Korištenje protokola HTTP3 u komunikaciji uređaja Interneta stvari

Čelar, Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 289

KORIŠTENJE PROTOKOLA HTTP3 U KOMUNIKACIJI UREĐAJA INTERNETA STVARI

Ivan Čelar

Zagreb, veljača 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 289

KORIŠTENJE PROTOKOLA HTTP3 U KOMUNIKACIJI UREĐAJA INTERNETA STVARI

Ivan Čelar

Zagreb, veljača 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 2. listopada 2023.

DIPLOMSKI ZADATAK br. 289

Pristupnik:	**Ivan Čelar (0036509531)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentor:	izv. prof. dr. sc. Marin Vuković

Zadatak: **Korištenje protokola HTTP3 u komunikaciji uređaja Interneta stvari**

Opis zadatka:

Internet stvari je mreža fizičkih uređaja koji su opremljeni senzorima, programskom podrškom i ostalim tehnologijama u svrhu povezivanja i razmjenjivanja podataka s ostalim uređajima i sustavima na Internetu. Zbog svoje specifičnosti u smislu ograničenih procesnih i memorijskih resursa, postoji više komunikacijskih tehnologija koje se koriste na takvim uređajima. Vaš zadatak je analizirati i usporediti različite komunikacijske tehnologije koje se primjenjuju u mrežama Interneta stvari, s naglaskom na nove tehnologije, kao što je protokol HTTP3. Na temelju istraživanja odaberite tehnologiju koju smatrate najprikladnijom za korištenje u gore navedenim uvjetima te implementirajte prototip koji koristi tu tehnologiju za testiranje komunikacije. Kao platformu koristite uređaj ograničenih resursa, kao što je Raspberry Pi. Analizirajte ostvarenu komunikaciju u različitim uvjetima te vrednujte i komentirajte rezultate.

Rok za predaju rada: 9. veljače 2024.

Sadržaj

Uvod

Tema ovog rada je korištenje aplikacijskog protokola HTTP3 u komunikaciji Interneta stvari, u nastavku IoT.

Ovaj rad se sastoji od prvog poglavlja koje sadrži definiciju i opis pojma mreže, OSI ili referentnog modela za otvoreno povezivanje sustava koji je najkorišteniji apstraktni opis arhitekture mreže te TCP/IP referentnog modela koji se koristi kao standard u internetskoj zajednici.

Zatim je u drugom poglavlju opisan pojam IoT, mreže fizičkih uređaja koji su opremljeni senzorima, programskom podrškom i ostalim tehnologijama u svrhu povezivanja i razmjenjivanja podataka s ostalim uređajima i sustavima na Internetu, koja zbog svoje specifičnosti u smislu ograničenih procesnih i memorijskih resursa, koristi više komunikacijskih tehnologija.

Treće poglavlje se sastoji od analize različitih komunikacijskih tehnologija koje se primjenjuju u IoT. Opisani su protkoli CoAP, MQTT, WebSocket te protokol HTTP u inačicama 1, 1.1, 2 i 3, s naglaskom na inačicu 3 te što je i kako HTTP3 rješava problem head-of-line blockinga koje imaju prethodne inačice protokola HTTP.

Četvrto poglavlje rada sačinjava opis implementacije prototipa klijenta i poslužitelja koji za komunikaciju koriste protokol HTTP3.

U petom poglavlju je opisana IoT okolina u kojoj je korišten prototip iz prethodnog poglavlja,

Šesto, posljednje, poglavlje ovog rada je analiza ostvarene komunikacije korištenjem prethodno opisanog prototipa u navedenoj IoT okolini u različitim uvjetima s vrednovanim i komentiranim rezultatima.

1. Mreža

U ovom poglavlju je definiran i opisan pojam računalne mreže s pokaznim primjerom mreže Interneta stvari koji će biti detaljnije upisan u drugom poglavlju. Osim mreže, ovo poglavlje navodi i opisuje OSI ili referentni model za otvoreno povezivanje mreže, najkorišteniji apstraktni opis arhitekture te TCP/IP model nazvan po glavna dva protokola koje koristi, istima će se koristit u nastavku rada pri opisu komunikacijskih tehnologija i protokola u trećem poglavlju koje se odnosi na iste.

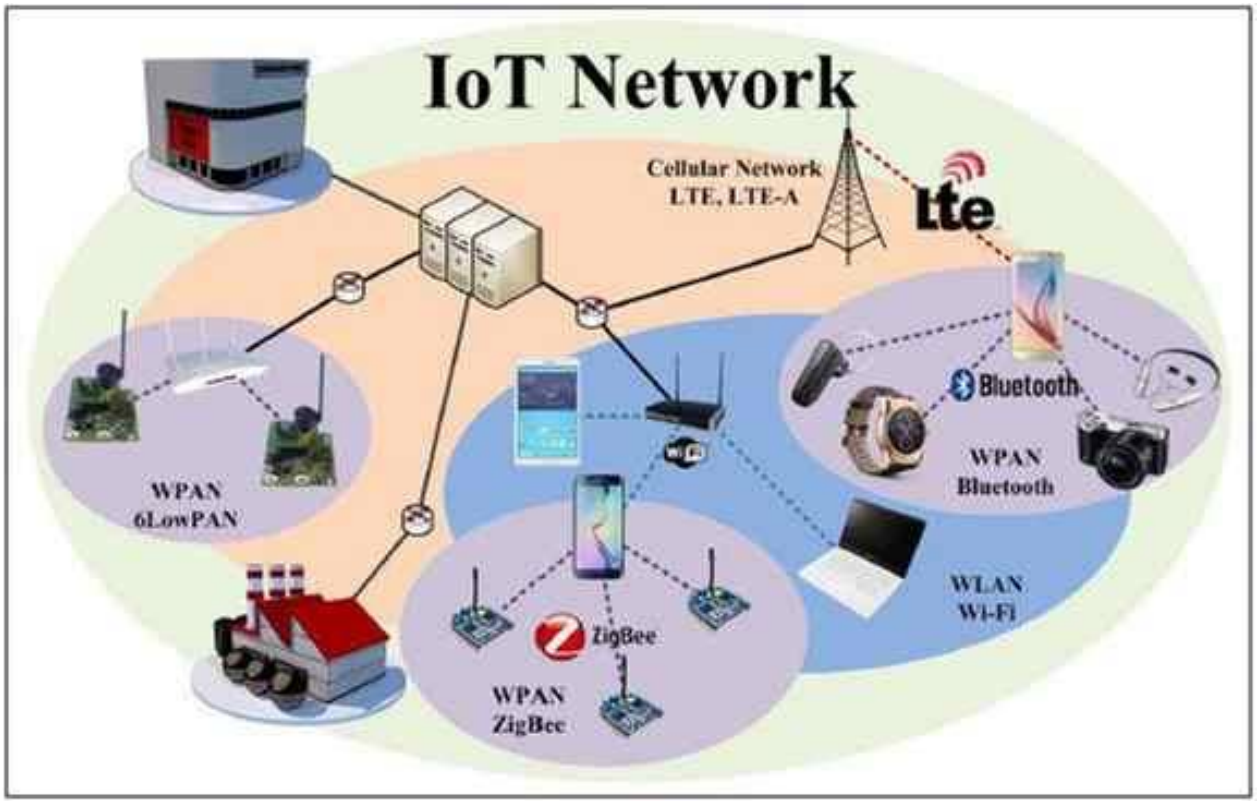
1.1. Računalna mreža

Računalna mreža je kolekcija autonomnih računala povezana jednom tehnologijom. Dva računala smatraju se povezanima ako mogu razmjenjivati informacije. Povezivanje ne mora biti isključivo putem bakrene parice, mogu se koristiti i optički vlakna, mikrovalovi, infracrveno svjetlo i komunikacijski sateliti. Mreže mogu biti različitih veličina i oblika. Obično su međusobno povezane kako bi stvarale veće mreže, pri čemu je Internet najpoznatiji primjer mreže više mreža. [1]

Česta se miješaju pojmovi računalne mreže i raspodijeljenih sustava. Ključna razlika je u tome što se u raspodijeljenom sustavu skup neovisnih računala korisnicima čini kao jedan koherentan sustav. Obično ima jedan model ili paradigmu koju predstavlja korisnicima. Često sloj softvera iznad operativnog sustava, nazvan middleware, je odgovoran za implementaciju ovog modela. Poznati primjer distribuiranog sustava je World Wide Web. Pokreće se na Internetu i predstavlja model u kojem sve izgleda kao jedna web stranica. [1]

U računalnoj mreži koherentnost, model i softver su odsutni. Korisnici su izloženi stvarnim strojevima, bez ikakvog pokušaja sustava učiniti više računala da izgledaju i djeluju na koherentan način. Ako računala imaju različito sklopovlje i različite operacijske sustave, to je potpuno vidljivo korisnicima. Ako korisnik želi pokrenuti program na udaljenom računalu, mora se prijaviti na to računalo i pokrenuti ga. [1]

U suštini, raspodijeljeni sustav je softverski sustav izgrađen na mreži. Softver mu daje visok stupanj kohezije i transparentnosti. Stoga, razlika između mreže i raspodijeljenog sustava leži u softveru, a ne u hardveru.[1]



Sl. 1.1 IoT mreža [16]

Na Sl. 1.1 je prikazana mreža Interneta stvari koja se sastoji od više drugih mreža (podmreža).

WPAN (Wide Personal Area Network) uključuje mreže poput Zigbee-a, Bluetooth-a, 6LowPAN, i sl. Na nešto većoj razini bežične mreže nalazi se bežična lokalna mreža (WLAN), koja uključuje Wi-Fi. Na razini iznad su mobilne komunikacijske tehnologije, LTE, 5G itd. Pametni telefoni i mobilni komunikacijski sustavi povezuju se s baznim stanicama koje pružaju povezivost s mrežom širokog područja (WAN), uključujući internet.

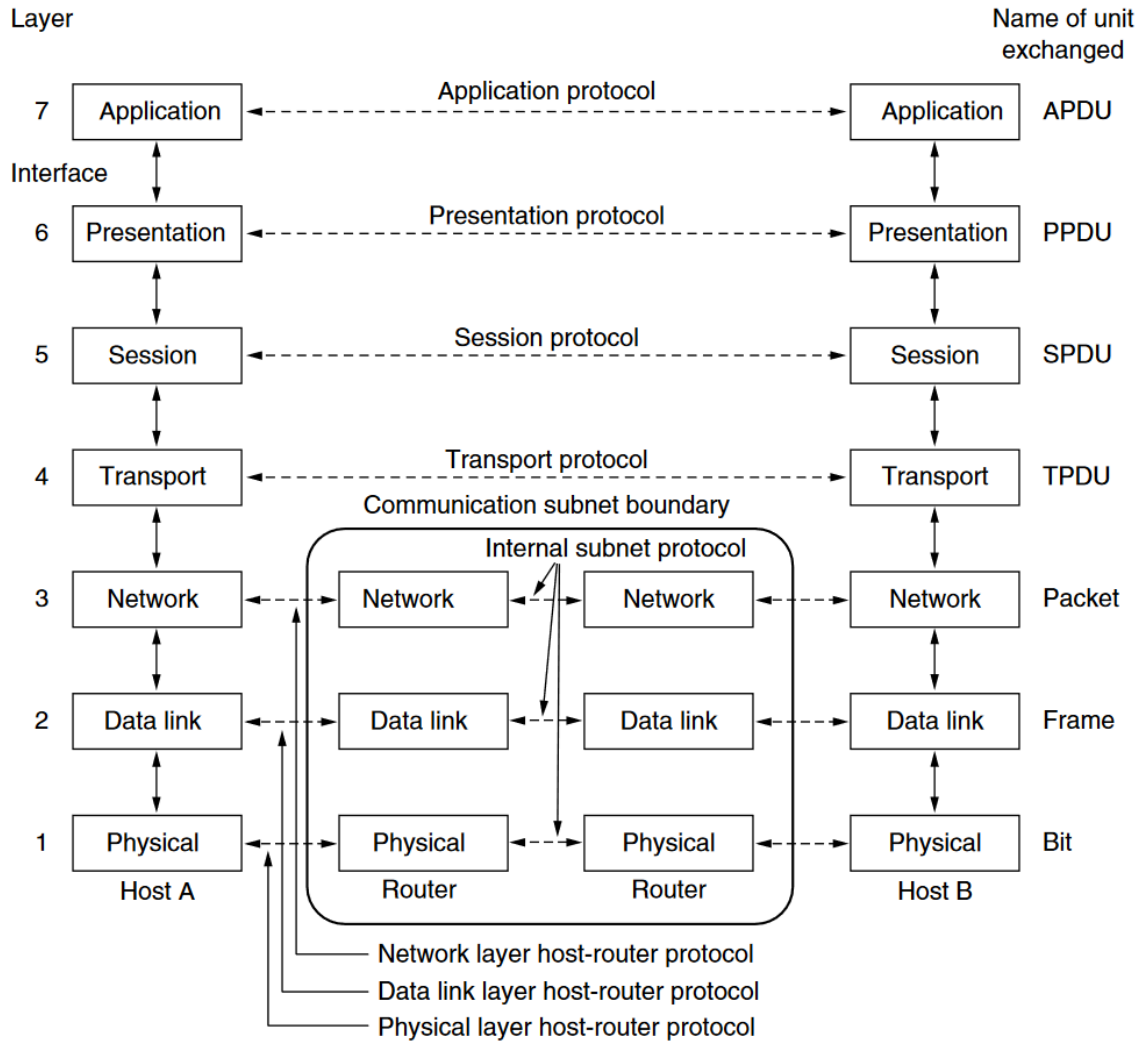
Primjer komunikacije u IoT mreži je pametni telefon opremljen Wi-Fi-jem ili Bluetoothom komunicira sa Zigbee uređajem WPAN mrežom, a pametni telefon može prenijeti informaciju preko mobilne mreže do bazne stanice koja će istu prenijeni na mrežu širokog područja, odnosno internet.

