Deletion

At any time, a user can request to delete their account. Administrators are obliged to fulfil this request, by performing the account deletion operation via the administrative portal – Fig. 5. During this process, all of the gateways assigned to the user will be un-assigned from the user account, and all personal data from the cloud will be deleted.

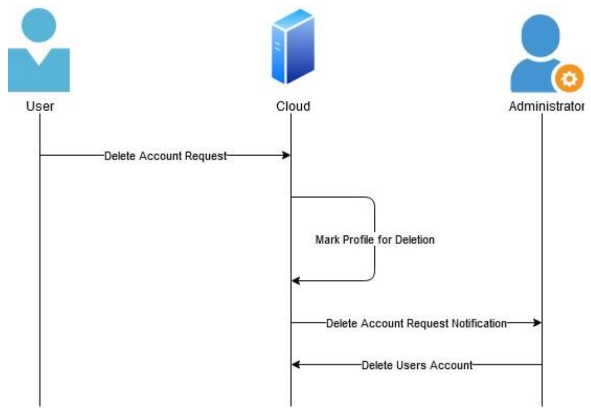


Fig. 5. Account deletion flow

However, the device usage data will be kept for analytics purposes. This data is in anonymized state, which means that it does not contain any information that can be traced to the original user.

IV. Functional Verification

A. Privacy Policy Acceptance and Modification

During the process of account creation, the user is asked to agree to the terms of service and the privacy policy.

The administrator can upload the new privacy policy and terms of service documents using the web portal for system administration – Fig. 6.

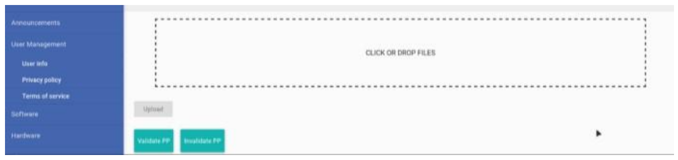
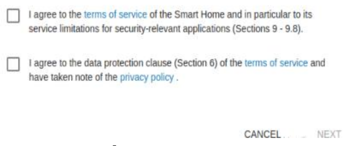


Fig. 6. Privacy policy and terms of service update.



☐ I agree to the terms of service of the Smart Home and in particular to its service limitations for security-relevant applications (Sections 9 - 9.8).

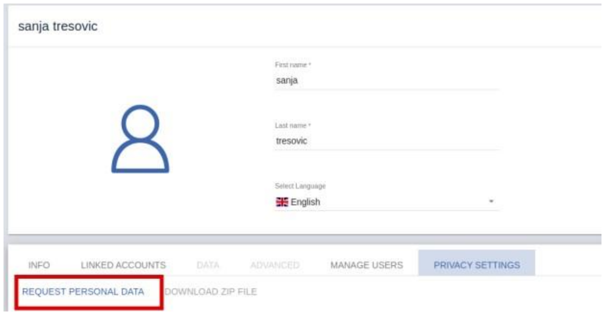
☐ I agree to the data protection clause (Section 6) of the terms of service and have taken note of the privacy policy .

Fig. 7. Modal dialog prompting the user to accept new privacy policy.

On next login attempt, every user will be prompted to accept new privacy policy and terms of service via modal dialog – Fig. 7. Until they accept, they will not be able to use the applications.

B. Data Export

On the user profile, a button is implemented which allows them to request the export of personal data. This button is disabled if another request is already processed. This tab also contains a link to personal data when collection task is finished – Fig. 8. Implications are, that a new data export request can be made after 15 days (guaranteed duration of the valid export link) plus the time needed to perform the data export request.



sanja tresovic

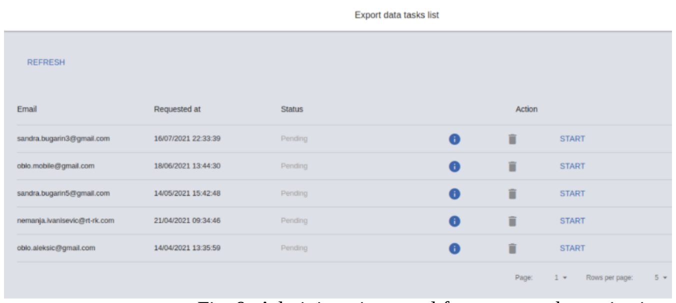
First name *
sanja

Last name *
tresovic

Select Language
English

Fig. 8. User requesting personal data export.

From the administrator side, the status of the pending, current and past data export tasks can be monitored, as in Fig. 9.



Export data tasks list

Email	Requested at	Status	Action		
sandra.bugarin3@gmail.com	16/07/2021 22:33:39	Pending	ⓘ	🗑	START
oblo.mobile@gmail.com	18/06/2021 13:44:30	Pending	ⓘ	🗑	START
sandra.bugarin5@gmail.com	14/05/2021 15:42:48	Pending	ⓘ	🗑	START
nemanja.ivanisevic@rt-rk.com	21/04/2021 09:34:46	Pending	ⓘ	🗑	START
oblo.aleksic@gmail.com	14/04/2021 13:35:59	Pending	ⓘ	🗑	START

Page: 1 Rows per page: 5

Fig. 9. Administrative panel for export task monitoring.

V. Performance testing

We have tested the implemented solution to asses the average time needed to prepare the export ZIP file with user data, depending on the data size. Typically, the size of the exported data is 5-10 MB, although for the setups with many devices it can increase up to 30 MB. The time needed for data export is presented in Fig. 10. It can be observed that the data export can be performed in less than 10 s for typical setups, while for the larger setups the time needed increases to the order of minutes. However, since the user will be informed by the notification when this process is finished, the performance of the solution is acceptable for the practical purposes.

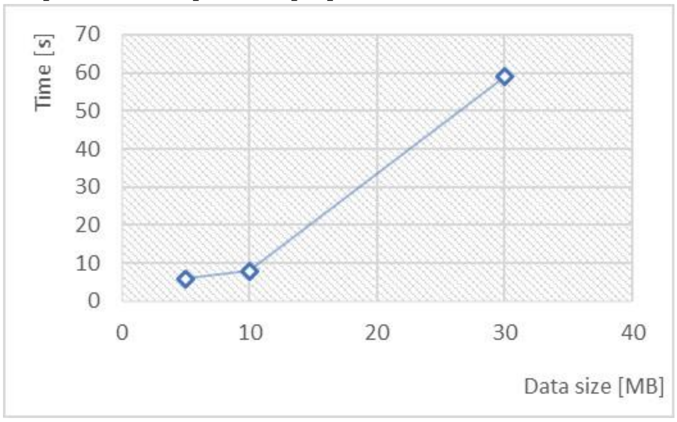


Fig. 10. Time needed to export data depending on the total size of the data file.

VI. Conclusion

In this paper, one implementation of the GDPR-compliant data handling in smart home solution has been presented. The cloud service was created, that handles the relevant aspects of data handling and user consent management, such as the update of terms of use and privacy policy, data export and account deletion. The implemented functionality has been verified, and it has been shown that the times needed for data export are acceptable. In the future work, this solution will be extended to allow users the finer granulation over the types of data collected and services enabled. For example, the users may want to opt out of the advanced functionalities, based on data analytics and machine learning, while still wishing to allow the exchange of data needed for the basic system operation.

Acknowledgment

This research has been supported by the Ministry of Education, Science and Technological Development through the project no. 451-03-68/2020-14/200156: “Innovative scientific and artistic research from the FTS activity domain”.

References

[1] A. Tsohou, E. Magkos, H. Mouratidis, G. Chrysoloras, L. Piras, M. Pavlidis, J. Debussche, M. Rotoloni, B. Gallego-Nicasio Crespo, “Privacy, security, legal and technology acceptance elicited and consolidated requirements for a GDPR compliance platform,” *Information and Computer Security*, vol. 28, no. 4, pp. 531-553, Oct. 2020

[2] P. Porambage, M. Ylianttila, C. Schmitt, P. Kumar, A. Gurtov, A. V. Vasilakos, “The Quest for Privacy in the Internet of Things,” *IEEE Cloud Computing*, vol. 3, no. 2, pp. 36-45, Mar. 2016

[3] S. Wachter, “The GDPR and the Internet of Things: a three-step transparency model,” *Law, Innovation and Technology*, vol. 10, no. 2, pp. 266-294, Sept. 2018

[4] S. Watcher, “Normative challenges of identification in the Internet of Things: Privacy, profiling, discrimination, and the GDPR,” *Computer Law & Security Review*, vol. 34, no. 3, pp. 436-449, June 2018

[5] C. Perera, R. Ranjan, L. Wang, S. Khan and A. Zomaya, “Big Data Privacy in the Internet of Things Era,” *IT Professional*, vol. 17, no. 3, pp. 32-39, May 2015

[6] A. D. Kounoudes, G. M. Kapitsaki, “A mapping of IoT user-centric privacy preserving approaches to the GDPR,” *Internet of Things,* vol. 11, no. 100179, Sept. 2020

[7] H. Lee, A. Kobsa, “Understanding user privacy in Internet of Things environments,” *Proc. of World Forum on Internet of Things (WF-IoT),* Dec. 2016

[8] Moataz Soliman, Tobi Abiodun, Tarek Hamouda, Jiehan Zhou, ChungHorng Lung, “Smart Home: Integrating Internet of Things with Web Services and Cloud Computing,” *International Conference on Cloud Computing Technology and Science,* Dec. 2013

[9] S. Zheng, N. Apthorpe, M. Chetty, N. Feamster. “User Perceptions of Smart Home IoT Privacy”, *Proc. of the ACM on Human-Computer. Interactiion.* vol. 2, no. CSCW, pp. 1-20, Nov. 2018

[10] K. Renaud,L. A. Shepherd, “How to make privacy policies both GDPRcompliant and usable,” *International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA),* Nov. 2018

[11] M. Matić, M. Tucić, M. Antić, R. Pavlović, “Using online third party geolocation services to improve smart home user experience,” Serbian Journal of Electrical Engineering vol. 17, no. 1, pp. 83-94, Feb. 2020

[12] S. Ivanović, M. Antić, I. Papp, N. Jović, “Data Acquisition, Collection and Storage in Smart Home Solutions,” *Proc. of 6th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN),* May 2019

Combined adaptive load balancing algorithm for parallel applications

Luka Filipović, Božo Krstajić, *Member IEEE*, Tomo Popović, *Senior Member IEEE*

***Abstract*— Development and improvement of efficient techniques for parallel task scheduling on multiple cores processors is one of the key issues encountered in parallel and distributed computer systems. The purpose of process distribution improvement in parallel applications is in increased system performance, reduced application execution time, reduced losses and increased resource utilization.**

This paper presents combined adaptive load balancing algorithm based on domain decomposition and master-slave algorithms and its core scheduling adaptive mechanism that handles load redistribution according obtained and analyzed data. Selection of distribution algorithm, based on collected parameters and previously defined conditions, proved to deliver increased performances and reduced imbalance. Results of simulations confirm better performance of proposed algorithms compared to the standard algorithms reviewed in this paper.

***Index Terms*— parallel programming, load balancing algorithm, tash scheduling, adaptive algorithm.**

I. Introduction

Distributed computer systems enables the delivery of computing resources necessary to solve complex problems with requirements that exceed the capabilities of the most powerful personal computers. High-performance computers, as one of the powerful elements of distributed computer systems, lead to complex solutions by using computer simulations enabling progress in all scientific fields. Parallel processing supports execution of several processes and instructions simultaneously, with a goal to save time and execute faster and more efficient complex applications in scientific and industrial applications. [1] [2].

The focus of many researches in the parallel processing field is process of finding optimal distribution of tasks in order to increase efficiency, reduce execution time of parallel applications and reduce communication time of computer resources. In order to achieve the highest parallel application efficiency, it is crucial to optimize the assignment of tasks to parts of the distributed computer system (cluster nodes and its CPU cores) and monitor their execution.

The subject of this research was combined adaptive algorithm (CAA) [3][4], which uses combination the static and dynamic load balancing algorithms to improve the performance of independent parallel tasks scheduling without significantly complicating the whole process. It uses an adaptive innovative mechanism for choosing load balancing algorithm for distribution of unexecuted autonomous tasks depending on the segments in which losses are the least and by limiting the algorithm at times when it causes losses.

II. Load balancing algorithms

Load balancing in parallel processing is defined as process of achieving parallelism by redistributing the load of parallel segments during the execution of a parallel program [5] [6]. The primary goal of load balancing algorithms is to find the optimal execution schedule that defines the initial execution time and the execution order of all tasks that run on a particular resource. Load balancing of parallel applications is process of reducing computation time achieved by reducing communication time, synchronization time between processes and waiting time due to uneven process distribution [7].

The imbalance of parallel applications most often occurs due to uneven load between cores, excessive communication between cores or waiting of group of cores for others to finish assigned jobs [8]. In a real distributed environment, resource load varies over time and it is not always possible to improve the use of resources that are completely free or equally loaded. It is not possible to determine or predict the length of processes that run on separate computers or delays due to communication between computers. Therefore, there is a longer execution of the parallel application and a decrease in resource utilization. The end of the execution of a parallel application or the beginning of the postprocessing phase directly depends on the execution time of the part of the application on the core that is assigned the most process or the processor with the lowest frequency.

Load balancing algorithms are divided as static and dynamic, depending on the type of job scheduling. Static load balancing algorithms have good usability and efficiency on homogeneous clusters while they execute tasks on all cores which have similar duration. Performance of programs using these algorithms is reduced at the end of the runtime without possibility of rescheduling. One of widely used static algorithms is domain decomposition algorithm. On the other side, dynamic algorithms can give better efficiency on heterogeneous system, but make unnecessary communication during executing time. The master slave algorithm is a one of the typical representatives of dynamic algorithms. Domain decomposition and master-slave algorithms have their advantages and disadvantages depending on the characteristics of the resource, the specific parallel application for which load balancing is performed and the duration of processes that are executed in parallel [9-11].

Luka Filipović is with the University Donja Gorica, Oktoih 2, Podgorica, Montenegro (e-mail: luka.filipovic@udg.edu.me).

Božo Krstajić is with the Faculty of electrical engineering, University of Montenegro, Džordža Vašingtona bb, Podgorica, Montenegro (e-mail: bozok@ucg.ac.me).

Tomo Popović is with the Faculty of Information Systems and Technologies, University Donja Gorica, Oktoih 2, Podgorica, Montenegro (e-mail: tomo.popovic@udg.edu.me).

Adaptive algorithms are advanced dynamic algorithms with adaptive strategy for task distribution scheme that is activated depending on the load change of the distributed system during operation.

III. Combined Adaptive Load Balancing Algorithm

The combined adaptive algorithm (CAA) is successor an improved version of combined algorithm (CA) [12]. It presents an adaptive decision model that selects an adequate algorithm based on data on the state of the resource on which the parallel application is running and the duration of finished tasks.

In the preprocessing phase, as in the CA algorithm, the input data is divided and tasks are prepared for execution. Before starting parallel simulations, the analysis of the distributed resource configuration is performed and the obtained data are used in the later analysis.

In the parallel processing of the combined adaptive load balancing algorithm, three execution phases stand out (Figure 1):

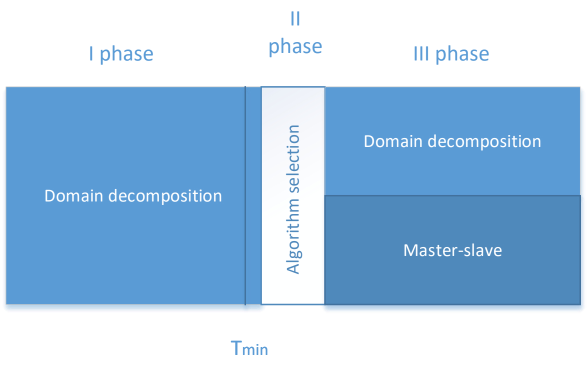


Fig 1. Execution phases of the proposed CAA algorithm

In the first phase of the combined adaptive algorithm, the domain decomposition algorithm is executed. It has the highest efficiency and the lowest losses in the initial phase of program execution. The algorithm stops working when the first (“fastest”) core completes the assigned job (T_{min}) and sends instructions to the other cores to stop working after completing the task they are processing at that point. The described procedure reduces the losses of the first execution phase to a minimum.

In the second phase of the algorithm, based on the amount and duration of performed tasks, cluster configuration and its load, an adaptive approach is used to select the algorithm for the scheduling of the remaining tasks in third phase. Upon initiating an interrupt at the end of the first phase, each CPU core sends to a predefined core a data containing the duration of the performed tasks. The predefined core receives the sent data and processes them, making an array with the number of executed tasks for each core and through executed and unexecuted tasks and selects the algorithm to be executed in the third phase according to the defined decision algorithm. The decision on the algorithm in third phase is made on the basis of the following parameters:

- the homogeneity of allocated resources,
- the total number of assigned cores,
- the numbers of completed tasks for each core individually and
- the execution time of each task individually.

The homogeneity of the allocated resources (the examination of whether they can be considered homogeneous or heterogeneous) is performed by comparing the performance values of the allocated nodes of the distributed resources. A measure of the performance of an individual resource can be core frequency, node memory or node network speed. Depending on the architecture of the distributed system and the type of tasks, one or more node performance measures can be taken. In the presented research, the core frequency (Hz) was used as a measure of the node performance of the distributed system.

The total number of assigned cores is defined when the application is started.

The number of completed tasks per core represents the part of the total number of tasks performed up to the moment T_{min}, when the first core performed the assigned tasks and initiated the interrupt, for each core separately. The data is expressed as a sequence whose number of elements is equal to the number of assigned cores, and the elements are the numbers of completed tasks for each core individually. The total number and type of tasks depends on the parallel application being executed and the input data, and the division is done before the parallel processing.

The execution time of each individual task is a matrix that contains the data on which core the task was executed and the duration of each task (ms) that was completed.

Based on the above parameters, the conditions for selection of an adequate distribution algorithm in the third phase can be defined. These conditions are defined by variables U_i that have binary values. Thus, the variable U_i takes the value 1 if the i-th condition is met, and otherwise U_i takes the value 0.

The first and eliminatory condition (U_e) for the selection of the distribution algorithm is the condition that the remaining number of tasks is less than or equal to the number of available cores. If the conditions U_e ($U_e = 1$) are met, the DD algorithm is selected for execution in the third phase, ie each of the remaining tasks is assigned one core for execution.

If the eliminator condition is not met ($U_e = 0$), the choice of algorithm is made based on a combination of the following conditions:

- U_1 - cluster homogeneity condition: this condition is fulfilled (U_1=1) if CPU cores of the same or approximate operating clock are assigned, ie. if the standard deviation of the operating clock of all cores is less than the set value;
- U_2 - number of cores condition: this condition is fulfilled ($U_2 = 1$) if the number of cores is less than a predefined number of cores, ie if the losses of the master core in the MS algorithm cannot be ignored;
- U_3 - condition of uniformity of the number of performed tasks: this condition is fulfilled ($U_3 = 1$) if the number of performed tasks for each core is approximate, ie. if the value of the standard deviation of the number of completed tasks per core is less than the predetermined value;
- U_4 - condition of uniformity of duration of performed tasks: this condition is fulfilled ($U_4 = 1$) if the duration of performed tasks per core is approximate, ie. the

value of the standard deviation of the execution time of each task per core is less than the predefined value.

The decision algorithm checks the fulfillment of conditions that depend on the values of the parameters. Choice of the algorithm itself adapts to the current performance of the allocated resources and the state of the performed tasks in the first phase. Thus, the proposed adaptive algorithm determines whether the domain decomposition or master-slave algorithm will be executed in the next phase based on the fulfillment of the defined conditions according to the principle: the more conditions are met, it determines the choice of DD algorithm in the third phase and vice versa.

In order to enable additional adaptation of the decision algorithm to a specific application and distributed system, each of the conditions can be weighted with real coefficients K_i, $K_i \in [0,1]$ which enables the exclusion of some conditions or assigning greater or lesser importance to some of the conditions. This does not apply to an eliminatory condition that is considered independently of the other conditions. The coefficients K_i are assigned a maximum value of 1 if this condition is fully taken into account, while $K_i = 0$ excludes the influence of this condition from the influence on the choice of algorithm. Coefficients should be defined separately for each application and distributed resource depending on previously obtained results and experiences.

Finally, based on the above conditions, we can define the decision function on the basis of which we select the algorithm in the third phase:

$$U = \sum_{i=1}^{4} Ki * Ui. \qquad (1)$$

The threshold value of the decision function U should also be defined, on the basis of which one or another algorithm is selected for the third phase (DD or MS). Since the maximum of the function U is achieved by the fulfillment of the conditions K_i*U_i and that determines the choice of the DD algorithm, then half of the maximum value of the function U is taken as the threshold value, ie

$$\mathrm{P} = \frac{\sum_{i=1}^{4} \mathrm{Ki}}{2}. \qquad (2)$$

Therefore, if it’s satisfied

$$U \geq P \qquad (3)$$

it is necessary to select the DD algorithm in the third phase or the MS algorithm if condition is not satisfied.

Figure 2. shows a schema of the decision making process for the selection of algorithm in second phase. As already mentioned, based on the presented parameters, defined conditions and coefficients, the algorithm for the distribution of tasks in the third phase is selected.

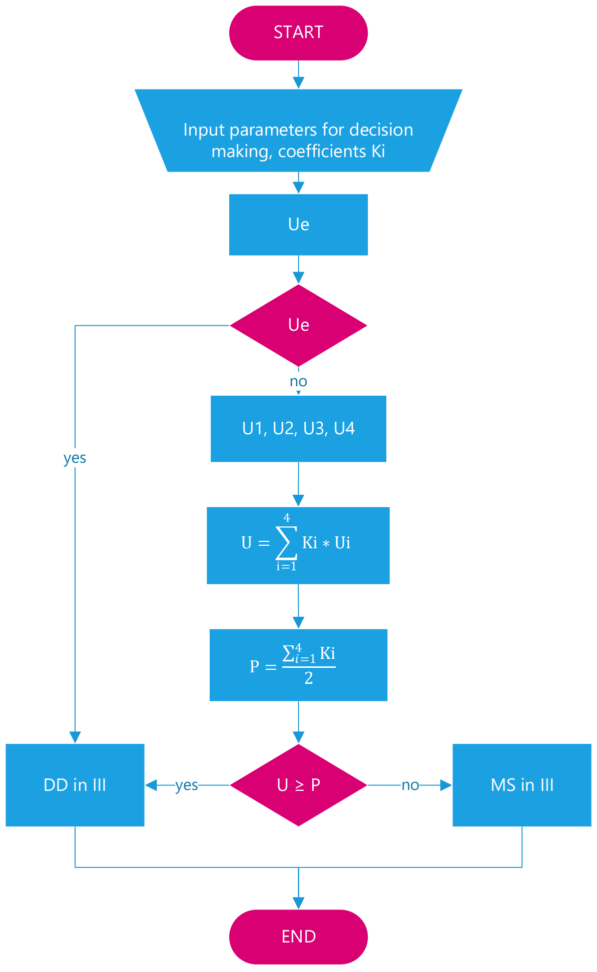

START
Input parameters for decision making, coefficients Ki
Ue
Ue
yes
no
U1, U2, U3, U4
$U = \sum_{i=1}^{4} Ki * Ui$
$P = \frac{\sum_{i=1}^{4} Ki}{2}$
DD in III
yes
U ≥ P
no
MS in III
END

Fig 2. Scheme of the decision making process for the selection of algorithm in Phase II

The selected algorithm (DD or MS) is executed in the third phase.

If the DD algorithm is selected, each core receives a portion of the list of unfinished tasks. Each core gets assigned one of the remaining tasks to solve if the remaining number of tasks is less than or equal to the available number of cores (condition Ue). Otherwise, the number of assigned tasks for each core is determined in proportion to the number of tasks completed in the first phase on each core separately.

In the case of selecting the MS algorithm, the core that performed the analysis in the second phase is determined as the master core. It contains information with a list of all unfinished tasks that are assigned to slave cores for execution in the third phase of the algorithm.

The proposed CAA algorithm will increase efficiency and shorten the execution time of parts of a parallel application in the third phase according to the interruption of the execution of the first phase, the analysis of the state of resources, the adaptation from the second phase and the redistribution of tasks.

The efficiency of the CAA algorithm has been improved due to process reallocation, reduced kernel latency for new instructions, and improved resource utilization by adapting the allocation to the distributed system architecture and application-specific. Therefore, the execution time of the proposed algorithm will be shorter than the execution time of the standard DD algorithm if measured under the same conditions. The CAA algorithm is similar to the CA algorithm in the case of deciding that a dynamic process allocation along with the MS algorithm is required in the third stage.

The disadvantages of the proposed CAA algorithm are the interruption of task execution at the end of the first phase and the duration of adaptation in the second phase. Interrupting the execution of tasks in the first phase may increase the duration of this phase if there are one or more tasks whose duration is significantly longer than the duration of other tasks. This phenomenon would cause an increase in the duration of the first phase, which may affect the performance of the entire algorithm. In that case, the efficiency would be the same as with the classical DD algorithm. The second phase, due to its short duration, cannot significantly affect the overall efficiency of the parallel application.

The proposed CAA works as a DD algorithm during the period of its maximum efficiency and stops working when its efficiency starts to decline. The proposed adaptive algorithm will have a significantly better performance than the domain decomposition algorithm in the case when the basic algorithm has low efficiency due to interruptions and redistribution of tasks.

The CAA algorithm will have better performance than the MS algorithm because the MS algorithm does not execute tasks on the master core and generates more communication losses than the proposed CAA algorithm. The MS algorithm will have lower efficiency than the proposed algorithm because it starts as a DD algorithm and redistributes and selects the algorithm for execution based on parameters in order to achieve better use of resources and efficiency.

In case of large losses during third phase, it is possible to re-initiate the interruption and repetition of the decision algorithm, ie adaptation based on new parameters, re-selection of the algorithm and its start to get the best use of resources.

IV. The analysis of simulation results

For the purposes of research and testing of the subject algorithms, a parallel version of the crossbar commutator performance simulator (CQ) [13] was used, as a numerically demanding example of a parallel application with several independent processes. The algorithms were tested on different distributed computing environments and run under different resource loads. Each simulation was performed ten or more times and the averaged results of the execution time are presented here. The performance of the combined adaptive algorithm was verified on the example of a 16-port CQ simulator with 1,000,000 requests and 3072 generated tasks. Simulations performed on the Paradox HPC cluster of the Institute of Physics in Belgrade. At the time of the simulation, the cluster consisted of 106 computing nodes based on two octa-core Xeon 2.6GHz processors with 32GB of RAM and NVIDIA® Tesla ™ M2090 cards. The performance of the combined adaptive algorithm is compared with the performance of the algorithms that make it up. Simulations were performed on 16, 32, 64 and 128 cores. The input files were copied to the nodes on which the simulations were run in the preprocessing phase, thus reducing the impact of communication between the nodes.

In the presented simulations, the value of standard deviation 10% of the average value of the core operating clock was used for condition U_1. A threshold of 32 cores is defined for condition U_2. For conditions U_3 and U_4, the value of the standard deviation is 25%. The coefficients used in these simulations are $K_1 = 0$, $K_2 = 1$, $K_3 = 1$ and $K_4 = 0.5$. Priority in decision making is given to the number of cores on which the simulation is performed and the number of performed tasks per core. A lower priority was given to the duration of the tasks, and due to the coefficient $K_1 = 0$, the influence of cluster homogeneity was not taken into account. The average results of parallel application execution with DD, MS and CAA algorithm for different number of used cores are shown in Figure 3.

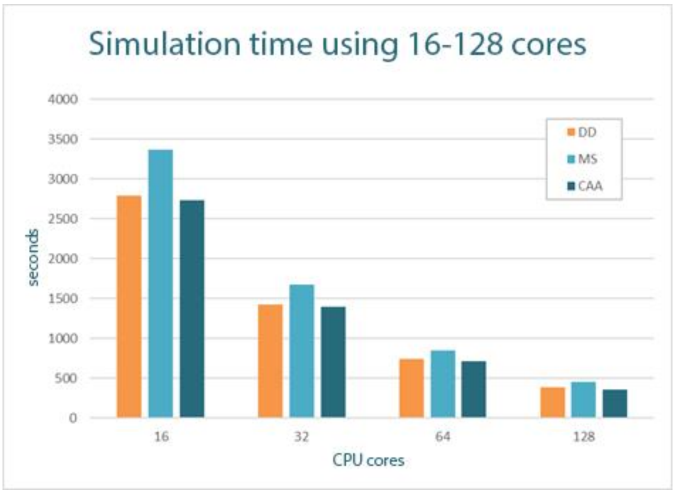


Figure 3. Average execution time of simulations using DD, MS and CAA algorithms on 16-128 cores

The combined adaptive algorithm completed simulations faster than the domain decomposition and master-slave algorithms in all conditions. The best results and the greatest benefits due to the redistribution of tasks were determined in cases of performing simulations on a number of cores. The simulations showed the longest execution time with the master-slave algorithm, especially on a small number of cores due to its previously described shortcomings.

The domain decomposition algorithm performed simulations faster than the master-slave algorithm. The input data was transferred before the simulations and most tasks were performed at approximately the same time, as shown in Figure 3. Therefore, the static distribution proved to be sufficient and the domain decomposition algorithm showed better performance than the master-slave algorithm.

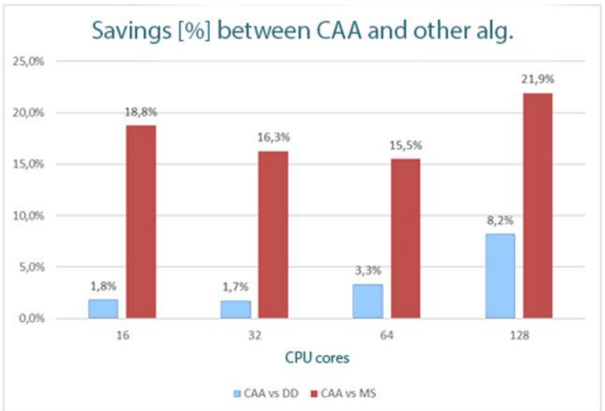


Figure 4. Savings during algorithm execution and comparison between combined algorithm and domain decomposition and master slave

Figure 4 shows the execution time savings between the combined adaptive algorithm and the algorithms that make it up. The domain decomposition algorithm required more time than the combined adaptive algorithm due to the static distribution throughout the execution process. The difference between the combined adaptive and domain decomposition algorithms ranges from 1.7% to 8.2%. The biggest difference was recorded when executing the application on 128 cores.

The differences between the combined adaptive algorithm and the master-slave algorithms are due to the loss of the master-slave algorithm due to the distribution of tasks and communication between cores during the entire program execution process. The execution time difference between the combined adaptive and master-slave algorithms ranges from 15.5% to 21.9%. The inability to execute tasks on the master core produced losses during execution on a smaller number of cores. Increased communication between cores throughout the execution of the simulation caused the largest difference between the results listed on 128 cores.

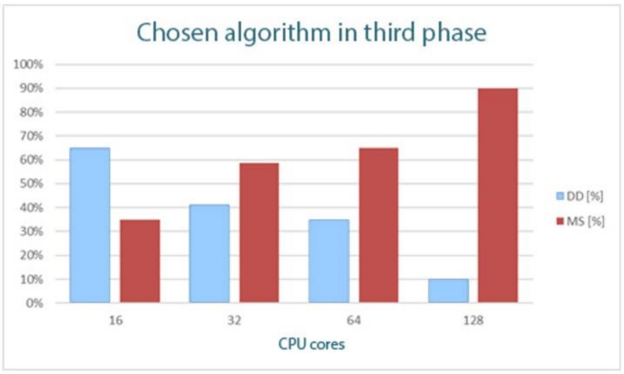


Figure 5. Selected algorithm in the third phase of CAA

Figure 5. shows the results of the selection of the algorithm in the second phase according to the received and analyzed data and the decisions made at the end of the second phase. The domain decomposition algorithm was chosen in most cases when the simulation was performed on 16 cores, because the execution was detected on less than 32 cores and an even number of tasks that needed to be redistributed. On the other hand, master-slave was chosen in cases of simulations on 32 or more cores because the decision algorithm from the second phase based on parameters discovered the number of available cores, different number and duration of performed tasks and selected this dynamic algorithm for the third phase.

V.Conclusion

The paper presents an original adaptive load balancing algorithm for parallel applications that combines the operation of static and dynamic algorithms. Domain decomposition and master slave algorithms were used on the basis for the proposed algorithm, as one of the most common algorithms in practice. As none of the algorithms provides good results in a wide range of applications and types of distributed systems, the following research was based on the idea of combining the mentioned algorithms in order to improve the parallelization performance without complication of the algorithm. Based on the identified advantages and disadvantages of standard algorithms, a combined adaptive algorithm is proposed. The idea of combined algorithms is to work in the phases when composite algorithms have the best performance. The advantages of the proposed solution are following:

- improved parallel application efficiency and cluster utilization in relation to basic algorithms due to task redistribution and reduced execution time;
- parameters and conditions for the selection of algorithms have been identified according to the status of resources and the point of execution of the application and determine a more adequate static or dynamic distribution of the process by an adaptive strategy
- weighting coefficients (K_i) adjust the adaptive load balancing algorithm and parallel application to the infrastructure
- applicability of the proposed adaptive part of the decision algorithm is possible in any load balancing algorithm and
- the proposed algorithm is applicable to all parallel applications consisting of several independent tasks.

The paper presents the results of executing domain decomposition, master-slave, combined and combined adaptive algorithm on different computer resources with the help of numerically demanding parallel application of CQ simulator. Comparison of the results of simulations with different loads and configurations of distributed resources confirms the better performance of the proposed algorithm in relation to the basic algorithms considered in the paper.

Acknowledgment

This research is supported in part by the EuroCC project, grant agreement grant agreement 951732 EuroCC-H2020-JTI-EuroHPC-2019-2.

References

[1] S. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigms, 2nd Edition, Pearson Education. Inc., 2007.
[2] B. Barney, Introduction to Parallel Computing, Lawrence Livermore National, 2012.
[3] L. Filipovic, “Combined adaptive load balancing algorithm for parallelization of applications”, PhD thesis, University of Montenegro, Faculty of Electrical engineering, 2019.
[4] L. Filipovic and B. Krstajic, "Combined load balancing algorithm in distributed computing environment," Information Technology and Control, vol. 45, no. 3, pp. 261-266, 2016.
[5] H. D. Karatza and R. C. Hilzer, "Parallel Job Scheduling in Homogeneous Distributed Systems," Simulation, vol. 79, 2003.
[6] J. Dinan, S. Olivier, G. Sabin, J. Prins, P. Sadayappan and C.-W. Tseng, "Dynamic Load Balancing of Unbalanced Computations Using Message Passing," in Parallel and Distributed Processing Symposium, 2007, IPDPS 2007, IEEE International, Long Beach, CA, USA, 2007.
[7] T. Rauber and G. Rünger, Parallel Programming for Multicore and Cluster Systems, Springer, 2010.
[8] D. Thiébaut, Parallel Programming in C for the Transputer, 1995.
[9] V. Sarkar. Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT Press, 1989.
[10] W. D. Gropp. Parallel Computing and Domain Decom-position. In: Fifth Conference on Domain Decompo-sition Methods for Partial Differential Equations, 1990, pp. 249-361.
[11] S. Sahni. Scheduling Master-Slave Multiprocessor Systems. IEEE Transactions on Computers, 1996, Vol. 45, No. 10, 1195-1199.
[12] L. Filipovic, B. Krstajic. Modified master-slave algorithm for load balancing in parallel applications. ETF Journal of Electrical Engineering, 2014, Vol. 20, No. 1, 74-83.
[13] M. Radonjic and I. Radusinovic, "CQ Switch Performance Analysis from the Point of Buffer Size and Scheduling Algorithms," in Proc. of 20th Telecommunication Forum TELFOR 2012, 2012

A Tool for Sentence Syntax Structure Markup for The Serbian Language

Teodora Đorđević, Suzana Stojković *University of Niš, Faculty of Electronic Engineering*

Abstract— **Syntax analysis is an important part of natural language processing. The biggest challenge to defining a natural language syntax analyzer is the inability to define unambiguous formal grammars that describe the language. Because of this, rule-based syntax analyzers need to be enhanced using statistics to allow us to predict which syntax tree is most likely. In order to do this, a corpus of tagged sentences in the target language is needed. The creation of this corpus is long and tedious work. Because of this, this paper implements a visual tool for creating such a corpus for the Serbian language. A component of this tool is the syntax analyzer, which generates all the possible syntax trees based on the defined grammar such that an expert may choose one of them. The expert may also create entirely new syntax trees.**

Index Terms—**Natural language processing (NLP); Syntax analysis; CYK; Annotated syntactic corpora; Serbian language**

I. Introduction

Natural language processing is a branch of computer science that teaches computers to understand and manipulate human language. Natural language processing is a combination of computer science, linguistics and machine learning. Many NLP techniques are already developed and applied for the English language but applying those techniques to different languages can be quite a challenge. Serbian language is under-researched in the context of natural language processing. Since the Serbian language and the English language do not belong to the same language group, many approaches designed for the English language need to be significantly modified in order to be used in the Serbian language or cannot be used at all.