Dodatno objašnjenje navedenih pojmova u opisu primjera su u drugom poglavlju koje se odnosi na IoT.

1.2. OSI model

OSI model temelji se na prijedlogu koji je razvila Međunarodna organizacija za normizaciju (ISO) kao prvi korak prema međunarodnoj standardizaciji protokola korištenih u različitim slojevima (Day i Zimmermann, 1983). Bio je revidiran 1995. godine (Day, 1995). Model se naziva OSI (Open Systems Interconnection) referentni model jer se bavi povezivanjem otvorenih sustava, odnosno sustava koji su otvoreni za komunikaciju s drugim sustavima. OSI model sastoji se od sedam slojeva. [1]

Iako se protokoli povezani s OSI modelom više ne koriste, sam model je zapravo vrlo općenit i i dalje važeći, a značajke razmatrane u svakom sloju i dalje su vrlo važne.



Sl. 1.2 OSI model prikazan u slojevima [1]

Načela koja su primijenjena kako bi se došlo do sedam slojeva mogu se sažeto sažeti na sljedeći način: [1]

- Sloj bi trebao biti stvoren tamo gdje je potrebna drugačija apstrakcija.
- Svaki sloj treba obavljati određenu funkciju.
- Funkciju svakog sloja trebalo bi odabrati s ciljem definiranja međunarodno standardiziranih protokola.
- Granice između slojeva trebale bi biti odabrane kako bi se minimizirao protok informacija između slojeva.
- Broj slojeva trebao bi biti dovoljno velik tako da različite funkcije ne moraju biti združene u istom sloju iz nužnosti, ali i dovoljno malen kako arhitektura ne bi postala nepraktična.

Fizički sloj bavi se prijenosom bitova preko komunikacijskog kanala. Problemi u dizajnu odnose se na osiguravanje da kada jedna strana šalje bit 1, druga strana ga prima kao bit 1, a ne kao bit 0. Izazovi u ovom sloju uključuju koje električne signale treba koristiti za predstavljanje bita 1 i bita 0, koliko nanosekundi traje jedan bit, može li se prijenos odvijati istovremeno u oba smjera, kako se uspostavlja početna veza, kako se zatvara kada obje strane završe, koliko pinova ima mrežni konektor i kako se koristi svaki pin. Ovi problemi u dizajnu uglavnom se bave mehaničkim, električnim i vremenskim sučeljima, kao i fizičkim prijenosnim medijem, koji se nalazi ispod fizičkog sloja. [1]

Glavni zadatak sloja podatkovne veze je ispravljanje grešaka koje dobije s fizičkog sloja. To postiže maskiranjem stvarnih pogrešaka kako ih mrežni sloj iznad ne bi vidio. Zadatak izvršava tako da pošiljatelj podatke razbija u okvire podataka i šalje okvire u nizu. Ako je usluga pouzdana, primatelj potvrđuje ispravan prijem svakog okvira slanjem povratnog okvira potvrde. Još jedan izazov koji se javlja u sloju veze podataka je kako spriječiti brzog predavača da preplavi spori prijemnik podacima. Moglo bi biti potrebno neko reguliranje prometa kako bi predavač znao kada prijemnik može prihvatiti više podataka. Mreže s emitiranjem imaju dodatni problem u sloju veze podataka: kako kontrolirati pristup dijeljenom kanalu. Poseban podsloj sloja veze podataka, podsloj upravljanja pristupom mediju, bavi se tim problemom. [1]

Mrežni sloj kontrolira rad podmreže. Ključni izazov sloja je određivanje kako se paketi usmjeravaju od izvora do odredišta. Rute mogu biti temeljene na statičkim tablicama koje

su "fiksirane" u mrežu i rijetko se mijenjaju, ili se češće mogu automatski ažurirati kako bi izbjegle neispravne komponente. Također se mogu odrediti na početku svake komunikacije. Naposljetku, mogu biti dinamične, određuju se iznova za svaki paket kako bi odražavale trenutačno opterećenje mreže. Ako je previše paketa prisutno u podmreži u isto vrijeme, mogu si smetati, stvarajući usko grlo. Upravljanje zagušenjem također je odgovornost mrežnog sloja, u suradnji s višim slojevima koji prilagođavaju opterećenje koje stavljaju na mrežu. Općenito, kvaliteta usluge koja se pruža (kašnjenje, vrijeme prijenosa, stabilnost itd.) također je izazov mrežnog sloja. Kada paket mora putovati iz jedne mreže u drugu kako bi došao do odredišta, mogu se pojaviti mnogi problemi. Adresiranje koje koristi druga mreža može biti različito od onoga koje koristi prva. Druga mreža možda uopće neće prihvatiti paket jer je prevelik. Protokoli se mogu razlikovati, i tako dalje. Na mrežnom sloju mora riješiti ove probleme i omogući povezivanje heterogenih mreža. U emitirajućim mrežama, problem usmjeravanja je jednostavan, pa je mrežni sloj često tanak ili čak nepostojeći. [1]

Osnovna zadaća transportnog sloja je prihvatiti podatke s viših slojeva, razdijeliti ih na manje jedinice ako je potrebno, proslijediti ih mrežnom sloju i osigurati ispravno stizanje svih podataka primatelju. Osim toga, sve to mora biti obavljeno učinkovito i na način koji izolira gornje slojeve od neizbježnih promjena u tehnologiji hardvera tokom vremena. Transportni sloj također određuje vrstu usluge koju će pružiti sloju sjednice i, konačno, korisnicima mreže. Najpopularnija vrsta transportne veze je kanal bez pogrešaka od točke do točke koji isporučuje poruke ili bajtove u redoslijedu u kojem su poslani. Međutim, postoje i druge moguće vrste transportne usluge, poput prijenosa izoliranih poruka bez jamstva o redoslijedu dostave i emitiranja poruka višestrukim odredištima. Vrsta usluge određuje se prilikom uspostave veze. Transportni sloj je stvarni end-to-end sloj; prenosi podatke sve od izvora do odredišta. Drugim riječima, program na izvoru komunicira sa sličnim programom na odredištu, koristeći zaglavlja poruka i upravljačke poruke. U nižim slojevima, svaki protokol je između stroja i njegovih neposrednih susjeda, a ne između krajnjeg izvora i odredišta, koji mogu biti razdvojeni mnogim usmjerivačima. Razlika između slojeva 1 do 3, koji su povezani, i slojeva 4 do 7, koji su *end-to-end,* prikazana je na Sl.1.2. [1]

Sloj sjednice omogućuje korisnicima na različitim računalima da uspostave sjedinicu između njih. Sjednice nude različite usluge, uključujući kontrolu dijaloga (praćenje tko ima red za prijenos), upravljanje tokenima (sprječavanje dva sudionika da istovremeno

pokušaju istu važnu operaciju) i sinkronizaciju (checkpointiranje dugih prijenosa omogućuje nastavak od mjesta gdje su stali u slučaju prekida i naknadnog oporavka). [1]

Za razliku od nižih slojeva koji se uglavnom bave premještanjem bitova, sloj prezentacije bavi se sintaksom i semantikom informacija koje se prenose. Kako bi omogućio komunikaciju između računala s različitim unutarnjim prikazom podataka, strukture podataka za razmjenu mogu se definirati na apstraktan način, zajedno sa standardnim kodiranjem koje se koristi. Sloj prezentacije upravlja tim apstraktnim strukturama podataka i omogućuje definiranje i razmjenu struktura podataka više razine. [1]

Sloj aplikacije sadrži različite protokole koji su često potrebni korisnicima. Jedan od široko korištenih aplikacijskih protokola je HTTP (HyperText Transfer Protocol), koji je osnova za World Wide Web. Kada preglednik želi web stranicu, šalje naziv stranice koju želi poslužitelju koji hosta stranicu koristeći HTTP. Poslužitelj potom šalje stranicu natrag. Drugi aplikacijski protokoli koriste se za prijenos datoteka, elektroničku poštu i mrežne vijesti. [1]

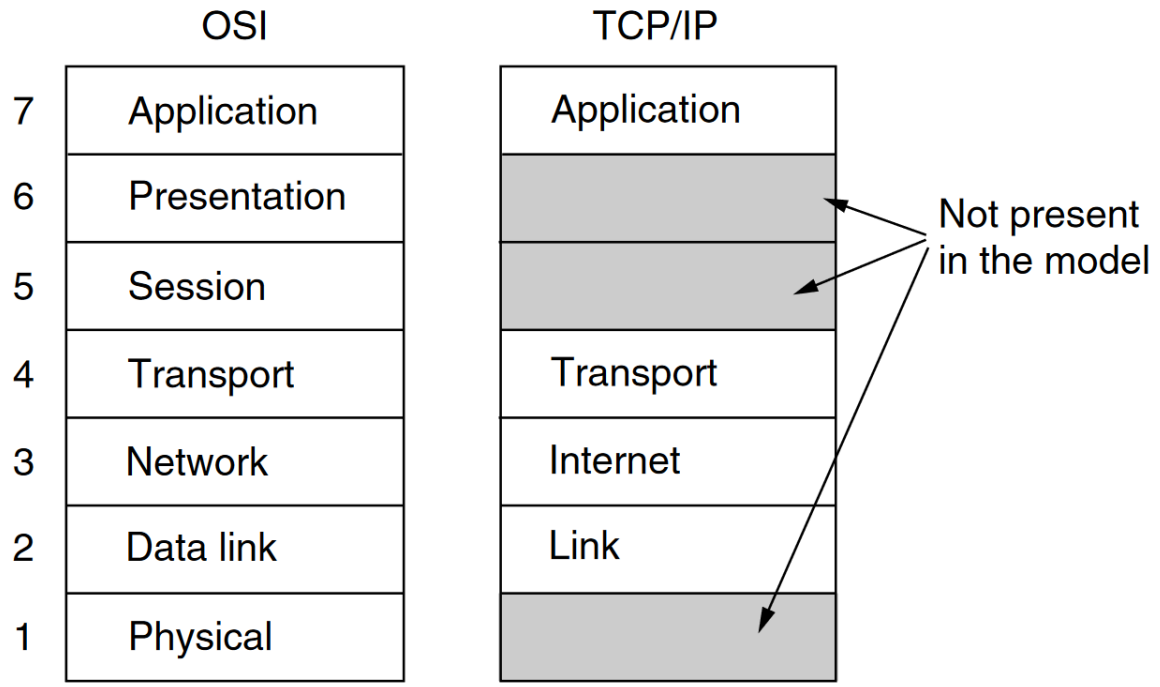
Protokol HTTP je važan u sklopu ovog rada i biti će detaljno obrađen u trećem poglavlju.

1.3. TCP/IP model

Sada ćemo se okrenuti od OSI referentnog modela prema referentnom modelu korištenom u "djedu" svih širokopojasnih računalnih mreža, ARPANET-u, i njegovom nasljedniku, internetu. ARPANET je bio istraživačka mreža koju je sponzoriralo ministartvo obrane SAD-a. Povezivao je stotine sveučilišta i vladinih institucija koristeći zakupljene telefonske linije. Kada su kasnije dodane satelitske i radio mreže, postojeći protokoli imali su problema s interoperabilnošću, pa je bila potrebna nova referentna arhitektura. Tako je, gotovo od početka, sposobnost povezivanja više mreža na jednostavan način bila jedan od glavnih ciljeva dizajna. Ova arhitektura kasnije je postala poznata kao TCP/IP referentni model, prema svojim dvama primarnim protokolima. Prvi put su je opisali Cerf i Kahn (1974), a kasnije su je izradili i definirali kao standard u internet zajednici (Braden, 1989). Filozofiju dizajna modela opisao je Clark (1988). [1]

S obzirom na zabrinutost ministartsva obrane da bi neki od njihovih dragocjenih računala, usmjerivača i internetskih priključaka mogli biti uništeni u trenu napada iz SSSR-a, još jedan glavni cilj bio je da mreža može preživjeti gubitak čvorova mreže, bez prekida postojećih veza. Drugim riječima, ministarstvo je željelo da veze ostanu netaknute dok god

su izvor i odredište funkcionalni, čak i ako su neki od računala ili prijenosnih linija između njih odjednom izvan funkcije. Osim toga, budući da su se predviđale aplikacije s različitim zahtjevima, od prijenosa datoteka do prijenosa govora u stvarnom vremenu, bila je potrebna fleksibilna arhitektura.



Sl. 1.3 Usporedba OSI i TCP/IP modela [1]

Ponovno će redom biti opisani slojevi odozdo prema gore.

Svi ovi zahtjevi doveli su do odabira mreže razmjene paketa temeljenom na bežičnom sloju koji se proteže preko različitih mreža. Najniži sloj u modelu, sloj veze, opisuje što veze poput serijskih linija i klasične Ethernet mreže moraju raditi kako bi zadovoljile potrebe ovog bežičnog internet sloja. To zapravo nije sloj u uobičajenom smislu riječi, već sučelje između računala i prijenosnih veza. Rani materijal o TCP/IP modelu ima malo toga za reći o tome.

Internet sloj ključna je komponenta koja povezuje cijelu arhitekturu. Prikazan je na Sl. 1.3 i otprilike odgovara OSI mrežnom sloju. Njegova je zadaća omogućiti računalima ubaciti pakete u bilo koju mrežu i neovisno putovanje paketa prema odredištu (potencijalno na različitoj mreži). Paketi mogu čak stići u potpuno različitom redoslijedu od onoga u kojem su poslani, u kojem slučaju je zadatak viših slojeva ponkvno ih posložiti, ako se želi

dostava u određenom redoslijedu. Napomena da "internet" se ovdje koristi u generičkom smislu, kao međuveza, iako je ovaj sloj prisutan i u internetu. [1]

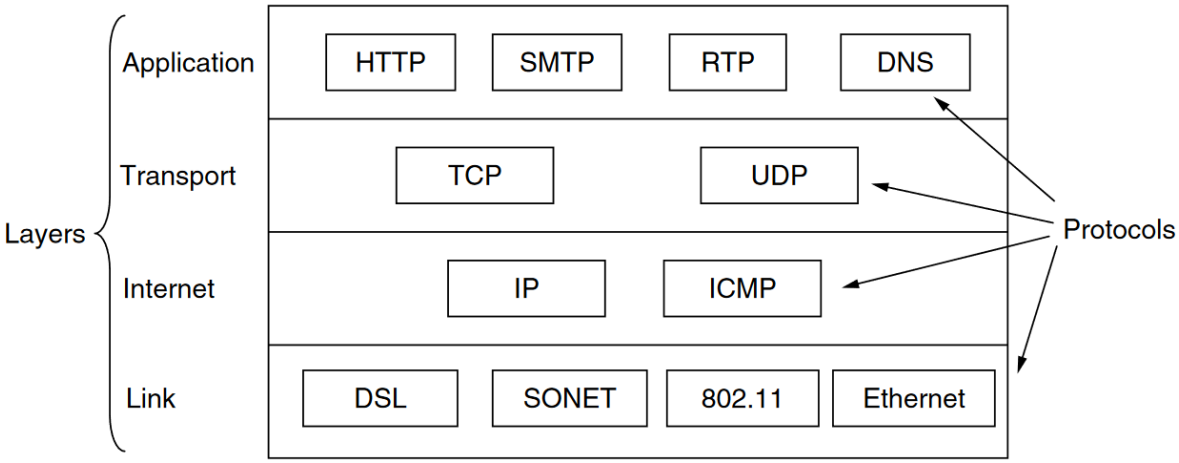
Analogija ovome je poštanski sustav. Pošiljatelj može ubaciti niz pisama u sandučić u jednoj zemlji i s malo sreće, većina će biti dostavljena na ispravnu adresu u odredišnoj zemlji. Pisma će vjerojatno putovati kroz jedan ili više međunarodnih poštanskih prolaza, ali to je transparentno korisnicima. Nadalje, činjenica da svaka zemlja (tj. svaka mreža) ima vlastite marke, preferirane veličine omotnica i pravila isporuke skrivena je od korisnika. Internet sloj definira službeni format i protokol paketa pod nazivom IP (Internet Protocol), zajedno s pridruženim protokolom nazvanim ICMP (Internet Control Message Protocol) koji mu pomaže u radu. Zadatak internet sloja je dostaviti IP pakete tamo gdje trebaju ići. Rutiranje paketa jasno je ključno pitanje ovdje, kao i zagušenje (iako se IP nije pokazao učinkovitim u izbjegavanju zagušenja). [1]

Sloj iznad internet sloja u TCP/IP modelu sada se obično naziva transportni sloj. Namijenjen je omogućavanju komunikacije vršnih entiteta na izvoru i odredištu računala, baš kao i u OSI transportnom sloju. Ovdje su definirana dva protokola za prijenos s kraja na kraj. Prvi, TCP (Transmission Control Protocol), je pouzdan orijentiran prema vezi protokol koji omogućuje da se niz bajtova koji potječe s jednog računala isporuči bez pogrešaka na bilo kojem drugom računalu na internetu. Segmentira ulazni niz bajtova u diskretne poruke i svaku proslijedi internet sloju. Na odredištu, TCP proces za primanje ponovno sastavlja primljene poruke u izlazni niz. TCP također upravlja kontrolom toka kako bi osigurao da brzi pošiljatelj ne može preplaviti sporog primatelja s više poruka nego što može obraditi. Drugi protokol u ovom sloju, UDP (User Datagram Protocol), je nesiguran, bezpovezнički protkol za aplikacije koje ne žele TCP-ovu slijednost ili kontrolu toka i žele pružiti vlastitu. Također se široko koristi za jednokratne, upit-odgovor upite i aplikacije tipa klijent-poslužitelj u kojima je promptna dostava važnija od točne dostave, poput prijenosa govora ili videa. Veza između IP, TCP i UDP prikazana je na Sl 1.4. [1]

TCP/IP model ne posjeduje sesijski i prezentacijski sloj. Nije se percipirala potreba za njima. Umjesto toga, aplikacije jednostavno uključuju sve funkcije sesije i prezentacije koje zahtijevaju. Iskustvo s OSI modelom pokazalo je da je ovaj pogled točan: ovi slojevi malo koriste većini aplikacija. Na vrhu transportnog sloja nalazi se sloj aplikacije. Sadrži sve protokole više razine. Prvi protokoli uključuju virtualni terminal (TELNET), prijenos datoteka (FTP) i elektroničku poštu (SMTP). Kroz godine dodani su mnogi drugi protokoli. Neki važni koje ćemo proučiti, prikazani na Sl 4., uključuju Domain Name

System (DNS) za mapiranje imena računala na njihove mrežne adrese, HTTP, protokol za dohvaćanje stranica na World Wide Webu, te RTP, protokol za dostavu medijskih sadržaja u stvarnom vremenu poput glasa ili filmova. [1]

Posebno je važno ovdje istaknuti mjesto HTTP protkola koji je bitan kasnije u ovom radu.



Sl. 1.4 Protokoli po slojevima TCP/IP modela [1]

2. Internet stvari

U ovom poglavlju je definiran pojam Interneta stvari, njegove karakteristike i arhitektura.

U implementacijskom dijelu ovog rada je korištena okolina Interneta stvari u kojoj je pokrenut HTTP3 klijent i poslužitelj prototip, stoga su u nastavku navedeni i pojašnjeni pojmovi vezani za Internet stvari.

2.1. Karakteristike Interneta stvari

Internet stvari opisuje mrežu fizičkih objekata - "stvari" - koji su ugrađeni sa senzorima, softverom i drugim tehnologijama kako bi se povezivali i razmjenjivali podatke s drugim uređajima i sustavima putem interneta.

IoT je širok i raznolik, može ga se smatrati kompliciranim zbog obilja komponenti i protokola koje obuhvaća. Umjesto tretiranja IoT-a kao jedne tehnološke domene, dobro je gledati na nj kao na skup različitih koncepata, protokola i tehnologija, koje su ponekad donekle ovisne o primjeni. Iako je širok spektar elemenata IoT-a osmišljen kako bi stvorio brojne koristi u područjima produktivnosti i automatizacije, istovremeno uvodi nove izazove, poput skaliranja ogromnog broja uređaja i količina podataka koje treba obraditi.

Razmjer uobičajene IT mreže obično se kreće u rasponu od nekoliko tisuća uređaja - pisača, mobilnih bežičnih uređaja, prijenosnih računala, poslužitelja i slično. Tradicionalni mreže, koji podržava pristup, distribuciju i jezgru (s podarhitekturama za WAN, Wi-Fi, podatkovni centar, itd.), dobro je shvaćen. Razmotrimo što se događa kada razmjera mreže pređe s nekoliko tisuća krajnjih točaka na nekoliko milijuna. Ova vrsta razmjera prethodno je viđena samo kod pružatelja usluga ranga 1. IoT uvodi model gdje se od prosječne komunalne usluge, tvornice, sustava prijevoza ili grada može lako tražiti podršku za mrežu ove veličine. Temeljem zahtjeva ove razine razmjere, IPv6 je prirodna osnova za IoT mrežni sloj. [2]

Tradicionalni modeli IT sigurnosti jednostavno nisu dizajnirani za nove načine napada koje donose visoko raspršeni IoT sustavi. IoT sustavi zahtijevaju dosljedne mehanizme autentifikacije, enkripcije i tehnike prevencije upada koje razumiju ponašanje industrijskih

protokola i mogu odgovoriti na napade na kritičnu infrastrukturu. Za optimalnu sigurnost, IoT sustavi moraju: [2]

- Biti sposobni identificirati i autentificirati sve subjekte uključene u IoT uslugu.
- Osigurati da su svi podaci korisnika dijeljeni između uređaja na kraju i pozadinskih aplikacija enkriptirani.
- Sukladnost s lokalnim zakonodavstvom o zaštiti podataka kako bi svi podaci bili zaštićeni i pohranjeni ispravno.
- Koristiti platformu za upravljanje povezanošću u IoT-u i uspostaviti sigurnosne politike temeljene na pravilima kako bi se odmah poduzela akcija ako se otkrije nenormalno ponašanje povezanih uređaja.
- Pristupiti sigurnosti na holistički način na razini mreže.

Većina IoT senzora dizajnirana je za jedan zadatak. Često imaju ograničenu snagu, procesorske i memorijske resurse, te šalju podatke samo kada je nešto važno. Zbog velikog broja tih uređaja i velikih, nekontroliranih okolina gdje se obično postavljaju, mreže koje pružaju povezivost često su vrlo nepouzdane i podržavaju vrlo niske brzine prijenosa podataka. Ako mreža ima ograničenja u performansama, rješenje je jednostavno: nadogradite na bržu mrežu. Ako je previše uređaja na jednoj mreži i utječe na performanse, jednostavno se može izraditi novu mrežu i nastaviti s rastom koliko treba. Međutim, ovaj pristup ne može zadovoljiti ograničenu prirodu IoT sustava. IoT zahtijeva novu vrstu tehnologija povezivosti koje zadovoljavaju i razmjere i ograničenja. [2]

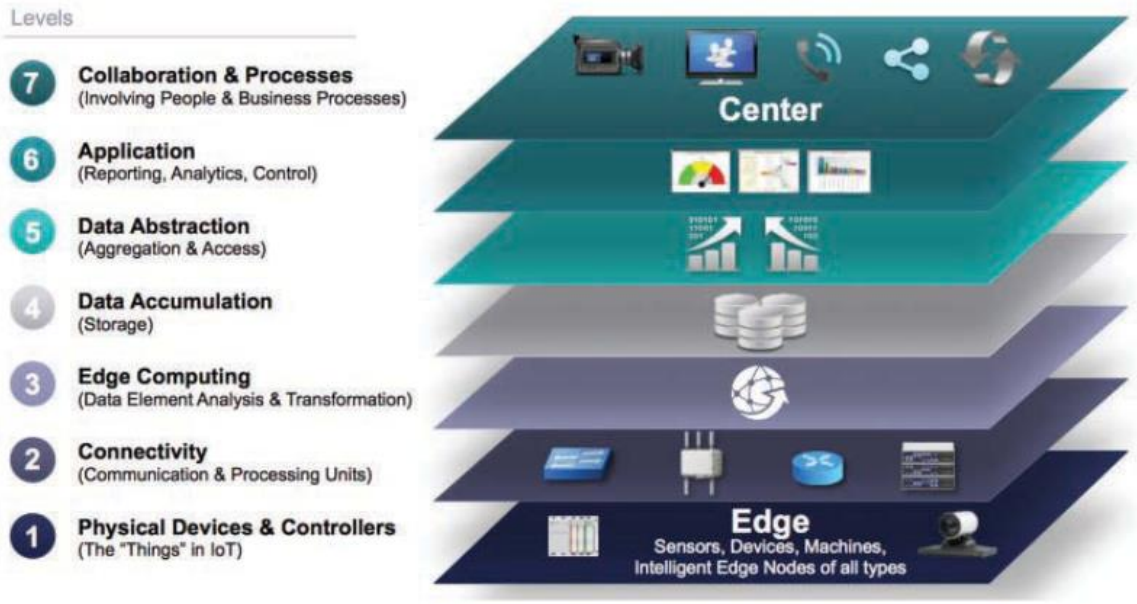
IoT uređaji generiraju ogromne količine podataka. Općenito, nisu važani nestrukturirani podaci koje generiraju uređaji na mreži. Međutim, u IoT-u su takvi podaci bitni jer omogućuju pružanje novih IoT usluga koje poboljšavaju korisničko iskustvo, smanjuju troškove i pružaju nove prihodne mogućnosti. Iako je većina podataka koje generira IoT nestrukturirana, uvidi koje pruža analitika mogu unaprijediti procese i stvarati nove poslovne modele Međutim, kada se svi ovi podaci kombiniraju, može postati teško upravljati i učinkovito analizirati. Stoga, IoT sustavi su dizajnirani s raspodijeljenom obradom podataka kroz arhitekturu, kako bi filtrirali i smanjili nepotrebne podatke te pružili najbrži mogući odgovor uređajima kad je to potrebno. [2]

Podrška za zastarjele uređaje u IoT sustavima, krajnji uređaji vjerojatno će biti na mreži vrlo dugo - ponekad desetljećima. Kako se implementiraju IoT mreže, trebaju podržavati starije uređaje već prisutne na mreži, kao i uređaje s novim mogućnostima. IoT mreža

mora biti sposobna za neku vrstu prijevoda protokola ili koristiti uređaj za povezivanje tih starih krajeva s IoT mrežom. [2]

2.2. Arhitektura Interneta stvari

IoTWF (vođen od strane tvrtki Cisco, IBM, Rockwell Automation i drugih) 2014. godine objavio je sedmoslojni referentni model za IoT. Iako postoje različiti IoT referentni modeli, onaj koji je predstavio IoT World Forum pruža čist i pojednostavljen pogled na IoT te uključuje računalstvo na rubu, pohranu podataka i pristup. Pruža sažet način vizualizacije IoT-a s tehničke perspektive. Svaki od sedam slojeva detaljno je razložen na specifične funkcije, a sigurnost obuhvaća cijeli model. Sl. 2.1 prikazuje referentni model IoTWF. [2]



Sl. 2.1 IotWf model Iot-a [2]

Kako je prikazano na Sl. 2.1, IoT referentni model definira skup razina s kontrolom koja teče iz središta (što može biti ili usluga u oblaku ili poseban podatkovni centar) prema rubu, koji uključuje senzore, uređaje, strojeve i druge vrste inteligentnih krajnjih čvorova. Općenito, podaci putuju prema gore kroz stog, potječući s ruba, i idu prema središtu.