Syntax analysis or parsing, in general, is the process of analyzing character strings according to the rules of a given formal grammar. It is typically encountered in fields of natural languages, computer languages or data structures. In Natural language processing, syntax analysis is one of the most important phases because it builds a great foundation to natural language understanding. Syntax analysis decides whether a sentence written in natural language conforms to the rules of a formal grammar and thus whether a sentence is valid or not. Designing a quality syntax parser is extremely significant for designing a semantical analyzer, since syntax parsing precedes semantic analysis. Also, syntax analysis has its own role in Rule-based Machine Translation, Information Extraction, Question Answering systems, etc.

Teodora Đorđević is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: teodora.djordjevic@elfak.ni.ac.rs).

Suzana Stojković is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: suzana.stojkovic@elfak.ni.ac.rs).

This paper describes designing a graphic tool for syntax analysis of the Serbian language based on the syntax analyzer designed in [1]. This tool is implemented as a web application that can analyze and visualize sentences using the implemented parser. It also enables the user to draw completely new syntax trees and save them.

II. Related Work

In linguistics, a corpus usually represents a collection of texts. The main purpose of a corpus is to be used as a tool in language study. In order to study and analyze language data, having a corpus is essential.

The English language is most widely researched language and thus the largest number of corpora exists for the English language.

Corpora contain texts that are sourced from natural contexts in order to be as close to the natural language as possible. There are many types of corpora that are used in natural language processing such as reference corpora, which are fairly balanced sets of texts that accurately describe a standard language, or specialized corpora which contain texts from a particular area, such as movie reviews, magazine texts, etc. A very important category of corpora are annotated corpora which contain additional information such as part of speech tags, lemmas, metadata, additional tags, etc. Annotated corpora can thus be used in supervised learning scenarios when attempting to infer this additional data based on the text given.

There are many publicly available corpora online for various languages. These corpora can be accessed either directly online via web browser, through specialized APIs to search the corpora, or can also be downloaded in their entirety.

Most modern language processing is done using computers. This means that modern corpora must be electronically readable documents. The first such document for the English language was The Brown Corpus of Standard American English [2]. This corpus consists of one million words of American English texts printed in 1961. In order to ensure high quality and to make the corpus useful for a wide range of applications, the corpus compiled texts from 15 different categories. Keeping in mind the huge increase in processing power, as well as that the Internet generates more linguistical data than ever before, this corpus is now considered small.

An example of a modern corpus of the English language that is quite big is the Corpus of Contemporary American English (COCA) [3]. COCA is probably the most widely used corpus of English, with over one billion words. Many corpora for the English language can be found at [4].

Besides the English language many other languages of the world are being researched in the field of syntax and semantic

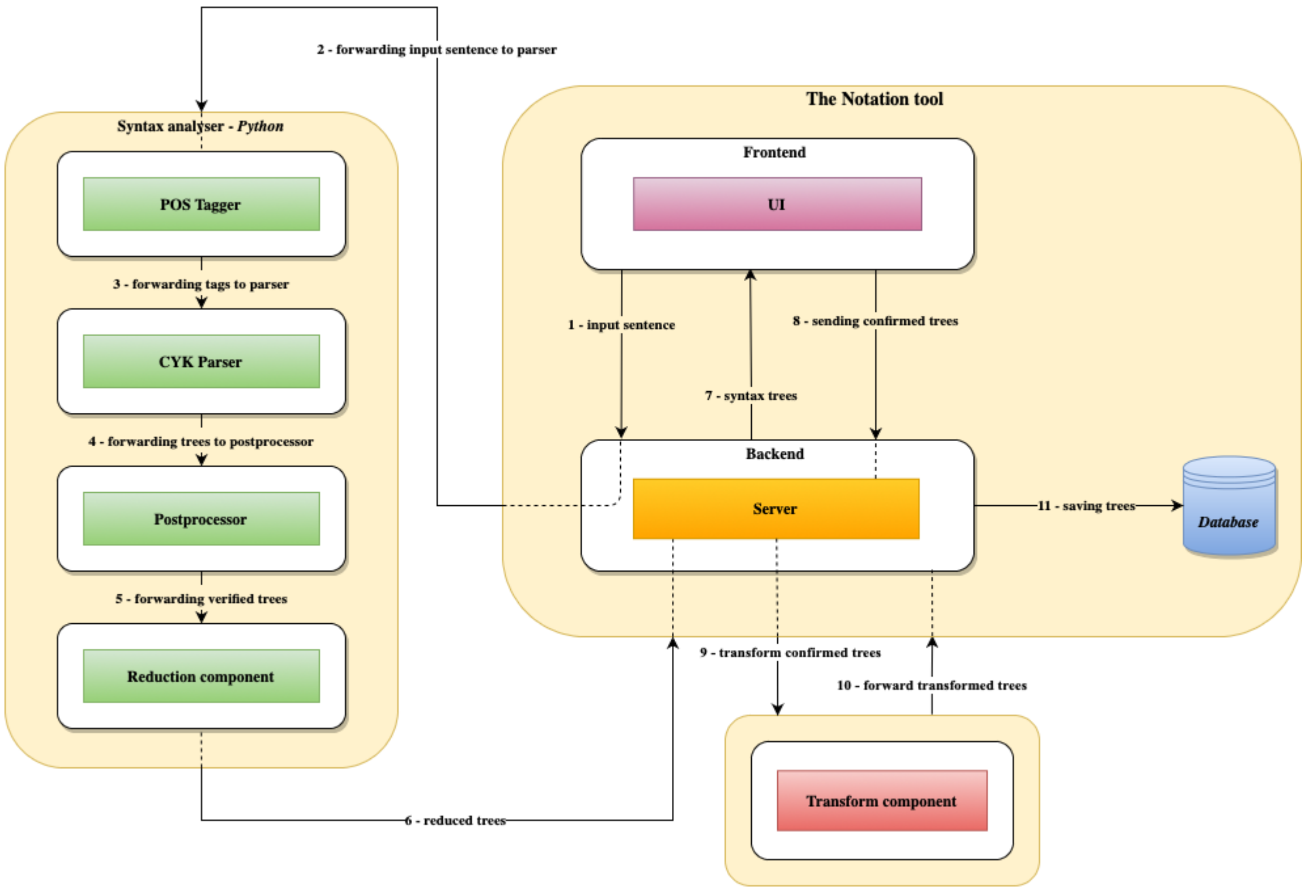

2 - forwarding input sentence to parser
Syntax analyser - *Python*
POS Tagger
3 - forwarding tags to parser
CYK Parser
4 - forwarding trees to postprocessor
Postprocessor
5 - forwarding verified trees
Reduction component
6 - reduced trees
The Notation tool
Frontend
UI
1 - input sentence
8 - sending confirmed trees
7 - syntax trees
Backend
Server
11 - saving trees
Database
9 - transform confirmed trees
10 - forward transformed trees
Transform component

Fig. 1. The Architecture of Notation tool

analysis and are thus being compiled into language specific corpora. One such corpus is the Quranic Arabic Corpus [5], which is an annotated linguistic document, that shows the Arabic grammar, syntax and morphology for each word in the Quran. The research paper [6] describes a specialized annotated corpus for the Chinese language, used for analyzing clinical texts. This corpus is annotated with part-of-speech tags, syntactic tags, entities, assertions, and relations.

For the Serbian language, given that it is spoken by only 12 million people in the world as a first language, there is not a large number of corpora such as English or Chinese. In the last few years, there has been development of open, freely available resources and technologies for computer processing of texts in the Serbian language. This includes annotated language corpora and some of the corpora are listed below.

1. *SETimes.SR* [7] – it is based on the *SETimes* parallel corpus of newspaper articles. This is a manually annotated corpus of texts written in the standard Serbian language. This corpus is used for training and evaluation of computer models on a number of natural language processing problems. It contains 3891 sentences. The *SETimes.SR* corpus is annotated using morphosyntactic notation, lemmas, syntactic dependencies, and named entities.
2. *srWac* [8] - the Serbian web corpus, which was built by crawling the *.rs* top-level domain in 2014. It contains 555 million tokens and over 25 million sentences arranged in about 1.3 million documents.
3. *MULTEXT-East* [9] - is a multilingual dataset for language research. This project consists of mainly Central and Eastern European languages, including Serbian.
4. *ReLDI-NormTagNER-sr 2.1* [10] - is a manually annotated corpus of Serbian tweets. It is meant for use in the fields of tokenization, sentence segmentation, word normalization, morphosyntactic tagging, lemmatization and named entity recognition of non-standard Serbian.

As mentioned earlier, there is a corpus that has marked dependency syntax at the sentence level, but there is still no corpus for the Serbian language that contains fully marked syntax trees. Precisely for this reason, the idea arose to create such a tool that will enable the creation of a corpus of syntactic trees for the Serbian language.

There are some visualization tools for drawing syntax trees, but they are mostly limited to inputting a syntax tree, and then getting a visualization of that tree. The tools which expect user to enter syntax trees and then visualize it are shown here [11, 12]. The tool where the user can draw syntax tree from scratch is TreeForm [13]. This tool offers wide palette of elements that can be drawn in order to create a syntax tree. There are several simpler solutions than TreeForm, such as [14][15]. All these tools are only intended for visualizing syntax trees. They do not support using a syntax parser in the background, which would generate syntax trees based on the entered sentence as suggested in this paper.

III. The Functionality Of The Notation Tool

A. *Syntax Analyzer*

The parser that was created in [1] achieved excellent performance and performed real-time parsing. This parser consists of three components:

1. **POS Tagger** – when a sentence is forwarded to the syntax analyzer it is necessary to extract POS tags first, because a syntax analyzer can only recognize tags, not actual words. This tagger is explained in detail in [1]. The tagger returns tags that have special meaning. For example, some of the valid tags are ‘nn’ (noun in nominative), ‘vm’ (main verb), ‘sl’ (preposition in front of locative). Every POS of the Serbian language has its own abbreviation, where every letter has its own meaning. These tags are then forwarded to syntax analyzer, and later displayed in syntax trees above actual words of the sentence.
2. **CYK Parser** – this parser is implemented to achieve optimal performances while analyzing sentences. Also, this parser, as defined in [1], is capable of recognizing all the syntax trees, like the parser in NLTK [16], but with significantly reduced parsing time.
3. **Postprocessor** – this layer is added because the number of syntax trees that are generated based on grammar designed for CYK Parser was large. To reduce this number, a series of rules is defined. These rules eliminate syntax trees that aren’t consistent with Serbian grammar. The postprocessing phase reduced the number of syntax trees by 54%.

The problem with this syntax analyzer, despite adding a postprocessing phase, is that it generated multiple trees for a single sentence. In order to solve this problem, it is necessary to add statistics that will enable to generate only one syntax tree as a result of a syntax analysis. To be able to implement statistical parsing, it is necessary to have a corpus of marked sentences, which is not the case for the Serbian language. For this reason, the idea of creating a visual tool arose. This tool will enable simple drawing and visualization of syntax trees and thus lead to the generation of a corpus of marked sentences that will be used further.

The notation tool works as follows:

1. The user enters the sentence they want to tag
2. The sentence is forwarded for processing to a parser that returns the resulting syntax trees
3. The syntax trees are displayed to the user
4. The user can choose one of the following options:
   - Select the correct tree,
   - Change the tree that is the most similar to the correct tree - by adding nodes, changing the node name, deleting nodes, and switching places with nodes, or
   - Create a new tree in case all the suggested trees are wrong
5. The correct tree is uploaded and stored in the database.

The architecture of the implemented system is shown in Figure 1.

The new component of the syntax analyzer is called reduction component. This component is added specifically for this tool.

The grammar created for the Serbian language contains a huge number of rules because the Serbian language is very complex. Considering that due to the implementation of the CYK algorithm, it was also necessary to transform the grammar so that it would be in Chomsky's normal form, a large number of auxiliary shifts were introduced. The syntax tree created in this way was too large to be displayed to the user of any system and this is the reason for introducing a reduction component.

This component aims to transform the syntax tree so that it no longer contains auxiliary rules, as well as that it does not contain shifts that have been introduced to make syntax analysis simpler and more robust.

The goal of reduction is to transform the syntax tree, generated by using a more complex grammar, into a simpler tree, corresponding to a simpler grammar. The main purpose for introducing the reduction component is to visualize the trees in a way that domain experts would expect by abstracting away implementation details. Also, reduced syntax trees are smaller and easier to display. After confirming the final tree for input sentence, it is necessary to return syntax tree to original form. This is achieved by using transform component. This component accepts syntax tree in simpler grammar and transforms that tree to original grammar. The transformed tree is then forwarded to backend application and saved in a database.

Figures 2 and 3 show how a part of the syntax tree looks with and without reduction. The reduced syntax tree is significantly smaller and thus much easier to display. An entire syntax tree without reduction would be impossible to fit in the page of the notation tool. The syntagm shown in figures 2 and 3 is “Moja divna drugarica”, meaning “my wonderful friend” in Serbian.

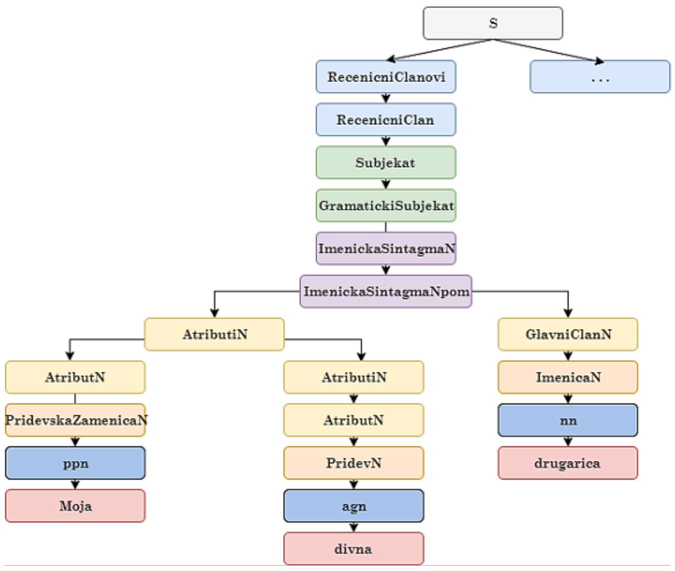


Fig. 2. Part of the syntax tree without reduction

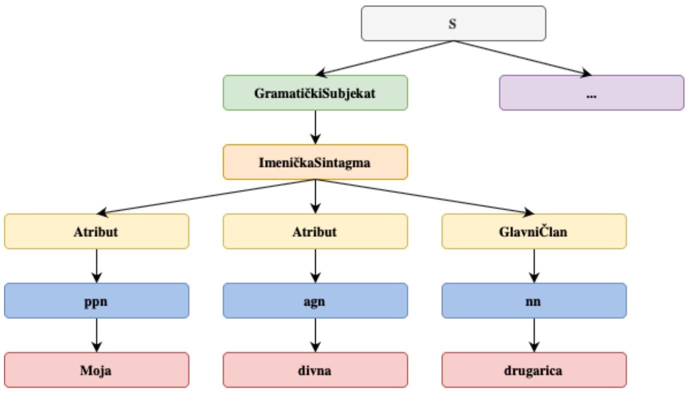


Fig. 3. Part of the syntax tree with reduction

B. *The Notation Tool*

This component is implemented as a web application so that users can use it as easily as possible. This approach was chosen to avoid any installation. The application itself is divided into three parts:

1. Frontend
2. Backend
3. Database.

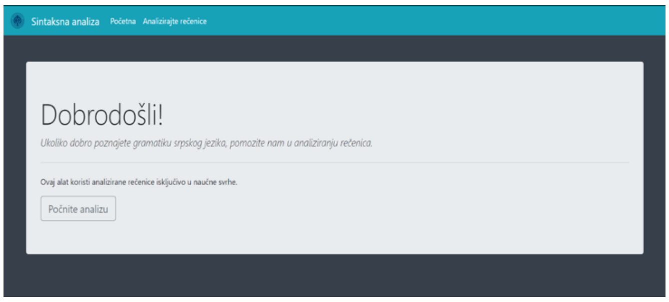
The role of the Frontend is to enable:

1. **Entering a sentence whose analysis should be performed**
2. **Displaying of all syntax trees generated by the parser**
3. **Selecting a syntax tree that is correct** - the user can view a list of all syntax trees that the parser returned and check the one that is correct. This sends a request to the server with the intention to save that tree in the database.
4. **Syntax tree modification** - if the syntax tree is not completely correct, but with a few minor changes it could become correct, the tool offers the possibility to make the following syntax tree changes:
   - *adding a new node* - it is necessary to select the node to which we want to add a new descendant and select the name that will be in that node. After interacting with the component, the tree structure is automatically updated to display the changes.
   - *deleting a node* - if it is necessary to remove a node, the tool offers the option to select that node and then delete it.
   - *renaming a node* - if the tree structure is adequate and an element is incorrectly recognized and it has the wrong name, the tool allows the user to rename that node.
   - *swapping nodes* - this option exists in case the nodes are correctly recognized but they have been misplaced. It is possible to swap the places of these nodes, but only if they have the same parent. This option was introduced because a new node is always added to the end of the list of children, and if the node is deleted, a new node should be added in its place. Since the new node is always added as the last child, this functionality allows the user to place the node in any arbitrary position.
5. **Drawing a completely new tree** - the tool offers space for drawing a new tree, where on one side of the control there is a list of possible nodes and arrows for connecting nodes, and on the other side there is a space for drawing - canvas. It is possible to transfer nodes from the palette to the drawing space, as well as to connect these nodes with arrows. When nodes are added and names are populated, the tool offers the ability to make a tree structure out of these nodes, as well as to send that tree further to the server to be stored in the database.
6. **Sending a tree to the database** – within the tool there is a service whose methods are called to interact with the server.

The role of the backend application is to enable:

1. **Route for frontend application where a sentence can be analyzed** – when a frontend application sends GET request the backend application forwards this sentence to the syntax analyzer described earlier. This syntax analyzer is written in Python, so it is necessary to call Python script which returns generated syntax trees for given input.
2. **Route for saving the chosen tree in the database** – when an expert reviewed the syntax trees and chose or drew the correct one. This syntax tree is saved along with tags and sentence that has been analyzed.

IV. The Examples Of The Notation Tool



Dobrodošli!

Ukoliko dobro poznajete gramatiku srpskog jezika, pomozite nam u analiziranju rečenica.

Ovaj alat koristi analizirane rečenice isključivo u naučne svrhe.

Fig. 4. Welcome page

Figure 4 shows the welcome page. There is a start analysis button that a user can click, and this will open a form for entering the sentence.

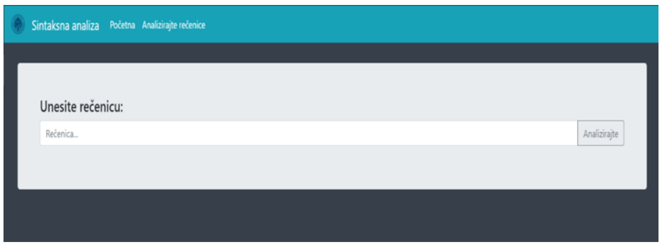


Fig. 5. Enter sentence form

Figure 5 shows a form where the user can enter a sentence for syntax analysis. That sentence is forwarded to the backend application. The backend application sends the sentence to the syntax analyzer by calling Python scripts.

The drawing of syntax trees was implemented using the canvas element in HTML and Canvas API in JavaScript. Below are shown pictures of different options which this tool offers.

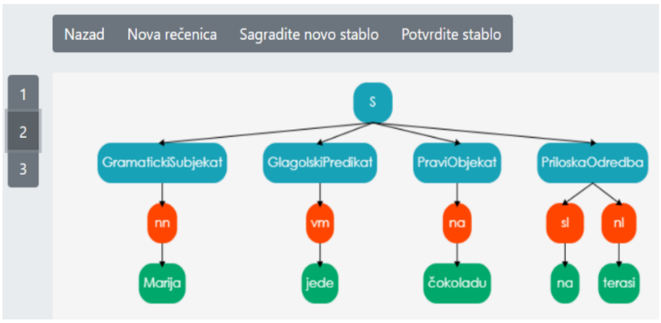
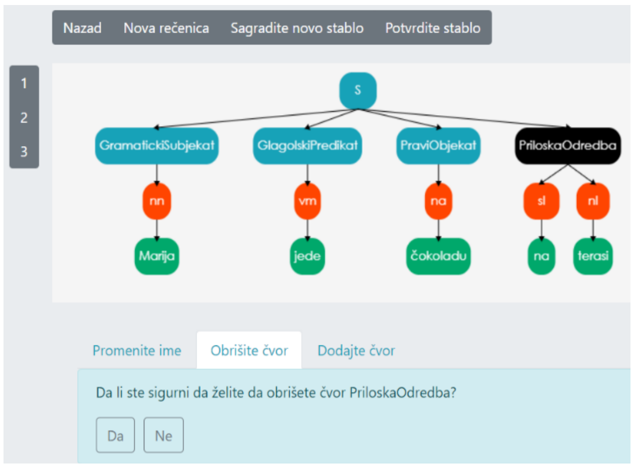


Fig. 6. One of the syntax trees that parser generated

Figure 6 shows the result that parser returned. As can be seen there are three syntax trees generated for this sentence. The second syntax tree is shown in figure 4. The start symbol of the grammar is S, the level below S represents syntactic structures. After that, there are POS tags that tagger returned and finally words of the sentence. The menu above the drawn syntax tree has three options:

1. Interrupting the current analysis and analyzing a new sentence (leads to the form where the sentence is entered),
2. A page where it is possible to build a completely new tree for the current sentence,
3. Confirmation of the current tree - forwarding the selected tree by sending a POST request to the server, where the syntax tree is sent as the request body, after which the syntax tree for the entered sentence is stored in the database.



Da li ste sigurni da želite da obrišete čvor PriloskaOdredba?

Fig. 7. Node removal

Figure 7 displays the menu for deletion of a node. First, it is necessary to select the node that is going to be altered. The selected node is colored black to stand out from other nodes. After node selection, the application opens the menu where the user can add a new node to the selected one, change the selected node's name or delete the selected node. Figure 7 shows that option for deletion is chosen. If the user wants to rename the node, it is necessary to select the rename option from the displayed menu. After that, the user needs to enter a new node name and confirm it. The third option in the menu is to add a new node. When a node to which a new node is added is selected, it is necessary to enter a name for the new node to be added.

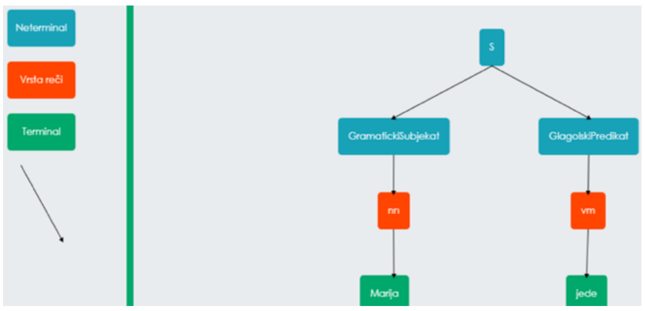


Fig. 8. Drawing syntax tree using canvas

Figure 8 shows the canvas where the user can draw a syntax tree from scratch.

V. Conclusion

The notation tool has been carefully created so that it has the simplest interface with the intention of being used primarily by domain experts - philologists. By using this tool, users are able to tag sentences in the simplest possible way, and thus quickly and efficiently create a corpus for the Serbian language.

After launching this site on the web, it is necessary to hire a set of domain experts who will tag sentences using the tool and create a corpus of sentences. After collecting a sufficient number of sentences, it is expected that these sentences will be used to further improve the Serbian language parser.

Acknowledgment

This work has been supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

References

[1] T. Đorđević, S. Stojković: "Different Approaches in Serbian Language Parsing using Context-free Grammars", Proceedings of 7th International Conference on Electrical, Electronic and Computing Engineering IcETRAN, Etno-Selo Stanišići, Bosnia and Herzegovina (Online conference), pp. 588-591, September 28-30. 2020.

[2] N. W. Francis, H. Kucera, "Brown Corpus Manual" , Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979.

[3] M. Davies, *The Corpus of Contemporary American English*, 2008, www.english-corpora.org/coca/, 13.06.2021.

[4] English Corpora, https://www.english-corpora.org, 12.06.2021.

[5] The Quranic Arabic Corpus, https://corpus.quran.com, 10.06.2021.

[6] B. He, B. Dong, Y. Guan, J. Yang, Z. Yang, Q. Yang, J. Cheng, C. Qu, "Building a comprehensive syntactic and semantic corpus of Chinese clinical texts", *Journal of Biomedical Informatics*, vol. 69, pp. 203-217, 2017.

[7] V. Batanović, N. Ljubešić, T. Samardžić, "SETimes.SR – A Reference Training Corpus of Serbian", Conference on Language Technologies & Digital Humanities, Ljubljana, Slovenia, pp. 11-17, 2018.

[8] N. Ljubešić, F. Klubička, "{bs,hr,sr}WaC - Web Corpora of Bosnian, Croatian and Serbian", *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, Gothenburg, Sveden, pp. 29-35, April, 26. 2014.

[9] MULTEXT-East, https://www.clarin.si/repository/xmlui/handle/11356/1041, 10.06.2021.

[10] ReLDI-NormTagNER-sr 2.1, https://www.clarin.si/repository/xmlui/handle/11356/1240, 05.06.2021.

[11] Syntax tree generator, http://mshang.ca/syntree/, 26.07.2021.

[12] phpSyntaxTree, http://www.tycho.iel.unicamp.br/phpsyntaxtree/?, 26.07.2021.

[13] TreeForm, https://sourceforge.net/projects/treeform/, 26.07.2021.

[14] Trees, https://www.ling.upenn.edu/~kroch/gifdir/Trees3-animation.GIF, 27.07.2021.

[15] Linguistic Tree Constructor, https://ltc.sourceforge.io/screenshots.html, 27.07.2021.

[16] S. Bird, E. Klein, E. Loper, *Natural Language Processing with Python*, Sebastopol, USA, O'Reilly Media, 2009

Modeling the ATP tour matches: A social networks analysis approach

Balša Knežević, Miloš Obradović, Predrag Obradović, and Marko Mišić, *Member, IEEE*

***Abstract*—Professional men’s tennis is a demanding sport which greatly benefits from various approaches to performance analysis. More specifically, a complex network theory can be used to model and explain the dynamics of players and tournaments, based on the recorded matches. In this paper, played matches are used to model a social interaction between players. Several undirected weighted networks are constructed to model the ATP tour matches from 2018 to 2020. Moreover, the three most dominant players on the tour (the “Big Three”) were observed and analyzed using ego networks approach. The chosen time frame further allowed for the exploration of impact of COVID-19 on the dynamics of the ATP tour. Different network properties were explored, such as small world phenomenon, core-periphery model applicability, community structure, and the rich club phenomenon. Our results based on network theory approach showed that analyzed networks expose similar topological properties, despite the lower numbers of tournaments held in the year 2020.**

***Index Terms*—collaboration network analysis; community detection; ego networks; men’s tennis; network modelling.**

I. Introduction

Computational analysis of the results of sports competitions, as well as the performance of teams and individual athletes, has long been present in various sports. The development of data science and artificial intelligence, as well as the possibility of processing large amounts of data, have enabled new approaches to analyze the performance of both teams and individual players. In addition to traditional statistical methods, new methods have been developed, such as collaboration analysis and various prediction techniques.

Several methods based on network science were successfully applied to the analysis of team performance in collective sports, such as football [1][2], basketball [3], and water polo [4]. Furthermore, applications in individual sports are known, such as men's [5][6][7] and women's tennis [8], boxing [9], chess [10], cricket [11], etc. The goal of this paper is to further explore the usage of complex network analysis methodology in the field of men’s tennis.

Balša Knežević is with the University of Belgrade - School of Electrical Engineering, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: balsa.knezevic@etf.bg.ac.rs).

Miloš Obradović is with the University of Belgrade - School of Electrical Engineering, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: miobra@etf.bg.ac.rs).

Predrag Obradović is with the University of Belgrade - School of Electrical Engineering, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: predrag.obradovic@etf.bg.ac.rs).

Marko Mišić is with the University of Belgrade - School of Electrical Engineering, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: marko.misic@etf.bg.ac.rs).

The world of tennis tournaments is a complex system that consists mainly of players, the tournaments they play and the matches they have played in those tournaments. Inherently, such a system is very convenient to model with an appropriate collaboration network. Most often, such a system is modeled by players representing the nodes of the network, while the matches that the players play are in some way depicted by the edges in the network.

In this paper, the state of men’s professional tennis in the three years from 2018 to 2020 is modeled and analyzed. Similar analyses have already been done in the past for men’s tennis in singles [5][7] and doubles [6]. In the meantime, great changes have taken place in the world of tennis. That primarily refers to more than a decade of domination of tennis players from the so-called "Big Three" to which Roger Federer, Rafael Nadal, and Novak Djokovic belong. Moreover, the COVID-19 virus pandemic affected the holding of tournaments in 2020, while tennis tournaments in 2018 and 2019 took place regularly. This allowed for comparative analysis and additional remarks on the impact of the COVID-19 virus pandemic. Therefore, various research methods have been applied in the paper, such as quantitative and qualitative analysis of the collaboration network, community detection, analysis of ego networks of members of the "Big Three", data visualization, etc.

The paper is divided into several sections. The second section describes the studied data sets and provides an overview of the used methodology. The third section presents the results of the research which are then discussed. Appropriate quantitative and qualitative analyses of the data, as well as the produced visualizations, are given. The last section provides guidelines for future work and a brief conclusion.

II. Data Sets and Methodology of Analysis

This section presents the primary dataset and transformations performed on it in order to construct the dataset used for analysis. Furthermore, this section contains the methodology of analysis.

A. Data sets

This paper analyzes the results of men’s singles matches played on the ATP tour in the period from 2018 to 2020. Although data from earlier years are available, this timeframe was chosen with the intent to include years 2018 and 2019 which are two consecutive years with regularly held tournament seasons, and the year 2020 which was influenced by the epidemic of COVID-19 disease. Thus, it is possible to determine the influence of pandemics, as the dataset includes both the influenced data points and the two years of regular tennis seasons used as reference points.

The match data was taken on 12/22/2020 from a repository maintained by Jeff Sackman [12] and forms the primary data set for analysis. At the time of analysis, the tennis season for 2020 was completed. The primary data set consists of files containing data on matches in singles competition in the specified period, a list of all players ever ranked on the ATP list, data on the ranking of active tennis players on the ATP list in the period from 2010 to 2020. Match data contains information about the tournament, players, match results with statistics, and the performance of both players during the match. According to the author, the primary data set is largely refined and complete, but there may be certain inconsistencies or incompleteness where the data was not available.

The primary data set contains data on 7117 matches in the specified period. In addition, the data set contains data on 54,975 players who at some point during the observed period had at least 1 point on the ATP list. If several players have the same number of points, then they can share the same ranking on the ATP list, depending on other parameters.

The secondary data set was formed based on the primary dataset, as a refined and cleaned version of it. The data cleaning was performed according to the needs and goals of the research. During the process of cleaning and refinement, some data not necessary for the research itself was intentionally omitted, such as data on players who did not play any matches in the observed period, certain contradictory data, as well as redundant information (columns) that were not used in the analysis. The final secondary data set included data on 7117 matches, as well as data on 581 players.

B. Methodology of analysis

Firstly, a thorough statistical analysis of the dataset was conducted. Analyzed properties include the average number of tennis matches in certain years, the average number of tournaments in which tennis players participate, and the ranking of tennis players depending on the number of matches or tournaments played. Most interesting results are presented in the following section. Following the statistical analysis, the refined data set of tennis players and their mutual matches were used to create multiple collaboration networks. These networks were then further studied using methods of complex network theory.

Tennis tournaments are grouped in a season that lasts for a whole year. Therefore, three independent networks were constructed, each holding data for a specific year (N-18, N-19, N-20). Additionally, to allow the analysis of the whole data set, the three networks were aggregated into N-T. The four networks together are referred to as N-series networks.

As per common practice in the field of social network analysis, the network is represented through a set of nodes that describe the actors within the social network and the edges that represent social relations. In the case of networks used in this paper, the nodes of the network are tennis players who played at least one official ATP match in the analyzed period. The two tennis players are connected if they have played at least one official ATP match. The weight of the edge represents the number of matches that the tennis players played with each other. The networks are undirected.

In addition to networks representing all players and matches, the ego networks of the members of the "Big Three" were constructed for each year. These consist of prominent ego nodes, their direct connections with the neighbors, as well as the mutual connections of the neighbors. Furthermore, these three ego networks were unified, and then aggregated They were used to analyze the core-periphery property and the topology of the core of the N-series networks.

Community detection was performed by the Louvain method over the entire network, as well as over the aggregated ego network. For this purpose, a set of filtered and reduced networks was constructed. Clustering strength was evaluated, and the rich club phenomenon was examined.

Python programming language was used to collect and refine data, model the network, and calculate specific metrics using the NetworkX package [13] for network analysis. Gephi [14] was used to visualize and determine network metrics.

III. Results

This section presents the results of the research. The first subsection explores the basic properties of N-series networks, while the rest explores the derived ego networks and community detection.

A. Basic properties of N-series networks

A statistical analysis of networks N-18, N-19, N-20, and N-T was conducted. Basic quantifiable features of those networks are presented in Table 1. As expected, the number of tournaments and matches held in 2020 is significantly lower due to the pandemic. This is further reflected in the weighted and unweighted degrees of nodes. However, looking only at the statistical data does not give the whole picture, as it would lead one to believe that year 2020 was significantly different from the previous two years. Only after applying the complex network theory methods discussed below one can give a proper conclusion about the impact of COVID-19 on the dynamics of the observed data sets.

TABLE I
Metrics of constructed N-series networks

	N-18	**N-19**	**N-20**	**N-T**
Players (nodes)	419	364	345	581
Edges	2489	2378	1325	5330
Matches	2974	2696	1447	7117
Tournaments total	138	123	67	328
Tournaments hard surface	81	80	46	207
Tournaments clay surface	47	34	20	101
Tournaments grass surface	10	9	1	20
Avg. weighted degree	13.79	15.28	8.39	24.50
Avg. unweighted degree	11.88	13.07	7.68	18.35
Network density	0.03	0.04	0.02	0.03
Avg. shortest path length	3.13	3.04	3.18	3.23
Diameter	11	9	9	10
Avg. clustering coefficient	0.17	0.19	0.14	0.26

Networks N-18, N-19, and N-20 have an exceptionally low density and a relatively low average shortest path length. Given the low average local clustering coefficient, the networks do not express the small-world property. This is in contrast with previous works in the field [15], but the discrepancies come from a completely different network model. These observations also stand for the aggregated network N-T, as the aggregation does not significantly increase the density nor strengthen the clustering.

Another interesting observation can be made about the average weighted and unweighted degrees. As shown in Table 1, the relative difference between weighted and unweighted degrees is small for networks N-18, N-19, and N-20. This shows that an average pair of tennis players rarely meet more than one time per season. Similarly, in the aggregate network N-T, the annual expected number of matches played by a pair of players is lower than 2. Given the bracket organization of tennis tournaments and loser-go-home policy, only the best players are expected to play multiple matches in a tournament. This leads to the probability of two players meeting in a tournament being quite low, even if they both play in the tournament. In addition, a low annual number of tournaments leads to a low number of annual matches and further decreases the possibility of two players meeting.

A further discussion on this topic can be made when tournament seeding is taken into consideration. The probability of the first and second seed in a tournament gets further artificially lowered, as they are seeded in opposite sides of the bracket and are unable to meet before the finals. If a pair of players is consistently seeded with the top two seeds, this can lead to a measurable decrease in the weight of the edge connecting them.

B. *Analysis of ego networks*

Looking only at the average number of matches played does not show the whole picture and unravel the true topology of the constructed networks. Therefore, a distribution of the number of matches played during the observed period has been calculated and is shown in Fig. 1. As can be seen, many players have only played one or two matches and are thus very isolated, suggesting a core-periphery topology.

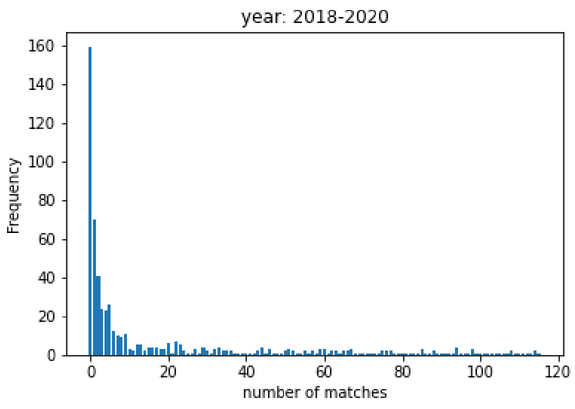


Fig. 1. Distribution of the number of matches played during the three years from 2018 to 2020. The distribution largely resembles a Pareto distribution.