Korištenjem ovog referentnog modela, postiže se sljedeće: [2]

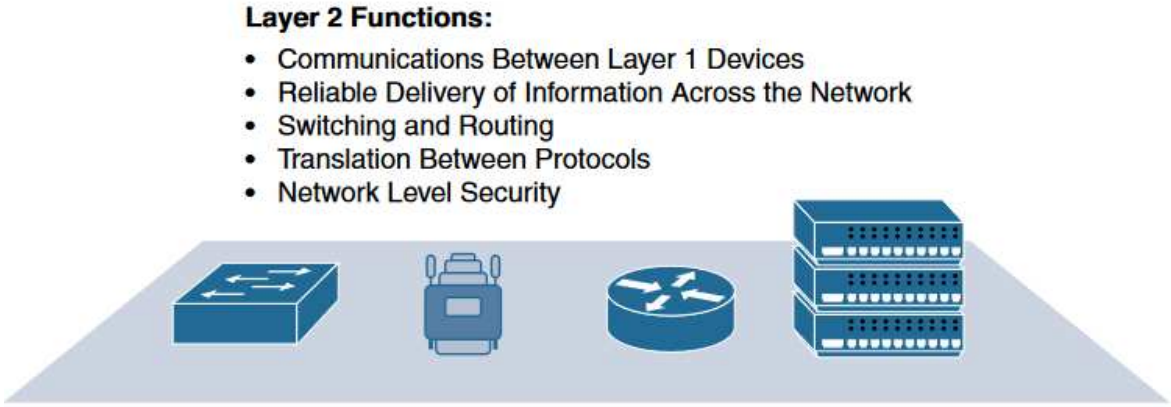
- Razlaganje IoT-a na manje dijelove

- Identificiranje različite tehnologije na svakom sloju i kako se one međusobno povezuju.
- Definicija sustava u kojem različiti dijelovi mogu biti pruženi od strane različitih dobavljača.
- Postojanje procesa definiranja sučelja koji vodi do interoperabilnosti.
- Određivanje sigurnosnog modela s više razina koji se provodi na prijelaznim točkama između razina.

Prvi sloj IoT modela je sloj fizičkih uređaja i kontrolera. Ovaj sloj je dom "stvari" u Internetu stvari, uključujući različite krajnje uređaje i senzore koji šalju i primaju informacije. Veličina ovih "stvari" može varirati od gotovo mikroskopskih senzora do ogromnih strojeva u tvornici. Njihova osnovna funkcija je generiranje podataka i mogućnost upita i/ili kontrole putem mreže. [2]

U drugom sloju IoT modela, fokus je na povezivosti. Najvažnija funkcija ovog sloja IoT-a je pouzdan i pravodoban prijenos podataka. Konkretno, to uključuje prijenos između uređaja na 1. sloju i mreže te između mreže i obrade informacija koja se odvija na 3. sloju (sloj računala na rubu mreže). [2]

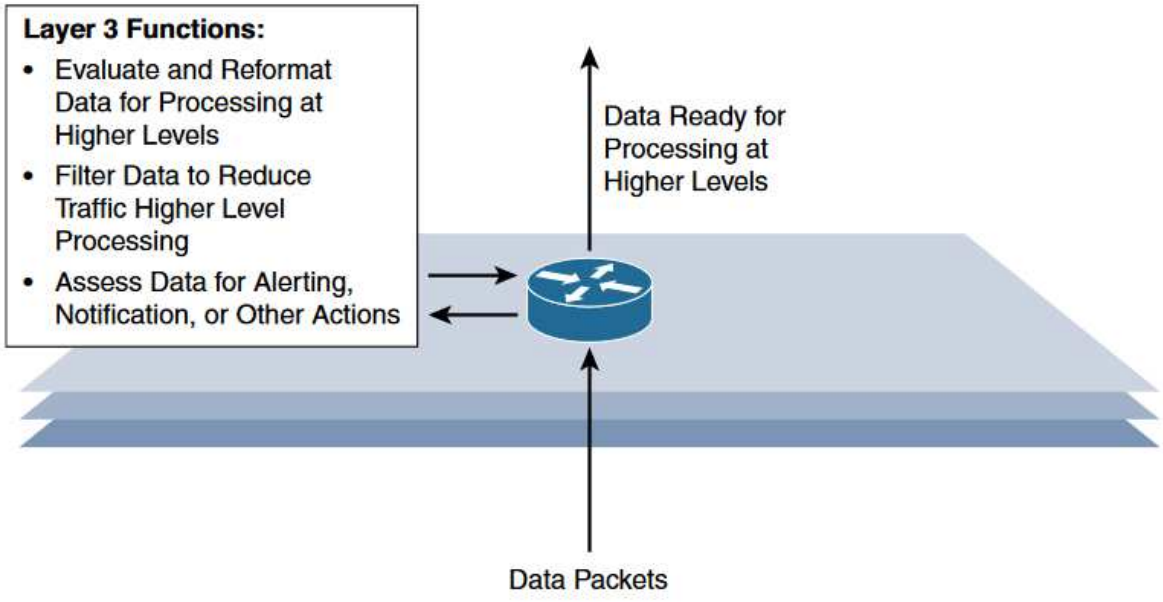
Sloj povezivosti obuhvaća sve mrežne elemente IoT-a i zapravo ne razlikuje se od rubne (mreže između senzora/uređaja i IoT gatewaya, o kojem će se raspravljati kasnije u ovom poglavlju), gatewaya i unutrašnje mreže. Funkcije sloja povezivosti prikazuje Sl. 2.2. [2]



Sl. 2.2 Sloj povezivosti IoT-a [2]

Rubno računanje je uloga trećeg sloja. Rubno računarstvo često nazivamo slojem računarstva u magli, također nazvano mrežom u magli ili zamagljivanje, to je decentralizirana računalna struktura smještenu između oblaka i uređaja koji proizvode

podatke. Na ovom sloju naglasak je na smanjenju podataka i pretvaranju mrežnih tokova podataka u informacije koje su spremne za pohranu i obradu od strane viših slojeva. Jedan od osnovnih principa ovog referentnog modela jest da se obrada informacija pokreće što je prije moguće i što bliže rubu mreže. Sl. 2.3 ističe funkcije koje obavlja 3. sloj IoT modela. Još jedna važna funkcija koja se odvija na trećem sloju je evaluacija podataka kako bi se vidjelo može li se filtrirati ili agregirati prije nego što se pošalje na viši sloj. To također omogućuje ponovno oblikovanje ili dekodiranje podataka, čime se olakšava dodatna obrada od strane drugih sustava. Dakle, ključna funkcija je procjena podataka kako bi se vidjelo jesu li premašene unaprijed definirane granice i treba li poslati akcije ili upozorenja. [2]



Sl. 2.3 Sloj rubnog računarstva IoT-a [2]

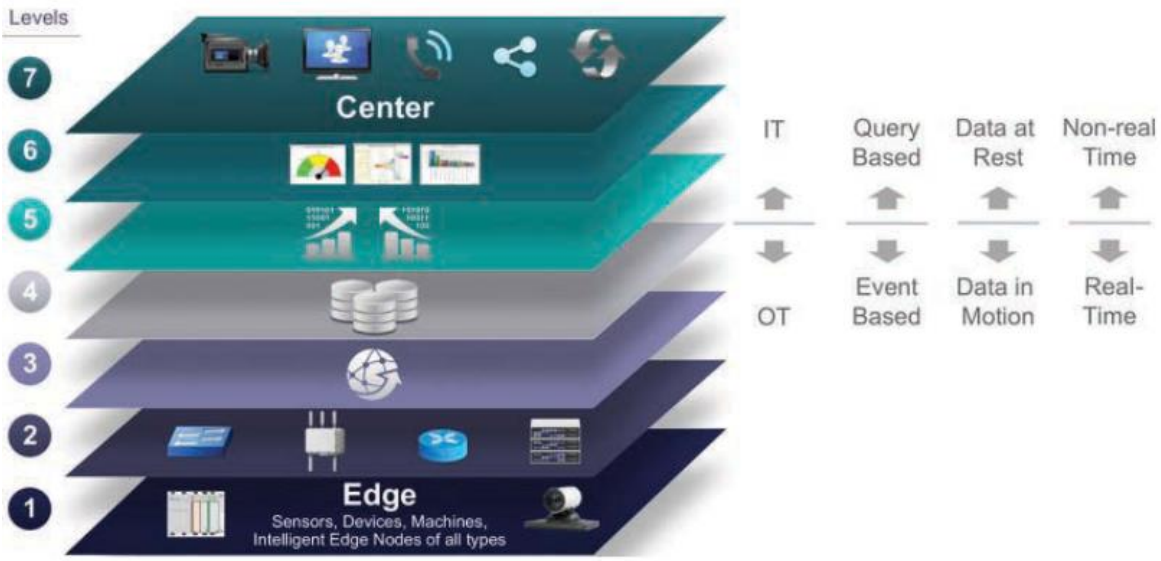
Četvrti sloj IoT modela vrši sakupljanje podataka i pohranjuje ih tako da su korisni za aplikacije kad je potrebno. Pretvara podatke temeljene na događajima u obradu temeljenu na upitima.

Sloj podatkovne apstrakcije ujedinjuje različite formate podataka i osigurava dosljednu semantiku iz različitih izvora. Potvrđuje da je skup podataka potpun i konsolidira podatke na jednom ili više mjesta pomoću virtualizacije.

Aplikacijski sloj tumači podatke pomoću softverskih aplikacija. Aplikacije mogu pratiti, kontrolirati i pružati izvještaje na temelju analize podataka.

Sloj suradnje i obrade konzumira i dijeli informacije aplikacija. Suradnja i komunikacija informacija IoT-a često zahtijevaju više koraka, što čini IoT korisnim. Ovaj sloj može promijeniti poslovne procese i donosi prednosti IoT-a.

Zanimljiv aspekt vizualizacije arhitekture IoT-a je organizacija odgovornosti duž IT i OT linija. Sl. 2.4 ilustrira prirodnu demarkacijsku točku između IT i OT na IoT modelu. Kao što prikazuje Sl. 2.4, sustavi IoT-a moraju prijeći nekoliko granica izvan funkcionalnih slojeva. Dno stoga uglavnom pripada domeni OT-a. U OT spadaju senzori i upravljačke jedinice,a vrh modela pripadaju poslužitelji, baze podataka i aplikacije. U prošlosti su OT i IT uglavnom bili neovisni i imali su malo potrebe za međusobnim komunikacijama. IoT mijenja taj paradigma. Na dnu, u slojevima OT-a, uređaji generiraju podatke u stvarnom vremenu prema vlastitoj stopi - ponekad ogromne količine svakodnevno. Ovo ne samo da rezultira ogromnom količinom podataka koji prolazi kroz IoT mrežu, već i sam volumen podataka sugerira da će aplikacije na vrhu sloja moći unositi toliko podataka po zahtjevanoj stopi. podaci se stoga moraju nakupljati ili pohranjivati na određenim točkama unutar IoT stoga. Slojevito upravljanje podacima na ovaj način diljem stoga pomaže gornjim četiri sloja obraditi podatke svojom brzinom. Kao rezultat toga, podaci u stvarnom vremenu "u pokretu" blizu ruba moraju biti organizirani i pohranjeni kako bi postali "podaci u mirovanju" za aplikacije u IT razinama. IT i OT organizacije moraju surađivati u ukupnom upravljanju podacima. [2]



Sl. 2.4 Podjela IoT modela duž IT i OT linija [2]

3. Komunikacijske tehnologije u Internetu stvari

Nakon uvoda u pojmove mreže i Interneta stvari u prethodnim poglavljima, ovo poglavlje je usredotočeno na komunikacijske tehnologije koje se najčešće koriste u komunikaciji OT slojeva IoTWF arhitekture Interneta stvari. Poglavlje sadrži definicije i opise protokola CoAP, MQTT, WebSocket i HTTP, s naglaskom na posljednji u inačicama 1, 1.1, 2 te 3 te služi kao uvod u implementacijski dio ovog rada, HTTP3 prototip, koji je opisan u sljedećem poglavlju.

3.1. CoAP

Protokol CoAP je jedna od najčešće korištenih protokola u komunikaciji Interneta stvari ,s stoga je u ovom potpoglavlu je naveden sažeti opsi CoAP protokola i njegovog modela slanja zahtjeva i odgovora kako bi se mogao usporediti s HTTP protokolom.