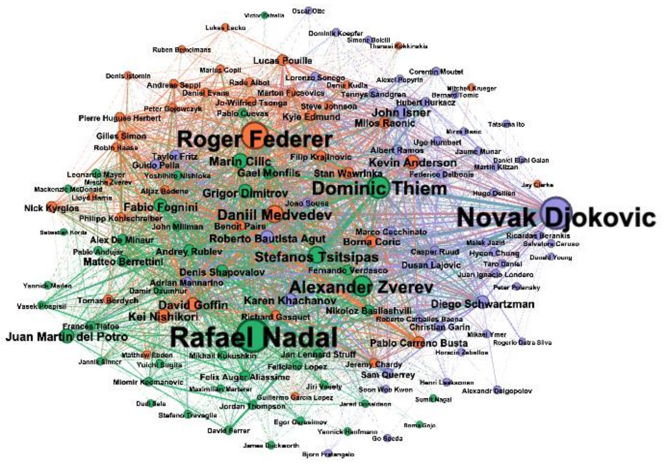


Fig. 2. EGO-T, a unified ego network of the "Big Three" for the period from 2018 to 2020. The size of the node represents weighted node degree and nodes are colored based on clustering.

As stated in the section about methodology, to check if N-18, N-19, N-20, and N-T networks follow the core-periphery model and unravel the topology of the cores of specified networks, several ego networks centered around the members of "Big Three" were constructed. Annual ego networks of Djokovic, Nadal, and Federer were then unified into EGO-18, EGO-19, and EGO-20. Together with these ego networks, their aggregated network EGO-T, shown in Fig. 2, was built.

Clustering the EGO-T network using the Louvain method [16] and tuning the resolution to give 3 clusters reveals a very interesting phenomenon. The original ego nodes bind stronger to some of the other nodes in the network than between themselves. This is in concert with the aforementioned observation about the bracket system and seeding principles influencing the edge weights on the very top of the ATP list.

Exploring the number of nodes and edges of N-series networks included in EGO-series networks can help us explore the properties of the core of N series networks. These statistics are therefore shown in Table 2.

TABLE II
METRICS OF EGO-SERIES NETWORKS

	EGO-18	EGO-19	EGO-20	EGO-T
Nodes	81	88	57	136
Nodes covered	19.33%	24.17%	16.50%	23.4%
Edges	691	744	202	2563
Edges covered	27.76%	31.28%	15.24%	48.08%

Given the percentage of all players and matches included in EGO-series networks, it is obvious that even EGO-T which aggregates other EGO networks and enhances the core property can not be considered a core by itself. Further exploring this topic, the Rombach core finding algorithm [17] was applied to find cores of N-18, N-19, N-20, and N-T, giving cores with 234, 200, 193, and 315 players, respectively. These cores are much larger than EGO series networks and include most of the players.

However, a remark has to be made about the EGO-T network and the percentage of matches included in it. Even though EGO-T is more than two times smaller than the core of N-T, it includes 48.08% of all matches recorded in N-T, which is an astounding amount. This means that matches between the players from EGO-T represent nearly half of all the ATP matches played from 2018 and 2020 and could be used to study some phenomena on a smaller, but representative, group of players, without drastically compromising the number of matches included in the data.

C. *Community detection and the rich club phenomenon*

To discover a more fine-grained structure in the constructed networks, in addition to exploring network cores, the Louvain method was used once again to find communities in N-18, N-19, N-20, and N-T. Before running the Louvain method, all nodes with degrees lower than 3 were removed from N-18, N-19, and N-20 to avoid the formation of forced and unnatural clusters due to modularity optimization. Characteristics of these reduced networks, aptly named R-18, R-19, and R-20 (R standing for "reduced"), are shown in Table 3. Moreover, a similar procedure was applied to N-T, removing all players with less than 5 matches during the three years, giving us R-T, a reduced network of total aggregated data.

TABLE III
METRICS OF R SERIES (REDUCED) NETWORKS

	R-18	R-19	R-20	R-T
Nodes	244	203	181	287
Nodes retained	58.23%	55.77%	52.46%	49.40%
Edges	2292	2190	1159	4889
Edges retained	92.09%	92.09%	87.47%	91.73%
Communities	9	6	8	7
Avg. clustering coeff.	0.22	0.24	0.18	0.32

The process of node removal is validated by looking at the percentage of nodes and edges retained in the reduced networks. As we can see in Table 3, 49.40% of players played 91.73% of matches during the observed three-year period. This phenomenon can also be seen in Figure 1. As the distribution of the number of matches loosely follows a Pareto distribution, it is to be expected that a rich-club phenomenon can express itself when considering the number of matches as "wealth". This is somewhat validated by looking at EGO-T, as it consists of a small group of players which bind strongly to each other and monopolize the number of matches over the observed period.

Communities formed by the Louvain modularity clustering are grouped by average rating during the period. This is to be expected, as players of similar ratings choose to play and qualify for the same class of tournaments and are more likely to meet each other. However, the clustering is still not strongly expressed, as can be seen from the average local clustering coefficients.

IV. CONCLUSION

Studying interactions of men's tennis players proved to be interesting in several aspects. Motivated by the available data, several undirected weighted networks with node metadata were constructed, analyzed, and characterized and multiple common phenomena in the field of complex network theory were explored. Those include small world phenomenon, core-periphery model applicability, community detection, and the rich club phenomenon. In addition, the authors' own experience with the topic helped explain many of the observed properties and the given explanations are one of the biggest results of this paper, as they give a much better understanding of the dynamics of men's tennis and are a result of social network analysis and network theory approach to the problem.

In addition, provided network models clearly show an impact of the COVID-19 pandemic on the tennis world, through a smaller number of matches and participants. However, the network theory methodology applied in this paper also shows that the topological properties of the data (such as clustering properties, rich club and small-world phenomena, core-periphery property) stay largely the same, which could not be inhered by naive statistical analysis of the primary data set.

This paper and the constructed networks form a strong basis for further exploration of the topic, including the analysis of mixing patterns in the data depending on the ratings of players, geographical locations of tournaments, affiliations of players, etc. Furthermore, the data in network form is much more suitable for solving some regularly asked questions in the field, such as ranking and match outcome prediction using graph convolutional networks or graph attention models. Lastly, the provided networks are an ideal model for the problem of choosing the representatives of the international tennis community, touching upon the problem of choosing the dominating set of the graph.

ACKNOWLEDGMENT

This work has been partially funded by the Ministry of Education, Science, and Technological Development of the Republic of Serbia. Grant numbers III44009 and TR32047.

REFERENCES

[1] T. Grund, "Network structure and team performance: The case of English Premier League soccer teams", *Social Networks*, 34(4), pp.682-690, 2012.
[2] J. Gama, M. Couceiro, G. Dias, V. Vaz, "Small-world networks in professional football: conceptual model and data", *European Journal of Human Movement*, 35, 85-113, 2015.
[3] F. M. Clemente, F. M. L. Martins, D. Kalamaras, R. S. Mendes. "Network analysis in basketball: Inspecting the prominent players using centrality metrics", *Journal of Physical Education and Sport*, 15(2), 212, 2015.
[4] P. Passos, K. Davids, D. Araújo, N. Paz, J. Minguéns, J. Mendes, "Networks as a novel tool for studying team ball sports as complex social systems", *Journal of Science and Medicine in Sport*, 14(2), 170-176, 2011.
[5] F. Radicchi, "Who is the best player ever? A complex network analysis of the history of professional tennis", *PloS one*, 6(2), e17249, 2011.
[6] K. Breznik, "Revealing the best doubles teams and players in tennis history", *International Journal of Performance Analysis in Sport*, 15(3), 1213-1226, 2015.
[7] U. Michieli, "Complex Network Analysis of Men Single ATP Tennis Matches", *arXiv preprint arXiv:1804.08138*, 2018.
[8] M. Kostić, D. Drašković, "Complex Network Analysis of Women's Singles Tennis Matches", Telecommunications Forum (TELFOR), Belgrade, Serbia pp. 1-4, IEEE, 2020.
[9] A. G. Tennant, C. M. Smith, J. E. Chen C, "Who was the greatest of all-time? A historical analysis by a complex network of professional boxing", *Journal of Complex Networks*, 8(1), cnaa009, 2020.

[10] N. Almeira, A. L. Schaigorodsky, J. I. Perotti, O. V. Billoni, "Structure constrained by metadata in networks of chess players", *Scientific reports*, 7(1), 1-10, 2017.

[11] S. Mukherjee, "Identifying the greatest team and captain—A complex network approach to cricket matches", *Physica A: Statistical Mechanics and its Applications*, 391(23), 6066-6076, 2012.

[12] J. Sackmann, Repository tennis_atp, available on: https://github.com/JeffSackmann, accessed: 22.12.2020.

[13] A. A. Hagberg, D. A. Schult, P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX", Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, CA USA, pp. 11–15, August, 2008.

[14] M. Bastian, S. Heymann, M. Jacomy, (2009, March). "Gephi: an open source software for exploring and manipulating networks", Proceedings of the International AAAI Conference on Web and Social Media, vol. 3, no. 1, 2009.

[15] H. Situngkir, "Small world network of athletes: Graph representation of the world professional tennis player", Available at SSRN 1001917, 2007.

[16] V. D. Blondel, J. L. Guillaume, R. Lambiotte, E. Lefebvre, "Fast unfolding of communities in large networks", *Journal of statistical mechanics: theory and experiment*, (10), P10008, 2008.

[17] M. P. Rombach, M. A. Porter, J. H. Fowler, P. J. Mucha, "Core-periphery structure in networks", *SIAM Journal on Applied mathematics*, *74*(1), 167-190, 2014.

File system performance comparison of native operating system and Docker container-based virtualization

Borislav Đorđević, *Member, IEEE*, Darko Gojak, Nikola Davidović and Valentina Timčenko, *Member, IEEE*

Abstract **- This paper aims to examine and compare the file system capabilities of container virtualization and the native host. Different virtualization categories are mentioned with a focus on OS level types. We have described the importance of container virtualization and its contribution to virtualization popularization. Also, the paper contains a detailed description of the Docker container-based virtualization, its mode of operation, as well as the advantages and disadvantages it possesses. Since the main purpose of this work is to measure the host and Docker file system throughput, one of the best open-source benchmarks is chosen and presented - FileBench, through which all tests were performed. With a practical example, we have shown the file system performance comparisons considering Docker containers and host physical machine.**

Keywords - Docker; containers; virtualization; benchmark; FileBench; file system; performance; comparison.

I. Introduction

There is rapid development in the IT industry, while hardware and software are changing daily. Hardware development is accompanied by software solutions that aim to make the most efficient use of performance. We strive for solutions that will meet today's standards, asking ourselves what the best use is and how to optimize the available resources so that the requirements and user needs are met.

Some of the most important characteristics in hardware manufacturing are the development costs and time [1]. The above brings us to one of the indispensable topics of today in the IT world - virtualization.

The question is whether virtualization is a better solution and how cost-effective it is, whether it is possible to achieve the desired results with virtualization, and what the limitations are.

There are several varieties of virtualization types, and it can be said for all of these varieties to be usable, with some being more simplified, that is, less decomposed than others. One of the variations is that virtualization can be divided into eight types: hardware virtualization, network virtualization, storage virtualization, memory virtualization, software virtualization, OS level virtualization, data virtualization, and desktop virtualization.

The type of virtualization covered in this paper is "OS level virtualization", whose instances are sometimes called containers. One of the most common associations when mentioning container instances is the well-known Docker [2]. In this paper, the Docker container's file system is examined and compared with the host file system performance.

As the popularity of container virtualization has been growing over time, so have questions about the performance of this type of virtualization. It is hard to talk about container virtualization without mentioning the increasingly prevalent Docker. The ease of installation and use, as well as simplicity of containers management, made Docker a good candidate for file system testing. Another benefit of using it is that Docker containers are lightweight, time savers (it takes less than a minute to build one instance) and besides that, they are consuming a small amount of disk space, so those instances will not affect the host significantly.

Thus, in this paper, the response of the file system of the native operating system and Docker container-based virtualization was researched, and then a comparison of the obtained results was made.

II. Related work, objective, and motivation

As hardware is developing fast today, in terms of storage size, its response speed, as well as processing power, there is an inevitable question about the efficient use of physical machines, which are in most cases underused, or their full potential is not reached [3]. In this regard, scientific research deals with the consideration of further efficiency enhancements possibilities and the mentioned issues.

There is a growing debate about whether virtual solutions are always better and whether they can be expected to largely compete with physical machines [4], [5]. As a big part of the hardware resources in many cases remain unused, there is a lot of room left for the possibility of implementing virtual instances and consideration of the most optimal use.

As the main goal of this paper is to compare the performance of the file systems, with equal settings and the same conditions of the benchmark used for host and Docker

Borislav Đorđević – Institute Mihailo Pupin, Volgina 15, 11000 Belgrade, Serbia, (borislav.djordjevic@pupin.rs)

Darko Gojak – VISER, School of Electrical and Computer Engineering of Applied Studies, Belgrade, Serbia (darkogojak@gmail.com)

Nikola Davidović – University of East Sarajevo, Faculty of Electrical Engineering, Vuka Karadzica 30, 71123 East Sarajevo, RS, BiH, (nikola.davidovic@etf.unssa.rs.ba)

Valentina Timčenko - Institute Mihailo Pupin, School of Electrical Engineering, Belgrade, Serbia (valentina.timcenko@pupin.rs)

containers, we resorted to the method of comparative analysis through FileBench workloads, where four were selected, namely: fileserver, webserver, varmail and randomfileaccess. In our opinion, these are some of the best options for file systems workload testing procedures.

After setting the hypothesis, where it was expected that the physical machine dominates in all fields of given loads comparing to the containers, we proceeded with the application of the experimental method and obtained results that fully justified the assumptions. Based on the comparative analysis method, the obtained results confirmed the initial estimates and expectations, which is proved through the given equations as well as through workloads.

For better understanding and a clearer picture of the container's service capacity, measurements were also performed by increasing the number of Docker instances that worked in parallel, starting from one, until reaching four instances, where all of those were used simultaneously. The decrease of their power was observed and examined.

III. Host OS and Docker

To install the Ubuntu 20.04 operating system on the host, in this case with hardware characteristics shown in Tables 1 and 2, 1024 MiB of RAM is required at least. With Desktop image, which is the most common, there is the ability to try Ubuntu without changing the current computer system. There is also a Server install image that can be only permanently installed on the machine, but without a graphical user interface.

Experienced users are increasingly opting for Ubuntu when it comes to container operations. We can say that the most important item for security, performance, and quality is the Linux kernel, which always has the latest versions of the kernel accompanied by up-to-date security features. All of the above-mentioned is the reason why the world's largest cloud operators opt for Ubuntu operating system to run their containers [6].

Most users will agree and say that Docker became synonymous with container technology, as it had the greatest impact on popularization. But container technology is not a new term, it has been built into Linux in LXC form for more than ten years, and similar virtualization at the operational level systems was offered by: FreeBSD jails, AIX Workload Partitions, and Solaris Containers [7].

Unlike hypervisor virtualization, container virtualization does not have a hypervisor that would be used as a layer of abstraction, isolation of operating systems and applications from the host operating system. There are two types of hypervisors: type 1, which is mounted directly on the hardware, whereas, on the other hand, we can say that the Docker engine is like type 2, which depends on the host operating system, where the Docker container would be in the virtual machine role (Figure 1) [8].

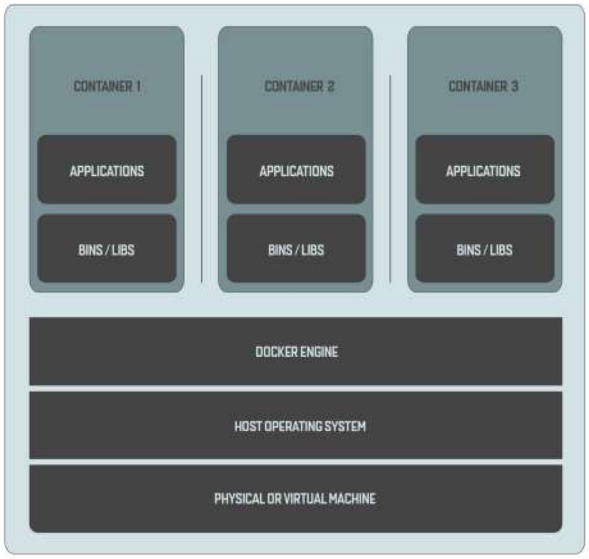


Fig. 1. Docker container-based virtualization

There is a belief that container virtualization is less secure compared with hypervisor virtualization because if weaknesses can be found in the host's kernel on which the containers are located, it could allow intrusion into the containers. The same can be said for the hypervisor, but since the hypervisor provides far less functionality than the Linux kernel (which usually implements file systems, networking, application process controls, etc.) it leaves much less space for attack. In recent years, great efforts have been made to develop software to improve container security. For example, Docker and other container systems now include a signing infrastructure that allows administrators to sign container images to prevent the deployment of unreliable containers [9].

Below is a simple description of docker client-server architecture. Docker client communicates through REST API, over network interface or UNIX sockets with Docker daemon which does building, running and distributing containers (Figure 2) [10]. It is not mandatory that Docker daemon has to run on the same operating system as the Docker client, which can also be connected to a remote daemon [11].

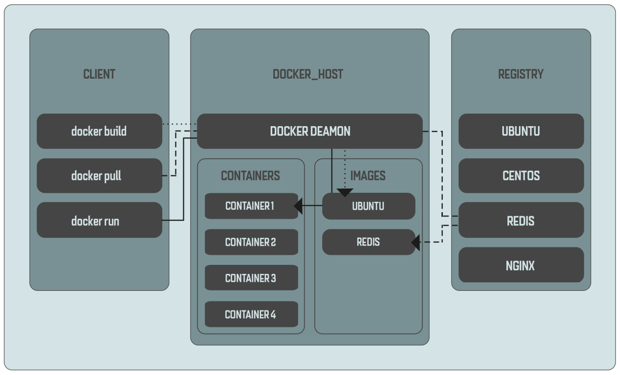


Fig. 2. Docker architecture

Some Linux distributions are designed for running containers and Docker such as Project Atomic [12], Photon OS, RancherOS, etc. [13]. Since 2016, Docker containers have also been able to run on Windows operating system and managed from any Docker client or through Microsoft PowerShell [14].

Docker can also work on popular cloud platforms [15], including Amazon Web Services, Google Compute Engine, Microsoft Azure, Rackspace, etc. [16].

IV. Hypothesis of Expected Behaviour

To make it easier to understand how the results were obtained, the following formulas were derived:

$$T_{WKLD} = T_{RR} + T_{SR} + T_{RW} + T_{SW} \quad (1)$$

In equation 1, the *TWKLD* notation stands for the total processing time for each workload. This is followed by a random - *TRR* and a sequential - *TSR* reading time, while the *TRW* notation indicates the random write time, and *TSW* stands for sequential write time. The following formula represents the expected file system access time for each individual workload:

$$T_W = T_{DIR} + T_{META} + T_{FL} + T_{FB} + T_J + T_{HK} \quad (2)$$

The *TW* notation above represents the total time required to complete all operations on the ongoing workload. The following notations represent the time required to complete all operations related to: directory - *TDIR*, metadata - *TMETA*, free list - *TFL*, file block - *TFB*, journaling - *TFJ* and house-keeping - *THK*. There are two candidates for file system performances that are covered in this paper and they are:

1. native HostOS
2. native HostOS + Docker engine + containers

1. The Ubuntu 20.04 operating system is installed on the host with its default file system, and since the Docker containers are running on it, the native host will play a major role in terms of file system performance. For a better comparison with the host, Ubuntu image is pulled and run on all four container instances. Thus, benchmark and the host file system characteristics depend on the time needed to process benchmark-generated workload, and are noted in the following formula as *TW*:

$$T_W(nHost) = f(Bench, hOS\_FS) \quad (3)$$

2. The docker engine has the biggest impact on performance after the host and its file system where everything takes place. As mentioned, HostOS, Docker engine, and containers run on the host file system, except for Docker volumes and self-storage. The benchmark, the host file system characteristics, and Docker engine mapping depend on the time needed to process benchmark-generated workload, in the following formula noted as *TW*:

$$T_W(DOCKER) = f(Bench, D\_engine, hOS\_FS) \quad (4)$$

The obtained performance results of the file system of the host and Docker container were predicted by the given formulas. So, as expected, the host was in the lead through all workloads, which was confirmed by the calculation from equation 3. There are small differences in throughput in all segments between the single running container and the host. This lag in the performance of the container was caused by the Docker engine, which was also confirmed by equation 4. After monitoring the throughput of these instances, the following conclusion was made:

Single Docker container is slightly behind the host performances by all measurements, while for any increase of containers running in parallel by one instance, the deterioration in throughput power should be expected.

V. Test Configuration and Benchmark Application

There are various tools, benchmarks that can measure performance in order to examine the capabilities of physical machines as well as the capabilities of virtual solutions. Some benchmark tools are open-source, while others are commercial solutions. Depending on the purpose of the tests, we can opt for one of the most adequate tools. For these measurements, a FileBench is chosen as one of the most suitable benchmarks.

FileBench is a storage and file system benchmark. It uses its own Workload Model Language (WML) that can allow I/O specification of application behavior. It is one of the best-known open-source tools, which, unlike most of the tools that mainly rely on predefined workloads (which cannot be changed in most cases), allows workload modifications as well as adaptation to the specificities of the purpose for which the testing is performed.

Installing a FileBench benchmark is quite simple after downloading the software package. However, on Ubuntu, it requires a few more commands than on Centos operating system, for instance, where it is possible to install it with a simple "yum install filebench" command. Additionally, there is a difference in the installation of the benchmark between two versions covered in this paper. In the first part of the installation, as the configuration files are not included in the repo, they have to be created. Therefore, for the last stable version, it is necessary to run the following commands if they are not installed, respectively: libtoolize, aclocal, autoheader, automake, --add-missing, autoconf.

The second part of the installation requires the installed gcc, flex and bison in order to run FileBench [17]. This part is the same as in the 1.5-alpha3 version, except that in this version it is the only step and it involves running the following commands, respectively: ./configure, make, make install.

In order to measure as accurately as possible and to obtain as better results as possible, Ubuntu 20.04 operating system was installed on the host (hardware shown in Tables 1 and 2) only for this file system test purpose, which after the installation of the benchmark had no other applications that could disrupt the operation of this tool in any way. Also, containers had nothing but installed FileBench.

After everything is set, there is still one thing left to do and that is disabling ASLR (address space layout randomization) by changing the value of randomize_va_space to "0" (zero), otherwise, the workloads will be blocked in the stage of running.

TABLE I
HARDWARE CONFIGURATION OF THE HOST

Component	Characteristics
Processor	AMD Ryzen 5 3600X, 3.8GHz – 4.4GHz, 6 Core, 12 Thread
Cache	L1 Cache 384KB, L2 Cache 3MB, L3 Cache 32MB
Memory	16Gb DDR4, 3200MHz
SSD	Kingston A2000 SA2000M8/500GB
Motherboard	GIGABYTE B450M DS3H

TABLE II
SSD characteristics

Capacity	500GB
DRAM	DDR4
Interface	NVMe™ PCIe Gen 3.0 x 4 Lanes
Form factor	M.2 2280
NAND	3D TLC
Sequential Read/Write	up to 2.200/2.000MB/s
Random 4K Read/Write	up to 180.000/200.000 IOPS

VI. TESTS AND RESULTS

Each measurement was done in three rounds per host and per each container instance, after which the average value was taken for results. The obtained measurements of individual container performances were then compared with the results obtained while testing the host. The throughput of each container was observed in cases when only one container instance was started, when two instances were running in parallel, and when three and then four containers were running at the same time.

File system performance tests were conducted on the latest stable version of FileBench - 1.4.9.1 and 1.5-alpha3 version where throughput was measured in MB/s. For the purposes of this experiment, four of the over fifty predefined workloads were selected. On both versions, the performance of the filesystem was tested via three workloads that were used to emulate applications, namely: fileserver, webserver and varmail. On the last stable version, an additional workload was included - radnomfileaccess. The following is a brief description of workloads that were used and covered with formulas (1) and (2): *Fileserver* – It mimics the elementary I/O activity of a file server. It performs a sequence of creating, deleting, adding, reading, writing, and attribute operations on a directory tree; *Webserver* - Mimics elementary I/O activity of a web server. Produces an open-read-close sequence on multiple files in a directory tree, plus appends a log file; *Varmail* - Imitates elementary I/O activity of a mail server that saves each e-mail in an isolated file (/var/ mail/server). It contains a set of multiple threads of the following operations in a particular directory: create-add-sync, read-add-sync, read, and delete; *Randomfileaccess* - Uses random variables that are user-defined entities, and these entities are formulated by a random distribution that is used to select a random value that is returned with each use [18].

It is hard not to mention virtual clusters when Docker containers are used. Testing could take on a completely different dimension if any container orchestration platforms such as Kubernetes were used, where containers would combine and pool their serving powers [19]. But the purpose of these tests was to compare the file system performance of the host and individual container.

The parameters shown in Tables 3 and 5 are set with default values. The values for the four specified parameters (number of files - nfiles, average file width, and size - (meandirwidth, meanfilesize), as well as the number of threads - nthreads) are the same in both versions of the benchmark. The time for executing each of the workloads is set to 60 seconds, which is the default value for most of the predefined workloads.

TABLE III
PARAMETERS OF THE SOURCE CODE *.F FILES (1.4.9.1 VERSION)

Workload (runtime 60s)	Fileserver	Webserver	Varmail	RFA
nfiles	10.000	1.000	1.000	10.000
meandirwidth	20	20	1.000.000	20
meanfilesize	128k	16k	16k	Random
nthreads	50	100	16	5

TABLE IV
BENCHMARK RESULTS (MB/S), 1.4.9.1 VERSION

Instance	Fileserver	Webserver	Varmail	RFA
Host	3866.6	1001.5	187.4	19081.8
1 container	3746.1	962.8	180	18190.1
2 containers	1764.4	695.9	158.3	9438.5
3 containers	1170.5	528.5	137.2	5300.7
4 containers	651.7	458.8	117.2	3809.8

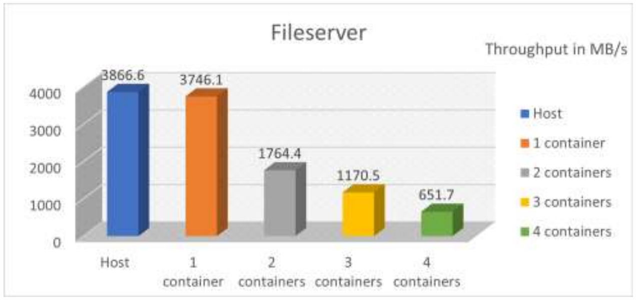


Fig. 3. Fileserver test results from Table 4

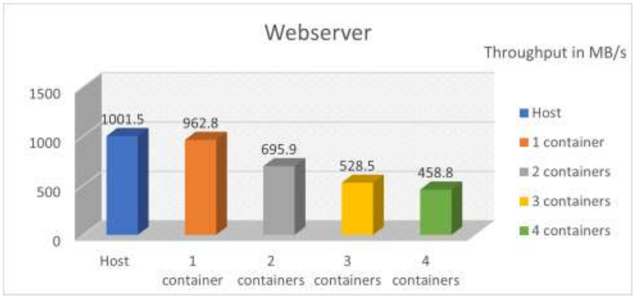


Fig. 4. Webserver test results from Table 4

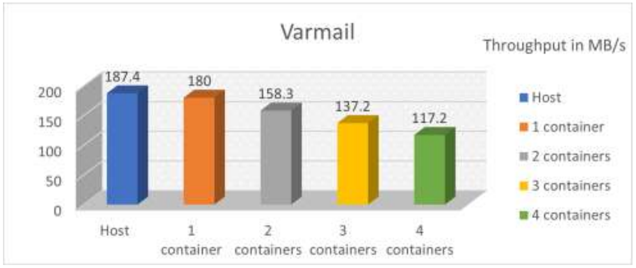


Fig. 5. Varmail test results from Table 4

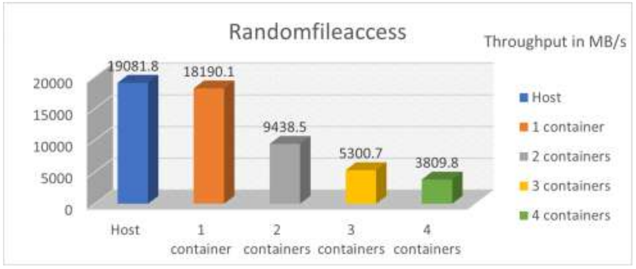


Fig. 6. Randomfileaccess test results from Table 4

A. Measurements performed on version 1.4.9.1

The host had better performance in all four categories which is shown in Table 4. The obtained results were proved by formulas (3) and (4). Starting with the fileserver environment, there is a small throughput difference of 3 % in favor of the host compared to a single container. Then, as expected, by increasing the number of containers by one, the serviceability also decreases, so that the performance of the two running containers drops by more than twice, i.e. 54%. Performance with three running containers deteriorated by 70% and with four instances the results showed it to be 83% (Figure 3).

For webserver tests, the results are as follows. The throughput at the host instance is 4% higher when compared to a single running container, while for two running containers that gap is 30%. With three and four containers in running state, we can see the degradation of 47% and 54%, respectively (Figure 4).

In the case of varmail environment, the single running container has lower performances by 4%, two containers by 16%, and three and four containers by 27% and 37% compared to the host (Figure 5).

The randomfileaccess workload also had poorer container results, showing performance declines of 5, 51, 72, and 80% when having 1, 2, 3, and 4 containers in running state, respectively (Figure 6).

TABLE V
PARAMETERS OF THE SOURCE CODE *.F FILES (1.5-ALPHA3 VERSION)

Workload (runtime 60s)	Fileserver	Webserver	Varmail
nfiles	10.000	1.000	1.000
meandirwidth	20	20	1.000.000
meanfilesize	128k	16k	16k
nthreads	50	100	16

TABLE VI
BENCHMARK RESULTS (MB/S), 1.5-ALPHA3 VERSION

Instance	Fileserver	Webserver	Varmail
Host	4072.6	3333.8	163.5
1 container	4007.1	3080.1	133.4
2 containers	1696.7	1563.5	111.2
3 containers	704.6	1160.8	88.5
4 containers	451.4	952.6	76

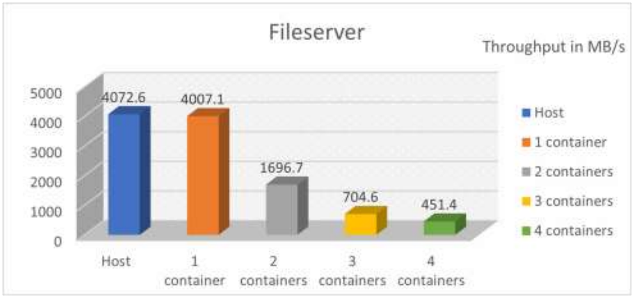


Fig. 7. Fileserver test results from Table 6

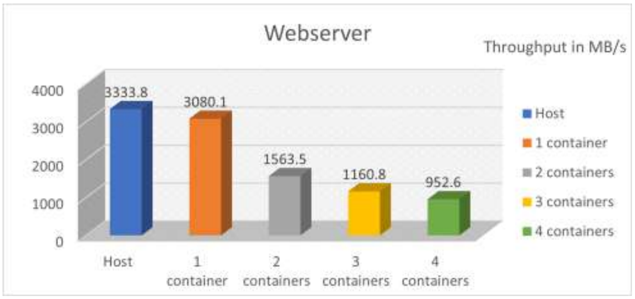


Fig. 8. Webserver test results from Table 6

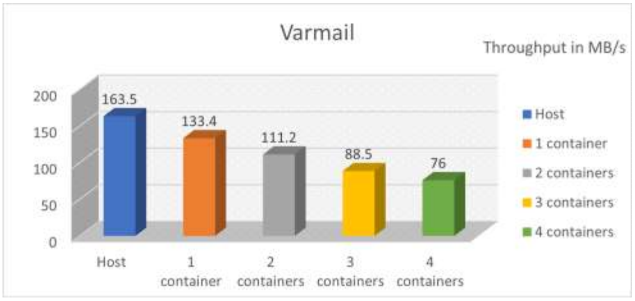


Fig. 9. Varmail test results from Table 6

B. *Measurements performed on version 1.5-alpha3*

Within a FileBench version 1.5-alpha3, the expected results were obtained, which is verified by formulas (3) and (4). As well as in the latest stable version the host dominates (Table 6). In the fileserver case, a single container performance does not significantly differ from the host and it is lower by 2%, while for two container instances in running state the drop is much bigger, 58%. For three and four instances it is 83% and 89%, respectively per container (Figure 7).

With webserver workload tests we have a throughput deterioration comparing to host, namely 8% for a single container, 53% in the case of two instances, 65% for three running containers, and 71% per instance in the case of four containers running (Figure 8).

As for varmail, the host throughput is higher by 18% compared to a single container, while for two instances there is gap of 32% per instance, it is 46% for three instances and 54% for all four containers (Figure 9).

VII. Conclusion

According to the shown tests, the host had better performance in all segments compared to Docker containers which justifies the hypothesis. During performance monitoring through all four workloads, a slight differences in throughput between the host and single container is noticeable. As we can see in the obtained measurement results, the increase of the number of container instances decreases their service power, which also differs from workload to workload. Those are expected results, and accordingly, depending on the load, we can determine whether containers are suitable and if they will meet the requirements for which container instances were originally intended.

This is only a small segment in testing the host and Docker container capabilities, as there are over forty predefined tests left, as well as many variations of modifying existing and writing your own workloads that can be processed. Since FileBench workloads can be easily managed it leaves a lot of room for future measurements and comparisons with the results of other benchmarks that are not so flexible in terms of tests.

Today, it is known that hardware development is increasingly focusing on multi-core solutions that can process many instructions in a very short time. That leaves plenty of room for further processing of power and resources, which is suitable for the normal and smooth operation of virtual solutions. Virtualization is not always the answer to everything, for some purposes virtualization simply does not achieve the desired results so in that case, the only choice is a physical machine. But in most cases, security, productivity, and cost-reducing benefits outweigh all problems, and therefore Docker virtual solutions and virtualization, in general, are increasingly gaining in popularity.

Acknowledgment

The work presented in this paper has partially been funded by the Ministry of Education, Science, and Technological Development of the Republic of Serbia.

Literature

[1] C. Walls, “Hardware and software development; what’s the cost?,” 2018 [online]: https://www.embeddedcomputing.com/technology/software-and-os/hardware-and-software-development-what-s-the-cost
[2] IBM Cloud Team, IBM Cloud. Containers vs. VMs: What’s the difference? IBM, 2020. [online]: https://www.ibm.com/cloud/blog/containers-vs-vms
[3] Spiceworks. The 2020 State of Virtualization Technology, 2019. [online]: https://www.spiceworks.com/marketing/reports/state-of-virtualization/
[4] K. Thompson, “Hardware vs. Software development: Similarities and Differences,” Cprime, 2015. [online]: https://www.cprime.com/resources/blog/hardware-vs-software-development-similarities-and-differences/
[5] T. Collins, “Virtual servers vs physical servers: Which is best? 10 March,” Atlantech, 2020. [online]: https://www.atlantech.net/blog/virtual-servers-vs-physical-servers-which-is-best
[6] Canonical. Why is Ubuntu #1 OS for containers? Ubuntu, 2018. [online]: https://ubuntu.com/containers
[7] S. Hogg, “Software Containers: Used More Frequently than Most Realize,”, Networkworkd, 2014 [online]: https://www.networkworld.com/article/2226996/software-containers--used-more-frequently-than-most-realize.html
[8] U. Hiwarale, “Anatomy of Docker,” [online]. 2018 Nov [Accessed 24 February 2021]. Available from: https://itnext.io/getting-started-with-docker-1-b4dc83e64389
[9] P. Rubens, “What are containers and why do we need them?,”, Cio, 2017 [online]: https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html
[10] N. Poulton, Docker Deep Dive, JJNP Consulting Limited, Lean Publishing. Leanpub book, 2018.
[11] Docker. Docker overview. [online]: https://docs.docker.com/get-started/overview/#docker-architecture
[12] SP. Kane, K. Matthias, “Atomic hosts,” in Docker: Up and Running. 2nd ed. O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472; 2018.
[13] JM. Scheuermann, “A Comparison of Minimalistic Docker Operating Systems,” Inovex, 2015. [online]: https://www.inovex.de/blog/docker-a-comparison-of-minimalistic-operating-systems/
[14] M. Friis, “Build and run your first Docker Windows Server container,” Docker, 2016. [online]: https://www.docker.com/blog/build-your-first-docker-windows-server-container/
[15] C. Ward, “The Shortlist of Docker Hosting,” CloudBees, 2016. [online]: https://www.cloudbees.com/blog/the-shortlist-of-docker-hosting/
[16] J. Turnbull, “Installing Docker,” In The Docker Book. CC BY-NC-ND 3.0; 2018.
[17] G. Amvrosiadis, “FileBench,” GitHub, 2016 [online]: https://github.com/filebench/filebench
[18] V. Tarasov, “Predefined personalities. [online]. 2016 Jul [Accessed 24 January 2021]. Available from: https://github.com/filebench/filebench/wiki/Predefined-personalities
[19] IM. Aidan, H. Sayers, “Using a Kubernetes cluster,” in Docker in Practice. Manning Publications Co. 20 Baldwin Road PO Box 761 Shelter Island, NY 11964; 2016.