3.1.1. Opis protokola CoAP

CoAP, punog naziva Constrained Application Protocol, je specijalizirani transportni protokol namijenjen za korištenje na čvorovima ograničenih resursa (procesnih i memorijskih) i ograničenim mrežama (nestabilna veza, velik broj izgubljenih paketa i sl.). Čvorovi često imaju 8-bitne mikrokontrolere s malim procesnim i memorijskim resursima, te se koristi na ograničenim mrežama poput IPv6 preko bežičnih osobnih mreža s niskom snagom (6LoWPAN) koje često imaju visoke stope pogrešaka i tipični protok od 10-ak kbit/s. Protokol je dizajniran za komunikaciju stroj sa strojem (M2M) kao što je to slučaj u najnižim slojevima Interneta stvari. [3]

CoAP pruža model interakcije zahtjevima i odgovorima između aplikacijskih krajnjih točaka, podržava ugrađeno otkrivanje usluga i resursa, te uključuje ključne koncepte weba poput URI-ja i medija interneta. CoAP je dizajniran za jednostavno integriranje s HTTP-om radi integracije s webom, dok zadovoljava specijalizirane zahtjeve poput podrške za multicast, jednostavna zaglavlja paketa što ga čini jednostavnim i pogodnim za korištenje u prethodno opisanim uvjetima s ograničenjima. [3]

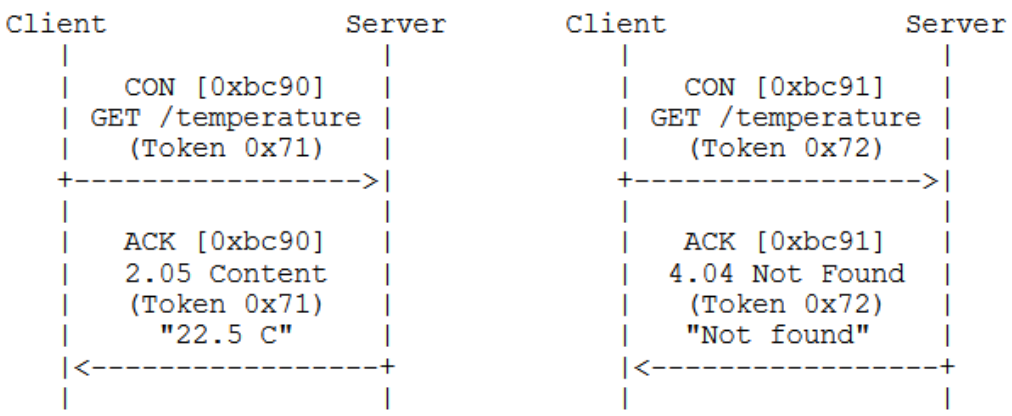
Model interakcije CoAP-a sličan je modelu klijent/server HTTP-a koji je opisan kasnije. Međutim, interakcije stroj-stroj obično rezultiraju implementacijom CoAP-a koja djeluje i kao klijent i kao poslužitelj. CoAP zahtjev je ekvivalentan HTTP zahtjevu i šalje ga klijent kako bi zatražio radnju (koristeći metode) na resursu (identificiranom URI-jem) na poslužitelju. Zatim poslužitelj šalje odgovor s kodom odgovora, a taj odgovor može uključivati prikaz resursa.

3.1.2. Model zahtjeva i odgovora CoAP protkola

CoAP poruke razmjenjuje asinkrono preko prijenosa orijentiranog na datagrame protokola UDP. To se postiže upotrebom poruka koji podržava opcionalnu pouzdanost. CoAP definira četiri vrste poruka: Confirmable, Non-confirmable, Acknowledgement, Reset (potvrđujuće, nepotvrđujuće, potvrda, reset). Nazivi metoda i kodovi odgovora u porukama čine ih nositeljima zahtjeva ili odgovora. Osnovne razmjene četiri vrste poruka donekle su neovisne o zahtjev/odgovor interakcijama; zahtjevi se mogu prenositi u potvrđujućim i nepotvrđujućim porukama, a se odgovori također mogu nositi i u njima, u tijelu poruka. [3]

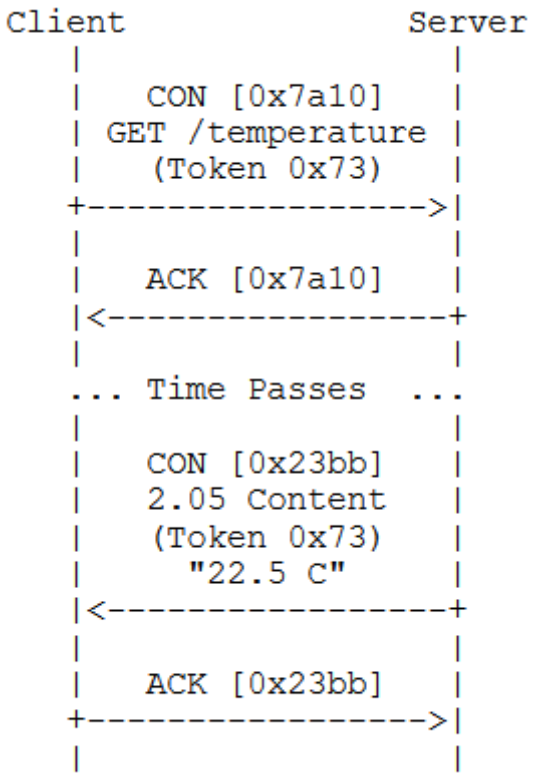
Semantika CoAP zahtjeva i odgovora prenosi se u CoAP porukama, koje uključuju ili naziv metode ili kod odgovora. Opcionalne (ili zadane) informacije zahtjeva i odgovora, poput URI-ja i tipa podatka koji se šalje, prenose se kao CoAP opcije. Token se koristi za podudaranje odgovora s zahtjevima neovisno o porukama koje se razmjenjuju. Važno je napomenuti da je Token koncept odvojen od ID-a poruke. [3]

Zahtjev se prenosi u potvrđujućoj (CON) ili nepotvrđujućoj (NON) poruci, a ako je odmah dostupan, odgovor na zahtjev prenesen u potvrđujućoj poruci prenosi se u rezultirajućoj potvrdi (ACK) poruci. Nema potrebe za posebnim potvrđivanjem sadržaja odgovora, jer će klijent ponovno poslati zahtjev ako je poruka potvrde koja prenosi sadržaj odgovora izgubljena. Dva primjera za osnovni GET zahtjev sa sadržajem odgovora prikazana su na Sl. 3,1, jedan uspješan, a drugi rezultira odgovorom s kodom 4.04 (Nije pronađeno). [3]



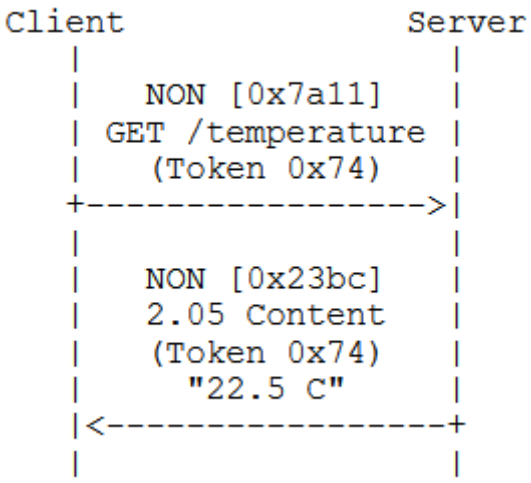
Sl. 3.1 Primjeri zahtjeva i odgvora CoAP protkola [3]

Ako poslužitelj nije u mogućnosti odmah odgovoriti na zahtjev prenesen u potvrđujućoj poruci, odgovara praznom potvrdnom porukom kako bi klijent mogao prestati ponovno prenositi zahtjev. Kada je odgovor spreman, poslužitelj ga šalje u novoj potvrđujućoj poruci (koja zatim treba biti potvrđena od strane klijenta). To se naziva odvojenim odgovorom, što je prikazano na Sl. 3.2.



Sl. 3.2 Primjer s GET zahtjevom i odvojenim odgovorom [3]

Ako se zahtjev pošalje u nepotvrđujućoj poruci, tada se odgovor šalje korištenjem nove nepotvrđujuće poruke, iako poslužitelj umjesto toga može poslati i potvrđujuću poruku. Ovaj tip razmjene poruka je prikazan na Sl. 3.3 u nastavku.



Sl. 3.3 Razmjena nepotvrđujućih poruka protokola CoAP [3]

CoAP koristi GET, PUT, POST i DELETE metode na sličan način kao HTTP, iako je detaljna semantika CoAP metoda gotovo u potpunosti slična semantici HTTP metoda, postoje razlike zbog kojih bi bilo potrebno koje proučiti specifikaciju iste, ali za potrebu ovog rada to nije potrebno jer CoAP protkol nije u fokusu implementacijskog dijela. [3]

Osim osnovne četiri, mogu se dodati metode u CoAP u odvojenim specifikacijama. Nove metode ne moraju nužno koristiti zahtjeve i odgovore u parovima. Čak i za postojeće metode, jedan zahtjev može rezultirati s više odgovora, npr., za multicast zahtjev. [3]

Podrška URI-ja u poslužitelju je pojednostavljena jer klijent već parsira URI i razdvaja ga na komponente poslužitelja, priključnice, putanje i upita, koristeći zadane vrijednosti za efikasnost. Kodovi odgovora čine mali podskup statusnih kodova HTTP-a s nekoliko dodanih CoAP-specifičnih kodova, poput kodova s prefiksom 6 koji služe za identifikaciju CoAP resursa i za lociranje istih. [3]

3.1.3. Zaključak o protokolu CoAP u Internetu stvari

Protokol CoAP zbog svojih karakteristika navedenih ranije je prikladan za korištenje u komunikaciji unutra okoline Interneta stvari zbog ograničenja iste, stoga je vrlo često korišten upravu u tu svrhu.

3.2. MQTT

U ovom potpoglavlju je opisan MQTT protkol, punog naziva Message Queuing Telemetry Transport, za razmjenu poruka za Internet stvari, objavi/pretplati model koji koristi pri komunikaciji te kontrolne poruke koje koristi. Namijenjen je za jednostavan prijenos poruka u objavi/pretplati obrascu koji je idealan za povezivanje udaljenih uređaja s ograničenim resursima. MQTT protkol je jedan od najkorišteniji protkola za komunikaciju u Internetu stvari.

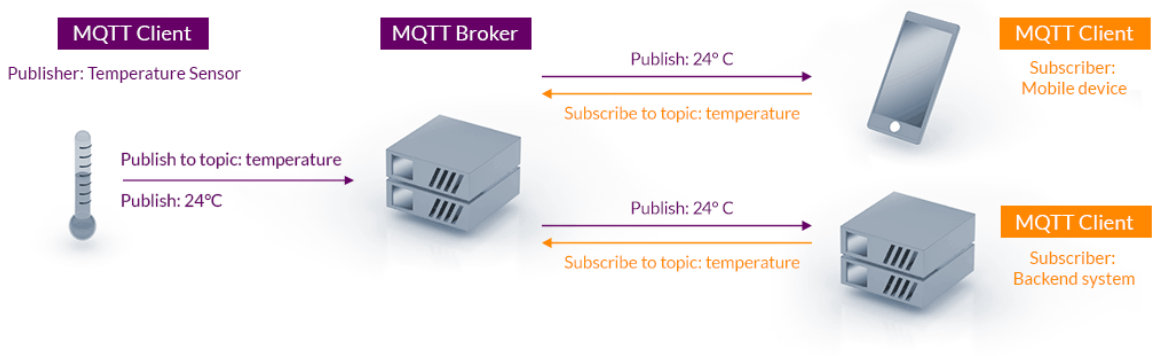
3.2.1. Opis protokola MQTT

MQTT-ov model razmjene poruka temelji se na temama na koje se mogu objaviti podaci i pretplata na iste preko kojih se dobiju podaci objavljeni na temi. Teme se identificiraju po nazivu na koji se poruke objavljuju i na koje se pretplaćuje. Teme su hijerarhijske i mogu sadržavati više razina odvojenih kosim crtama, poput putanja do datoteka u operacijskom sustavu. [4]

Pretplatom klijent određuje s koje teme klijent želi primati poruke. Kada klijent pretplaćuje na temu obaviještava posrednika (MQTT Broker) da želi primati poruke objavljene na toj temi. Posrednik zatim prati pretplatu i proslijeđuje bilo koje poruke objavljene na toj temi pretplaćenom klijentu. Klijent može istovremeno pretplatiti na više tema te tema može imati više pretplatnika. To omogućuje fleksibilan i skalabilan sustav razmjene poruka. Osim tema i pretplata, MQTT također podržava wildcard znakove koji se mogu koristiti za pretplatu na više tema koje odgovaraju određenom obrascu. Dva tipa zamjenskih znakova su zamjenski znak za jednu razinu (+), koji odgovara jednoj razini u temi, i zamjenski znak za više razina (#), koji odgovara svim razinama nakon određene razine u temi. MQTT-ov model razmjene poruka pruža fleksibilan i skalabilan način objave i pretplate na poruke koristeći teme i pretplate. Upotreba wildcard znakova dodaje dodatni sloj fleksibilnosti, omogućujući pretplate na više povezanih tema koristeći samo jednu pretplatu. [4]

Na Sl. 3.4 u nastavku je prikazana objavi/pretplati arhitektura MQTT koju čine posrednik (MQTT Broker) i klijenti (MQTT Client), od kojih jedan klijet, temperaturni senzor, objavljuje na temu s nazivom „temperature“ podatak o temperaturi, dok druga dva klijetna, koji su u ovom slučaju mobilni uređaj i backend sustav, pretplaćeni na temu „temperature“

te im posrednik šalje podatke koje je na temu „temperature“ objavi klijent temperaturni senzor.



Sl. 3.4 Prikaz objavi/pretplati arhitekture MQTT [17]

3.2.2. Objavi/pretplati model protkola MQTT

MQTT koristi model s posrednikom u kojem se klijenti povezuju s posrednikom, a poruke se objavljuju na temama. Pretplatnici zatim mogu pretplatiti na teme i primati objavljene poruke na istima.