Performance comparison of native host vs. ESXi hypervisor-based virtualization

Borislav Đorđević, *Member, IEEE*, Srđan Milenković, Nikola Davidović and Valentina Timčenko, *Member, IEEE*

Abstract – The main objective of this paper is performance comparison of hypervisor-based virtualization with VMware ESXi virtual machines and native host machine. From all performance classes, for the needs of this research we have chosen the evaluation of the file system performance. The measurements are carried out under equivalent conditions and by a unique test method, using the Filebench software, which guarantees equality and independence from the impact of hardware and operating system characteristics. As the base operating system we have used CentOS 7.7 with the latest updates, while ESXi 6.7 was used as the hypervisor. Performances are compared for the native host machine and ESXi server with one, two and three virtual machines (VM) running simultaneously. We have also analysed the expected behaviours, verified the assumption with Filebench testing software, and provided the concluding remarks for this papers research topic.

Key words – Virtualization; Filebench; Hypervisors; ESXi; VMware; CentOS; Virtual Machines

I. INTRODUCTION

In IT world, the term virtualization refers to the act of creating a virtual version of something, or it is the process of creating and running a virtual instance of a computer resource in a layer abstracted from the actual hardware. It is used to describe virtual computer hardware platforms, storage devices, network resources, server infrastructure, etc. We can experience virtualization in almost all segments of today's computer technology. The main idea behind virtualization is a very simple and came from the corporative approach: the need to satisfy the increase in the utilization of available hardware resources, while at the same time reducing the costs of the infrastructure. Virtualization did exist as a technology even some 30 years ago, but the hardware of those days could not exploit the full usage that virtualization brought, so it was disregarded until progress was made in computer technology giving to virtualization a new meaning, shaping it to what it looks today. Nowadays, thanks to this technology it is possible to run multiple independent operating systems on one physical server. Some of the benefits that virtualization provides are primarily related to saving the necessary physical space that would be needed for the accommodation of the devices and also the electrical energy consumption that would inevitably be used for powering such devices. Today, the use of virtualization in a simple way increases server availability and isolation, making it one of main reasons why these technologies are so popular [1]. When using these technologies it is important to mention that the level of hardware utilization of servers without virtualization is in the range of 15% of its maximum capacity, while with the use of virtualization technologies the utilization raises to more than 70%. These technologies however come with a price, or to be exact, with the retention or even increasing availability of resources, while it is realistic to expect a somewhat lower performance of virtualized systems when compared to the non-virtualized systems, which is the main topic of this paper.

There are several virtualization types: virtualization of hardware, software, desktop, data, network, memory, storage, etc. We are focused on hardware virtualization. Hardware virtualization implies the use of a hypervisor, a layer that acts as a mediator between the host and virtual machine, which is nothing more than a simulated computing environment that can, but does not have to be equal to the physical environment that it simulates. In addition to the classification by the location of the hypervisor layer, the hardware virtualization also depends on what type of virtualization is provided, and can be categorized as: full, hardware-assisted, and paravirtualization.

Full (native) virtualization is a virtualization technique that completely simulates the underlying hardware. Hardware-assisted virtualization (Intel VT-x or AMD-V) is platform virtualization approach that enables efficient full virtualization using help from hardware capabilities, primarily from host processors. In this situation, the processor simulates hardware that does not have to be the same as physical. Paravirtualization is an enhancement of virtualization technology in which a guest operating system is modified prior to the installation inside a virtual machine in order to allow all guest OS within the system to share resources and successfully collaborate, rather than attempt to emulate an entire hardware environment [2].

The remainder of this paper will be structured as follows. Section II provides a brief description of the technologies that are mentioned in the paper and a short review of related work for this project. Section III provides the description of the

Borislav Đorđević – Institute Mihailo Pupin, Volgina 15, 11000 Belgrade, Serbia, (borislav.djordjevic@pupin.rs)
Srđan Milenković - School of Electrical and Computer Engineering of Applied Studies, Vojvode Stepe 283, 11000 Belgrade, Serbia, (smilenkovic1992@gmail.com)
Nikola Davidović – University of East Sarajevo, Faculty of Electrical Engineering, Vuka Karadzica 30, 71123 East Sarajevo, RS, Bosnia and Herzegovina, (nikola.davidovic@etf.ues.rs.ba)
Valentina Timčenko - Institute Mihailo Pupin, School of Electrical Engineering, Belgrade, Serbia, (valentina.timcenko@pupin.rs)

performance-measuring tool that we have used for this experiment. In Section IV, we present a short description of the architecture of the used hypervisor. Section V presents the hypothesis and methodology used to achieve performance comparison. In Section VI, we present the test environment and configuration for the experiment. Test results for our benchmarks' tests are presented in Section VII. In Section VIII, we draw conclusions to the work made in this paper.

II. Related work and objective

This paper is primarily devoted to analysis of the performances of hypervisor-based virtualization with one of the most commonly used hypervisors. The hypervisor serves as a layer between the virtual machine's operating system and the host's physical memory, providing data integrity and isolation of VMs. Thanks to hardware-assisted virtualization which is accomplished via EPTS (extended page tables, for Intel chipsets) or RVI (rapid virtualization indexing, for AMD) we have a large increase of speed compared to software memory virtualization [3]. The paper considers advantages of using virtual machines while creating modern network infrastructure; as well as describes an experiment using common test environments and programs for measuring and analysis of hypervisors and their performances. Benchmarking is a popular approach nowadays for many devices and general I/O performance analysis, whereas the special attention is put on the problem of fast input/output support [4-6].

Main contribution of this paper is the examination of the performances of the native host operating system and hypervisor-based virtualization of VMware ESXi [7] [8]. As the technology that was used for this research is relatively new, there are not many references in literature that research with similar environments, tools, and test characteristics. The goal of this paper is to examine the file system performance of the generated workload through Filebench software tool for: (1) mail server scenario which is dominated by random read and random write components; (2) web server scenario where random read components dominate; (3) file server scenario in which both random and sequential components are equally represented; and (4) random file access scenario dominated by random read component [9].We have set up a model for the file system performance analysis of the native host and ESXi based virtual machines. The results of this experiment should give us a full picture of how the performance of a native machine compares to the performance of a hypervisor-based virtual machine.

III. Filebench

Filebench is a software test environment (usually called a benchmark) used to measure the performance of various parts of an operating system. What sets Filebench apart from other benchmarks is the fact that it is equipped with several predefined workloads, which allows users to easily test their systems in various forms (most popular forms being a mail server or a file server) [10]. Presently many benchmarks hard code the workloads they generate quite rigidly, meaning that a user can specify some of the basic workload parameters, but cannot really control the execution flow of the workload in detail. Filebench gives its users freedom to define workloads using a Workload Model Language (WML). WML is mainly composed of four main parts: fileset, process, thread, and flowop.

A standard Filebench test is executed in two stages: fileset pre-allocation and a workload execution. First part of any workload execution is defining a fileset that it uses. A fileset is a named collection of files and to define it a user must specify its name, path, number of files, and a few other optional attributes that can be included in a filesets creation. After defining a fileset the next step are the processes in WML that represent real UNIX processes which are created by Filebench during the test. Every process is made of one or more threads representing an actual POSIX threads and every thread executes a loop of flowops. A single flowop is a representation of a file system operation that is translated to a system call by Filebench.

The ending of a WML file usually contains one of two "run" commands (run and psrun) that tell Filebench to allocate the defined filesets, prepare the required number of UNIX processes and threads, and start a cycled flowops execution. After completing a run, Filebench gives a number of different metrics, where the most important one for the user is operations per second. This is the total number of executed flowop instances (in all processes and threads) divided by the time it took for a full run of the workload. To generate a workload and start the measurement of a particular part of the system, one must execute the *filebench -f workload.f* command.

IV. ESXi Hypervisor

VMware ESXi (Elastic Sky X "integrated") is a type-1 hypervisor developed by VMware for deploying and serving virtual machines that was made from its predecessor ESX. Type-1 hypervisors run directly on the host's hardware to control the given hardware and to manage guest operating systems, and for this reason they are mainly called bare metal hypervisors (Figure 1). A guest operating system runs on another level above the hypervisor.

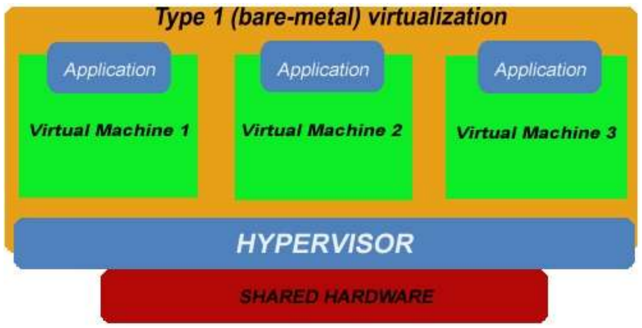


Fig. 1.Type-1 (bare metal) hypervisor

VMware ESXi is a hypervisor that runs on the host server hardware without the underlying operating system. ESXi provides a virtualization layer that abstracts the CPU, storage, memory and networking resources of the physical host into multiple virtual machines. That means that applications running in virtual machines can access these resources without direct access to the underlying hardware. VMware refers to the hypervisor used by VMware ESXi as VMkernel and it receives requests from virtual machines (as processes that run on top of it) for resources and presents the requests to the physical hardware [12-14]. The kernel also provides means for running all processes on the system, including management applications and agents as well as virtual machines. It has control of all hardware devices on the server, and manages resources for the applications as shown in Figure 2 [15]. The main processes that run on top of VMkernel are:

- Direct Console User Interface (DCUI) — the low-level configuration and management interface, accessible through the console of the server, used primarily for initial basic configuration.
- The VMM, virtual machine monitor, which is the process that provides the execution environment for a virtual machine, as well as a helper process known as VMX. Each running virtual machine has its own VMM and VMX process.
- Various agents are used to run and enable high-level VMware Infrastructure management from remote applications.
- The Common Information Model (CIM) system is the interface that enables hardware-level management from remote applications via a set of standard APIs.

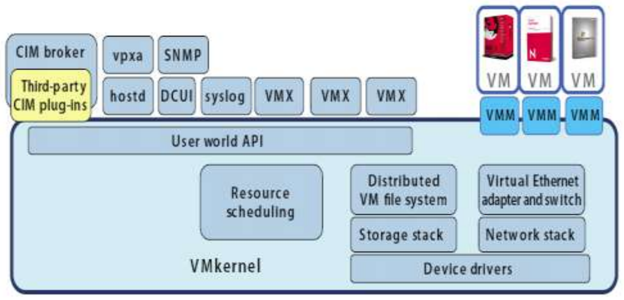


Fig.2.VMware ESXi architecture

V. Hypothesis of Expected Behaviour

Since we are using a Type-1 hypervisor that works directly on hardware, the total processing time for each workload T_W can be described by the following equation:

$$T_W = T_{RR} + T_{RS} + T_{WR} + T_{WS} \quad (1)$$

where T_{RS} and T_{RR} represent sequential and random read time respectively, while T_{WR} and T_{WS} represent random and sequential write time respectively. For every specific workload we have an expected access time for the file system which includes five components as shown in following equation:

$$T_{WORKLOAD} = T_D + T_M + T_{FL} + T_{FB} + T_J + T_{HK} \quad (2)$$

where $T_{WORKLOAD}$ represents the overall time for finishing all operations on the current workload, and T_D, T_M, T_{FL}, T_{FB}, T_J, T_{HK} represent time needed for completing all operations related to directory, metadata, free list, file block, journaling and house-keeping operations in the file system, respectively.

In this study we have a specific situation where there are two sides which have identical settings of the operating system (CentOS) and the file system (XFS), used in the performance testing: (1) Native machine (hostOS) and (2) ESXi + VMs(guestOS).

1. Native hostOS: The time to process the generated workload depends on the benchmark interaction with the hostOS file system and also the characteristic of the file system. Total time to process the workload, $T_{W(native)}$ is defined as:

$$T_{W(native)}=f(benchmark,\ hostOS\_FS) \quad (3)$$

2. ESXi + VMs(guestOS): The time to process the generated workload ($T_{W(ESXi)}$) in this case depends on the benchmark interaction with guestOS file system, the characteristic of the file system and the virtualization processing component of the ESXi hypervisor (ESXi_proc) is as in the following formula:

$$T_{W(ESXi)}=f(benchmark,\ guestOS\_FS,\ ESXi\_proc) \quad (4)$$

Since we use the same settings as the native machine for our virtual machines, the benchmark interaction and characteristics of the file system on the guest will be the same as the ones on the native machine. The virtualization processing component depends on the virtualization type and hypervisor processing as in the following formula:

$$ESXi\_proc = f\,(virt\_type,\ hyp\_proc) \quad (5)$$

In the context of virtualization type, ESXi uses full virtualization, which is further enhanced with one of the technologies (depending on hosts' CPU) for hardware assisted virtualization. In the context of the hypervisor processing it is important to consider the delay, which represents the time required for the hypervisor to receive requests from virtual hardware of a guest OS and forward them to the hosts' hardware for proccessing. The delay can be explained as following: virtual machines generate workload, which passes from a VM through the hypervisor onto the hosts' hardware. First the benchmark application generates the workload which is passed on for further processing to the hypervisor. The second part happens inside the hypervisor and is defined as the interaction between guest workload and VM image file. Generated workload is passed on the hypervisor, which maps it into requests for VM large image files. Lastly the hypervisor's mapping process generates input files as requests for real disk drivers on the hosts' hardware. The time needed for generating those requests depends on the hypervisor's file system and caching capabilities.

The expected outcome according to formula (3) is that the native host will perform better than ours ESXi virtual machines. Virtual machines have a complex data path, formula (5), where data must pass through guest OS file system and the hypervisor onto machine hardware. Therefore, it is expected that a degradation of the ESXi VM performance will happen compared to the native host machine, formula (4).

We have investigated a few cases for the this paper: firstly, the performace of a native host machine, then the performance of a single ESXi VM running and lastly the performance of several virtual machine running at the same time. In general, we expect:

- Native host to perform better when compared to ESXi with one virtual machine running.

- Running several instances of the ESXi virtual machines, n*ESXi VMs ($n=1,2,3$...), should have a significant performance degradation compared to the native host.

VI. Test environment configuration

The assumption of an adequate testing is the application of a single hardware configuration, the same operating system, and measurement methodology for all test procedures as mentioned before. The hardware configuration contains all the components necessary for a modern-day computer, and in this case, it is a home-based system of the newer generation (Table 1). CentOS version 7.7 is selected as the operating system, which is currently one of the most popular Linux distribution.

During the installation process we opted for Gnome graphical interface installation option with essential packages and programs for a graphical environment. The XFS file system characteristics and layouts are shown in Table 2. Filebench is a program designed to measure the performance of file systems and storages, and it is capable of generating multiple workload types that simulate environments when using certain servers/services such as mail, web, file, database, etc. Before starting tests, we made sure that all available updates were installed. Each virtual machine was given 4 GB of RAM.

TABLE I
Hardware configuration of the test PC

MB	Gigabyte B75M-D2V
RAM	DDR3 1330 MHz, 16 GB
CPU	Intel
Model	Pentium G860
Cores	2 /2 threads
Speed	3.00 GHz
Cache(L1,L2,L3)	2x32kB; 2x256kB, 3MB
SSD	Samsung SSD 860 EVO
Interface	SATA 6Gbps
Capacity	250 GB
OS	CentOS 7.7.1908.el7

This benchmark behaviour is controlled using files with the extension *.f that are written in Workload Model Language, that can be edited in any text editor. The use for individual measurements involves putting a command from a terminal with root privileges using the name of the *.f file as an argument.

TABLE II
Fs layout

File system	Size	Mount
/dev/mapper/data-root	35 GB	/
/dev/sda1	4 GB	swap
/dev/sda2	1024 MB	*/ boot*

VII. Tests and results

The focus of this paper was to measure the performance of hard disks and data-flow in one of the more popular virtualization systems, especially in cases where several instances of virtual machines are being used. The main idea was: as the number of instances increases, there is a significant drop in performance and this drop is constant on any hardware-software configuration. Benchmark of the host computer without virtualization was taken as a reference point for file system performance in these tests.

A number of modified files of the source code ***fileserver.f***, ***webserver. f***, ***randomfileaccess.f*** and ***varmail.f*** were used during the tests, which are thus testing the files, web and the mail server environments, respectively. The changes were taken into consideration when setting the benchmark parameters in a way to provide as realistic as possible exploitation conditions. And while the location (/ bench), the I/O block size (iosize = 1M) and the average size of the add-on (meanappendsize = 16k) are common denominator for all tests, the parameters such as the number of files (nfiles), the average depth of the directory (meandirwidth) the average file size (meanfilesize), cache and the number of threads (nthreads) are changed on a case-by-case basis (with * .f files). The defined settings are retained throughout the entire benchmark test and are displayed in Table 3. For an easier view in the following table the name of each benchmark workload has been abbreviated with their initials (file server (FS), web server (WS), mail server (VMail) and random file access (RFA).

TABLE III
Settings of the source code in the *.f files

	FS	WS	VMail	RFA
nfiles	10.000	1.000	1.000	10.000
meandirwidth	20	20	1.000.000	20
meanfilesize	16k	16k	16k	
nthreads	50	100	16	5
cached				false

The duration of each test was 120 seconds, which is also stated in the *.f files, with the goal of acquiring the most realistic results. Special attention was paid to keep the OS clean and the impact of any external subject on system components was reduced to the minimum. After performing a reference measurement of the host computer without virtualization, ESXi was installed and three virtual machines were generated. Tests were conducted in a way that one virtual machine was first started and measured, then two and three machines simultaneously. From the generated data, the final conclusions were made by calculating the average values of the results.

TABLE IV
BENCHMARK RESULTS (MB/S)

	FS	WS	VMail	RFA
Host	401.6	127.9	51.4	7379.5
1VM	230.4	67.6	45.7	3646.0
2VM	122.3	42.8	24.4	2126.9
3VM	78.2	24.9	14.8	1498.6

Table 4 shows the data we collected from workloads running in the test environment (again we used the same abbreviations like in Table 3). Data from Table 4 are shown on the next few figures, with remarks on the performance displayed in each. All of the measures shown in the following figures are displayed in megabytes per second (MB/s).

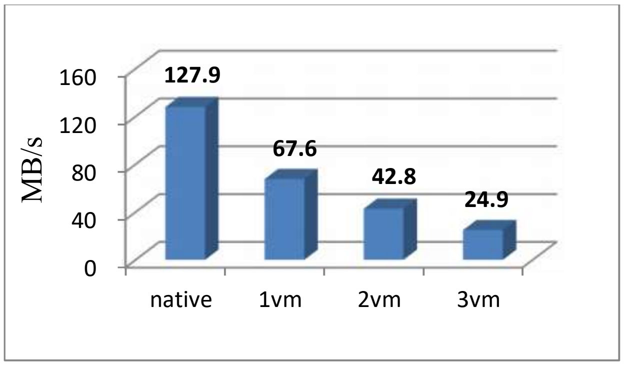


Fig. 3. Webserver.f workload test results

The characteristics of the webserver.f workload with our specification (100 threads) is that random reads dominate, there are some random write components, while the sequential components are not present. Here we observe that native host OS performs much better than in the case with one instance of the ESXi virtual machine (Figure 3). In the case of this workload, instantiation of more than one ESXi virtual machine brings some performance degradation but not significant.

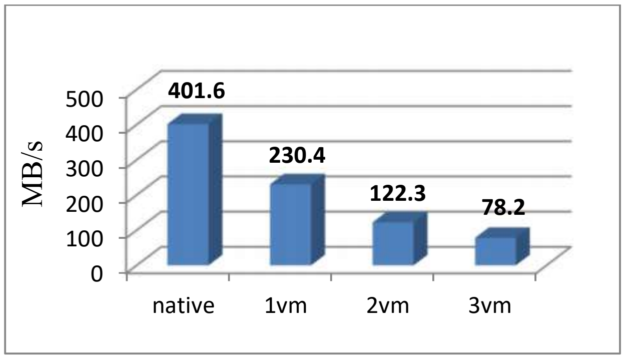


Fig. 4. Fileserver.f workload test results

The characteristics of the fileserver.f workload with fifty threads, are that both random and the sequential components dominate, but there is also a large number of I/O requests and much heavier data flow. A general notion is that, in the case for one virtual machine instance, the ESXi is significantly weaker than in the case of the native host OS. In the case of fileserver.f workload, when two virtual machines are instanced, the performance is further degraded by approximately the same amount as in the previous case (with one virtual machine). Instancing a third virtual machine brings very little performance degradation (Figure 4).

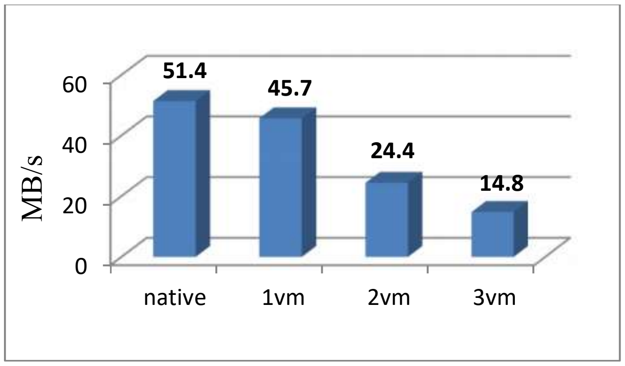


Fig. 5. Varmail.f workload test results

The characteristics of the varmail.f workload with our specification (16 threads) are that the components of random read and write are dominating, while the sequential components are not present, as it is shown in Figure 5. The special characteristic is that the components of the random write are synchronous, so each write will end up on the disk. The general notion for one instance is that performances of ESXi virtual machine are close to the native host OS. However, synchronous entries cancel the effects of cashing, so there are minor differences between native host OS and one instance of ESXi virtual machine. In the case of varmail.f workload, the instantiation of more than one ESXi virtual machines does not bring significant performance degradation.

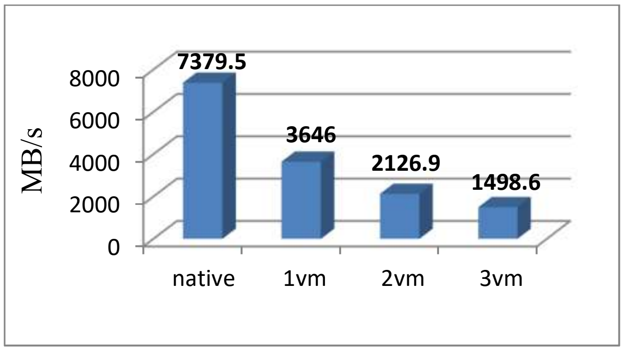


Fig. 6. Randomfileaccess.f workload test results

The characteristics of the randomfileaccess.f workload with five threads are that random reads dominate, while the sequential components are not present as shown on figure 6. We set up this workload so that cache would not be used. As we observe, the native host OS performs significantly better compared to one instance of ESXi virtual machine running. After starting second and third instances of ESXi virtual machines, we were able to observe that performance degradation is still present but it is not too significant.

The acquired benchmark results are fully expected and in line with the theoretical assumptions. The ESXi hypervisor and the hardware assisted full virtualization model show clear limitations on the data flow, in particular with the increase in the number of active virtual machines that cause even greater sharing of processors' resources and its increased use for hardware simulation. The addition of new instances of virtual machines is even more decreasing the achieved data flow, which means that new virtual machines cannot be added to the indefinite, as the performance of the whole system degrades per virtual machine added.

VIII. CONCLUSION

The introduction of virtualization has led to major changes in the use and deployment of information technology. Virtualization technology has a significant impact on reducing hardware investment as well as reducing operating costs, while also providing many additional benefits other than server consolidation. The great expansion of cloud computing in recent years has also contributed to the accelerated development of virtualization technologies and in the foreseeable future virtualization will always have an increased application in information technologies. It is also reasonable to expect, given the development of information technology today, that virtualization techniques will continue to improve and that the performance gap between virtualized systems and native systems will narrow in the future.

The results of our measurements showed that a native machine works convincingly better in most cases than a virtual machine based on ESXi hypervisor and full virtualization, as we assumed in our hypothesis of expected behaviour. Virtual machines running on the ESXi hypervisor have lower performance than a native machine, and in three of the four tests the performance degradation is approximately 50% when we have only one instance of a virtual machine running. The performance degradation is even greater with the introduction of more virtual machines. In one of the tests (mail server), the performance degradation between a native and a single virtual machine is not large, but with the introduction of new virtual machines into the test environment, the performance degradation of virtual machines becomes extremely pronounced. Future research may include a different approach where instead of comparing native machine vs. virtualized one, we compare different types of similarly structured virtualized machines or systems. With this research, we have proven that virtual systems still cannot reach the performance of non-virtualized systems, but as technologies evolve at an accelerated pace, we hope that in the future the performance of virtual machines will reach or be equal to regular non-virtualized machines.

ACKNOWLEDGEMENT

The work presented in this paper has partially been funded by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

LITERATURE

[1] C. Jiang, B. Luo, J. Wang, J. Zhang, Y. Wang, W. Shi, "Energy efficiency comparison of hypervisor," Proc. 2016 Seventh International Green and Sustainable Computing Conference *(IGSC)*, pp. 1-8, 2016.

[2] Correia, "Hypervisor based server virtualization" in *Encyclopaedia of information Science and Technology*, IGI Global, 2015, pp. 1182-1187.

[3] W. Huang, J. Liu, B. Abali, D. K. Panda, "A case for high performance computing with virtual machines," Proc. of the International Conference on Supercomputing. ACM, pp. 125-134, 2006.

[4] G. Casale, S. Kraft, D. Krishnamurthy, "A model of storage I/O performance interference in virtualized systems," Proc. of 31st International Conference on Distributed Computing Systems Workshops, pp. 34-39, 2011.

[5] J. Che, Q. He, Q. Gao, D. Huang, "Performance Measuring and Comparing of Virtual Machine Monitors," Proc. of 5th International Conf. Embedded and Ubiquitous Computing. EUC2008, Vol. 2, Piscataway, NJ, USA, pp. 381–386, 2008.

[6] A. Bhatia and G. Bhattal, "A comparative study of various hypervisors performance," International Journal of Scientific and Engineering Research, vol. 7, no. 12, pp. 65-71, 2016.

[7] J. Hwang, S. Zeng, F. Y. Wu, and T. Wood, "A component-based performance comparison of four hypervisors," Proc. of 2013 IFIP/IEEE International Symposium. IEEE, pp. 269-276, 2013.

[8] Pousa, Duarte; Rufino, José, "Evaluation of type-1 hypervisors on desktop-class virtualization hosts," IADIS International Journal on Computer Science and Information Systems. ISSN 1646-3692. 12:2, p. 86-101, 2017.

[9] H. Kazan, L. Perneel, M. Timmermann, "Benchmarking the performance of Microsoft Hyper-V server, VMWare ESXi and Xen hypervisors," Journal of Emerging Trends in Computing and Information Sciences, vol. 4, no. 12, pp. 922-933, 2013.

[10] Filebench project, Available: https://github.com/filebench/filebench/wiki

[11] VMware vSphere Documentation: https://docs.vmware.com/en/VMware-vSphere/index.html

[12] VMware ESXi 6.7 project: https://docs.vmware.com/en/VMware-vSphere/6.7/vsphere-esxi-vcenter-server-67-storage-guide.pdf.pdf

[13] VMware, Inc. white paper. "Virtualization Overview". https://www.vmware.com/pdf/virtualization_considerations.pdf

[14] VMware, Inc. white paper. "The Architecture of VMware ESXi," p. 3, 2020. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/ESXi_architecture.pdf

ESXi and Proxmox: FileSystem Performance Comparison for Type-1 Hypervisors

Borislav Đorđević, *Member, IEEE*, Valentina Timčenko, *Member, IEEE,* Nenad Nedeljković, Nikola Davidović

***Abstract* — This paper presents the comparison of two representatives of type 1 hypervisors: Proxmox VE and VMware ESXi. Hypervisor acts like a lightweight operating system and runs directly on the host's hardware. The measurements are carried out on the same server and under the equivalent conditions, with the Linux Ubuntu 20.10 as the guest operating system using the Filebench 1.5-alpha1 software. The goal of this paper is to show an impact of different number of virtual machines on the performances of various file system and highlight the best combination. The results have been illustrated in graphical form.**

Keywords* — *Virtualization; Hypervisor; Proxmox; ESXi; Filebench; virtual machine

I. Introduction

The virtualization is considered as one of the most important topics in IT. It allows a single computer/server to use multiple operating systems simultaneously. It also helps in reducing the costs, because they can run multiple different services on a single server, leading to more efficient server utilization, easier system maintenance, and reduced hardware. As the power of a computer unit has significantly increased since 1960s when the IBM's presented its visionary idea of virtualization, this solution became popular in system implementation and maintenance [1].

There are several approaches for virtualization in IT environments: hardware, software, desktop, data, network, memory, storage, etc. The hardware virtualization implies the use of a hypervisor, which is an additional layer that lies between hardware and operating system (OS) and makes a slight delay for when accessing the resources for virtualized environment, providing lower performances when compared to bare metal or non-virtualized system [2], [3].

Actually, hypervisor is specialized firmware and/or software installed on single hardware that allows hosting of the VMs.

There are two types of hypervisors (Figure 1): type 1, that is executed directly on hardware and manages guest OSs (ESXi, Proxmox); and type 2 that is executed on the host OS (VirtualBox, VMware Workstation).

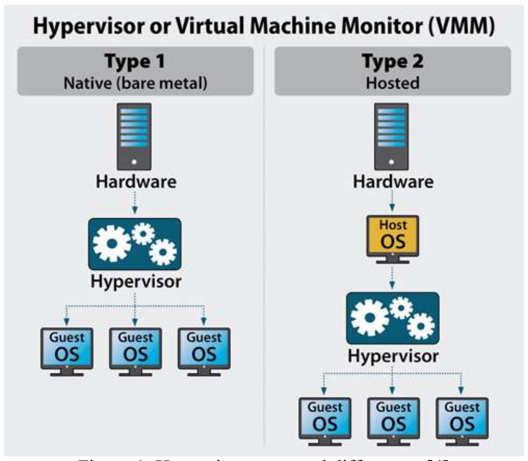


Figure 1. Hypervisor types and differences [4]

As the type 1 hypervisor has direct access to hardware, while type 2 hypervisor accesses hardware through host OS, we assume that type 1 hypervisor provides more scalability, reliability, and better performance [5].

II. Related Work, Objective and Motivation

This research is focused on the performance comparison of two type-1 hypervisors and results analysis. Since virtualization is the primary solution for systems ranging from small firms to large corporations, the arising question is: what is the best solution on the market? Some recent research addresses this issue from different perspectives, mostly considering VMware, KVM and Hyper-V hypervisors, and basing the results on Filebench or Bonnie++ [6]. This paper can provide a new picture of the situation since almost no research has focused on the Proxmox solution versus a commercial solution such as ESXi.

The primary goal of this paper is to compare performance using ESXi and Proxmox hypervisors on identical hardware, same VM parameters and the same guest OS – Linux Ubuntu 20.10 with ext4 as main file system (FS). Also, the disk we are testing has contained one of the three FSs: ext4, xfs or btrfs. Since we have used a Filebench workloads for testing, our idea was to find the best FS for each test. Selected workloads are: varmail, webserver and fileserver.

We have defined the mathematical model, measured the performances and interpreted the obtained results based on the mathematical model and hypotheses.

Borislav Đorđević - Institute Mihajlo Pupin, Volgina 15, 11000 Belgrade, Serbia, (borislav.djordjevic@pupin.rs)
Valentina Timčenko - Institute Mihajlo Pupin, Volgina 15, 11000 Belgrade, Serbia (valentina.timcenko@pupin.rs)
Nenad Nedeljković - VISER, School of Electrical and Computer Enginering of Applied Studies, Belgrade, Serbia (nedeljkovic1nenad@gmail.com)
Nikola Davidović – University of East Sarajevo, Faculty of Electrical Engineering, Vuka Karadzica 30, 71123 East Sarajevo, RS, BiH, (nikola.davidovic@etf.unssa.rs.ba)

III. Mathematical model

Variable T_W is calculated in accordance with the equation (1), and shows the total processing time for each workload.

$$T_W = T_{RR} + T_{SR} + T_{RW} + T_{SW} \tag{1}$$

Variables T_{RR} and T_{SR} represent random and sequential read time, and T_{RW} and T_{SW} random and sequential write time. There is an expected access time for every specific workload for the FS, which include following components:

$$T_{WORKLOAD} = T_{DIR} + T_{META} + T_{FL} + T_J + T_{HK} \tag{2}$$

$T_{WORKLOAD}$ represents the overall time for finishing all operations on the current workload, T_{DIR} the time needed to run all directory-related operations, T_{META} the time needed to complete all metadata operations, T_{FL} the time needed to go through all free lists operations, T_{FB} the time needed to carry out direct file blocks operations, T_J the time needed to complete journaling operations and T_{HK} the time needed to run housekeeping operation within the FS [7].

We have two candidates whose performances we compare:

1. Proxmox + VMs (guest OS).
2. ESXi + VMs (guest OS).

1. Proxmox + VMs (guest OS): The time to process the generated workload ($T_{W(Proxmox)}$) in this case depends on the benchmark interaction with guestOS FS, the characteristic of the FS, Virtual Hardware processing and the virtualization processing component of the Proxmox hypervisor (PVE-proc) is calculated in accordance to the following formula:

$$T_{W(Proxmox)} = f(BENCH,\ guestOS\text{-}FS,\ VH\text{-}proc,\ PVE\text{-}proc,\ hostOS\text{-}FS) \tag{3}$$

2. ESXi + VMs (guest OS): The time to process the generated workload ($T_{W(ESXi)}$) in this case depends on the benchmark interaction with guestOS FS, the characteristic of the FS, Virtual Hardware processing and the virtualization processing component of the ESXi hypervisor (ESXi-proc) is calculated in accordance with the following formula:

$$T_{W(ESXi)} = f(BENCH,\ guestOS\text{-}FS,\ VH\text{-}proc,\ ESXi\text{-}proc,\ hostOS\text{-}FS) \tag{4}$$

Since we are using the same settings for VMs on both hypervisors, the virtualization processing component will depend on the virtualization type and hypervisor processing as provided in the following formula:

$$PVE\text{-}proc = f\,(virt\_type,\ hyp\_proc) \tag{5}$$
$$ESXi\text{-}proc = f\,(virt\_type,\ hyp\_proc) \tag{6}$$

We are predicting the following:

- Based on the practical experience, it is expected that ESXi will produce better performance.
- Multiple VMs to have a significant performance drop compared to just one VM.

IV. File Systems

Linux FS is generally a built-in layer of a Linux OS used to handle the data management of the storage.

A. EXT4

The ext4 (fourth **ext**ended filesystem) is a journaling FS for Linux, and is developed as the extension of the ext3 [8]. It has the following characteristics [9]:

- Maximum FS size of up to 1 EB and maximum file size of nearly 16 TB.
- Hashed B-tree organizes and finds directory entries.
- Online defragmentation tool (e4defrag), which performs defragmentation of individual files or the whole FS.
- Easily detectable corruptions of files by metadata checksumming.

B. XFS

XFS is a high-performance journaling FS created by Silicon Graphics, Inc (SGI) in the last decade of 20th century [10]. It has the following characteristics [9]:

- Maximum FS size and maximum file size of nearly 8 EB.
- B+ tree organizes and finds directory entries.
- Delayed allocation for minimizing fragmentation and increasing performance.
- Implemented direct I/O for high throughput and non-cached I/O for DMA devices.

C. BTRFS

Btrfs ("better FS", "b-tree F S") is a copy-on-write (COW) FS based on B-trees. It was initially designed at Oracle Corporation in 2007 for the use in Linux [11]. It has the following characteristics [9]:

- Maximum FS size and maximum file size of nearly 16 EB.
- B-tree organizes and finds directory entries.
- Online defragmentation, offline FS check.
- Background based fixing errors on redundant files.

V. VMware ESXi and Proxmox

ESXi is an enterprise-class, type-1 hypervisor developed by VMware for deploying and serving VMs (Figure 2). It runs directly on hardware and significantly improves system performance [12]. The major part of architecture is VMkernel and processes that run on top of it. VMkernel has control of all hardware devices on server, manages resources and handles system processes. It receives requests from VMs for resources and presents the requests to the physical hardware [13].

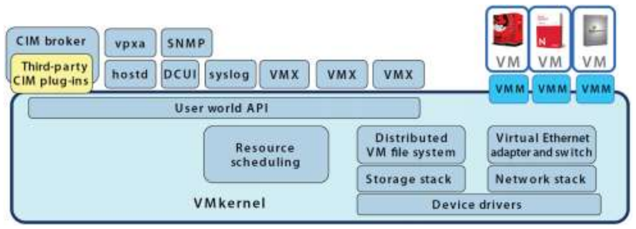


Figure 2. ESXi architecture [14]

The main processes that run on top of VMkernel are: [13]

• Direct Console User Interface (DCUI) — the low-level configuration and management interface, accessible through the console of the server, used primarily for initial basic configuration.