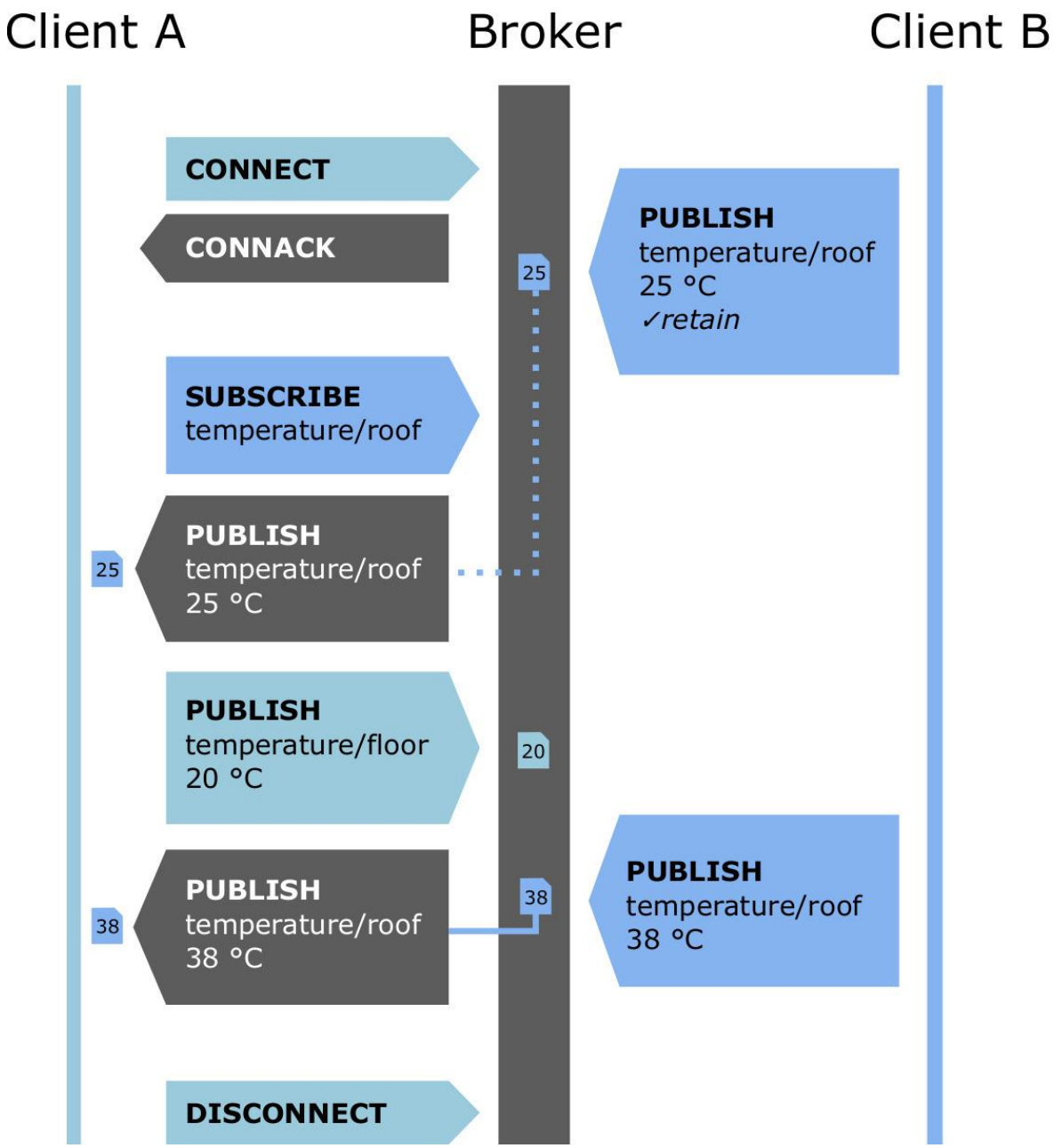
Arhitektura objavi/pretplati nudi alternativu tradicionalnim modelu zahtjev/odgovor. U pristupu zahtjev/odgovor, klijent izravno komunicira s poslužiteljem, stvarajući usko grlo koje umanjuje performanse. S druge strane, model objavi/pretplati razdvaja objavitelja poruke od pretplatnika. Objavitelj i pretplatnik nisu svjesni postojanja drugog. Kao treća komponenta, posrednik upravlja vezom između njih. Ovaj razvodnjeni pristup proizvodi brži i učinkovitiji proces komunikacije. Uklanjanjem potrebe za izravnom komunikacijom između objavitelja i pretplatnika, arhitektura objavi/pretplati uklanja razmjenu IP adresa i portova. Također pruža razdvajanje, omogućujući rad na obje komponente bez prekida tijekom objavljivanja ili primanja. Arhitektura objavi/pretplati nudi tri dimenzije razdvajanja za optimalnu učinkovitost: [4]

- Prostorna razdvojenost: objavitelj i pretplatnik ne moraju poznavati jedan drugoga.
- Vremenska razdvojenost: objavitelj i pretplatnik ne moraju raditi istovremeno
- Sinkronizacijska razdvojenost: obje komponente ne moraju prekidati rad tijekom objavljivanja ili primanja.

Jedna od najznačajnijih prednosti programske arhitekture objavi/pretplati je njena sposobnost filtriranja svih dolaznih poruka i njihova ispravna raspodjela pretplatnicima, čime se uklanja potreba da objavitelj i pretplatnik znaju za postojanje jedan drugoga.

Protokol MQTT koristi protokol TCP na transportnom sloju za slanje poruka.

3.2.3. Kontrolne poruke protokola MQTT



Sl. 3.5 Razmjena MQTT poruka između klijenata i posrednika [18]

Na Sl. 3.5 je prikazana razmjena poruka protokola MQTT objašnjenih u nastavku.

Protokol MQTT podržava kontrolne poruke tipova CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT te AUTH. [4]

Stoga što protokol MQTT nije u fokusu ovog rada, u nastavku će biti samo sažeto objašnjenje kontrolne poruke koje su prikazane na Sl. 3.5.

Kontrolnu poruku CONNECT klijent šalje posredniku kako bi se uspostavila veza TCP veza među klijentom i posrednikom za uspostavu dvosmjerne komunikacije na koju u slučaju uspješnoh povezivanja posrednik odgovara porukom CONNACK čime je veza klijenta i posrednika uspostavljena. [4]

Dodatne dvije poruke protokola MQTT koje su zanimljive u sklopu ovog rada su poruke PUBLISH i SUBSCRIBE jer se objavi/pretplati model koristi i u implementacijskom dijelu ovog rada. [4]

SUBSCRIBE poruka je poruka kojom klijent brokeru javlja da se želi pretplatiti na temu s nazivom koji je sadržan u tijelu poruke, sa Sl. 3.5 je to „temperature/roof“. Na tu poruku posrednik odgovara porukom SUBACK, što nije prikazano na istoj, čime potvrđuje klijentu pretplatu na temu. [4]

PUBLISH poruka služi za slanje podataka u oba smjera između klijenata i posrednika. PUBLISH porukom klijent može objaviti sadržaj poruke na temu ili od posrednika dobiti sadržaj objavljen na temu ako je na nju pretplaćen. Glavni elementi tijela poruke su naziv teme za koju se podatak šalje, bilo objava ili odgovor pretplatniku, te sam sadržaj. [4]

Detaljna objašnjena svih kontrolnih poruka te njihova struktura se mogu pronaći u literaturi [4] iz koje je prethodno izvučeno. Za potrebe ovog rada je prethodni sadržaj poglavlja MQTT dovoljan.

3.2.4. Zaključak o protokolu MQTT u Internetu stvari

Protokol MQTT zbog svojih karakteristika navedenih u prethodnim potpoglavljima je također prikladan izbor za komunikaciju unutar okoline Interneta stvari, ponajviše zbog korištenja objavi/pretplati athitekture te je analogno istoj napravljena arhitektura i u implementacijskom dijelu ovog rada.

3.3. WebSocket

U ovom potpoglavlju je opisan komunikacijski protkol WebSocket. WebsSocket nije toliko često korišten protkol u Internetu stvari, zbog karakteristika koje su iznese u nastavku, a najčešće se koristi u slučaju kada je potrebna stalna veza među uređajima Interneta stvari te je naveden u ovom radu jer poslužitelj HTTP3 prototipa koji je opisan u sljedećem poglavlju podržava protokol WebSocket za pretplaćivanje na teme u svom objavi/pretplati modelu.

3.3.1. Opis protokola WebSocket

Protokol WebSocket je dizajniran da zamijeni postojeće tehnologije za dvosmjernu komunikaciju koje koriste HTTP kao transportni sloj kako bi iskoristile postojeća infrastrukturu (proksiji, filtriranje, autentikacija). Takve tehnologije implementirane su kao kompromisi između učinkovitosti i pouzdanosti jer HTTP prvotno nije bio namijenjen za dvosmjernu komunikaciju. WebSocket pokušava ostvariti ciljeve postojećih dvosmjernih HTTP tehnologija u kontekstu postojeće HTTP infrastrukture; stoga je dizajniran da radi preko HTTP priključaka 80 i 443 te podržava HTTP proksije i posrednike, čak i ako to implicira određenu složenost specifičnu za trenutno okruženje. Međutim, dizajn ne ograničava WebSocket na HTTP, pa bi buduće implementacije mogle koristiti jednostavniji handshake preko zasebnog priključaka bez stvaranja potpuno novog protokola. Posljednje rečeno je važno jer obrasci prometa interaktivne komunikacije ne podudaraju se sa standardnim HTTP prometom i mogu uzrokovati neuobičajena, prevelika opterećenja na nekim komponentama. [5]

Protokol se sastoji od dva dijela, rukovanja i slanja podataka.

Nakon što su klijent i poslužitelj uspješno obavili rukovanje, počinje dio prijenosa podataka. Ovo je dvosmjerni kanal komunikacije gdje svaka strana može, neovisno o drugoj, slati podatke po želji. Nakon uspješnog rukovanja, klijenti i poslužitelji prenose podatke naprijed i natrag u konceptualnim jedinicama nazvanim "porukama" u ovoj specifikaciji. Svaka poruka je sastavljena od jednog ili više okvira. Svaki okvir koji pripada istoj poruci sadrži istu vrstu podataka. Općenito govoreći, postoje tipovi za tekstualne podatke, binarne podatke i kontrolne okvire (koji nisu namijenjeni za prijenos podataka aplikaciji, već za signaliziranje na razini protokola, poput signaliziranja da bi

veza trebala biti zatvorena). Protokol definira šest vrsta okvira i ostavlja deset rezerviranih za buduću upotrebu. [5]

3.3.2. Početno rukovanje

Kada klijent započne WebSocket vezu, šalje svoj dio rukovanja. Poslužitelj mora analizirati barem dio ovog rukovanja kako bi dobio potrebne informacije za generiranje dijela rukovanja poslužitelja. Klijentovo otvarajuće rukovanje sastoji se od sljedećih dijelova. Ako poslužitelj, tijekom čitanja rukovanja, utvrdi da klijent nije poslao rukovanje koje odgovara opisu u nastavku, uključujući, ali ne ograničavajući se na, bilo kakva kršenja ABNF gramatike specificirane za komponente rukovanja, poslužitelj mora zaustaviti obradu klijentovog rukovanja i vratiti HTTP odgovor s odgovarajućim greškom. [5]

- HTTP/1.1 ili viši GET zahtjev, uključujući URI zahtjeva koji bi trebao biti tumačen kao /naziv resursa/ (ili apsolutni HTTP/HTTPS URI koji sadrži /naziv resursa/).
- Polje zaglavlja |Host| koje sadrži autoritet poslužitelja.
- Polje zaglavlja |Upgrade| koje sadrži vrijednost "websocket", tretira se kao ASCII vrijednost bez obzira na velika i mala slova.
- Polje zaglavlja |Connection| koje uključuje sadržaj "Upgrade", tretira se kao ASCII vrijednost bez obzira na velika i mala slova.
- Polje zaglavlja |Sec-WebSocket-Key| s vrijednošću kodiranom u base64, koja, kada se dekodira, ima duljinu od 16 bajtova.
- Polje zaglavlja |Sec-WebSocket-Version|, s vrijednošću 13.
- Po želji, polje zaglavlja |Origin|. Ovo polje zaglavlja šalju svi pretraživači. Pokušaj uspostave veze koji nedostaje ovog polja zaglavlja ne bi trebao biti tumačen kao zahtjev pretraživača.
- Po potrebi, polje zaglavlja |Sec-WebSocket-Protocol|, s popisom vrijednosti koje označavaju koje protokole klijent želi koristiti, poredane po željenom redoslijedu.
- Po potrebi, polje zaglavlja |Sec-WebSocket-Extensions|, s popisom vrijednosti koje označavaju koja proširenja klijent želi koristiti.

- Po potrebi, moguće su i druga zaglavlja, poput onih korištenih za slanje kolačića ili zahtjeva za autentifikaciju na poslužitelju. Nepoznata zaglavlja se ignoriraju.