• The VM monitor, which is the process that provides the execution environment for a VM, as well as a helper process known as VMX. Each running VM has its own VMM and VMX process.

• Various agents used to enable high-level VMware Infrastructure management from remote applications.

• The Common Information Model (CIM) system: CIM is the interface that enables hardware-level management from remote applications via a set of standard APIs.

VMware uses VMFS. It is a special high-performance clustered FS. The main feature of this segment is ability to be shared by being simultaneously mounted on multiple servers. The VMFS datastore can be extended to span over several physical storage devices that include SAN LUNs and local storage. This feature allows you to pool storage and gives you flexibility in creating the datastore necessary for your virtual machines. [12]

Proxmox **V**irtual **E**nvironment – PVE (Figure 3) is a bare-metal hypervisor (runs directly on the hardware), to run VMs and containers. It is an open-source project, developed and maintained by Proxmox Server Solutions GmbH. For maximum flexibility, they implemented two virtualization technologies: full virtualization with KVM (Kernel-based Virtual Machine) and container-based virtualization (LXC) [15].

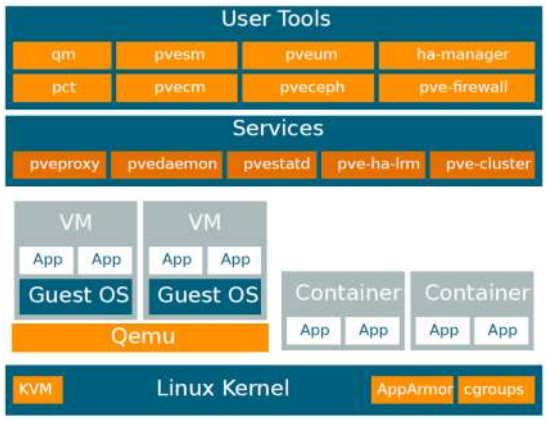


Figure 3. Proxmox architecture [15]

Proxmox uses a Linux kernel and is based on the Debian GNU/Linux Distribution. The source code is released under the GNU Affero General Public License, version 3. KVM was the first hypervisor to become part of the native Linux kernel (2.6.20). It is implemented as a kernel module, allowing Linux to become a hypervisor simply by loading a module. Benefits from the changes to the mainline version of Linux is optimization of hypervisor and the Linux guest Oss [16].

Proxmox natively supports running LXC (LinuX Containers) containers from the UI. These are similar to docker containers but behave more like a traditional VM.

Performance of KVM virtualization was the focus of this paper.

The main features for Proxmox VE [17]:

- Live migration;
- High availability;
- Scheduled backup;
- Command-line (CLI) tool;
- Flexible storage;
- OS template.

VI. Testing

The assumption of adequate testing is the application of a single hardware configuration, the same OS, and measurement methodology for all tests. The used server configuration has respectable hardware components although it is does not represent the latest technology.

The OS used is Ubuntu version 20.10, the latest instalment of Linux distribution (Table 1). During the installation process, we opted for minimal installation option which installs only essential packages and programs. The system disk uses EXT4 while the test disk is EXT4, XFS, or BTRFS.

All tests were performed using Filebench tool. Latest release of Filebench software was installed following instructions provided on the official GitHub repository of this project. Filebench is a program designed to measure the performance of FS and storage, and it can generate multiple workload types that simulate environments when using certain servers/services such as mail, web, file, database, etc. [18]. Before starting any tests, we made sure that all available updates were installed. Each VM was given 4 GB of RAM and 4 CPU cores.

TABLE I
Server test environment

HP ProLiant DL380 G7	
Component	***Characteristic***
CPU	2 x Intel Xeon E5540 QuadCore 2.53GHz
RAM	32GB DDR3
Storage Controllers	HP Smart Array P410i
Hard Drive 1	HP 10K SAS 146GB(DG0146)
Hard Drive 2	HP 7.2K SAS 500GB(MM0500)
PVE hostOS-FS	ext4
ESXi hostOS-FS	VMFS

The VM parameters are shown in Table 2. All used VMs have identical characteristics.

TABLE II
Virtual Machine Parameters

Component	***Characteristic***

vCPU	4
RAM	4GB
Disk	12GB + 32GB
OS	Linux Ubuntu 20.10
FS	ext4
Tested FS	ext4/xfs/btrfs

The focus of this paper is on measuring disk performance by comparing two hypervisors combined with three different FSs using 1, 2, or 3 VMs at the same time. It is expected that, as the number of VMs increases, performance will decline significantly in any combination.

Filebench is a very powerful and very flexible tool able to generate a variety of FS - and storage-based workloads. It implements a set of basic primitives like *create file*, *read file*, *mkdir*, *fsync* and uses WLM (the Workload Model Language - WML) to combine these primitives in complex workloads [18].

The files used for our benchmark were *varmail.f*, *webserver.f*, and *fileserver.f*. Those files are included in the Filebench software installation package, and were minimally edited to suit our needs.

The duration of each the tests was set to 120 seconds, which is the only change we made in *.f files with the goal of making the most realistic results. During the test execution, it was ensured that the impact of any external subject on system components was reduced to the minimum. The benchmark is run 3 times and the average value of the test is taken as final.

First, Proxmox VE was installed on server and nine VMs were generated, 3 for every FS. Tests were conducted in a way that one VM was first started and measured, then 2 and 3 VMs simultaneously. After that, disk is formatted and ESXi was installed. By the same principle, everything is applied to ESXi. From the generated data, the final conclusions were made by calculating the average values of the results.

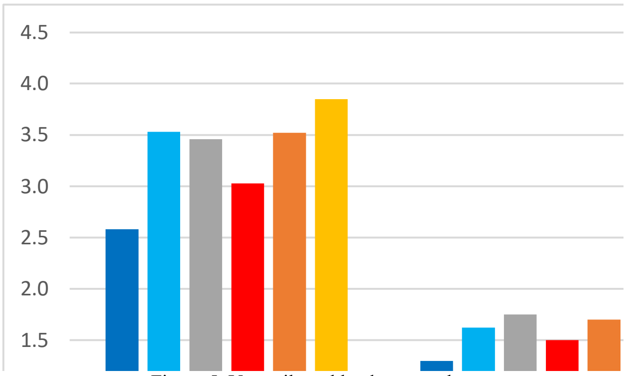


Figure 5. Varmail workload test results

TABLE III
BENCHMARK VARMAIL RESULTS

Varmail	1VM - (MB/s)	2VM - (MB/s)	3VM - (MB/s)
esxi - ext4	2.6	1.3	0.9
esxi - xfs	3.5	1.6	1.2
esxi - btrfs	3.5	1.8	1.0
pve - ext4	3.0	1.5	1.1
pve - xfs	3.5	1.7	1.2
pve - btrfs	3.9	1.9	1.5

Figure 5 and Table 3 show Varmail test results. Varmail emulates I/O activity of a simple mail server that stores each e-mail in a separate file (/var/mail/ server). The workload consists of a multi-threaded set of create-append-sync, read-append-sync, read and delete operations in a single directory. 16 threads are used by default [19].

For the Varmail workload, which is characterized by the dominant random reads and random writes, where random writes are represented by the synchronous transfers covered by equations (3) and (4), the main differences are components 3 (VH-proc), 4 (hypervisor-proc) and 5 (hostOS-FS).

When looking at the number of VMs, the combination of pve-btrfs was the best in each category, while esxi-btrfs and esxi-xfs had the same overall results with the ESXi hypervisor. We can conclude that the BTRFS FS is the best choice for a mail server.

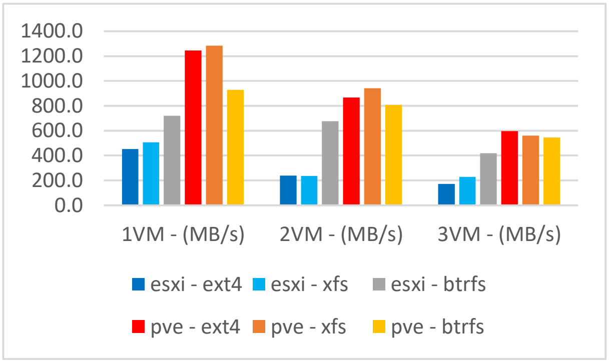


Figure 6. Webserver workload test results

TABLE IV
BENCHMARK WEBSERVER RESULTS

Webserver	1VM - (MB/s)	2VM - (MB/s)	3VM - (MB/s)
esxi - ext4	453.5	238.4	171.2
esxi - xfs	507.2	235.1	228.3
esxi - btrfs	720.0	677.5	419.1
pve - ext4	1243.9	864.9	595.0
pve - xfs	1284.1	940.5	561.8
pve - btrfs	928.8	808.0	544.7

Figure 6 and Table 4 show Webserver test results. Webserver emulates simple web-server I/O activity and produces a sequence of open-read-close on multiple files in a directory tree plus a log file append. 100 threads are used by default [19]. The Webserver workload is characterized by a dominant random read component as covered in equations (3) and (4), while the main differences are components 3 (VH-proc) and 5 (hostOS-FS). In both cases, VH-proc is Full-Hardware virtualization, but in Proxmox it is realized through QEMU. A large difference in performance in favor of Proxmox was observed in this test. The overall results of pve-xfs is 2.87 times better than esxi-xfs.

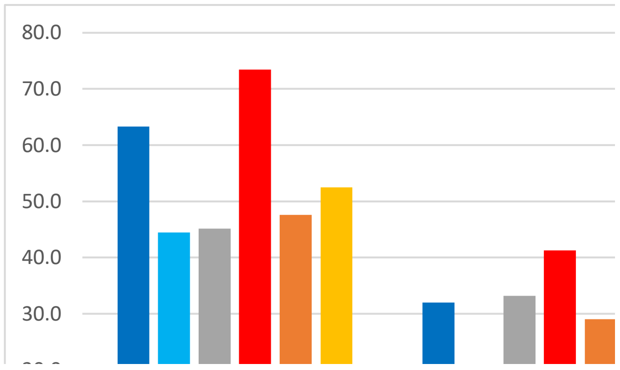


Figure 7. Fileserver workload test results

TABLE V
BENCHMARK FILESERVER RESULTS

Fileserver	1VM - (MB/s)	2VM - (MB/s)	3VM - (MB/s)
esxi - ext4	63.3	32.0	17.4
esxi - xfs	44.4	16.7	13.8
esxi - btrfs	45.1	33.2	10.7
pve - ext4	73.4	41.3	21.4
pve - xfs	47.6	29.0	18.3
pve - btrfs	52.5	26.4	20.4

Figure 7 and Table 5 show Fileserver test results. Fileserver - Emulates simple file-server I/O activity. This workload performs a sequence of creates, deletes, appends, reads, writes and attribute operations on a directory tree. 50 threads are used by default [19].

For the Fileserver workload, which is characterized by all kinds of data transfers, when considering equations (3) and (4), the main difference is component 5 (hostOS-FS). As ESXi uses VMFS, which is a clustered FS and represents a higher level of abstraction, while Proxmox uses EXT4, and the best FS in this test was EXT4, we conclude that this ruled in favor of Proxmox.

As in the previous two tests, this time too Proxmox came out as the winner but with a slightly smaller difference. We also have a match in the choice of FS: EXT4 gave the best overall results in both hypervisors.

VII. CONCLUSION

In this paper, we tested two respectable type 1 hypervisors: the commercial VMware ESXi solution and the open-source solution - Proxmox. Although it was expected that, due to its importance and big impact in the IT world, ESXi would provide better results, this did not happen. Proxmox won each comparator hypervisor + file system test. This was best seen during the webserver test where they were better almost 3 times and the third (VH-proc) and fifth (hostOS-FS) components of formulas (3) and (4) came to the fore.

If we only look at the performance of the FS, we get an interesting distribution. EXT4 performed best on fileserver, XFS on webserver, and BTRFS on varmail test.

For all 3 workloads we noticed that Proxmox is significantly better than ESXi. In the context of formulas (3), (4), (5), (6), we consider that the first two components, BENCH and guestOS-FS, in equations (3) and (4) have the same effect on for both hypervisors. The 3rd and 4th components, VH-proc, PVE-proc and ESXi-proc, differ significantly, where we notice that Proxmox is better. However, the main reason for Proxmox's victory is the 5th component (hostOS-FS). ESXi used a higher level of abstraction such as VMFS which slowed it down in this case, while Proxmox used a basic level of FS such as EXT4.

When we summarize all the test results, the used virtual machine operating system and hypervisors hostOS-FS, we can say that Proxmox is more optimized for Linux distribution.

The Proxmox virtualization system can be particulary useful for people starting their own business in small steps, without requiring additional costs. This does not mean that large companies do not use it. As already mentioned, this is an Open-Source solution and help for some of the possible problems can be found in a community where the number is unknown. If you still want to be insured, you can subscribe to the team of people behind this solution - Proxmox Server Solutions GmbH on more than favorable terms.

Interesting ideas for future work and research is to add fast Solid State Disks, comparative analysis of hypervisors using container virtualization or testing a different hypervisor such as Xen and Microsoft Hyper-V to determine which one achieves the best results.

ACKNOWLEDGMENT

The work presented in this paper has partially been funded by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

REFERENCES

[1] S. Meier, *IBM Systems Virtualization: Servers, Storage, and Software*, First edition, Redpaper, 2008.

[2] F. Bazargan, C. Yeun, J. Zemerly, “State-of-the-Art of Virtualization, its Security Threats and Deployment Models”, International Journal for Information Security Research. 3. 10.20533/ijisr.2042.4639.2013.0039, 2013.

[3] N. Yaqub, “Comparison of Virtualization Performance: VMware and KVM“, Master Thesis, Department of Informatics, Uviversity of Oslo, Norway, 2012.

[4] C. Taylor, 2020, What is a Hypervisor Server?, accessed 14 May 2021, https://www.serverwatch.com/virtualization/hypervisor-server/

[5] P. Vasconcelos, F. Araújo, G. Freitas, T. Marques, “KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems”, International Journal of Multimedia and Image Processing. 6. 10.20533/ijmip.2042.4647.2016.0039, 2016.

[6] B. Đorđević, N. Maček, V. Timčenko, “Performance Issues in Cloud Computing: KVM Hypervisor’s Cache Modes Evaluation”, Vol. 12, No. 4, pp 147-165, 2015. http://uni-obuda.hu/journal/Dordevic_Macek_Timcenko_60.pdf,

[7] D. Vojnak, B. Đorđević, V. Timčenko, S. Štrbac, “Performance Comparison of the type-2 hypervisor VirtualBox and VMWare

Workstation", 27th Telecommunications Forum (TELFOR), Belgrade, Serbia, pp. 1-4, doi: 10.1109/TELFOR48224.2019.8971213, 2019.

[8] THE EXT4 FILE SYSTEM, accessed 16 May 2021, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/storage_administration_guide/ch-ext4

[9] D. Vujičić, D. Marković, B. Đorđević, S. Ranđić, "Benchmarking Performance of EXT4, XFS and BTRFS as Guest File Systems Under Linux Environment", 2016.

[10] THE XFS FILE SYSTEM, accessed 16 May 2021, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/storage_administration_guide/ch-xfsBtrfs

[11] O. Rodeh, J. Bacik, C. Mason, (2013). "BTRFS: The linux B-tree filesystem", ACM Transactions on Storage (TOS). 9.10.1145/2501620.2501623, 2013.

[12] E. Sosa, *Mastering VMware NSX® for vSphere®*, First edition, Indianapolis, Indiana, John Wiley & Sons, 2020.

[13] VMware, "The Architecture of VMware ESXi", Technical White Paper, 2007.

[14] B. Laurent, 2020, VMs Everywhere, accessed 24 May 2021, https://littlecorner.info/post/virtualization/

[15] Proxmox Server Solutions Gmbh, "Proxmox VE Administration Guide", 2021.

[16] S. Aiiy, "Comparative analysis of proxmox VE and xenserver as type 1 open source based hypervisors", International Journal of Scientific and Technology Research. 7. 72-77, 2018.

[17] S. Cheng, *Proxmox High Availability*, First edition, Birmingham, England, Packt Publishing, 2014.

[18] V. Tarasov, E. Zadok, S. Shepler, "Filebench: A Flexibile Framework for File System Benchmarking", Vol. 41, No. 1, 2016.

[19] Filebench, 2017, accessed 23 April 2021, https://github.com/filebench/filebench/wiki

Snort IDS system visualization interface

Nadja Gavrilovic, Vladimir Ciric, Nikola Lozo
University of Nis, Faculty of Electronic Engineering, Nis, Serbia

***Abstract*—Over the past decades, the rapid Internet development and the growth in the number of its users have raised various security issues. Despite numerous available security tools, the exchange of data over the Internet is becoming increasingly insecure. For this reason, it is of great importance to ensure the security of the network in order to enable the safe exchange of confidential d ata, a s w ell a s t heir i ntegrity. O ne o f t he most important components of network attack detection is an Intrusion Detection System (IDS). Snort IDS is a widely used intrusion detection system, which logs alerts after detecting potentially dangerous network packets. The next step in successful network protection is the analysis of logged alerts in search of deviations from normal traffic t hat m ay i ndicate a n i ntrusion. T he g oal of this paper is to design and implement a visualization interface that graphically presents alerts generated by Snort IDS, classifies them according to the most important attack parameters, and allows the users to easily detect possible traffic i rregularities. An environment in which the system has been tested in real-time is described, and the results of attack detection and classification are given. One of the detected attacks is analyzed in detail, as well as the method of its detection and its possible consequences.**

***Index Terms*—IDS, snort, network intrusion detection, visualization interface**

Nadja Gavrilovic is with the Faculty of Electronic Engineering, University of Nis, Aleksandra Medvedeva 14, Nis, Serbia (e-mail:nadja.gavrilovic@elfak.ni.ac.rs).
Vladimir Ciric is with the Faculty of Electronic Engineering, University of Nis, Aleksandra Medvedeva 14, Nis, Serbia (e-mail:vladimir.ciric@elfak.ni.ac.rs).
Nikola Lozo is with the Faculty of Electronic Engineering, University of Nis, Aleksandra Medvedeva 14, Nis, Serbia (e-mail:nikolalozo@elfak.rs).

Performance comparison of homomorphic encryption scheme implementations

Goran Đorđević, *AET Europe The Netherlands, ETF Beograd, Milan Marković, Panevropski Univerzitet Apeiron Banja Luka, Pavle V. Vuletić, ETF Beograd*

***Abstract* — Homomorphic Encryption allows third party to receive encrypted data and perform arbitrarily computations on that data while it remains encrypted, despite not having the secret decryption key. This enables many new secure applications in cloud environments. For a long time, a key issue with the homomorphic encryption was its low performance which made it unusable in production environments. Advances in the last ten years in the field of homomorphic encryption resulted in several new schemes and software libraries which implement them. These homomorphic schemes have improved performance, but there is still a question whether the improvements would justify their use in production environments. In this paper we evaluated features and performances of several new homomorphic encryption mechanisms: BGV, BFV and CKKS.**

***Keywords* — Homomorphic Encryption; Performance; Secure Multiparty Computation.**

I. Introduction

Homomorphic encryption allows computations on ciphertext without the knowledge of the secret key, or more precisely it allows performing computations on the encrypted data, without decrypting them [1]. Homomorphic encryption allows a third party (e.g., cloud, service provider) to perform certain computable functions on the encrypted data while preserving the features of the function and format of the encrypted data and without being able to see its content. Homomorphism of the first asymmetric encryption algorithms (RSA) over some mathematical operations (e.g. multiplication) was known since these algorithms were invented almost fifty years ago. Such schemes which support partial set of mathematical operations are known as partially homomorphic. Cryptographic mechanisms that support arbitrary level of computations on ciphertext (multiplication, addition, rotation) without the knowledge of the secret keys are known as Fully Homomorphic Encryption (FHE) systems. The increased popularity of cloud-based services on one side and the need to preserve data privacy led to the new interest in homomorphic encryption research which would enable secure multiparty computation in the cloud environment. One could imagine the use of AI or machine learning algorithms on the data which is encrypted and invisible to the AI system provider, thus preserving data privacy only for the data owner. An example of such a scenario where homomorphic encryption mechanisms are deployed is given in Figure 1. In this example the user sends and stores the data in the encrypted form on the cloud server. The data is processed on the server in the encrypted form, and the results which remain in the encrypted form are sent back to the user who can decrypt the data and use the result.

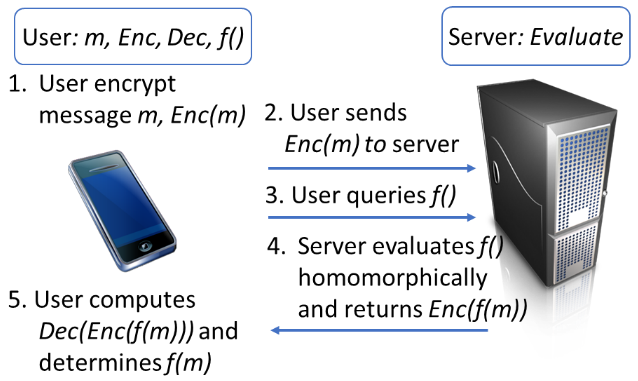

User: *m, Enc, Dec, f()*
Server: *Evaluate*
1. User encrypt message *m, Enc(m)*
2. User sends *Enc(m)* to server
3. User queries *f()*
4. Server evaluates *f()* homomorphically and returns *Enc(f(m))*
5. User computes *Dec(Enc(f(m)))* and determines *f(m)*

Fig. 1. An example of client-server HE scenario

The biggest obstacle for the use of homomorphic encryption schemes was the fact that there were no FHE mechanisms which had reasonable performance. Computations were by several orders of magnitude slower than the operation on unencrypted data which made any such solution too resource expensive. However, in the last ten years a breakthrough happened in the area and a set of new homomorphic encryption schemes emerged. Modern fully homomorphic encryption schemes use complex algorithms on lattice structures and Ring-LWE (Ring Learning With Errors) mechanism [2]. In addition to the homomorphic property, it is believed that these algorithms are resistant to quantum computer attacks because nowadays there are no known algorithms that would use the properties of quantum computers to break these algorithms in polynomial time. Following the appearance of new fully homomorphic encryption schemes, a set of programming APIs and libraries which implement different schemes emerged as well. In this paper we are assessing the set of capabilities and performance of three homomorphic encryption schemes (BGV, BFV, CKKS) and are discussing the suitability and constraints of these schemes for use in the cloud-based environments for secure multiparty computations. Performance assessment of the new FHE schemes has not been explored a lot in the literature. We believe that this paper will provide a better insight into the current state of the work on FHE and its suitability for real use case scenario deployments.

The paper is organized as follows. Section II gives an overview of the related work in the field of the performance evaluation of the FHE schemes. The most important properties of homomorphic encryption and the classification of the homomorphic encryption schemes are presented in Section III. The main features and description of modern HE schemes: BGV [3], BVF [4] and CKKS [5] are elaborated in Section IV. In Section V is shown results of experimental analysis. Conclusions are given in Section VI.

II. RELATED WORK

Related work about the FHE schemes is spread across the papers in the relevant sections, while this section contains only those papers which were dedicated to FHE performance evaluation. Experimental results related to BGV scheme with value of ciphertext modulus q=130 are given in [1]. Viand et al. in [6] compare the features of Palisade, Microsoft SEAL, and HELib homomorphic encryption libraries. In addition, this paper gives statistical compiler tests of BVF scheme implemented in the SEAL library in a graphical form without presenting precise numerical values. Melchor et al. [7] compared the performance of three libraries HELib, SEAL and FV-NFLlib for large plaintext moduli of up to 2048 bits. Finally, Lepoint et al. [8] compare the performance of two older homomorphic schemes. Unlike the previous work, in this paper we give experimental results for BGV with broader range of values ciphertext modulus q and results for other modern homomorphic schemes: BFV and CKKS that are not covered in [1].

III. PROPERTIES OF HOMOMORPHIC ENCRYPTION

There are four main types of homomorphic schemes [1]:

- Partially Homomorphic Encryption (PHE). The PHE scheme enables either any number of addition or any number of multiplication operations over encrypted data.
- Somewhat Homomorphic Encryption (SHE) allows both addition and multiplication, but it can perform a limited number of operations. “Somewhat” means it works for some functions *f*.
- Fully Homomorphic Encryption. The scheme allows any number of addition or multiplication operations. “Fully” means it works for all functions *f*. An FHE scheme can evaluate unbounded depth.
- Levelled Homomorphic Encryptions (LHE). This scheme can evaluate arbitrary polynomial-size circuits.

Homomorphic Encryption should support two main homomorphic operations:

- Additive Homomorphic Encryption;
- Multiplicative Homomorphic Encryption.

Homomorphic encryption is additive, if [9]:

$$\text{Enc}\ (m_1 + m_2) = \text{Enc}\ (m_1) + \text{Enc}\ (m_2);\ \forall m_1, m_2 \in M.$$

Homomorphic encryption is multiplicative, if [9]:

$$\text{Enc}\ (m_1 * m_2) = \text{Enc}\ (m_1) * \text{Enc}\ (m_2);\ \forall m_1, m_2 \in M.$$

The most popular classes of homomorphic schemes are (given with their main properties):

- Boolean circuit (Fastest Homomorphic Encryption in the West (FHEW) [10] and Fast Fully Homomorphic Encryption over the Torus (TFHE) [11]):
  - Plaintext data are coded as bits;
  - Computations are performed by using Boolean circuits.
- Modular integer arithmetic (BGV, BFV):
  - Plaintext data are coded as integer modulo a plaintext;
  - Computations are expressed as integer modulo arithmetic.
- Approximate number arithmetic (CKKS):
  - Plaintext data are coded as real (or complex) numbers;
  - Computations are performed in a way similar to floating-point arithmetic but dealing with fixed-point numbers.

Modern HE mechanisms are based on usage of lattice cryptography with errors LWE [12]. Lattices have an important role in modern cryptography, especially in the context of the research on post-quantum cryptography. It is known that the factoring problem which was discovered to be solvable in polynomial time on a quantum computer by Shor can be applied to the widely used asymmetric cryptographic schemes (RSA, DH). At the moment of writing this paper there was no report in the literature which claimed that it can break lattice-based cryptographic algorithms using quantum computer algorithms.

The newest HE algorithms are applied structured lattices i.e. Ring-LWE mechanism [2]. The Ring-LWE reduces key length and computation time. The ring implementation is based on power-of-two cyclotomic rings:

$$\text{R}_q = \mathbb{Z}_q\ /\ \langle \text{x}^n + 1\rangle$$

The optimized Residue Number System (RNS) variants of algorithms show significant performance gain compared to their earlier respective implementations [13]. The RNS works with native (machine-word size) integers because it is faster than multi-precision integer arithmetic. It breaks rings of large bit-width integers into a parallel set of rings (<64-bit residues) allowing very efficient computation on 64-bit CPU architecture.

Large modulus q is represented as product of integers:

$$q = \prod_{i=1}^{k} qi$$

Modulus q is a functional parameter that determines how many computations are allowed without the appliance bootstrapping procedure [14].

One of the properties of the homomorphic encryption schemes is that they add noise to a ciphertext in the encryption process. Homomorphic operations (especially multiplication) increase the noise. If the noise becomes too large, the resultant ciphertext can become undecryptable. Noise budget is the total amount of noise that can be added until the decryption fails [15]. The bootstrapping is the procedure of "refreshing" a ciphertext by running the decryption function on it homomorphically, resulting in a reduced noise.

All considered homomorphic encryption schemes support the following homomorphic operations:

- Addition;
- Multiplication;
- Rotation.

IV. Homomorphic Schemes

The BGV scheme was proposed [3]. BGV is a levelled HE scheme, meaning that the parameters of the scheme depend on the multiplicative depth that the scheme is capable to evaluate. Multiplicative depth determines how many sequential multiplications can be performed.

The BFV scheme [4] is a homomorphic cryptographic scheme based on the Ring-LWE problem in a lattice.

The CKKS scheme [5] is known as Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAAN). Supported operations in the scheme are shown in Figure 2. The CKKS scheme enables computations on vectors of complex values.

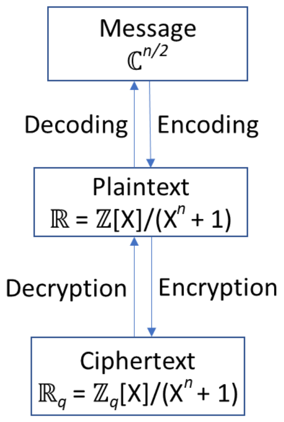

Message
$\mathbb{C}^{n/2}$
Decoding
Encoding
Plaintext
$\mathbb{R} = \mathbb{Z}[X]/(X^n + 1)$
Decryption
Encryption
Ciphertext
$\mathbb{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$

Fig. 2. Operations in CKKS

The CKKS is an approximate homomorphic encryption scheme with the following features:

- Dec (Enc(m)) ≈ m;
- Dec (ct_1 * ct_2) ≈ Dec (ct_1) * Dec (ct_2);
- Noise bounds are determined by the parameter set.

In the CKKS scheme noise is considered as a part of numerical error in approximate computation. It supports homomorphic rounding-off.

In all above-mentioned schemes the following homomorphic operations are implemented [16]:

- Public key encryption:
  $$PubEncrypt(pk, M) \rightarrow C$$
  The public encryption algorithm takes as input the public key (pk) of the scheme and any message M from the message space. The algorithm outputs a ciphertext C.
- Decryption:
  $$Decrypt(sk, C) \rightarrow M$$
  The decryption algorithm takes as input the secret key of the scheme (sk), and a ciphertext C. It outputs a message M from the message space.
- Homomorphic addition:
  $$EvalAdd(Params, ek, C_1, C_2) \rightarrow C_3$$
  EvalAdd is an algorithm that takes as input the system parameters *Params*, the evaluation key (ek), two ciphertexts C_1 and C_2, and outputs a ciphertext C_3.
- Homomorphic multiplication:
  $$EvalMult(Params, ek, C_1, C_2) \rightarrow C_3$$
  EvalMult is an algorithm that takes as input the system parameters *Params*, the evaluation key ek, two ciphertexts C_1 and C_2, and outputs a ciphertext C_3.

The evaluation key is needed to perform homomorphic operations over the ciphertexts. The evaluation key is used in in the following homomorphic operations: relinearization (multiplication) and rotation. Any entity that has only the evaluation key cannot learn anything about the messages from the ciphertexts only [16].

An example of homomorphic encryption with asymmetric key cryptography by using BGV [3], BVF [4], and CKKS [5] schemes is shown in Figure 3.

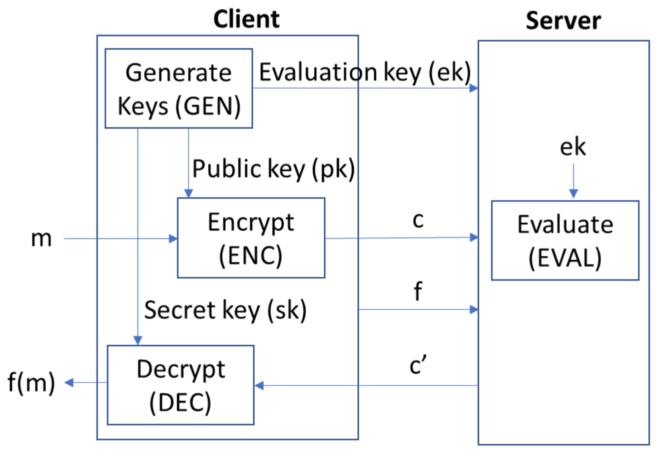


Fig. 3. Homomorphic encryption with asymmetric keys

V. Experimental Analysis

In the experimental analysis we evaluated the time needed for execution of the following homomorphic operations: Public key encryption (Table II), Decryption (Table III), Homomorphic addition (Figure 4), and Homomorphic multiplication (Figure 5). Homomorphic encryption libraries implement the above-mentioned cryptographic operations of a scheme and expose a higher-level API. We evaluated the use of the following homomorphic schemes:

- BGV,
- BVF and
- CKKS;

that are implemented in the following open-source libraries respectively:

- Microsoft SEAL [17];
- Palisade [14];
- HELib [18] [19].

HELib is a C++ open source library that implements both the BGV [3] and CKKS [5] homomorphic encryption schemes. HELib library, published in 2013 by Halevi and Shoup, was the first homomorphic encryption library.

Palisade [14] is multi-threaded library written in C++ 11. It uses the NTL library [20] to accelerate underlying mathematical operations. Palisade supports more schemes, including BFV, BGV, CKKS. It also supports multi-party extensions of certain schemes and other cryptographic primitives like Proxy Re-Encryption (PRE) and digital signatures [6].

Microsoft Simple Encrypted Arithmetic Library (SEAL) [17] is a homomorphic encryption library that allows additions and multiplications to be performed on encrypted integers or real numbers. Microsoft SEAL is written in C++11 and contains a .NET wrapper library for the public API. The latest available version 3.6.2 is developed in C++17.

Table I gives an overview of the publicly available open-source libraries with implemented HE algorithms. Palisade implements Boolean circuits Fully Homomorphic Encryption (FHE) schemes: FHEW and TFHE. In the FHE mechanisms it uses bootstrapping procedure [14] (noise refreshing procedure) with the application of the appropriate bootstrapping keys. The FHEW and TFHE schemes are not implemented in the HELib and Microsoft SEAL libraries.

TABLE I
HE ALGORITHMS IN OPEN-SOURCE LIBRARIES

Library/ HE scheme	Palisade	HELib	SEAL
BGV	√	√	
BFV	√		√
CKKS	√	√	√
FHEW	√		
Threshold FHE	√		

The homomorphic encryption code was executed on a PC with:

- 2194.84 MHz 8-core CPU;
- 16 GB RAM;
- Ubuntu 20.04 LTS.

Tables II and III and Figures 4 and 5 show the results of encryption, decryption, HE addition and HE multiplication tests respectively, where:

- Times in the last three columns (HE Library) are expressed in microsecond (μs);
- Each operation was executed 1000 times and the times presented are the times to execute 1000 iterations;
- We used 128-bit homomorphic encryption security level;
- Ciphertext dimension is n;
- Ciphertext modulus is q.

Ciphertext dimension n shall be chosen on basis of desired security level and value of ciphertext modulus q. If ciphertext modulus q is bigger than noise budget it enables implementation more complex homomorphic evaluation function f i.e. implementation the function with bigger depth.

Palisade library implements modular arithmetic schemes: BGV and BVF with 128-bit security level beginning from ciphertext dimension $n = 2048$.

The public key encryption operation in BFV scheme has the best performance when the SEAL library is used. Performance difference depends on the ciphertext dimension: while the SEAL encryption is three times faster for the ciphertext dimension of 2048, when the ciphertext dimension is 32768, this factor is 1.3 times. The encryption operation has the best performance in BGV scheme when the Palisade library is used. Performance difference ratio decreases with the increase of the ciphertext dimension. The encryption operation in CKKS scheme for ciphertext dimension $n \geq 8192$ has the best performance when the HELib library is used, whereas in case of lower dimension n the best results are achieved by using SEAL library.

TABLE II
PUBLIC KEY ENCRYPTION

HE scheme	HE parameters		HE library		
	n	$\log_2 q$	Palisade	HELib	SEAL
BFV	1,024	27	-	-	272
BGV	1,024	27	-	1,783	-
CKKS	1,024	27	585	482	257
BFV	2,048	54	1,557	-	506
BGV	2,048	54	1,560	3,608	-
CKKS	2,048	54	1,173	997	479
BFV	4,096	109	3,519	-	1,687
BGV	4,096	109	3,493	7,833	-
CKKS	4,096	109	2,753	2,288	1,926
BFV	8,192	218	7,773	-	4,838
BGV	8,192	218	8,116	17,817	-
CKKS	8,192	218	7,538	4,664	5,688
BFV	16,384	438	24,050	-	16,252
BGV	16,384	438	25,926	44,796	-
CKKS	16,384	438	23,183	12,581	19,344
BFV	32,768	881	77,553	-	59,457
BGV	32,768	881	78,639	109,340	-
CKKS	32,768	881	76,406	39,890	71,373

TABLE III
SECRET KEY DECRYPTION

HE scheme	HE parameters		HE library		
	n	$\log_2 q$	Palisade	HELib	SEAL
BFV	1,024	27	-	-	63
BGV	1,024	27	-	13,047	-
CKKS	1,024	27	415	3,159	10
BFV	2,048	54	159	-	127
BGV	2,048	54	133	49,096	-
CKKS	2,048	54	809	5,104	19
BFV	4,096	109	420	-	416
BGV	4,096	109	353	192,351	-
CKKS	4,096	109	1,432	14,279	72
BFV	8,192	218	940	-	1,484
BGV	8,192	218	1,012	763,178	-
CKKS	8,192	218	6,038	48,960	290
BFV	16,384	438	2,370	-	5,904
BGV	16,384	438	3,690	3,033,690	-
CKKS	16,384	438	13,776	183,254	1,166
BFV	32,768	881	7,330	-	24,919
BGV	32,768	881	14,941	12,003,497	-
CKKS	32,768	881	51,960	701,913	4,826

The decryption operation in CKKS scheme has the best performance by using SEAL library. The decryption operation in CKKS scheme when using SEAL is approximately 10 times faster than when Palisade is used and more than 100 times faster than when HELib is used.

The secret key decryption operation in BGV scheme performs better by several orders of magnitude in the Palisade than in the HELib library.

The decryption operation in BFV scheme for ciphertext dimension $n \geq 8192$ has better performance when Palisade library is used, whereas in case of lower dimension n better results are achieved by using SEAL library.

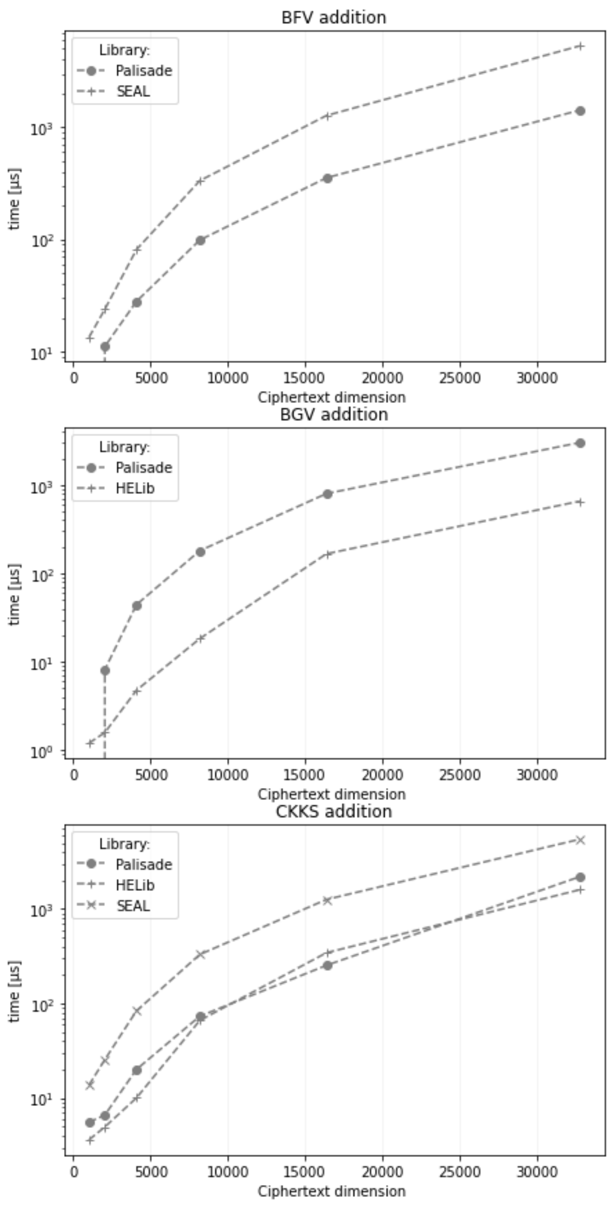


Fig. 4. Homomorphic encryption – addition operation time

The ciphertext addition in CKKS scheme has the best performance in the HELib library. The ciphertext addition in CKKS scheme has better performance in the Palisade than in the SEAL library, but the differences are generally smaller than for the decryption operation.

The ciphertext addition in BFV scheme has significantly better performance (more than 2 times faster) in the Palisade than in the SEAL library.

The ciphertext addition in BGV scheme has significantly better performance (more than 4 times faster) in the Palisade than in the SEAL library.

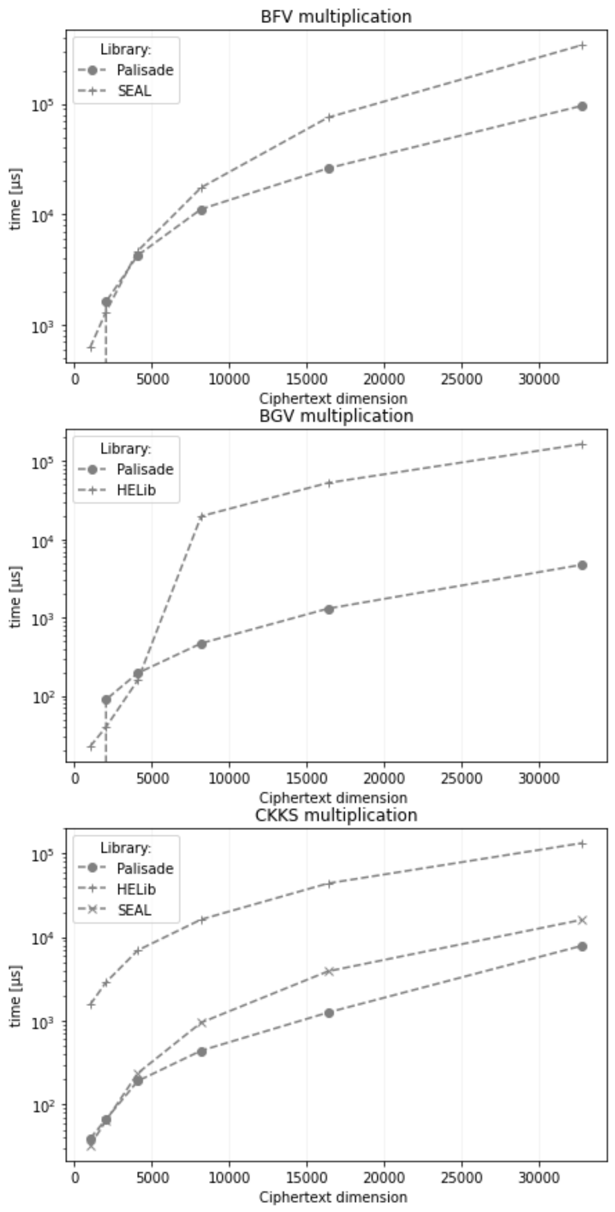


Fig. 5. Homomorphic encryption – multiplication operation time

The ciphertext multiplication is much more complex and more time consuming than ciphertext addition. Figure 4 presents the time needed for performing homomorphic multiplication without relinearization procedure.

The cyphertext multiplication in CKKS scheme for ciphertext dimension $n \geq 8192$ has the best performance when implemented in the Palisade library whereas in case of lower dimension n the better results are achieved using SEAL library.

The cyphertext multiplication in BGV scheme for ciphertext dimension $n \geq 8192$ has significantly better performance (more than 3 times faster) when implemented in the Palisade library than in the HELib whereas for lower ciphertext dimensions better results are achieved by using HELib library.

The cyphertext multiplication in BFV scheme for ciphertext dimension $n \geq 4096$ has better performance in the Palisade than in the SEAL whereas for lower ciphertext dimensions slightly better results are achieved by using SEAL library.

In addition, we compared execution time of homomorphic operations with no security level versus operations with 128-bit security level. We have measured execution time of homomorphic operations in CKKS (approximate arithmetic) and BGV (integer modulo arithmetic) schemes that are implemented in the Palisade library.

In the experiments we have got similar ratio of results for both schemes, so we present only results related to CKKS scheme.

In the tests we have performed homomorphic operations by using following scenarios:

1. No security level with ciphertext dimension n=512;
2. 128-bit security level with ciphertext dimension n=32768.

Each operation was executed 1000 times. In both scenarios it is used same value of ciphertext modulus q.

We have got following results of homomorphic operations (CKKS scheme):

- Public key encryption operation is about 69 times faster in scenario 1;
- Private key decryption operation is about 46 times faster in scenario 1;
- Homomorphic addition operation is about 45 times faster in scenario 1;
- Homomorphic multiplication operation is about 48 times faster in scenario 1.

VI. Conclusions

Homomorphic encryption allows performing computations on the encrypted data, without decrypting them. The paper compares the time needed to execute homomorphic operations, like, public key encryption, secret key decryption, addition and multiplication implemented in the open-source libraries: Microsoft SEAL, Palisade, and HELib. The operations are compared for BGV, BFV and CKKS homomorphic encryption schemes implemented in the libraries.

Homomorphic operations that are performed at client side: public key encryption and secret key decryption if it is used BGV scheme (integer arithmetic) have the best performance when using methods that are implemented Palisade.

Homomorphic operations that are performed at the server side: addition and multiplication are fastest when Palisade library is used for all three tested schemes, except for BGV addition and higher ciphertext dimensions in which cases HELib has slightly better performance.

Execution time of homomorphic operations with no security level versus operations with 128-bit security level was performed and showed that all the operations are still by two orders of magnitude slower than when no security is used which presents an issue when complex machine learning or AI calculations are required.

The performance of current fully homomorphic encryption schemes, especially for large parameters, can still be improved. Further improvement can be achieved by implementation low-level homomorphic operations in an assembly language which is executed on a hardware platform. Also it can be achieved better performance if homomorphic operations are implemented in hardware platforms like Graphics Processing Unit (GPU), Application-Specific Integrated Circuit (ASIC), and Field-Programmable Gate Array (FPGA).

Literature

[1] A. Acar, H. Aksu, A. Selcuk, and M. Conti, "A Survey on Homomorphic Encryption Schemes: Theory and Implementation," ACM Comput. Surv. 1, 1, Article 1, http://dx.doi.org/10.1145/3214303, 2018.
[2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," Journal of the ACM (JACM) 60, no. 6, 2013.
[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully Homomorphic Encryption without Bootstrapping," Cryptology ePrint Archive, Report 2011/277. https://eprint.iacr.org/2011/277, 2011.
[4] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," IACR Cryptology ePrint Archive, 2012:144, 2012.
[5] J.H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," Cryptology ePrint Archive, Report 2016/421, https://eprint.iacr.org/2016/421, 2016.
[6] A. Viand, P. Jattke, A. Hithnawi, "SoK: Fully Homomorphic Encryption Compilers", IEEE Symposium on Security and Privacy 2021.
[7] C. Aguilar Melchor, M. Kilijian, C. Lefebvre, T. Ricosset, "A Comparison of the Homomorphic Encryption Libraries HElib, SEAL and FV-NFLlib," in: Lanet JL., Toma C. (eds) Innovative Security Solutions for Information Technology and Communications. SECITC 2018. Lecture Notes in Computer Science, vol 11359. Springer, Cham. https://doi.org/10.1007/978-3-030-12942-2_32, 2019.
[8] T. Lepoint, M. Naehrig, "A Comparison of the Homomorphic Encryption Schemes FV and YASHE," in: Pointcheval D., Vergnaud D. (eds) Progress in Cryptology – AFRICACRYPT 2014. AFRICACRYPT 2014. Lecture Notes in Computer Science, vol 8469. Springer, Cham. https://doi.org/10.1007/978-3-319-06734-6_20, 2014.
[9] T. Maha, S. Hajji, and A. Ghazi, "Homomorphic encryption applied to the cloud computing security," in Proceedings of the World Congress Engineering, vol. 1, pp. 4-6, 2012.
[10] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in E. Oswald and M. Fischlin, editors, Advances in Cryptology – EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, So_a, Bulgaria, April 26-30, 2015, Proceedings, Part I, volume 9056 of Lecture Notes in Computer Science, pages 617-640. Springer, 2015.
[11] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster packed homomorphic operations and e_cient circuit bootstrapping for tfhe," in Advances in Cryptology-ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security, pages 377-408. Springer, 2017.
[12] O. Regev, "The learning with errors problem," in Blavatnik School of Computer Science, Tel Aviv University Invited survey in CCC, 2010.
[13] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," Cryptology ePrint Archive,Report 2018/931, https://eprint.iacr.org/2018/931, 2018.
[14] Y. Polyakov, K. Rohloff, G.W. Ryan, and D. Cousins, "PALISADE Lattice Cryptography Library User Manual (v1.10.6)", 2020.
[15] S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya, "A Review of Homomorphic Encryption Libraries for Secure Computation," http://arxiv.org/abs/1812.024, 2018.
[16] M. Albrecht, M. Chase, H. Chen and others, "Homomorphic encryption standardization," homomorphicencryption.org, 2018.
[17] K. Laine, "Simple Encrypted Arithmetic Library 2.3.1," 2017.
[18] S. Halevi and V. Shoup, "Algorithms in Helib," in Advances in Cryptology – CRYPTO 2014, J. A. Garay and R. Gennaro, Eds, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 554–571, 2014.
[19] S.Halevi, V. Shoup, "HElib design principles," 2020.
[20] V. Shoup and others, "NTL: A library for doing number theory," http://www.shoup.net/ntl.

Comparison of Message Queue Technologies for Highly Available Microservices in IoT

Marko Milosavljević, Milica Matić, Neven Jović, Marija Antić

Abstract— **Internet of Things (IoT) solutions connect large numbers of devices, which generate various data and control messages asynchronously. In the IoT system cloud, these messages need to be queued in order to control the processing load and prevent the overload in cases of traffic bursts. On the other hand, one of the requirements the IoT cloud needs to fulfill is the high availability. Therefore, multiple instances of services accepting and processing the messages generated by the devices are needed. There are various message queue technologies available today, but they all have their limitations. In this paper, we compare the performance of Apache Kafka and RabbitMQ in the scenario of the highly available IoT cloud data processing.**

Index Terms— **message queue; high availability; load balancing; internet of things.**

I. INTRODUCTION

In the past decade, the world is witnessing the expansion of Internet of Things (IoT) solutions. Within IoT systems, different devices are connected to perform a certain function together. IoT use-cases are various, such as smart transport, smart fabrics, smart cities, smart homes, etc.

In order to collaborate, the devices need to be able to exchange data such as commands and state change reports. Although the expansion of IoT has led to the development of technologies such as ZigBee, Z-Wave, WiFi or Bluetooth Low Energy, which enabled the connection of many different actuators and sensors into large local mesh networks, in order for an IoT solution to achieve its purpose, the existence of the cloud component is also needed. The cloud allows remote control and monitoring of the local networks, but it can also provide advanced features which require processing of larger quantities of historical system data, or the interaction with components responsible for customer management, software update and third-party services.

As the data from the IoT system is generated asynchronously [1], and processing it requires a certain amount of time, mechanisms are needed to control the cloud load. Usually, this control is achieved by deploying various message queueing systems, that allow to communicate between different components of the cloud, and react to messages generated by the end devices [2]. Message queuing technologies which are available today differ in terms of the performance guarantees they offer, and depending on the actual use-case, metrics such as latency, disk space, RAM memory or processor usage may be a limiting factor [2], [3]. The comparison of Kafka and Apache Pulsar has been performed by the authors in [4], and it has been shown that, although Apache Pulsar may achieve better results in terms of resource usage, the maturity of the solution, available documentation, and possibility to integrate with other data processing tools, may be a reason to favor Kafka in the commercial deployment scenarios. On the other hand, Kafka and RabbitMQ have been compared in [5], to show that RabbitMQ has its advantages in terms of the achieved throughput on a single server instance, but the scaling options are on Kafka's side.

In this paper, we explore the possibility of replacing the already implemented RabbitMQ message queueing within the smart home system cloud [6],[7], with Apache Kafka. Within the deployed smart home cloud, messages generated by end devices are processed by multiple cloud services. As the number of supported features is growing, so is the number of the cloud services that process these messages. Also, some of the messages need to be processed by multiple of these services. Additionally, as the number of users grows, the system needs to be scaled up, and, as already said, Kafka has its advantages in this domain. The paper is organized as follows: in Section II, the elements of smart home system and its cloud architecture are introduced, then the overview of RabbitMQ and Kafka is given in Section III and Section IV. Finally, the performed tests and their results are presented in Section V.

II. SMART HOME CLOUD DATA BUFFERING

In the existing smart home solution, the end devices within the household use technologies such as ZigBee, Z-Wave and ONVIF/IP to connect to the home gateway – Fig. 1. The gateway is responsible to execute the core system logic: it implements the middleware which represents all of the devices in the same way, regardless of the communication technology they use in the local network, and allows them to work together, according to the automation rules set up by the user. To communicate with the user applications and cloud backend, the gateway uses MQTT protocol. MQTT conveys commands issued by the user, system control messages, and reports about device state changes. Control messages are processed on the cloud side, for the purpose of system

Marko Milosavljević is with OBLO Living, Novi Sad, Narodnog fronta 21a, Serbia (e-mail: marko.a.milosavljevic@ obloliving.com).

Milica Matić is with the Faculty of Technical Sciences, University of Novi Sad, Serbia (e-mail: milica.matic@rt-rk.uns.ac.rs).

Neven Jović is with OBLO Living, Novi Sad, Narodnog fronta 21a, Serbia (e-mail: neven.jovic@ obloliving.com).

Marija Antić is with the Faculty of Technical Sciences, University of Novi Sad, Serbia (e-mail: marija.antic@rt-rk.uns.ac.rs).

administration, upgrade, backup and restore. Also, reports about device state changes are stored to provide user with the information about the history of system usage [7].

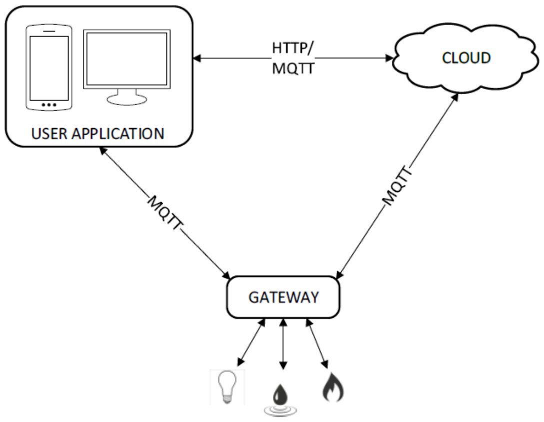


Fig. 1. Smart home system components and communication between them.

The observed smart home cloud system solution has the microservice-based architecture. It is highly available (HA), which means that the entire system is fault tolerant, i.e. that there are multiple instances of every microservice running [6]. In order to prevent problems with MQTT messages processing due to the overload of cloud system, or the failure of some instances, temporary data buffering is necessary. In the temporary data buffering module all important messages are first queued, allowing relevant microservices to process them at their own pace.

In the current implementation, RabbitMQ is used for the purpose of data buffering. The incoming MQTT messages are parsed by the B2Q (Broker to Queue) microservice, and directed to the appropriate RabbitMQ queues, based on the information they contain. All of the instances of one cloud microservice share the load of processing the messages from the RabbitMQ queue they are associated with. The problem here represents the fact that if one message needs to be processed in multiple ways (i.e. it is relevant as the input for multiple cloud microservices), it has to be replicated to multiple queues. Therefore, in this paper we explore the possibility of replacing RabbitMQ with Apache Kafka. We implement the B2K (Broker to Kafka) microservice, which publishes messages to Kafka queues, that the processing microservices are subscribed to, and we compare the performance of the two implementations.

A. *RabbitMQ*

RabbitMQ is a message queue manager, which has originally implemented the Advanced Message Queuing Protocol (AMQP). Later it was extended to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queue Telemetry Transport (MQTT), and other protocols, but AMQP remains the default and the most widely used one.

RabbitMQ messages can convey any kind of information, from a simple text message to a message with information about processes important for the system. Message broker stores the message into the queue, until the application fetches it for processing. Message queuing allows web servers to avoid the overload, as they can control the number of the messages that are processed simultaneously. It is also useful for distributing messages to multiple consumers sharing the load and providing fault tolerance.

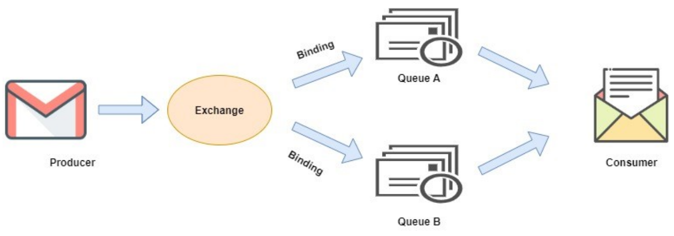


Fig. 2. RabbitMQ message delivery mechanism.

Producer applications create the messages, but the messages are not published directly to a queue. First, the producer sends the message to the RabbitMQ exchange running on the broker – Fig. 2. The exchange is responsible for routing the messages to different queues, based on the configured bindings and routing keys. Four types of exchanges exist - direct, topic, fanout and headers exchange. In the direct exchange, the message is routed to the queue whose binding key matches the routing key of the message. The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding. The fanout exchange routes messages to all of the queues bound to it. The headers exchange uses the message header attributes for routing. Consumers subscribe to the queues and process the messages from them. All consumers subscribed to the same queue will share the load of processing the messages from that queue. The messages are deleted from the queue after processing.

B. *Apache Kafka*

Apache Kafka is an event streaming platform. It is elastic, distributed, highly scalable and fault-tolerant. Similar to RabbitMQ, Kafka has the client and server side. Kafka clients and servers communicate using TCP protocol.

Kafka implements the publish/subscribe mechanism, and allows processing streams of events as they arrive into the system or retrospectively, but also allow to store streams of events as long as they are needed.

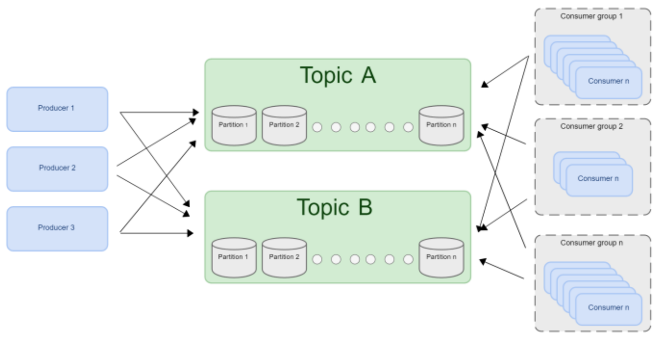


Fig. 3. Kafka message processing mechanism.

Similar to RabbitMQ, the Apache Kafka clients can act as producers and consumers – Fig. 3. Producers represent client applications that write (publish) events to Kafka. On the other hand, consumers are subscribing to topics, reading and writing events. Producers and consumers are not aware of each other. They work completely independently, and that is a key design to achieve high scalability. Therefore, producers will never need to wait for consumers.

When data is written to Kafka, it is written in the form of an event containing the key, value, timestamp and optional metadata. Events are stored in topics. The durability of events inside Kafka's topic is configurable. Unlike RabbitMQ, Kafka events can be read whenever they are needed, because events are not deleted after consumptions. Events can be stored as long as needed. Storing data for a long time does not affect Kafka.

Topics in Kafka are partitioned, and one Kafka topic can have any number of partitions defined in the Kafka configuration file. Events are ordered inside the partition in the exactly same order as they were written, and one consumer can process data from one partition only. However, the data stored in one partition can be processed by multiple consumers belonging to different consumer groups, i.e. one message can be processed multiple times, without the need to duplicate it. Offset is an integer number that is used to maintain the current position of a consumer inside partition. Every topic can be replicated, so that there are dozens of brokers that have a copy of data. This makes data fault-tolerant and highly-available.

III. Testing and Results

Tests were designed to measure CPU load of smart home system servers when RabbitMQ and Apache Kafka are used for data buffering. RabbitMQ and Kafka brokers were run on the 8-core Intel i7 processor with 8 GB of RAM memory.

TABLE I
RabbitMQ test results

Setup	**CPU usage on 8 cores [%]**		
	average	**maximum**	**deviation**
16 producers 8 queues 0 consumers	324	640	108
16 producers 8 queues 8 consumers	410	794	197
16 producers 8 queues 16 consumers	486	800	167
16 producers 8 queues 32 consumers	553	800	147

To test the RabbitMQ buffering, 16 producer B2Q processes were created, that published messages to 8 queues. The messages from these queues were processed by a variable number of consumers (0, 8, 16, 32). Producers were configured to publish messages every 1 ms. Test results are presented in Table I.

RabbitMQ reached CPU limit after 16 consumers, but was able to continue working stably, while the setup with 32 consumers stopped working after ten minutes. The throughput of the system was approximately 11000 messages per second. Maximum CPU usage was 800%, i.e. all eight cores were used 100%.

To test Kafka performance, 16 producers were created, which published to the variable number of partitions (32, 64, 128). Since Kafka allows only one consumer per partition, the number of consumers was also varied from 0 to 128. Test results are presented in Table II.

In any of test cases limit of Kafka maximum CPU load was not reached. It can be observed that the CPU usage deviation is smaller than in RabbitMQ case. Therefore, the server stays stable, even as the number of messages that are stored in Kafka increases with time.

TABLE II
Kafka test results

Setup	**CPU usage on 8 cores [%]**		
	average	**maximum**	**deviation**
16 producers 32 partitions 0 consumers	210	573	65
16 producers 32 partitions 32 onsumers	202	347	43
16 producers 64 partitions 0 consumers	186	473	90
16 producers 64 partitions 64 consumers	208	360	37
16 producers 128 partitions 0 consumers	150	300	93
16 producers 128 partitions 128 consumers	480	553	53

IV. Conclusion

This paper gave a brief description of some of the message queueing technologies that can be used for flow control and load balancing in the IoT scenario. RabbitMQ and Apache Kafka were deployed within the smart home system cloud, and their performance was tested for a variable number of consumers.

The presented test results indicate that data buffering in Kafka is highly stable and has the lower average CPU usage. At any point of testing, maximum CPU usage was never reached. Therefore, in our further work we will focus on integrating Kafka in the data collection and storage module of the smart home system. Using Kafka will allow us to process the same messages multiple times, without the need to duplicate data. This, in turn, opens the possibility to create advanced data processing scenarios which may bring added value to the users of the smart home system.

V. Acknowledgment

This research (paper) has been supported by the Ministry of Education, Science and Technological Development through the project no. 451-03-68/2020-14/200156: “Innovative scientific and artistic research from the FTS (activity) domain”.

References

[1] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev and P. E. Heegaard, “Modeling of Aggregated IoT Traffic and Its Application to an IoT Cloud,” *Proceedings of the IEEE*, vol. 107, no. 4, pp. 679-694, April 2019

[2] G. Fu, Y. Zhang and G. Yu, “A Fair Comparison of Message Queuing Systems,” *IEEE Access*, vol. 9, pp. 421-432, Jan. 2021

[3] H. Wu, Z. Shang and K. Wolter, “Performance Prediction for the Apache Kafka Messaging System,” *Proc. of IEEE HPCC/SmartCity/DSS*, Aug. 2019

[4] S. Intorruk and T. Numnonda, “A Comparative Study on Performance and Resource Utilization of Real-time Distributed Messaging Systems for Big Data,” *Proc. of IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, July 2019

[5] P. Dobbelaere and K. S. Esmaili, “Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper,” *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*, June 2017

[6] M. Matić, E. Nan, M. Antić, S. Ivanović and R. Pavlović, “Model-Based Load Testing in the IoT System,” *Proc. of International Conference on Consumer Electronics (ICCE-Berlin)*, Sept. 2019

[7] S. Ivanović, M. Antić, I. Papp, N. Jović, “Data Acquisition, Collection and Storage in Smart Home Solutions,” *Proc. of 6th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN)*, May 2019

Design of a Network Topology Using CISCO NSO Orchestrator

Mioljub Jovanovic
Wireless Communications Research Group
University of Westminster
London, UK.
Cisco Systems, Diegem, Belgium
m.jovanovic@my.westminster.ac.uk
mjovanov@gmail.com

Milan Cabarkapa
Department of Telecommunications
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia.
cabmilan@etf.rs

Djuradj Budimir
Wireless Communications Research Group
University of Westminster
London, UK.
School of Electrical Engineeing
Univrsity of Belgrade, Serbia.
d.budimir@wmin.ac.uk
d.budimir@etf.rs

***Abstract*— This paper presents the design of a network topology using CISCO NSO orchestrator. The mismatch problem solution between a network service and its monitoring is proposed. Applying the proposed approach, the telemetry efficiency ratio parameter greater than 40 is achieved. All tests are performed in the real experimental conditions using CISCO NSO orchestrator.**

Keywords—intent based network; intent-aware monitoring agent; model-driven telemetry; service assurance

I. Introduction

NFV orchestrators (e.g., Tacker [1], Cloudify [2], ONAP [3], CISCO NSO [4]) are a crucial part for the dynamic and optimal management and orchestration of various virtualized network resources (e.g., VMs, Virtualized Network Functions). 5G technology, empowered by NFV and SDN, presents a new dimension of complexity that must be addressed by service assurance [5].

Using these orchestration software having higher level of abstraction, the rapid connectivity and provisioning could be achieved at lower prices while letting to operators possibility to build, arrange and preserve network service [6], [7].

Communication Service Provider (CSP) networks – such as Virtual Evolved Packet Core are subject to very dynamic configuration change. Provisioning, modification and termination of packet data services are being done in rapid pace in order to keep up with dynamic environment needs and cater to main business drivers, such as IoT, Video etc. SDN technologies using Network Slicing approach are foundation for such a dynamic environment, allowing automated and programmatic configuration of network services [5].

Traditionally network services are being monitored by deployment of probes which generate traffic and provide feedback on the status of the service. Due to such rapid changes in network service configuration there is open question in regard to monitoring and assuring provisioned services: What is the right approach to take in order to monitor the network which constantly changes? How to ensure network service is operational and carefully selecting probes to monitor network service? [5], [8].

Monitoring using active probes face challenges such as introduction of synthetized traffic within the data flow, end to end monitoring only with no understanding of the data path, lack of comprehension of the configuration intent etc [9]-[11]. Generated traffic using probes should resemble real traffic of the network service, however even with almost perfect synthetized traffic, there is substantial possibility that real network service traffic could be impacted, but probe does not detect such a problem since probe is not part of the actual real data flow [12]. Therefore, there is a gap in regard to monitoring and assurance of the actual network service data flow, with all network elements data traverses on the path between endpoints.

We are proposing solution based on Intent Based Networking (IBN). The proposed approach consists by:

- Extraction of configuration intent by analysing of the network service configuration.
- Discovery of the network elements along the network service data path.
- Leveraging existing network monitoring capabilities of network elements, along with probes and Model Driven Telemetry (MDT) to get more accurate information on the status of desired Network Service.

Research methodology used in understanding benefits of proposed monitoring approach involves qualitative approach, comparative analysis of existing - probe based monitoring and proposed solution based on Intent Based Networking (IBN). By using the proposed approach, we have achieved higher than 40 for the telemetry efficiency ratio parameter.

II. Intent Based Network Methodology

Role of Intent Based Network is transforming Business Intents into configuration changes. As depicted in Fig. 1, Intent at high level represents one or set of different requirements which describe service or network.

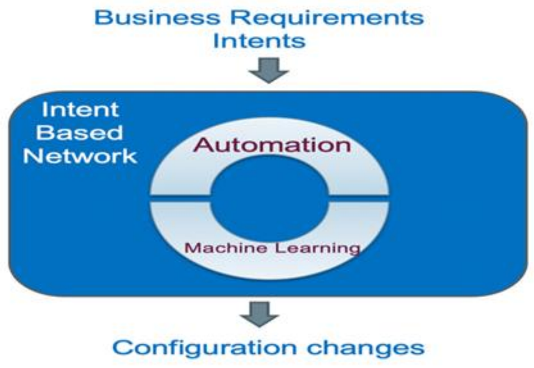


Fig. 1. Intent Based Network – high level description

Those requirements are then being analysed by set of steps, processes or algorithms in order to convert/render high-level requirements into lower form of abstraction, which could then be used to configure computer network elements in order to enable needed service. As business intent is being transformed into configuration on devices it's important to enable monitoring of the network services in order to have understanding whether desired service is operational and functioning in accordance to the business requirements – key performance indicators (KPIs). As graphically demonstrated in Fig. 2, traditionally in legacy network such monitoring would mean enabling monitoring on different data points including but not limited to: SNMP, Netflow/SFlow, telemetry and even Command Line Interface outputs (CLI). Acquiring data from different sources would certainly improve visibility on the state of the network, yet it would greatly impact efficiency and would aplify amount of telemetry data transferred over the network, but without providing clear answer on whether the intent has been fulfilled and whether network service is running and operational as per pre-defined KPIs [13].

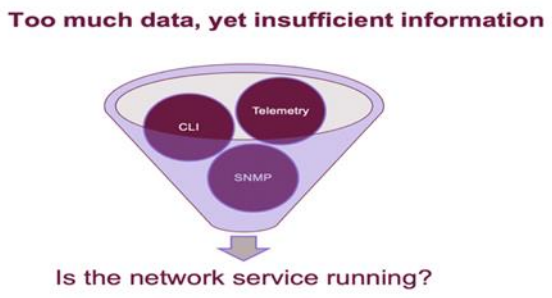

Too much data, yet insufficient information
CLI
Telemetry
SNMP
Is the network service running?

Fig. 2. Main query is - Is the Network Service running according to the pre-defined KPIs?

III. Experimental Setup

Experimental setup consists of the following routers: Simulated customer premises routers (CE), provider core routers (P) and provider edge routers (PE). Fig.3, shows the network with service models which is configured using the orchestration network architecture.

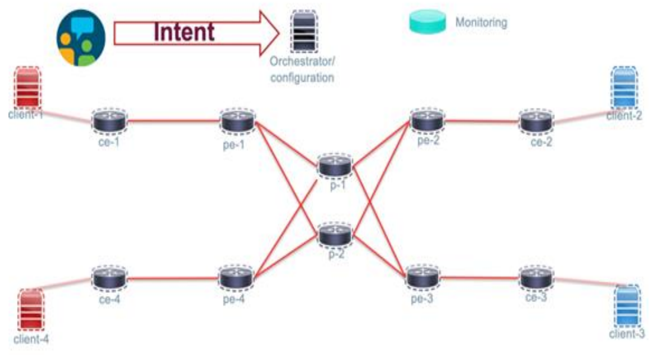


Fig. 3. Business Intent communicated to the orchestrator

In Fig. 4 orchestrator is configuring devices in order to fulfil desired service intent. Orchestrator uses Netconf protocol to access and configure network elements which are taking part in the data path to enable desired service.

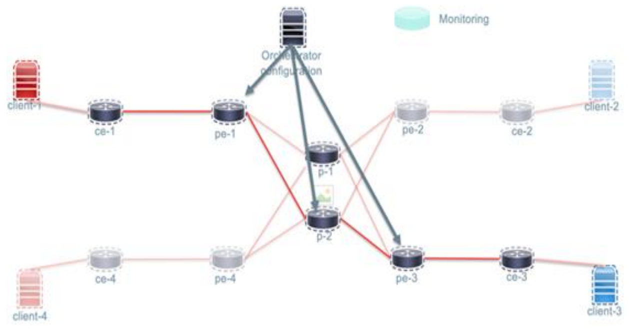


Fig. 4. Orchestrator sends configuration to network devices

In the provided example, actual intent is to establish communication – tunnel service between ce-1 and ce-3 network device in order to enable communication between Client-1 and Client-3. In order to traverse path between Client-1 and Client-3, data packets need to cross pe-1, p-2 and pe-3 as shortest path between the endpoints. Of course, this trajectory may be different in function of routing protocols and connectivity in function of time, but topology discovery and update events will be discussed in future work. At this time, we are focusing on the fixed path through the experimental network and assuming there would not be topology changes throughout the experiment shown in Fig. 5.

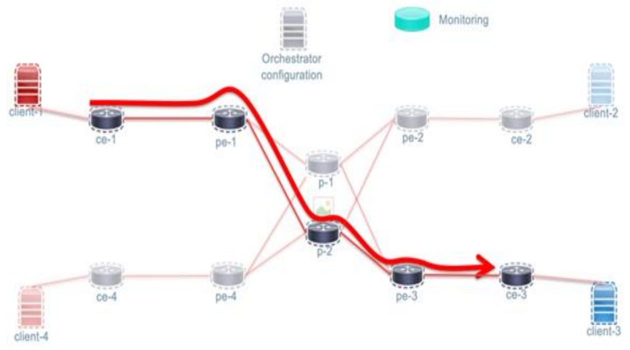


Fig. 5. Service is configured. Question: Service running within acceptable KPIs? Question: Is configuration model mapped to monitoring model?

In Fig. 6 we can observe each of the network devices streaming telemetry data to the collector, monitoring platform which is receiving and processing all telemetry data.

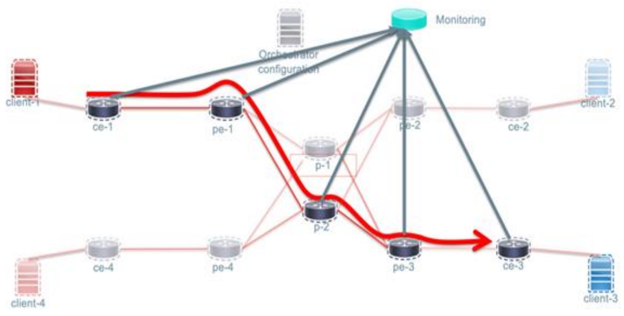


Fig. 6. Telemetry data streamed to Monitoring/Analytics platform. 250000 different stats per router (740 kbps of data)

Thanks to the fact involved network elements are already using Model Driven Telemetry processing data points by collector is simpler. However, as there are so many different data points which are being monitored on devices, there may be information overload since on average router there could easily be 250000 different monitored data points. Such as large number of collected data points could essentially mean that amount of generated telemetry data may be significantly high and could pose challenge for network infrastructure as well as could cause impact to collector processing capacity.