Kada klijent uspostavi WebSocket vezu s poslužiteljem, poslužitelj mora dovršiti sljedeće korake kako bi prihvatio vezu i poslao rukovanje poslužitelja. [5]

- Ako se veza događa na HTTPS (HTTP preko TLS) portu, obavite TLS handshake preko veze. Ako to ne uspije, tada zatvorite vezu; inače, sva daljnja komunikacija za vezu mora se odvijati kroz enkriptirani tunel.
- Poslužitelj može izvršiti dodatnu autentifikaciju klijenta
- Poslužitelj može preusmjeriti klijenta koristeći statusni kod s prefiksom 3. Napomena, ovaj korak može dogoditi zajedno s, prije ili nakon opcionalnog koraka autentifikacije opisanog ranije.
- Poslužitelj mora provjeriti i odrediti sljedeće informacije: U zaglavljima |Origin|, |Sec-WebSocket-Key|, |Sec-WebSocket-Version|, identikator usluge koju nudi poslužitelj te proširenja protkola koje protokola koje održava te u slučaju da klijent pošalje neispravne ili pogrešne podatke poslužitelj mora odbiti daljnu komunikaciju i zatvoriti vezu WebSocketom.
- Ako poslužitelj odluči prihvatiti dolaznu vezu, mora odgovoriti s valjanom HTTP odgovorom koji označava sljedeće: Statusni kod 101, zaglavlje |Upgrade| sa sadržajem „websocket“, zaglavlje |Connection| sa sadržajem „Upgrade“, zaglavljem |Sec-WebSocket-Accept| koje ima sadržaj opisan u literaturi koja je korištena kao izvor u ovom potpoglavlju, zaglavlje |Sec-WebSocket-Protocol| koje sadrži verziju protkola te zaglavlje |Sec-WebSocket-Extensions| ako je pri rukovanju klijent poslao zahtjev za korištenjem prošitenja.

Ovim se završava rukovanje poslužitelja. Ako poslužitelj dovrši ove korake bez prekida rukovanja WebSocketom, poslužitelj smatra da je veza WebSocketa uspostavljena i da je veza WebSocketa u OPEN stanju. U ovom trenutku, poslužitelj može početi slati (i primati) podatke, kao što to može i klijent. [5]

3.3.3. Slanje i primanje podataka protkolom WebSocket

Okviri s podacima identificirani su opcodeovima gdje je najznačajniji bit opcodea 0. Trenutno definirani opcodeovi za okvire s podacima uključuju 0x1 (Tekst), 0x2 (Binarni). Opcodovi 0x3-0x7 rezervirani su za daljnje okvire s podacima koji još nisu definirani.Okviri s podacima nose podatke na razini aplikacije i/ili na razini proširenja. Opcode određuje interpretaciju podataka: [5]

- Text: podaci su tekstualni podaci kodirani kao UTF-8. Napomena: određeni tekstualni okvir može uključivati djelomični niz UTF-8, međutim, cijela poruka morala bi sadržavati valjan UTF-8.
- Binarni: podaci su proizvoljni binarni podaci čiju interpretaciju isključivo određuje sloj aplikacije.

Da bi primio WebSocket podatke, krajnja točka sluša mrežnu vezu. Dolazni podaci moraju biti parsirani kao WebSocket okviri. Ako je primljen kontrolni okvir, okvir mora biti obrađen prema definiciji obrade kontrolnih paketa. Nakon primanja okvira s podacima krajnja točka mora zabilježiti vrstu podataka kao što je definirano opcode-om. Podaci iz ovog okvira definirani su kao podaci poruke. Ako okvir čini nefragmentirana poruka, kaže se da je primjena WebSocket poruka s navedenom prethodno određenom vrstom i podacima. Ako je okvir dio fragmentirane poruke, podaci iz sljedećih okvira s podacima su spojeni kako bi se oblikovali ukupni podaci. Kada se primi zadnji fragment kako je naznačeno FIN bitom (frame-fin), kaže se da je primljena WebSocket poruka s podacima sastavljenim spajanjem fragmenata u prethodnim okvirima vrsta određena iz prvog okvira koji sadrži prvi fragment podataka. Naknadni okviri s podacima moraju biti interpretirani kao pripadajući novoj WebSocket poruci. Proširenja mogu promijeniti semantiku kako se podaci čitaju, posebno uključujući što čini granicu poruke. Proširenja, osim dodavanja podatak o proširenju prije podatka u poruci, mogu također modificirati podatke (na primjer, komprimiranjem). Poslužitelj mora ukloniti maskiranje za okvire s podacima primljenim od klijenta. [5]

3.3.4. Zatvaranje veze

Za zatvaranje WebSocket veze, zatvara se osnovna TCP vezu. Trebalo bi koristiti metodu koja čisto zatvara TCP vezu, kao i TLS sesiju, ako je primjenjivo, odbacujući bilo kakve preostale bajtove koji su mogli biti primljeni. Krajnja točka može zatvoriti vezu putem bilo

koje dostupne metode kada je potrebno, poput slučaja napada. Osnovna TCP veza, u većini normalnih slučajeva, trebala bi biti prvo zatvorena od strane poslužitelja, tako da zadržava stanje TIME_WAIT, a ne klijent. U abnormalnim slučajevima (poput nedobivanja TCP zahtjeva za zatvaranje veze od poslužitelja nakon razumnog vremenskog razdoblja), klijent može pokrenuti TCP zatvaranje. Kao takvo, kada je poslužitelju naređeno da Zatvori WebSocket vezu, trebalo bi odmah pokrenuti TCP zatvaranje, a kada je klijentu naređeno da isto učini, trebalo čekati TCP zatvaranje od poslužitelja. [5]

Za pokretanje završnog rukovanja Zatvaranja WebSocket-a s statusnim kodom i opcionalnim razlogom zatvaranja, krajnja točka morat će poslati kontrolni okvir za zatvaranje, s postavljenim statusnim kodom i razlogom. Nakon što je krajnja točka poslala i primila kontrolni okvir za zatvaranje, krajnja točka trebala zatvoriti WebSocket vezu kako je opisano u prethodnom odlomku.

Kada se pošalje ili primi kontrolni okvir za zatvaranje, kaže se da je pokrenuto rukovanje tatvaranja WebSocket-a i da je veza WebSocket-a u stanju zatvaranja (CLOSING). [5]

Kada je osnovna TCP veza zatvorena, kaže se da je veza WebSocket-a zatvorena i da je veza WebSocket-a u stanju zatvoreno (CLOSED). Ako je TCP veza zatvorena nakon što je završeno rukovanje zatvaranja WebSocket-a, kaže se da je veza WebSocket-a zatvorena čisto, a ako veza WebSocket-a nije mogla biti uspostavljena, također se kaže da je veza WebSocket-a zatvorena, ali ne čisto. [5]

Dodatna objašnjena razloga zatvaranja WebSocket veze kao i statusni kodovi su objašnjeni u literaturi [5] vezanoj za ovo poglavlje, ali za potrebe ovoga rada nije potrebno dalje opisivati isto.

3.3.5. Zaključak o protokolu WebSocket u Internetu stvari

Protokol WebSocket zbog korištenja TCP veze te zbog složenog procesa rukovanja pri otvaranju i zatvaranju veze u okolini Interneta stvari s ranije navedenim ograničenjima nije najpogodnije rješenje za korištenje pri komunikaciji uređaja okoline. WebSocket je koristan u primjeni gdje je potrebno imati stalnu vezu između uređaja ako za tim ima potrebe, a u implementacijskom dijelu ovog rada poslužitelj uz HTTP3 podržava i komunikaciju WebSocketom te je stoga isti naveden i pojašnjen u ovom potpoglavlju.

3.4. HTTP

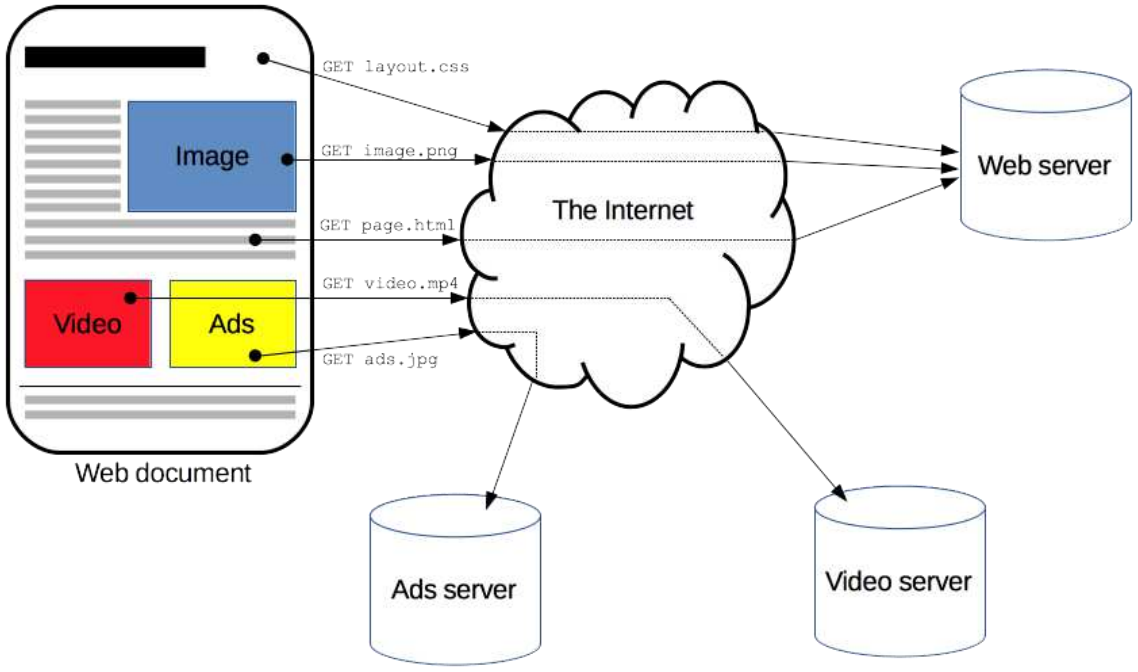
U ovom potpoglavlju je opisan protokol HTTP, punog naziva Hyper Text Transfer Protocol, koji je u fokusu ovoga rada, te su navedene osnovne značajke HTTP protkola koje su zajedničke svim inačicama, a u nastavku su u svakom potpoglavlju objašnjenje razlike i unapređenja u odnosu na prethodnu inačicu protokola HTTP.

3.4.1. HTTP općenito

Hyper Text Transfer Protocol, skraćeno HTTP, je beskontekstualni protokol na aplikacijskom sloju za raspodijeljenje, suradničke, hipertekstualne informacijske sustave. U ovom potpoglavlju je opisana opća arhitekturu HTTP-a, uspostavljena zajednička terminologija i definirani aspekti protokola koji su zajednički svim inačicama. U ovoj definiciji sadržani su osnovni elementi protokola, mehanizmi proširivosti te "http" i "https" sheme uniformih identifikatora resursa (URI).

HTTP je skup beskontekstualnih, protokola na aplikacijskom sloju athitekture zahtjev/odgovor koja dijeli generičko sučelje, proširive semantike i samoopisne poruke kako bi se omogućila fleksibilnu interakcija mrežnih sustava. HTTP skriva detalje o tome kako je usluga implementirana tako što klijentima predstavlja jednolično sučelje neovisno o vrstama pruženih resursa. Slično tome, poslužitelji ne moraju biti svjesni svrhe svakog klijenta: zahtjev se može razmatrati izolirano umjesto da bude povezan s određenom vrstom klijenta ili unaprijed određenim slijedom koraka aplikacije. To omogućuje da se implementacije općeg namjenskog koriste učinkovito u različitim kontekstima, smanjuje složenost interakcije i omogućuje neovisnu evoluciju tijekom vremena. HTTP je također dizajniran za upotrebu kao posrednički protokol, gdje proxy serveri i preusmjerivači mogu prevesti informacijske sustave koji nisu HTTP u generičko sučelje. Jedna posljedica ove fleksibilnosti je što se protokol ne može definirati u smislu onoga što se događa iza sučelja. Umjesto toga, ograničeni smo na definiranje sintakse komunikacije, namjere primljene komunikacije i očekivano ponašanje primatelja. Ako se komunikacija promatra izolirano, tada bi uspješne radnje trebale biti vidljive kroz odgovarajuće promjene u opaženom sučelju koje pružaju poslužitelji. Međutim, budući da se više klijenata može istovremeno aktivirati, možda čak i s različitim ciljevima, ne možemo zahtijevati da takve promjene budu opažene izvan okvira jednog odgovora. [6]

HTTP pruža jedinstveno sučelje za interakciju s resursom, bez obzira na njegovu vrstu, prirodu ili implementaciju, slanjem poruka koje manipuliraju ili prenose prikaze rezultata. Svaka poruka je ili zahtjev ili odgovor. Klijent konstruira poruke zahtjeva koje komuniciraju njegove namjere i usmjerava te poruke prema identificiranom poslužitelju. Poslužitelj osluškuje zahtjeve, parsira svaku primljenu poruku, tumači semantiku poruke u vezi s identificiranim ciljnim resursom i odgovara na taj zahtjev s jednom ili više poruka odgovora. Klijent pregledava primljene odgovore kako bi vidio jesu li njegove namjere provedene, određujući što učiniti sljedeće na temelju statusnih kodova i sadržaja primljenih poruka. HTTP semantika uključuje namjere definirane svakom metodom zahtjeva, proširenja tih semantika koja se mogu opisati u zaglavljima zahtjeva, statusne kodove koji opisuju odgovor i druge kontrolne podatke i metapodatke resursa koji se mogu dati u poljima odgovora. Semantika također uključuje metapodatke reprezentacije koji opisuju kako sadržaj treba biti interpretiran od primatelja, zaglavlja zahtjeva koja mogu utjecati na odabir sadržaja, i razne algoritme odabira koji se zajednički nazivaju "pregovaranje o sadržaju". [6]



Sl. 3.6 Primjer dohvaćanja web dokumenta protokolom HTTP [19]

Na Sl. 3.6 se može vidjeti primjer kako se korištenjem GET metoda protokola HTTP dohvaća sadržaj web dokumenta. Metode protokola HTTP i način razmjene poruka su objašneni u nastavku.