Instead of monitoring all relevant and non-relevant data points, causing unnecessary increase of traffic and compute resources to process large amount of data, we're proposing significant reduction in amount of telemetry data by ensuring that only minimal set of relevant data points is exported from the network devices by means of intent-aware monitoring agent (IAMA). Data reduction task is accomplished by deploying IAMA locally to the network devices, thus leveraging local area network (LAN) links and avoiding use of wide-are links (WAN) for large amount of data points. IAMA is aware of the service details and is also capable of receiving telemetry data. As represented on IAMA architecture in Fig. 7, service intent is received by from the orchestrator while MDT is received from network devices.

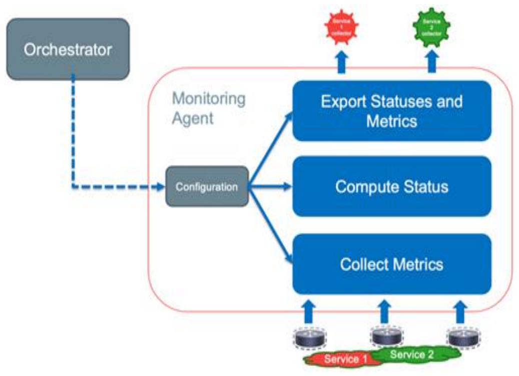


Fig. 7. Intent-aware monitoring agent architecture

IAMA is performing analysis on the received datasets and series of computations in order to determine actual state of the service. Steps performed by IAMA: collecting MDT, processing and exporting reduced – yet more relevant MDT is called IAMA pipeline. Final result of IAMA pipeline is significantly reduced amount of MDT containing only high-level status of the monitored service, as per pre-defined Key-Performance Indicators (KPIs).

IV. Results

Measuring objective was to determine how much data is actually received via MDT under usual telemetry export, with typical data points for router such as environmental, interface stats etc. Result of this work outlines amount of measured data after performing analysis of the incoming telemetry and mapping to service aware MDT. All routers and all incoming data points were taken into account.

TABLE I. Experimental Results

	Intent-Aware Monitoring Efficiency			
	Total MB	**Rate 1 min in kbps**	**Rate 5 min in kbps**	**Rate 15 min in kbps**
Incoming from routers	5200	740.7	700.1	711.7
This work	130.8	17.3	17.2	17.1
Outgoing to Analytics platform	224.9	29.5	29.1	29.3
This work efficiency ratio	40.9	42.8	40.6	41.7

As outlined in Table I, demonstrated experimental results have reduced the amount of incoming MDT from routers from 5.2 GB to 130 MB, while preserving relevant information which is – is service running and operational per pre-defined KPIs.

V. Conclusion

The design of a network topology using CISCO NSO orchestrator has been presented in this paper. The solution about mismatch problem between a network service and its monitoring has been proposed. The telemetry efficiency ratio parameter of more than 40 has been achieved. The amount of telemetry data has been reduced by injecting service aware information in MDT and removing all overhead MDT data points which do not need to be exposed to the network operator who is monitoring the service. Of course, full MDT can also be enabled if desired.

References

[1] https://docs.openstack.org/tacker/latest/

[2] https://cloudify.co/

[3] https://www.onap.org/

[4] https://www.cisco.com/c/en/us/solutions/service-provider/solutions-cloud-providers/network-services-orchestrator-solutions.html

[5] Anil Rao, "Reimagining service assurance for NFV, SDN and 5G", White paper, Analysis Mason, 2018.

[6] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and Orchestration Challenges in Network Functions Virtualization," IEEE Communications Magazine,vol. 54, no. 1, pp. 98–105, Jan 2016.

[7] A. J. Gonzalez, G. Nencioni, A. Kamisiski, B. E. Helvik, and P. E. Heegaard, "Dependability of the NFV Orchestrator: State of the Art and Research Challenges," IEEE Communications Surveys Tutorials, pp. 1–23, 2018.

[8] M. Pattaranantakul, R. He, Z. Zhang, A. Meddahi and P. Wang, "Leveraging Network Functions Virtualization Orchestrators to Achieve Software-Defined Access Control in the Clouds," in IEEE

Transactions on Dependable and Secure Computing, pp. 1-14, Nov. 2018.

[9] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia and P. Casas, "A Survey on Big Data for Network Traffic Monitoring and Analysis," in IEEE Transactions on Network and Service Management, vol. 16, no. 3, pp. 800-813, Sept. 2019.

[10] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities. J. Internet Serv. Appl., 9(16), 2018.

[11] Cisco Systems, Inc, "GitHub Network Telemetry Pipeline," Cisco Systems, Inc, 2017. [Online]. Available: https://github.com/cisco/bigmuddy-network-telemetry-pipeline

[12] M. Jovanović, M. Čabarkapa, B. Clause, N. Nešković, M. Prokin, B. Đurađ, Model driven telemetry using Yang for next generation network applications, 5th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN) 2018, pp. 1186 - 1189, Palić, Serbia, June, 2018.

[13] B. Claise, J. Clarke, and J. Lindblad "Network Programmability with YANG: The Structure of Network Automation with YANG, NETCONF, RESTCONF, and gNMI", Addison-Wesley Book, 1st edition, 2019.

Visualization of microscopic morphological characteristics used for determination of infectious molds

Mina Milanović, Aleksandar Milosavljević and Marina Ranđelović

***Abstract*—Invasive fungal infections (IFI) and systemic fungal infections (SFI), caused by molds are on the rise, based on data from literature. Diagnostics of those infections can sometimes be inefficient; they require a longer period of time in laboratory procedures and sometimes may lead to late diagnosis or misdiagnosis, which can result in patient's critical condition or even mortality. The goal of this research is to develop a neural network model that will perform identification of molds, and thus accelerate the process of diagnostics. A classifier has been developed, using an EfficientNet-B1 deep convolutional neural network (CNN) and sample images obtained at the Department of Microbiology and Immunology, Medical faculty, University of Niš, Serbia, archives. We applied Grad-CAM visualization to determine morphological characteristics used by the model to classify samples.**

***Index Terms*—molds identification, fungal infection, convolutional neural networks, deep learning, Grad-CAM.**

I. Introduction

Ability of fungus to start a pathological process in the host organism is as a specific phenomenon, according to numerous authors, because, excluding groups of dermatophyte molds and tropical fungi, these microorganisms does not need pathogenicity for their dissemination and survival in nature [1]. Among 400.000 species of fungi known in the nature, around 50 kinds can cause invasive fungal infections(IFI), that are characterized by very high morbidity (serious clinical case) and mortality. Numerous reasons have contributed to the increase of number of infections among humans, and incidences of IFI caused by molds are constantly growing. The most important reasons are complex procedures and medical interventions, intensive treatments with antibacterial drugs, cytostatics, immunosuppressants; longer lifespan of a humans, increase in the number of patients at high risk due to primary diseases and treatment, the appearance of resistance in fungi and certainly the establishment of mycological analyzes and higher diagnostic efficiency, i.e. more successful diagnostic procedures in a microbiology [2].

Mina Milanović – Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: mina.milanovic@elfak.rs).

Aleksandar Milosavljević – Faculty of Electronic Engineering, University od Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: aleksandar.milosavljevic@elfak.ni.ac.rs).

Marina Ranđelović - Department of Microbiology and Immunology, Medical faculty, University of Niš, Blvd Zorana Djindjica 81, 18000 Niš, Serbia (e-mail: marina87nis@gmail.com).

Fungi are eukaryotic microorganisms. In nature, they are widespread, living in soil and water, on organic materials as saprophytes, as symbionts or parasites of animals, plants, or human [3]. Based on the structure, all fungi can be primarily divided into yeasts -unicellular fungi with basal cell blastoconidia (blastospora) and multicellular fungi (molds) with a basic hypha cell. Molds classification is performed on the basis of structure, i.e. macroscopic and microscopic morphological characteristics. Differences between the morphology of molds, hypha structure, production of different conidiae (spores), enable a diagnostic procedure for their identification [4].

This goal of this project is to develop a neural network model that will perform identification of molds, and thus accelerate the process of diagnostics. No similar projects, involving determination of molds or their morphological characteristics, could been found during our research, and beside some rapid tests that can be used only for most common types of infections, whole process of determination is manual and sometimes takes days, so providing an application that can accelerate the process can be very beneficial. During recent years, number of infections caused by more rare species of fungi has drastically increased, which was a motivation for a project like this, which includes classification of so far neglected types of fungi.

Sample collection has contained high resolution images, which needed manual preparation for training, as described in chapter II. Prepared dataset was expanded before training and EfficiencyNet-B1 architecture of convolutional neural network (CNN) has been used for developing and training the model, which makes the core of the classifier, as presented in chapter III. Results and discussion of the results, including visualization of decision making process using Grad-CAM method, have been shown in chapter IV. Conclusion and planned further steps have been described in chapter V.

II. DATA

A. *Dataset description*

Fungi, based on morphology are classified in group of yeasts -unicellular fungi with basal cell blastoconidia (blastospora) and multicellular fungi (molds) with a basic hypha cell. Molds can be primarily divided into dermatophytic and non-dermatophytic fungi [5].

Dermatophytic molds, in other words, dermatophytes, are causative agents of superficial fungal infection of skin, hair and nail with prevalence of 22-25% worldwide.

Other group of molds caused invasive fungal infection (IFI) and in recent years incidence of these diseases has been on the rise [6].

Diagnostics of infections caused by dermatophytic and non-dermatophytic fungi can sometimes be inefficient; they require a longer period of time in laboratory procedures and sometimes may lead to late diagnosis. In case of SFI (systemic fungal infection) late diagnosis or misdiagnosis can lead to wrong treatment, as well as not implementing measures for preventing the spread of infection [7]. On the other hand late diagnosis or misdiagnosis of IFI can result in patient's condition impairment or even mortality. In our previous paper [8], we considered only fungi genera that cause invasive infections, but we extended the dataset so types of fungi that cause systematic fungal infections are included too.

In both groups molds classification was performed on the basis of structure, i.e. macroscopic and microscopic morphological characteristics. Expert's knowledge and experience are needed for differentiation and identification of isolated fungi in laboratory practice.

B. *Morphologial differences of fungi*

Microscopic morphological characteristics of Dermatophytes are:

i) ***Microsporum*** spp. are characterized by segmented hyphae, numerous macroconidiae that are thick walled, rough, present microconidia;

ii) ***Trichophyton*** spp. are characterized by segmented hyphae, rare macroconidiae that are thin walled and smooth, numerous microconidiae;

iii) ***Epidermophyton*** spp. are characterized by segmented hyphae, numerous macroconidiae that are thin and thick walled, smooth and microconidiae are not formed.

Microscopic morphology of non-dermatophytic genera is characterized by:

i) ***Aspergillus*** spp.: Septate hyphae with unbranced conidiophores which ending with swollen vesicule that is covered with flak-shaped phialides on which are chains of mostly round sometimes rough conidia;

ii) ***Penicillium*** spp.: Septate hyphae with branched or unbrenched conidiophores that have secondary branches known as metulae or prophialides on which are phialides with chains of conidiae (Figure 1);

iii) ***Fusarium*** spp.: Septate hyphae with formation of canoe shaped or sickle shaped multiseptate macroconidia that are produced from phialides on unbranched or branched conidiophores;

iv) ***Alternaria*** spp.: Septate, dark hyphae with septate conidiophpores and formation of large macroconidiae which have transverse and longitudinal septations;

v) ***Mucor*** spp.; Wide and practically non-septate hyphae, speorangiophores are long, often branched and bear terminal round spore-filled sporangia.

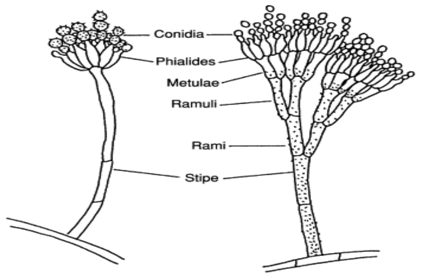


Figure 1. Penicillium morphology

C. *Preparation of dataset images for training*

Machine processes a picture as an array of pixels and numbers, so classification of images can be a rather difficult job, especially in cases where brightness is not the best, position of camera changes or the object is not fully present on the picture, which doesn't present a problem for a person. But, like a human, machine learns in the same manner, with examples of different categories with labels, so it eventually can recognize the patterns on the images.

For our model, we extracted examples of eight fungal genera, which are *Aspergillus* spp., *Fusarium* spp., *Epidermophyton* spp., *Alternaria* spp., *Microsporum*spp., *Penicillium*spp., *Trichophyton*spp. and *Mucorales* spp. (Figure 2). Images have been made at the Department of Microbiology and Immunology, Medical faculty, University of Niš, Serbia, laboratories, where molds have been isolated from patient materials, examined on microscopes and then photographed.

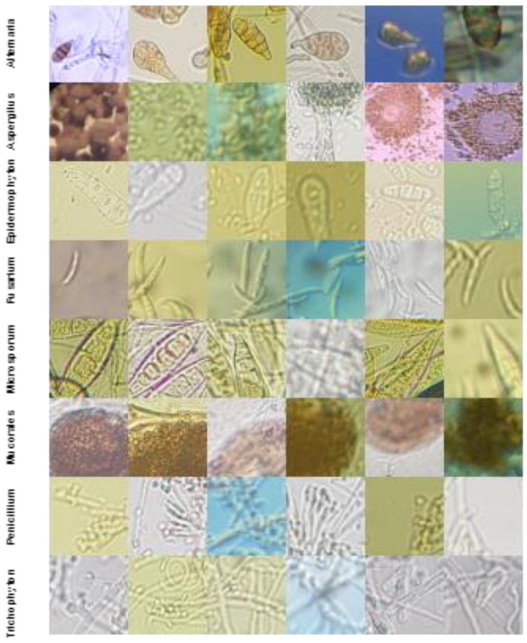


Figure 2. Examples of the dataset images

After preparing the images, which includes manually cutting the high resolution (3024 x 4032 pixels) sample images obtained from Department of Microbiology and Immunology and selecting ones which contain significant molds parts, it is necessary to determine which percentage of them will be used for training, and which for evaluation, since these sets have to be different so results of evaluation can be regular. After manual preparation, there were 6918 images, from which we used around 80% for training and the rest of the images (20%) for evaluation. In Table I, details of dataset used for training are presented.

TABLE I
Details of used dataset

Number of classes	Number of samples	Number od samples per class	Number of images after preparation	Images used for training	Images used for validation
8	492	50-65	6918	5603	1315

III. Method Description

For a neural network to learn to recognize certain patterns in images, it is necessary to create examples so it can learn from them. Sample images of patient materials with molds are high resolution, taken on microscopes, and they have to be cut, because of the GPU limitations when it comes to neural network training, and also to make more examples for network to learn. To obtain small resolution images, it was necessary to cut original images into the set of smaller images, suitable for training. After cutting the images, and manually eliminating the ones that don't contain mold patterns, it was decided to expand the dataset so examples can be more informative.

Operations that are used on the images to widen the dataset and provide multiple examples from one image are called augmentations [9]. Using different brightness, rotation, translation, flipping of the images, etc., we made more examples for training (Figure 3). In the end of this process, dataset became more informative and training could be started.

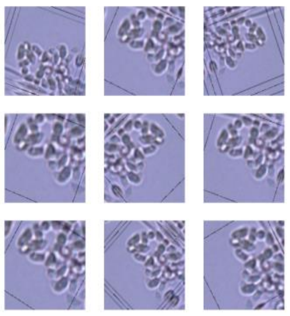


Figure 3. Augmentation of an image gives more images for training

Image classification is a very common problem, present in many different fields of expertise, and traditional approach to this problem is crafting a feature extractor that can be used for training a classifier [10-13]. Earlier solutions used artificial neural networks (ANNs) [14], but major advantages in this area have been made in recent years with development of convolutional neural networks (CNNs) [15]. CNNs represent an aggregation of three architectural ideas, local receptive fields, shared weights and spatial subsampling, which makes them more consistent in terms of translation and distortion [16].

During recent years, many different types of convolutional neural network architectures have been developed, but the one that gave the best result while training our model is EfficientNet. EfficientNet has a family of models (B0 to B7) and during training we tried various variants, where B1 showed the best results, based on accuracy measured. This models, introduced in 2019, by Tan and Le [17], are among the most efficient models, and their innovation lays in heuristic way to scale the model (compound scaling), making them a good combination of efficiency and accuracy [18].

Unlike conventional scaling methods (b-d on Figure 4) that arbitrary scale a single dimension of the network, compound scaling method uniformly scales up all dimensions. In this method, appropriate scaling coefficients are determined with grid search, which discovers relationships between different scaling dimensions. Applying those coefficients to baseline network gets the desired target model size [19].

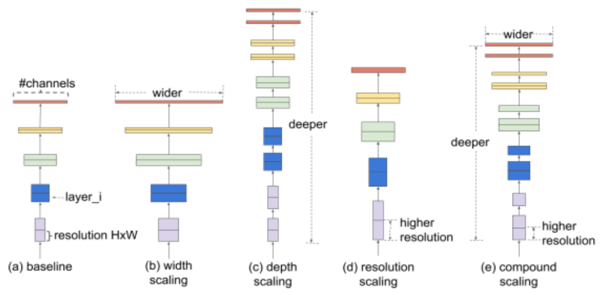
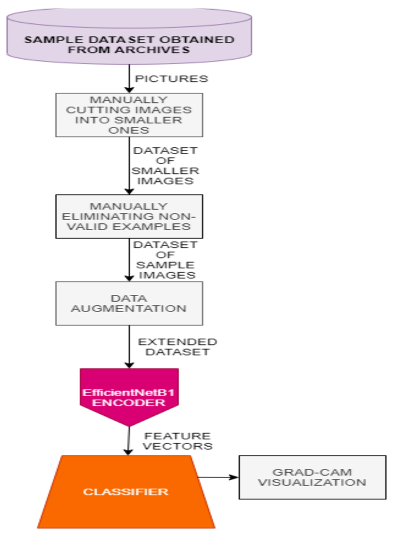


Figure 4. Comparison of different scaling methods [17]

Programming language Python [20] and library Keras have been used for training the model. Keras library [21], implemented in Python, has an interface which can be used for creating and training neural network models, including EfficientNet family. Keras is a deep learning API, running on top of the machine learning platform TensorFlow [22]. They were developed with a focus on enabling fast experimentation.


SAMPLE DATASET OBTAINED FROM ARCHIVES
PICTURES
MANUALLY CUTTING IMAGES INTO SMALLER ONES
DATASET OF SMALLER IMAGES
MANUALLY ELIMINATING NON-VALID EXAMPLES
DATASET OF SAMPLE IMAGES
DATA AUGMENTATION
EXTENDED DATASET
EfficientNetB1 ENCODER
FEATURE VECTORS
CLASSIFIER
GRAD-CAM VISUALIZATION
Figure 5. Solution diagram

Model has been compiled with ***RMSprop*** algorithm [23] for optimization (optimizers module),

sparse_categorical_crossentropy type of error (losses module), and the only parameter of metric during learning has been set as accuracy.

EfficientNet-B1 architecture model makes the core of this solution. After training of this model, feature vectors are obtained, which are then used to form a classifier. Classifier can then be used to determine which of 8 classes of molds new input images belong to. Diagram of current solution is shown in Figure 5.

Adjusting parameters of Keras functions and starting the training with different number of epochs, results at these phase of the project show that the trained model after twenty one epochs gives the best results, with 95,74% validation accuracy in classification of images (Figure 6). In our previous paper [8], with a slightly different (including only invasive fungi infections) and drastically smaller dataset, we got the accuracy of around 92%, which shows that we reached a very good improvement with new model. Also, in our previous work we haven't tried EfficientNet neural networks, which gave the best accuracy for our, now expanded, dataset.

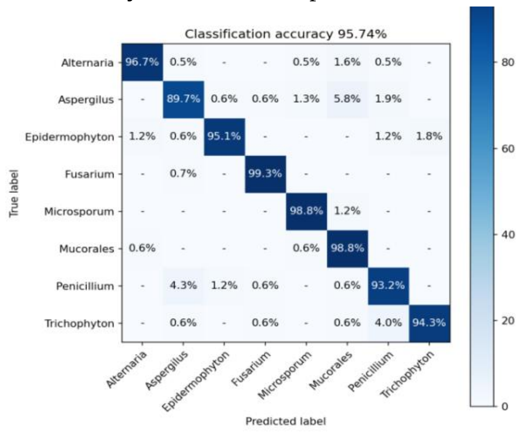


Figure 6. Confusion matrix showing accuracy in %

IV. Results and Discussion

Because of specific nature of the dataset and sample making, model has not been compared and tested with other datasets or models. In Table II, average results for each fungi genera have been presented.

TABLE II

Results for different fungi genera

Fungi genera spp.	Accuracy [%]	Samples placed correctly/samples per class
Alternaria	96,7	177/183
Aspergillus	89,7	139/155
Epidermophyton	95,1	155/163
Fusarium	99,3	144/145
Microsporum	98,8	166/168
Mucorales	98,8	161/163
Penicillium	93,2	151/162
Trichophyton	94,3	166/176

After validation of the model, it has also been tested manually, showing that the results for most images are accurate. Figure 7 shows confusion matrix, which contains accuracy results per classification class, showing problematic areas too. The most misclassifications happened for *Apergillus* spp. genera, for which we had the least number of clear images, which points out that more images have to be obtained or existing images should be sharpened, so better accuracy can be achieved.

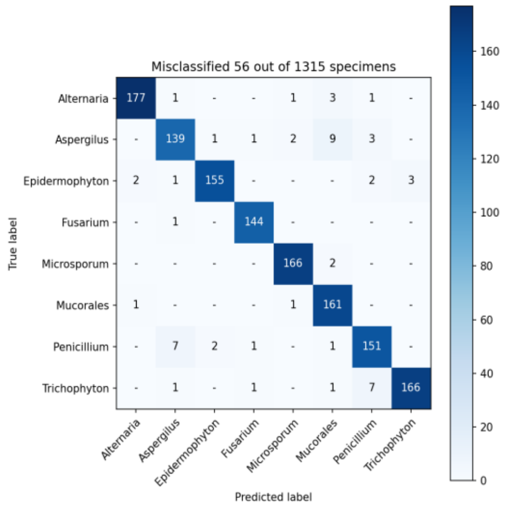


Figure 7. Confusion matrix

Taking into consideration that neural networks learn from examples, from which they learn patterns, and that some sample molds images contain not only significant parts used for diagnostics, but also other parts of materials (for example plain parts of the branches, end of slides on the microscope, different base colors) it is important to verify those learned patterns to be sure that classification, and later diagnostics, performed by the model is valid.

Grad-CAM method is a technique used for visualization of decisions from CNN models, making the decision making process transparent and understandable [24]. This method uses gradients of a target concept (in our cases molds) flowing into final convolutional layer in a network, so it can highlight regions of significance. This way, part of the image which had lead to decision of the classifier is highlighted.

Based on the majority of heat maps got from Grad-CAM method, decisions made by our classier have been done on significant parts of mold samples. Figure 8 shows the examples.

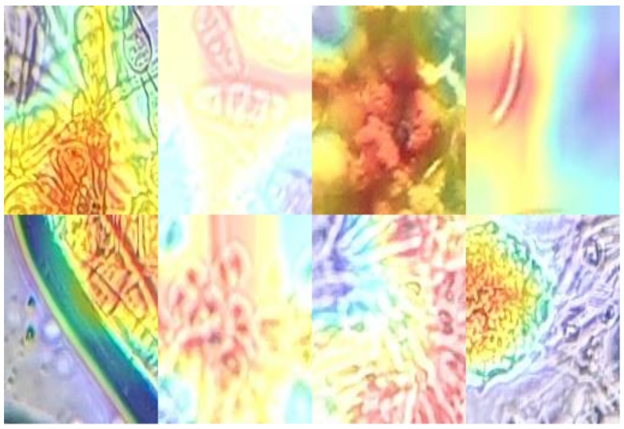


Figure 8. Examples of good pattern recognition visualization

Grad-CAM method is very useful in terms of concluding which of the test images have been misclassified because of the wrong pattern recognition in wrong part of the image (Figure 9). In our case, most of the misclassification happened because of poor quality of input images, because some of them are taken by mobile phones brought close to the microscope oculus, which can result in blurry image. In this way, visualizing the decision making process pointed out that maybe images should be sharpened before processing.

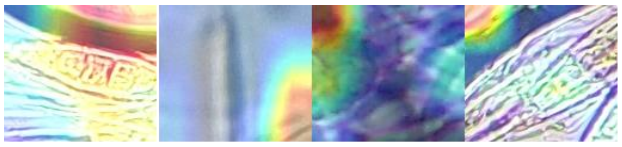


Figure 9. Examples of bad pattern recognition on blurry samples

V. CONCLUSSION

In this paper, we described developing a identification model which, based on accuracy results and testing, presents a solid base for developing an application that can be used in practice and drastically accelerate the process of diagnostics.

Grad-CAM method used to visualize the decision making process has proven to be a very efficient method of evaluation of the model, not only in terms of validating the "thinking" process of the classifier, but to point out flaws and cases where errors happen.

Future development of the model and application will involve developing an algorithm that can reach the decision based on high resolution photo, from which number of smaller images will be cut, and then classification will be performed on each of the small sample images. This approach will increase precision of the diagnostics, since the decision will be a ruling of the mayor, rather than determination based on one small sample.

ACKNOWLEDGMENT

We would like to thank the Department of Microbiology and Immunology, Medical faculty, University of Niš, Serbia, for all resources, samples and advices given during the work on this project.

REFERENCES

[1] Tasić S, Pešić S: Gljivičneinfekcije, dijagnoza I mogućnosti terapije, Punta Niš, Medicinski fakultet Niš, 2006 (odluka nastavno naučnog veća 14-376-5/2-2, 2006, udžbenik) R-12-5 M-44-2.

[2] Otašević S, Tasić-Miladinović N. Tasić A: Medicinska parazitologija-udžbenik sa CD-om. Univerzitet u Nišu- Medicinski fakultet Niš 2011 14-6735-6/2-4 2011.

[3] Valentina Arsić-Arsenijević, Marina Milenković, SuzanaOtašević, Dušan Pavlica. Medicinska mikologija i parazitologija. Društvo medicinskih mikologa Srbije, Centar za inovacije u mikologiji Beograd, V. Arsić-Arsenijević, Beograd 2012. ISBN 978-86-915391-1-5.

[4] Davise HL. Medically important fungi-a guide to identification, 3rd edn. Washington: American Society for Microbiology; 1995.

[5] S Tasić-Otašević , M Golubović, S Đenić, A Ignjatović, M Stalević, S Momčilović, M Bojanović , V Arsić-Arsenijević Species distribution patterns and epidemiological characteristics of otomycosis in Southeastern Serbia. J Mycol Med; 2020 Jun 30;101011. doi: 10.1016/j.mycmed.2020.101011.

[6] Otašević S, Momčilović S, Stojanović NM, Skvarč M, Rajković K, Arsić-Arsenijević V. Non-culture based assays for the detection of fungal pathogens. J Mycol Med. 2018 Jun;28(2):236-248. doi: 10.1016/j.mycmed.2018.03.001. Epub 2018 Mar 29.

[7] Pesic Z, Otasevic S, Mihailovic D, Petrovic S, Arsic-Arsenijevic V, Stojanov D, Petrovic M. Alternaria-Associated Fungus Ball of Orbit Nose and Paranasal Sinuses: Case Report of a Rare Clinical Entity. Mycopathologia. 2015 Aug;180(1-2):99-103. doi: 10.1007/s11046-015-9881-6. Epub 2015 Mar 7.

[8] Mina Milanović, Aleksandar Milosavljević, Determination of molds isolated from patient materials, based on the microscopic morphological characteristics, ICIST, March 2021 (proceedings in preparation)

[9] How to Load Large Datasets From Directories for Deep Learning in Keras by Jason Brownlee. Available online: https://machinelearningmastery.com/how-to-configure-image-data -augmentation-when-training-deep-learning-neural-networks/ (accessed on 20 May 2021).

[10] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification," IEEE Trans. Syst. Man. Cybern., vol. SMC-3, no. 6, pp. 610–621, Nov. 1973.

[11] J. A. Jose and C. S. Kumar, "Genus and Species-Level Classification of Wrasse Fishes Using Multidomain Features and Extreme Learning Machine Classifier," Int. J. Pattern Recognit. Artif. Intell., Mar. 2020.

[12] P. J. D. Weeks, M. A. O'Neill, K. J. Gaston, and I. D. Gauld, "Species-identification of wasps using principal component associative memories," Image Vis. Comput., vol. 17, no. 12, pp. 861–866, 1999.

[13] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual Categorization with Bags of Keypoints," in Workshop on statistical learning in computer vision, ECCV, 2004.

[14] I. Kanellopoulos and G. G. Wilkinson, "Strategies and best practice for neural network image classification," Int. J. Remote Sens., vol. 18, no. 4, pp. 711–725, Mar. 1997.

[15] Y. LeCun et al., "Handwritten digit recognition with a back-propagation network," papers.nips.cc, pp. 396–404, 1990

[16] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series.MIT Press, Cambridge," Handb. brain theory neural networks, vol. 3361, no. 10, 1995

[17] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan, Quoc V. Le, International Conference on Machine Learning, 2019

[18] Image_classification_efficientnet_fine_tuning. Available online:https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/ (accessed on 25 May 2021).

[19] EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling. Available online: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html(accessed on 25 May 2021).

[20] Python. Available online: https://www.python.org/ (accessed on 26 May 2021).

[21] Keras: The Python Deep Learning Library. Available online: https://keras.io (accessed on 26 May 2021).

[22] TensorFlow. Available online: https://www.tensorflow.org/ (accessed on 26 May 2021).

[23] Understanding RMSprop — faster neural network learning. Available online: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a (accessed on 26 May 2021).

[24] Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra, Dec. 2019

Freelancing blockchain: A practical case-study of trust-driven applications development

Milan Radosavljevic, Aleksandar Pesic, Nenad Petrovic, Milorad Tosic

***Abstract*—Nowadays, a large amount of work is done by freelancers across various areas – from graphical design and music composition to data input and software development. However, many issues appear due to participation of several third parties together with different rules and policies imposed by different platforms. On the other side, the emerging blockchain technology provides the execution of transactions in a trustable, decentralized, but still transparent manner. In this paper, we demonstrate a case-study where blockchain is adopted to eliminate the barriers and make freelancing more convenient and profitable at the same time. As an outcome, a proof-of-concept implementation of blockchain-based freelancing platform relying on Ethereum and Solidity smart contracts is presented that provides practical pointers for trust0driven applications development.**

***Index Terms*— Blockchain, Ethereum, Solidity, Freelancers**

I. Introduction

In today's business world, everything is based on trust. Any monetary transaction, ownership or arrangement. This trust, however, is provided in a very specific way - by the role of a third party, i.e. an institution of trust. In money transactions these are banks, in ownership relations there are cadastres and similar state institutions, in the case of any type of contract, there are courts. The positive side of these institutions, i.e. third parties, is that all parties to all these agreements trust them and expect protection in case of any unexpected occurrences. On the other hand, the appearance of third parties brings with it a lot of negative effects, so you often end up in a waiting list in order to make payments or get a certificate of ownership of a real estate and the like. Mistakes made by these institutions themselves are also very common, and as a rule they fall on the common man as a burden. There is bureaucracy, inefficiency, mistakes, enormous costs that at some point completely make the role of these intermediaries meaningless. The question is, is it possible to exclude third parties from future business, and still preserve that positive factor that they brought with them. In the last few years, there has been a development of a technology called *blockchain*, whose primary idea is just this. Decentralized system with minimal costs, efficient, transparent and yet safe enough to take on the role of e.g. banks. [1]

The share of freelancers in today's business world is large. A large amount of work is completed by freelancers, and as a freelancer, everyone has the opportunity to work in a huge number of domains as an independent. So, people who are engaged in graphic design, writing stories, teaching foreign languages and, of course, programming, earn their living by freelancing. Current platforms for freelancers are safe and the most popular places where freelancers can find work are [2] [3] [4]. The problem that arises is the possession of a third party and some rules that must be followed on certain platforms. Blockchain provides an opportunity to eliminate these problems and as a new solution sets some new boundaries and presents some new problems that are obtained by introducing this solution.

In this paper, it is explored how the blockchain technology can be leveraged to eliminate barriers when it comes to freelancing. As main outcome of this research, we introduce prototype implementation of freelancing platform based on Ethereum blockchain technology and Solidity smart contracts.

II. Background

A. Blockchain

Blockchain is a "cryptographically secure transactional singleton machine with a shared state". Cryptographically secure means that the creation of digital currency is provided by complex mathematical algorithms that are practically impossible to break. With the help of these algorithms it is almost impossible to cheat the system (e.g. creating fake transactions, deleting transactions etc.). A transactional singleton machine means that there is one canonical instance of the machine responsible for all transactions created in the system. In other words, there is one global truth that everyone believes in. Shared status means that the status stored on this machine is shared and available to everyone.

B. Ethereum blockchain platform

Ethereum blockchain is practically a transaction-based state machine. As it is known, the state machine receives inputs and, based on the current state, passes into new states. With Ethereum's state machine, we start from the "initial state". This is practically the state before any transaction occurred on the network. When transactions are executed, the initial state passes to some final state. At any time, the final state represents the current state in the Ethereum network. The state of Ethereum has millions of transactions. These transactions are grouped into "blocks". Each block

Milan Radosavljevic is student at Faculty of Electrical Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: milan.radosavljevic@elfak.rs).
Aleksandar Pesic is student at Faculty of Electrical Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: alpesh@elfak.rs)
Nenad Petrovic is teaching assistant at Faculty of Electrical Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: nenad.petrovic@elfak.ni.ac.rs)
Milorad Tosic is full professor at Faculty of Electrical Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: milorad.tosic@elfak.ni.ac.rs)

contains some set of transactions and each block is cryptographically chained together with the previous blocks, which can be seen in Figure 1.

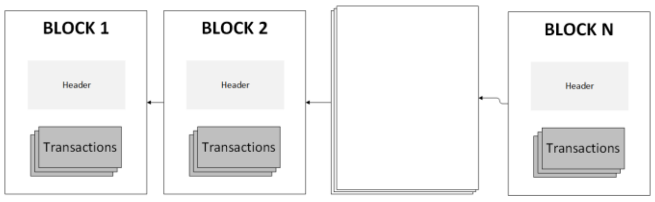


Fig. 1. Preview of Ethereum blockchain.

To cause a transition from one state to another, the transaction must be valid. For a transaction to be considered valid, it must go through a validation process known as mining. Mining is the process when a group of Ethereum nodes (more precisely computers) spend their computing resources to create a block of valid transactions. Any Ethereum network node that declares itself a miner can try to create and validate the block. All miners try to create and check blocks at the same time. Every miner provides mathematical "proof" when submitting a block, and this proof acts as a guarantee: if the proof exists, the block must be valid. The process of validation of each block by the miner who is supposed to provide mathematical proof is called *proof of work*.

C. Concept of fees

One very important concept in Ethereum is the concept of fees. Any calculation that occurs as a result of a transaction on the Ethereum network is charged a fee. The fee is paid in denominations called "gas". Gas is a unit used to measure the fees required for a particular calculation. Two factors determine how much it takes to pay for an action: the gas price, and how much gas that action requires. The important part is that Ethereum gas prices aren't fixed. Gas prices are determined by supply and demand. The busier the Ethereum network, the higher the gas price. The amount of gas required for each transaction depends on how complex the transaction is. Gas prices are denoted in gwei, which itself is a denomination of ETH. Gwei is 10^{-9} ETH. It is possible to set a gas limit for each transaction. Gas limit refers to the maximum amount of gas you are willing to consume on a transaction.

D. Solidity smart contracts

A smart contract is a contract that is performed by itself together with the terms of the contract to which the parties have agreed. The terms of the contract are written directly in the code of that smart contract. And the contract itself and the 'consent' of the participants exists throughout the Ethereum network. Practically smart contracts are programs that are immutable and deterministic. They depend on the context of the Ethereum Virtual Machine and the decentralized global network. The contract controls the execution of transactions, which are public and non-refundable. In essence, contracts are reduced to programs, which are modeled on traditional contracts, '*if it happens, then do it*'. The contract is executed on many computers to ensure reliability and trust. Smart contracts provide autonomy, trust, speed, security and money savings.

III. Related Work

The blockchain was invented in 2008 by a group or an individual, and this information is still unknown to the public. Therefore, we can consider that blockchain, and at the same time Ethereum, is a newer technology that poses some new challenges and problems in front of us. Much research has been conducted in academia as well as in industry to explore the benefits of smart contracts as well as the worlds in which they are applicable. There are many smart contract platforms on the market with different features that suit certain applications. In [5], the authors focus on the technique of using blockchain to store vaccination records, which is secure and efficient and is based on smart contracts found on the Ethereum platform. In [6], a system based on blockchain was proposed, which refers to workers who are temporarily employed in companies. In that way, employees are provided with a fair and legal salary for their work obligations, as well as protection if the employer becomes a debtor. The author's work in [7] gives a focus on climate change and proposes a solution which, with the use of blockchain, would reduce global warming while keeping records of the crown impression of the product. In [8], the authors provided an overview of all the challenges that smart contracts would face in the future.

IV. Solution Overview

A. System Architecture

The application architecture consists of three parts: client side, Web3 interface and server side. The server side is located on the Ethereum blockchain network and uses Solidity smart contracts which are accessed via Web3 interface on the client side. The client side is located in the browser and uses HTML, CSS and Javascript programming language. Also, the client side contains the Web3.js Javascript library through which it communicates with Solidity smart contracts as can be seen in Figure 2.

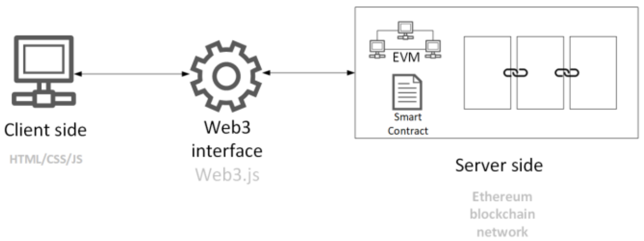


Fig. 2. Representation of system architecture.

B. Tools used for system implementation

1) Ganache

Ganache is a local Ethereum blockchain that runs on a local computer. Intended for the development and testing of smart contracts and decentralized applications in a secure and deterministic environment. Provides ten externally owned accounts for testing purposes. The application contains a graphical interface and can also be used as a console application.

2) Metamask

Metamask is a software that allows you to own a cryptocurrency wallet and allows you to interact with the Ethereum blockchain. Metamask provides the ability to store and manage account addresses on the browser. It also allows us to connect securely to decentralized applications. Using this software enables multi-user browser behavior.

3) Remix IDE

Remix IDE is an open source web and desktop application. It enables the rapid development of Solidity smart contracts and contains a large number of plugins as well as a graphical interface. The Remix IDE is used for contract development, but also as a platform for learning programming on the Ethereum platform. The Remix IDE is part of a Remix project that develops a handful of tools related to Solidity smart contracts. It is written in the Javascript programming language and allows you to run and test contracts in a web browser. It also allows you to test, debug and deploy contracts as well as many other useful options. [9]

V. Implementation

As indicated in the System Architecture chapter, the system consists of a server and a client side. The server side consists of smart contracts written in the Solidity programming language. The UML diagram in Figure 3 shows the organization of the contracts and the structures used in the system. The main component is a *FreelancerContract* contract that uses *Service*, *FreelancerStructure* and *Offer* data structures. This component represents one Freelancer who is registered in the system. While *PlatformContract* acts as a repository and it contains all registered freelancers in the system. In addition to the data structure that stores basic information about the freelancer, there are also structures for services and offers. *Service* is what a freelancer offers, while an *Offer* acts like a real job offer.

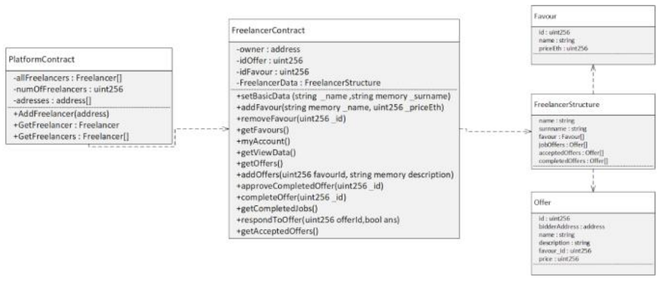


Fig. 3. UML contract diagram on Ethereum platform.

The client side of the application is written in the Javascript programming language. On the client side there are two proxy classes that correspond to the contracts on the server side, and also uses the Web3.js library to communicate with the server side. Practically one function on the client side calls one function in the contract. There are also functions that are handlers for the events which are broadcasted in contracts. The UML class diagram of the client side can be seen in Figure 4.

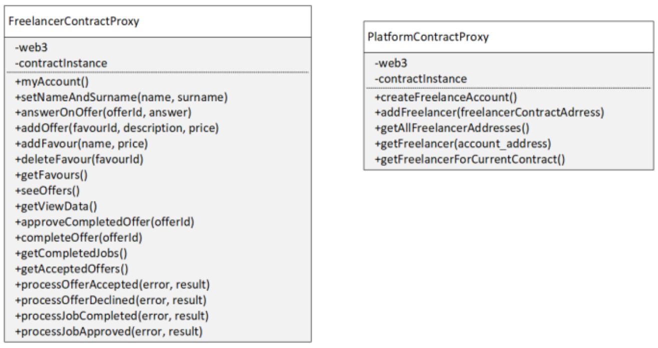

FreelancerContractProxy
-web3
-contractInstance
+myAccount()
+setNameAndSurname(name, surname)
+answerOnOffer(offerId, answer)
+addOffer(favourId, description, price)
+addFavour(name, price)
+deleteFavour(favourId)
+getFavours()
+seeOffers()
+getViewData()
+approveCompletedOffer(offerId)
+completeOffer(offerId)
+getCompletedJobs()
+getAcceptedOffers()
+processOfferAccepted(error, result)
+processOfferDeclined(error, result)
+processJobCompleted(error, result)
+processJobApproved(error, result)
PlatformContractProxy
-web3
-contractInstance
+createFreelanceAccount()
+addFreelancer(freelancerContractAdrress)
+getAllFreelancerAddresses()
+getFreelancer(account_address)
+getFreelancerForCurrentContract()

Fig. 4. UML class diagram on the client side.

VI. Results

Figure 5 shows the initial view of the client application. The application offers the ability to create a new account and navigation that allows to navigate through the entire application.

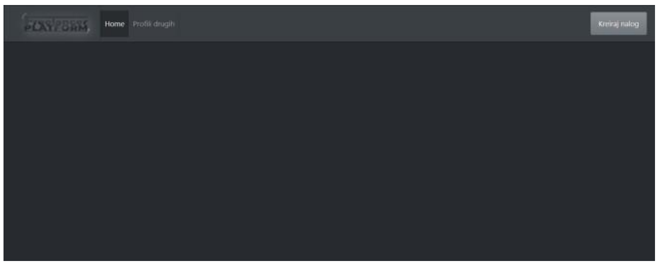


Fig. 5. Initial view of the application.

After creating the account, you get the view as in Figure 6 and the application allows the user to set the name and last name, add services he is offering and have an insight into the job offers that are offered to him, offers accepted by the user and offers waiting for client's approval.

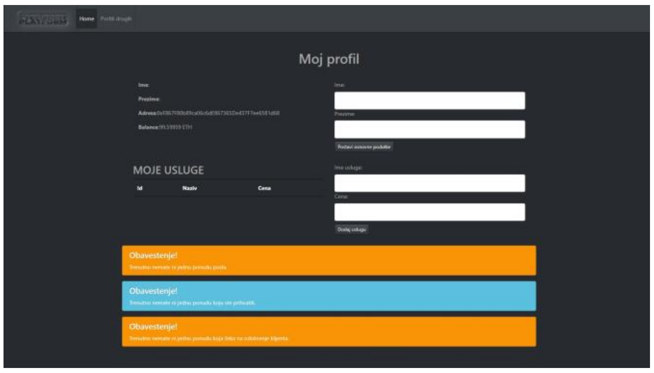
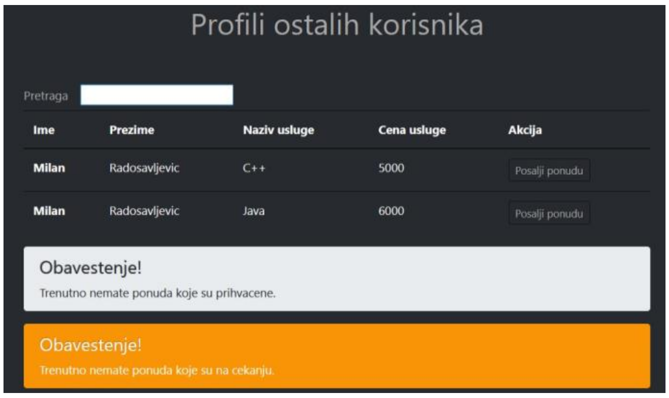


Fig. 6. View of the application after creating the account.

Clients can see the services of all freelancers on the Profiles of Others page, which can be selected from the navigation. The view of the page can be seen in Figure 7. The application provides the ability to search all offered freelancer services that are in the system and the ability to send a service offer to a specific freelancer that will be displayed on the homepage of the freelancer for whom the offer is intended.



Profili ostalih korisnika

Pretraga

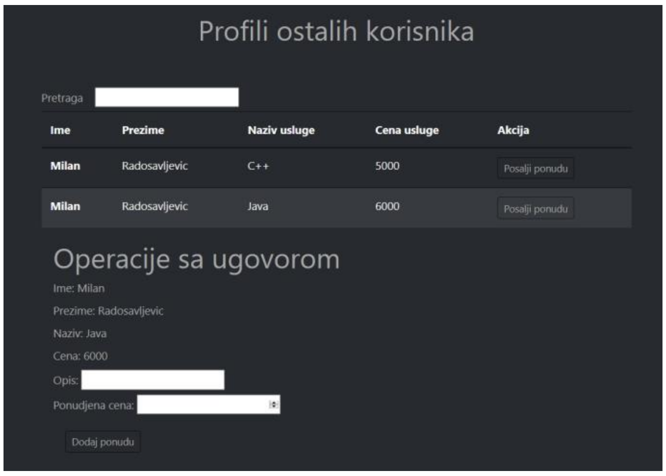
Ime	Prezime	Naziv usluge	Cena usluge	Akcija
Milan	Radosavljevic	C++	5000	Posalji ponudu
Milan	Radosavljevic	Java	6000	Posalji ponudu

Obavestenje!
Trenutno nemate ponuda koje su prihvacene.

Obavestenje!
Trenutno nemate ponuda koje su na cekanju.

Fig. 7. View of page *Profiles of other users*.

When submitting an offer it is necessary to add a description and offer a price for a particular service. Based on the submitted offer, the freelancer will decide whether to accept the offer for the job. Form for sending the offer can be seen in Figure 8. After the offer is sent, the offered price is transferred from the client's account to the smart contract account. In case the freelancer rejects the offer, Ether will be returned to the customer account.



Profili ostalih korisnika

Pretraga

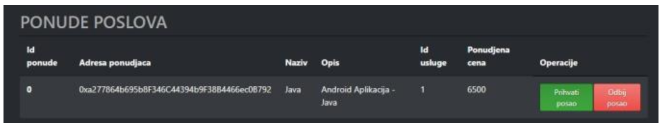
Ime	Prezime	Naziv usluge	Cena usluge	Akcija
Milan	Radosavljevic	C++	5000	Posalji ponudu
Milan	Radosavljevic	Java	6000	Posalji ponudu

Operacije sa ugovorom

Ime: Milan
Prezime: Radosavljevic
Naziv: Java
Cena: 6000
Opis:
Ponudjena cena:
Dodaj ponudu

Fig. 8. Appearance of the offer submission form.

After the accepted offer, freelancer completes the offer and sends it to the client for approval, after the approved offer, the client accepts the completed work and only then the offered price is transferred to the freelancer's account. The layout of the offer table can be seen in Figure 9.



PONUDE POSLOVA

Id ponude	Adresa ponudjaca	Naziv	Opis	Id usluge	Ponudjena cena	Operacije
0	0xa277864b695b8F346C44394b9F3884466ec08792	Java	Android Aplikacija - Java	1	6500	Prihvati posao / Odbij posao

Fig. 9. Appearance of the table with job offers.

VII. CONCLUSIONS AND FUTURE WORK

A platform for supporting a freelancing work community is developed using Solidity smart contracts on the Ethereum platform. The implementations process is used as a case-study for learning practical aspects of trust-driven applications development. Practical experiences and results are presented in this paper.

This paper gives our first experiences and more systematic approach is needed particularly validation and evaluation that are planned for future work. Regardless of absence of a full scientific rigor, we present our experiences that could be useful for future work in the field of trust-based applications development.

The good side of the solution presented in this paper is that it exploits advantages of the blockchain such as reliability, security and speed. Due to the fact that since its launch, the Ethereum platform has not had downtime due to consensus and decentralized approach. Hence the fact that this is another advantage, more precisely the advantage that the platform is always online and working. This further implies that the applications running on the blockchain do not have a downtime, including this one. The solution has an intuitive user interface that is capable of expansion. Even though the application was developed as a proof-of-concept, it shows high potential for commissioning in a real environment.

One of disadvantages of the approach is that every action that changes the state of the blockchain uses gas that requires compensation from the user, as stated in the chapter *Background*. Hence, some actions performed in the application are not free. In the current prototype, freelancer can not deliver product to the client who hired him. In future work this shortcoming could be fixed by connecting accounts with Github service, for example. It is possible to further optimize the speed of the application, on both client and server side, and to minimize the fee spent for performing actions on the server side. The obtained solution proves that it is possible to reach a satisfactory solution at an acceptable price.

ACKNOWLEDGMENT

This work has been supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

REFERENCES

[1] Uvod u blockchain [online], Blog, Đorđe Ivanović, 2018. Source: https://blog.itkonekt.com/2018/07/30/uvod-u-blockchain/
[2] Freelancer [online], freelance recruitment platform, https://www.freelancer.com
[3] Upwork [online], freelance recruitment platform, https://www.upwork.com
[4] Fiverr, freelance recruitment platform [online], https://www.fiverr.com
[5] S. K. Deka, S. Goswami, A. Anand, "A Blockchain Based Technique for Storing Vaccination Records", IEEE Bombay Section Signature Conference (IBSSC), pp. 135-139, 2020.
[6] A. Pinna, S. Ibba, "A blockchain-based Decentralized System for proper handling of temporary Employment contracts", Proceedings of the 2018 Computing Conference vol. 2, pp. 1-6, 2019.
[7] A. M. Rosado da Cruz, F. Santos, P. Mendes, E. Ferreira Cruz, "Blockchain-based Traceability of Carbon Footprint" Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS), pp. 1-10, 2020.
[8] Sh. Wang, Y. Yuan, X. Wang, J, Li, R. Qin, F. Wang, "An Overview of Smart Contract: Architecture, Applications, and Future Trends", IEEE Intelligent Vehicles Symposium, pp. 108-113, 2018.
[9] Remix IDE documentation [online]. Source: https://remix-ide.readthedocs.io/en/latest/

Comparative analysis of intra-board synchronous serial communication interfaces

Predrag Petronijević, *Vlatacom Institute of High Technologies, Milutina Milankovica 5, 11070 Belgrade, Serbia*
Vladimir Kuzmanović, *Faculty of Mathematics, University of Belgrade*

***Abstract* — Designing custom-made hardware for special purposes is a challenging process. During the development, it is essential to take into consideration the required performance of the device, component availability on the market as well as the final price of the developed and assembled product. Almost every modern hardware consists of various sensors, memories, AD/DA converters and a microcontroller to control and manage the interaction off all those devices. Based on the purpose of the device being developed, the engineer has to make a decision on the components that will be used in the final product. For this decision to be justifiable, the engineer needs to have a very high level of knowledge regarding the intricate world of interfaces required to establish the intercommunication of the components inside the device. Modern sensors, memories and AD/DA converters usually require some form of a high-speed serial interface, synchronous or asynchronous. In this paper we will analyze the three most commonly used serial synchronous communication interfaces: I^{2}C, SPI and SPORT. Also, we will explain the hardware and software properties and limits of every mentioned synchronous serial interface. Finally, the benefits and drawbacks of the chosen communication interfaces will be considered and conclusions drawn.**

***Index Terms* — computer engineering, embedded systems, sensors, synchronous serial communication**

I. Introduction

One aspect of designing new hardware is defining its application and the other aspect is defining a set of features the final product has to meet. The desired set of features can be divided into a set of operational and environmental limits, e.g. thermal resistance or voltage, and a set of desired performance characteristics, e.g. bandwidth or noise levels. This set of features limits the number of possible components that can be used in the design of the hardware. Even when limited with operational and performance characteristics, the choice of available hardware components is enormous due to a large number of manufacturers. Making the correct choice of hardware in order to meet the desired characteristics requires extensive knowledge [1-2]. One key decision to make is the choice of the right communication interface that will be used for intercommunication of the chosen components.

Modern devices usually consist of various sensors, memories, AD/DA converters and many other components controlled by a microcontroller. This control is achieved by establishing intercommunication between the microcontroller and every component inside the device. In modern devices, this communication is digital and standardized to conform to one or more of the standard communication interfaces in use today [3-4].

Interfaces used today can be divided into categories based on the way the data is transferred between the devices. Two criteria can be used for this division. The first criterion is defined by the number of channels used in the transmission of the data. If the data is transmitted bit by bit in a specific order over a single channel, such transmission is called serial. If the data is sent as multiple bits at the same time over multiple channels, such transmission is called parallel. The other criterion is defined by the way the data is sent. If the data is sent in the form of a byte or a single character with start and stop bits added to the data, such transmission is called asynchronous because it does not require synchronization. If the data is sent in the form of groups or frames, such transmission is called synchronous because it requires synchronization between sender and receiver. Synchronous transmission is more reliable and full-duplex, while asynchronous transmission is half-duplex [5].

Inter-Integrated Circuit (I^{2}C) was discussed by Patel et al. [6], Lynch et al. [7] and Blum [8]. Wootton in [9] described the use of Serial Peripheral Interface (SPI) as a means of communication between the CPU and various peripheral devices. Gay in [10] described the properties of SPI and its operation was described by Dogan in [11]. SPI and I^{2}C were compared in [12-13]. SPI, I^{2}C and UART were analyzed in [14-15].

In this paper we will address and compare the three most commonly used synchronous serial communication interfaces for intercommunication between various devices. Besides the well-known and widely used I^{2}C and SPI protocols, we will also introduce Analog Devices proprietary SPORT protocol and perform comparative analyses of the three serial protocols. Section 2 of the paper introduces all three interfaces with their hardware and software properties and requirements. The next section analyses the benefits and drawbacks of these serial interfaces. Finally, section 4 draws conclusions on serial interfaces described in the paper.

Predrag Petronijević is with Vlatacom Institute of High Technologies, Milutina Milankovica 5, 11070 Belgrade, Serbia (e-mail: predrag.petronijevic@vlatacom.com).
Vladimir Kuzmanović is with the Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia (e-mail: vladimir_kuzmanovic@matf.bg.ac.rs).

II. SYNCHRONOUS SERIAL COMMUNICATION

A serial communication protocol in which data is sent as a continuous stream at a constant rate is described as synchronous serial communication. For communication to be called synchronous it is required that the clocks are synchronized in both the transmitting and receiving devices. The term synchronized refers to the clocks running at the same rate, which enables the receiver to sample the signal at the same intervals used by the transmitter. Synchronization of clocks permits the omission of start and stop bits. As a consequence, more information can be passed over a circuit per unit of time than with asynchronous serial communication.

Serial communication can be established via a communication channel or a computer bus. It is mostly used for long-distance communication and computer networks where parallel communication is impractical. The development of technology has made serial computer buses more common at shorter distances, mostly as a basis for cheap and simple intra-board communication between two or more integrated circuits on the same printed circuit board connected by signal traces and not external cables.

The three most commonly used synchronous serial communication protocols for intra-board communication are inter-integrated circuit (I^2C), serial peripheral interface (SPI) and Analog Devices synchronous serial peripheral port (SPORT).

A. Inter-Integrated Circuit (I^2C)

Inter-Integrated Circuit (I^2C) is a synchronous, multi-master, multi-slave, packet-switched and single-ended serial communication bus invented in 1982 by Philips Semiconductors. Today it is widely used for interfacing with lower speed peripheral integrated circuits from a microcontroller in short distance intra-board communication.

I^2C emphasizes design simplicity and low manufacturing costs over speed. It is usually used for accessing low-speed AD/DA converters, controlling small displays, reading diagnostic sensors, etc. I^2C enables the microcontroller to control a network of devices with just two general-purpose input-output pins and software. Many other serial protocols offer similar functionality but require more pins and signals to interconnect multiple devices [16].

Hardware requirements for establishing I^2C communication are rather simple. Two bidirectional open collector or open drain lines with typical voltages of +5V or +3.3V are required for connecting the devices. These two lines are called Serial Data Line (SDA) and Serial Clock Line (SCL). I^2C bus speed can range from 10 kbit/s to 5 Mbit/s depending on the revision of the protocol. The bit rate is defined for transfer between master and slave without taking into consideration any protocol overhead. The overhead includes a slave address and usually a register within the slave device and finally per byte acknowledge (ACK/NACK) bits. This makes the actual bitrate lower than the bitrate used would imply. High-speed I^2C is widely used in embedded systems, while lower speed version is used in personal computers.

The reference design is a bus with clock (SCL) and data (SDA) lines with 7-bit addressing to which the devices are connected. Devices connected to the bus are referred to as nodes. The number of nodes is limited by the address space and by the total bus capacitance of 400pF. This restricts communication distances to a few meters. In practice, I^2C is restricted to intra-board communication due to its relatively high impedance and low noise immunity which requires a common ground potential.

There exist two roles for the node on the bus: master and slave. The device is referred to as the master if it generates the clock and initiates communication with the slaves. The device is referred to as the slave if it receives the clock and responds when addressed by the master. The protocol supports multiple masters and multiple slaves on the same bus. Also, the roles of the device can be changed during its operation.

The protocol defines four modes of operation for a given device on the bus: master transmits, master receives, slave transmits and slave receives. Usually, each device on the bus will use a single role with two predefined modes of operation. Besides 0 and 1 data bits, the I^2C defines special signals which represent message delimiters. These signals are called START and STOP signals which are distinct from data bits. The communication between devices is as follows:

- The master is in master transmit mode and initiates the transmission by sending the START signal followed by a 7-bit address of the slave it wants to communicate with which is followed by a single bit designating whether the master wants to write to or to read from the slave.
- If the slave with the given address exists on the bus it responds with the ACK bit for that address. Then, the master continues to transmit either in transmit or receive mode according to the bit set while the slave continues in complementary mode.

The address and the data over the I^2C bus are sent in MSB mode. The START signal is a high-to-low transition of the data line (SDA) with the clock (SCL) line high. The stop signal is a low-to-high transition of SDA with SCL high. All other transitions of SDA take place with SCL low. The device which is in transmitting mode writes the data byte by byte to the SDA line. The device in receive mode sends the ACK bit after every byte. I^2C transmission may consist of multiple messages. The master terminates a message with a STOP signal if it is the end of the transaction. If the master wants to retain control of the bus for another message it sends another START signal.

I^2C physical layer is shown in Figure 1.

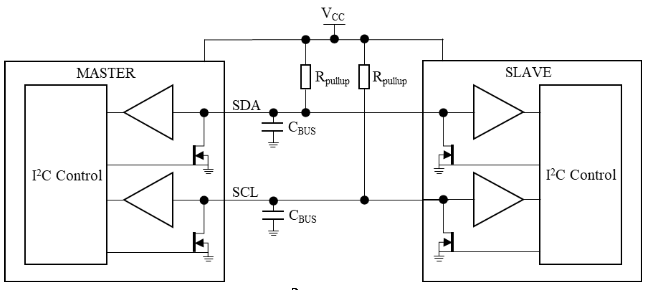


Figure 1. I^2C physical layer

B. Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial communication interface developed by Motorola in the mid-1980s [17]. It is used for short-distance communication in embedded systems. It is typically used for interfacing with memories, liquid crystal displays, sensors and AD/DA converters.

Devices communicating over SPI are organized as a master-slave architecture with a single master. The communication is achieved in full-duplex mode. The master device creates the frames for reading and writing. SPI supports multiple slave devices through selection with individual slave select lines. Sometimes, these lines are called chip select (CS) lines. The SPI bus specifies four logic signals:

- Serial clock (SCLK) – output from the master.
- Master Out Slave In (MOSI) – data output from the master.
- Master In Slave Out (MISO) – data output from the slave.
- Slave/Chip Select (SS/CS) – output from the master, active low.

For the communication to be established between devices, MOSI on a master device connects to MOSI on a slave device. Slave/Chip Select line is used instead of software addressing concept. Sometimes, MOSI on a slave device is labeled as Serial Data In (SDI) and MISO is labeled as Serial Data Out (SDO). This signal naming convention is used as an unambiguous way of labelling the pins of master and slave devices.

The SPI bus can operate with a single master device and one or more slave devices. Most slave devices have tri-state outputs so their MISO becomes high impedance when the device is not selected. This allows multiple slave devices to share common bus segments with each other.

For the communication to start, the master device has to configure the clock signal using a frequency supported by the slave. Then the master has to select the desired slave device with the logic level 0 on the appropriate SS/CS line. If the slave device requires a waiting period, the master device has to wait for at least that period of time before it starts issuing clock cycles on the SCLK line. During each cycle on the SCLK line, a full-duplex transmission occurs. The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This form of operation is maintained even when one-directional data transfer is intended.

Besides configuring the clock frequency, the master also needs to configure the clock polarity (CPOL) and clock phase (CPHA) with respect to the data. CPOL determines the clock polarity. CPOL value 0 defines a clock signal which idles at logic level 0 and each cycle consists of a pulse of 1. This translates to the leading edge being rising and the trailing edge is falling. CPOL value 1 defines the opposite. The clock idles at logic level 1 and each cycle consists of a pulse of 0. CPHA determines the timing of the data bits relative to the clock pulses. CPHA value 0 defines that the “out” side changes the data on the trailing edge of the preceding clock cycle, while the “in” side captures the data on the leading edge in the clock cycle. CPHA value 1 defines the opposite. The “out” side changes the data on the leading edge of the current clock cycle while the “in” side captures the data on the trailing edge of the clock cycle.

Finally, SPI supports word sizes that are not limited to 8-bit words but can range up to 32-bit words. Also, message size is arbitrary, as is its contents and purpose. The signal lines are shared between multiple devices, except for the slave select line which is unique per slave.

Its versatility, high speed and easy implementation coupled with board real estate savings compared to parallel buses have made it popular in many applications today. SPI interface is widely used in embedded systems for interfacing various sensors, control devices, memories and liquid crystal displays.

SPI physical layer is shown in Figure 2.

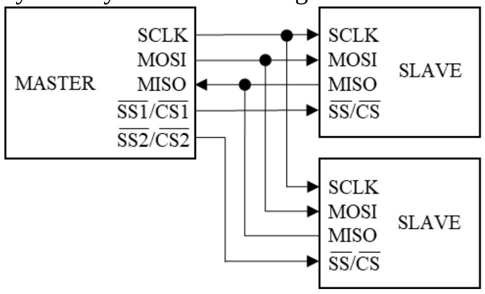

MASTER
SCLK
MOSI
MISO
$\overline{SS1}/\overline{CS1}$
$\overline{SS2}/\overline{CS2}$
SLAVE
SCLK
MOSI
MISO
$\overline{SS}/\overline{CS}$
SLAVE
SCLK
MOSI
MISO
$\overline{SS}/\overline{CS}$

Figure 2. SPI physical layer

C. Synchronous Serial Peripheral Port (SPORT)

Synchronous Serial Peripheral Port (SPORT) is Analog Devices proprietary synchronous serial communication interface that supports a variety of serial data communication protocols. Key features of SPORT are continuously running clock and serial data words from 3 to 32 bits in length either most- or least-significant bit first. The protocol also supports two synchronous transmit and two synchronous receive data signals which double the total supported data stream. Finally, frames are synchronized with configurable synchronization signals [18].

For the SPORT interface to be established between two devices, the standard defines the following eight signals:

- Transmit Data Primary (DT0)
- Transmit Data Secondary (DT1)
- Transmit Clock (TSCLK)
- Transmit Frame Sync (TFS)
- Receive Data Primary (DR0)
- Receive Data Secondary (DR1)
- Receive Clock (RSCLK)
- Receive Frame Sync (RFS)

The values for clocks are independent and can be calculated by dividing the SCLK of the microcontroller with the correct value. The SPORT clocks are calculated with the following formula:

$$SPORTCLK = \frac{SCLK}{(2 \cdot (SPORTCLKDIV + 1))}$$

The smallest value the divisor SPORTCLKDIV can have is zero and the greatest value is 65535. TSCLK and RSCLK are independent and thus can have different values of SPORTCLKDIV. Depending on the value of SCLK and SPORTCLKDIV, the clock values for SPORT can be as high as 60 MHz or as low as 1 kHz. By default, the primary transmit and receive channels are enabled while the secondary transmit and receive channels are disabled.

Frame sync signal can be divided into early frame sync and late frame sync. Early frame sync is active for one clock pulse and then deactivates. Once the signal has been deactivated, valid data will be available. Late frame sync signal frames valid data and is active for the length of time that valid data is available. The signal is deactivated once the word to transmit or receive is fully sent.

SPORT protocol is proprietary and is supported by a majority of Analog Device microcontrollers and various types of integrated circuits for numerous applications. Such applications range from AD/DA converters, sensors, memories, health applications, smart industries, etc. Also, with a range of clock and frame synchronization options, the SPORT interface allows a variety of serial communication protocols and provides a glueless hardware interface to many industry-standard data converters and CODECs [19-20].

SPORT physical layer is shown in Figure 3.

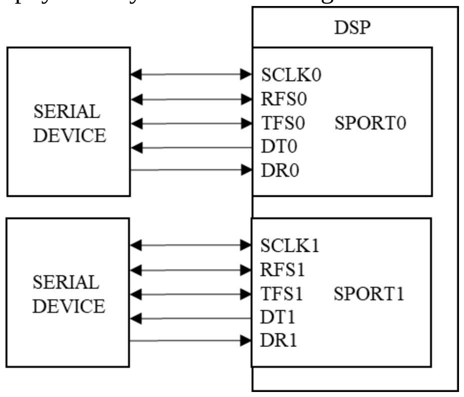


Figure 3. SPORT physical layer

III. Comparative Analysis

I^2C, SPI and SPORT all are synchronous bidirectional serial interfaces with considerable differences. The first obvious difference is the number of signals needed to establish communication between devices. The signals and number of lines required for establishing communication with each interface are displayed in table 1.

Table 1. Signals required for establishing communication

I^2C	SPI	SPORT
SDA Serial Data	MOSI Master Out Slave In	DT Serial Data Transmit
SCL Serial Clock	MISO Master In Slave Out	DR Serial Data Receive
	SCLK Serial clock	TFS Transmit Frame Sync
	SS Slave select	RFS Receive Frame Sync
		TCLK Transmit Clock
		RCLK Receive Clock

Considering the number of signals it is obvious that SPI and SPORT are full-duplex, while I^2C is half-duplex. Also, one other property to note is that I^2C is a multi-master multi-slave interface, while SPI and SPORT are single-master multi-slave interfaces.

Data transfer should also be considered when choosing the protocol to be used in the final product. The limits for data transfer are displayed in table 2.

Table 2. Data transfer limits

I^2C	SPI	SPORT
100 kbit/s – 5 Mbit/s Predefined values depending on version	Depending on the implementation Usually in range n x MHz to 10n x MHz n – number of devices connected to a single master	SCLK/2 Mbit/s SCLK – processor clock frequency

The advantages of I^2C over SPI and SPORT are the ease of linking multiple devices and the fact that cost and complexity do not scale up with the number of devices. The limitation of I^2C is numerous. The first is its slave addressing scheme and its relatively low number of possible addresses which may lead to address collisions. One other limitation is the number of supported speeds which need to conform to a certain standard. Since I^2C is a shared bus there exists a possibility that a single device could hang the entire bus. This happens if any device holds the SDA or SCL lines low, which prevents the master from sending START and STOP signals and reset the bus. Also, starvation is possible where a slower device starves the bandwidth needed by faster devices and thus increases latencies when other devices are addressed. Taking all this into consideration it is advisable to use I^2C for communication with on-board devices that are accessed only occasionally with no need for low latencies and high-speed bidirectional communication.

The advantages of SPI over I^2C and SPORT are complete protocol flexibility with variable size words and arbitrary choice of message size, contents and purpose. Also, hardware interfacing is easy. Slaves do not need a unique address since they are addressed with a per slave chip select line and slave devices do not need precision oscillators since they use the master's clock. Disadvantages compared to I^2C are the increased number of pins required for communication and the lack of slave ACK which enables the master to transmit data to nowhere without knowing it. Also, SPI protocol supports only one master, does not have a formal standard so validating conformance is impossible and does not support dynamically adding nodes. Taking all this into consideration, SPI is applicable in situations where the data transfer is organized in packets of arbitrary size and full-duplex. Also, it is applicable when there are a number of slaves communicating with the same SPI modes, because frequent changes of SPI mode severely impact the performance of communication.

Compared to the other two protocols, the main advantage of SPORT protocol is the support for multichannel transmits and receives of up to 128 channels. Also, a wide selection of data sizes is also a benefit as is the programmable polarity of both frame sync signals and data receive and transmit clocks. Finally, significantly higher data rates and double-buffered data registers that allow continuous data stream are a big advantage compared to both SPI and I^2C. The main disadvantages of SPORT are the fact that it is proprietary and supported only by Analog Devices products and that the complexity of supporting software components can be higher than that of competing schemes.

IV. Conclusion

In this paper we presented the three most commonly used synchronous serial protocols. The introduction showed that the engineer needs to have a broad knowledge regarding communication protocols to be able to make the right choice on the protocol to be used with respect to the desired operational and performance limits as well as to justify the proposed design. I^2C, SPI and SPORT are presented in detail and their properties, requirements and applications are discussed. Finally, the benefits and drawbacks of all three mentioned protocols are compared and analyzed which led to the conclusion on the suitability of the protocols in various scenarios. In the future, we intend to further research asynchronous communication protocols and their properties as well as inter-board communication protocols. We will focus on Controller Area Network (CAN) and Universal Asynchronous Receive Transmit (UART).

Acknowledgment

The research is founded by the Vlatacom Institute of High Technologies under project *#161 V155MM*.

References

[1] J. Staunstrup, W. Wolf, "Hardware/Software Co-Design: Principles and Practices," Springer Science & Business Media, 1997.

[2] P. Horowitz, W. Hill, "The Art of Electronics, 3rd edition," Cambridge University Press, 2015.

[3] J. Cowley, "Communications and Networking: An Introduction," Springer, 2007.

[4] IBM Corporation, "Data Communications Primer," Form C20-1668-0.

[5] J. Patrick, "Serial Protocols Compared," Embedded Staff, May 31, 2002, available at: https://www.embedded.com/serial-protocols-compared/.

[6] S. Patel, P. Talati, S. Gandhi, "Design of I2C Protocol," International Journal of technical innovation in Modern Engineering & Science, Vol 5, no. 3, pp. 741-744, 2019.

[7] K. M. Lynch, N. Marchuk, M. L. Elwin, "I^2C communication," Embedded Computing and Mechatronics with the PIC32, Newnes, 2016.

[8] J. Blum, "The I^2C Bus," Exploring Arduino; Tools and Techniques for Engineering Wizardry, 2nd Edition, 2019.

[9] C. Wootton, "Serial Peripheral Interface (SPI)," Samsung Artik Reference, Apress, Berkeley, 2016.

[10] W. W. Gay, "SPI Bus," Mastering the Raspberry Pi, Apress, Berkeley, 2014.

[11] I. Dogan, "Serial Peripheral Interface Bus Operation," SD Card Projects Using the PIC Microcontroller, Newnes, 2010.

[12] F. Leens, "An introduction to I^2C and SPI protocols," IEEE Instrumentation & Measurement Magazine, vol. 12, no.1, pp. 8-13, 2009.

[13] D. V. Gadre, S. Gupta, "Serial Communication: SPI and I2C," Getting Started with Tiva ARM Cortex M4 Microcontroller, Springer, 2017.

[14] A. Subero, "USART, SPI and I2C: Serial Communication Protocols," Programming PIC Microcontrollers with XC8, Apress, Berkeley, 2017.

[15] S. Shanthipriya, S. Lakshmi, "Design and verification of low speed peripheral subsystem supporting protocols like SPI, I2C and UART," ARPN Journal of Engineering and Applied Sciences, vol. 12, pp. 7368-7391, 2017.

[16] A.K. Oudjida, M.L. Berrandjia, R. Tiar, A. Liacha, K. Tahraoui, "FPGA Implementation of I2C & SPI Protocols: a Comparative Study," DOI: 10.1109/ICECS.2009.5410881, 2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009), Yasmine Hammamet, Tunisia, 13-16. December, 2009.

[17] Motorola, Freescale, NXP, "SPI Block Guide v3.06," 2003.

[18] B. Prabhalika, M. Kiran Kumar, "Fpga implementation of Design and verification Synchronous serial port(S-PORT)," ISSN: 2321-9939, International Journal of Engineering Development and Research IJEDR, India, 2013.

[19] A. Vasudev Prabhugaonkar, J. Rayala, "Interfacing AD7676 ADCs to ADSP-21365 SHARC® Processors," Engineer-to-Engineer Note EE-248, Analog Devices, Rev 1, October 7, 2004.

[20] "Implementing UART Using the ADuCM3027/ADuCM3029 Serial Ports," Application Note AN-1435, Analog Devices, Norwood, MA, 2017.