HTTP komunikacija se zasniva na modelu zahtjev/odgovor pri čemu na svaki poslani zahtjev klijenta poslužitelj mora odgovoriti s prikladnim odgovorom. Svi HTTP zahtjevi su sačinjeni od sljedećih elemenata: [6]

- Metoda: radnja koju se želi izvršiti nad nekim resursom i ovaj element je obvezan.
- Domaćin (Host): vezu na poslužitelja, poveznica ili ip adresa, na kojem će se izvršiti zahtjev, obavezan element zahtjeva
- Putanja: putanja do resursa na kojem se vrši radnja iz metode na hostu, identificira resurs na poslužitelju preko mrežne putanje na tom poslužitelju te je također obvezan element zahtjeva.
- HTTP inačica: inačica protokola HTTP koja se koristi u komunikaciji.
- Zaglavlja: element HTTP zahtjeva koje prenosi dodatni kontekst i metapodatke o zahtjevu. Ovaj element nije obavezan u svakom zahtjevu.
- Niz upita (Query String): dio URL-a koji pridružuje vrijednosti navedenim parametrima, kako bi poslužitelju naznačilo koji sadržaj treba prenijeti ili što s njim učiniti. Navodi se na kraju URL-a odvojen znakom „?“ nakon kojeg se navode nazivi parametara te se simbolom „=“ pridružuju vrijednosti parametrima. Moguće je poslati više parametara u nizu upita povezanih znakom „&“. Također neobavezan element upita.
- Tijelo: sadrži podatke koje se prenosi od klijenta do poslužitelja. Nije obvezno.

HTTP odgovor je slične strukture kao i zahtjev te se sastoji od: [6]

- HTTP inačice: jednako zahtjevu, obvezan element
- Statusni kod: kod koji predstavlja uspješnost izvođenja radnje zahtjeva, obvezan element
- Statusna poruka: kratki infromativni opis statusnog koda namijenjen korisnikovom lakšem razumijevanju statusnog koda. Također obvezan element.
- Zaglavlja: jednako kao i u zahtjevu, neobvezan element.
- Tijelo: jednako zahtjevu, sadrži podatke koji se prenose od poslužitelja klijentu. Nije obvezno.

U nastavku su navedene metode protokola HTTP i njihova značenja.

HTTP definira skup metoda zahtjeva kako bi naznačio željenu radnju koja treba biti izvršena za određeni resurs. Iako također mogu biti imenice, ove metode zahtjeva ponekad se nazivaju i HTTP glagolima. Svaka od njih implementira različiti semantički značaj, ali grupe metoda dijele neke zajedničke značajke. Metoda zahtjeva može biti sigurna, idempotentna ili pogodna za spremanje u privremenu memoriju.

Metode HTTP-a su: [6]

- GET: zahtijeva prikaz određenog resursa. Zahtjevi korištenjem GET metode bi trebali samo raditi dohvat podatka.
- POST: šalje podatke na određeni resurs, često uzrokujući promjenu stanja ili druge radnje na poslužitelju.
- PUT: zamjenjuje prikaze resursa podacima iz poslanog zahtjeva.
- PATCH: djelomično izmjenju prikaz resursa podacima iz poslanog zahtjeva.
- DELETE: briše naznačeni resurs.
- HEAD: zahtjeva odgovor kao i kod zahtjeva s metodom GET, ali bez tijela odgovora.
- CONNECT: uspostavlja dvosmjernu komunikaciju s resursom. Može se koristiti za tuneliranje TCP veze te kao rezultat zahtjeva s ovom metodom se može uspostaviti dvosmjerna veza kojom i klijent i poslužitlej mogu slati podatke.
- OPTIONS: zahtjeva dopuštene mogućnosti komunikacije s danim resursom ili poslužiteljem.
- TRACE: odgovor na zahtjev s ovom metodom je mrežna putanja do resursa i primarno služi kao mehanizam za otkrivanje i ispravljanje pogrešaka.

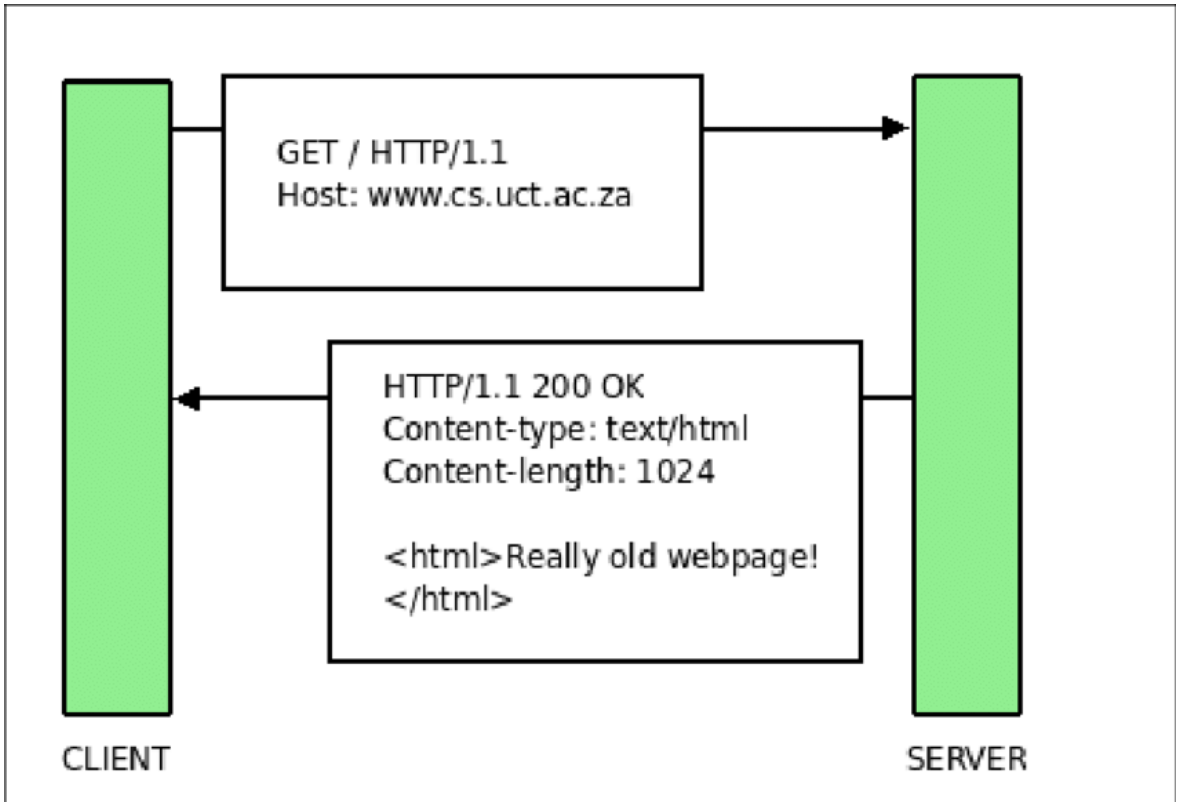
Statusni kodovi HTTP odgovora naznačuju je li određeni HTTP zahtjev uspješno izvršen. Kodovi su u rasponu od 100 do 599, a odgovori su grupirani u sljedeće klase po statusnim kodovima: [6]

- Informativni: kodovi 100 do 199.
- Uspješni: kodovi 200 do 299.
- Preusmjeravanje: kodovi 300 do 399.
- Greška klijenta: kodovi 400 do 499.

- Greška poslužitelja: kodovi 500 do 599.

U svakoj klasi postoje su već unaprijed definirane statusne poruke za statusne kodove, dok su preostali kodovi ostavljeni za mogućnost dodavanja novih statusnih poruka ovisno o primjeni. U sklopu ovog rada nije potrebno navoditi već definirane statusne kodove HTTP odgovora jer se isti mogu pronaći u priloženoj literaturi [6].

Na Sl. 3.7 je prikazan primjer HTTP zahtjeva s metodom GET, inačicom HTTP/1.1, hostom „www.cs.uct.ac.za“ te putanjom „/“. Odgovor poslužitelja je također u inačici HTTP/1.1, sa statusnim kodom „200“, opisom statusnog koda „OK“, zaglavljima „Content-type“ u kojem je sadržan tip podataka koji su u tijelu odgovora te „Content-length“ zaglavlje koje označava duljinu tijela odgovora izraženu u bajtovima. Tijelo odgovora sadrži html datoteku sadržaja „Really old webpage!“.



Sl. 3.7 Primjer HTTP zahtjeva i odgovora. [20]

U ovom potpoglavlju je još dodatno potrebno spomenuti HTTPS, odnosno, HTTP preko TLS/SSL-a, pošto je TLS/SSL sastavni dio HTTP3 protkola korištenog pri izradi prototipa u praktičom dijelu ovog rada. Glavni cilj TLS-a je pružiti siguran kanal za komunikaciju, jedini zahtjev za prijenosni sloj je pouzdan, poredan podatkovni tok.

Konkretno, siguran kanal trebao bi pružiti sljedeće osobine: [6]

- Autentifikacija: Strana poslužitelja kanala uvijek je autentificirana; strana klijenta se opcionalno autentificira. Autentifikacija se može dogoditi putem asimetrične kriptografije (npr. RSA, algoritam digitalnog potpisa eliptične krivulje (ECDSA) ili algoritam digitalnog potpisa Edwards-ove krivulje (EdDSA)) ili simetričnog unaprijed podijeljenog ključa (PSK).
- Povjerljivost: Podaci poslani preko kanala nakon uspostave vidljivi su samo pošiljatelju i primatelju. TLS ne skriva duljinu podataka koje prenosi, iako krajnje točke mogu dodati isječke u TLS zapise kako bi sakrile duljine i poboljšale zaštitu protiv tehnika analize prometa.
- Cjelovitost: Podaci poslani preko kanala nakon uspostave ne mogu biti mijenjani od strane napadača bez otkrivanja.

Siguran kanal mora imati navedena svojstva čak i slučaju kada napadač ima potpunu kontorlu na mrežom kojom pošiljatelj i primatelj komuniciraju.

TLS se sastoji od dvije sastavne komponente: [12, 13]

- Protokol rukovanja koji autentificira komunicirajuće strane, pregovara o kriptografskim načinima i parametrima, te uspostavlja zajednički materijal (ključeve) za enkripciju. Protokol rukovanja je dizajniran da odoli manipulaciji; aktivni napadač ne bi trebao biti u mogućnosti prisiliti parove da pregovaraju o različitim parametrima nego što bi to učinili da veza nije pod napadom.
- Protokol zapisa koji koristi parametre uspostavljene rukovanjem kako bi zaštitio promet između komunicirajućih parova. Protokol zapisa dijeli promet u seriju zapisa, pri čemu je svaki neovisno zaštićen koristeći ključeve za enkripciju.

Dakle, TLS je neovisan o aplikacijskom protokolu; viši protokoli mogu koristiti ispod sebe TLS-a transparentno. Međutim, TLS standard ne specificira kako protokoli dodaju sigurnost pomoću TLS-a; kako pokrenuti rukovanje TLS-om i kako tumačiti autentifikacijske certifikate razmijenjene ostavlja se na procjenu dizajnera i implementatora protokola koji se izvode u sloju iznad TLS-a. Sve inačice TLS-a uključuju mehanizam verzioniranja koji omogućuje klijentima i poslužiteljima da međusobno pregovaraju o zajedničkoj inačici koju podržavaju oba sudionika u komunikaciji. [13]

U implementacijskom dijelu ovog rada je korištena inačica TLS-a 1.3, više detalja o istoj se može pronaći u priloženoj literaturi [13]. TLS je unaprijeđenje SSL-a [11].

U sljedećim potpoglavljima će biti opisane specifičnosti svake inačice protokola HTTP s usporedbom i poboljšanjima u odnosu na prethodnu.

3.4.2. HTTP/0.9 i HTTP/1

Prvotno samo HTTP, kasnije dodana inačica u naziv HTTP/0.9 je bio prva inačica protkola HTTP u kojem su se zahtjevi sastojali od samo jedne linije i samo metode GET s putanjom do resursa. Puni URL nije bio uključen u HTTP/0.9 jer su protokol, poslužitelj i priključnica već bili postavljeni pri uspostavi veze. Odgovori HTTP/0.9 su se satojali samo od html datoteke koja bi bila poslana kao odgovor. Zahtjevi i odgovori nisu imali zaglavlja, a odgovori nisu imali stausne kodove pa bi u slučaju neuspjeha odgovor sadržavao izgeneriranu html datoteku s opisom uzroka neuspješne obrade zahtjeva. [6]

Inačica 1 je uvela slanje inačice protokola HTTP u zahtjevu, vraćanje statusnog koda u odgovoru kako bi se moglo razlikovati uspješne od neuspješno obrađenih zahtjeva, uvedena su zaglavlja kako bi povećala fleksibilnost i proširivost samog protokola. Uvođenje zaglavlja je omogućilo slanje odgovora u drugom formatu, ne samo kao html datoteke, slanjem zaglavlja „Content-type“. [6]

HTTP/0.9 i HTTP/1 su koristili TCP vezu na transpotnom sloju što je uzrokovalo nastajanje head-of-line blocking problema opisanog u posljednjem potpoglavlju ovog poglavlja. [6]

3.4.3. HTTP/1.1

HTTP/1.1 je prva standardizirana inačica objavljena nedugo nakon HTTP/1.

Ovaj protkol je razjasnio nejasnoće i uveo brojna unapređenja: [8]

- Ista TCP veza se mogla ponovno koristiti, što je štedjelo vrijeme. Više nije bilo potrebno ponovno otvarati TCP vezu kako bi se prikazali resursi ugrađeni u pojedinačni originalni dokument.
- Dodana je tehnika pipelininga. To je omogućilo slanje drugog zahtjeva prije nego što je odgovor na prvi u potpunosti prenesen. To je smanjilo kašnjenja komunikacije.
- Rascjepkavanje odgovora u više pojedinačnih odgovora koji bi se na odredištu spajali u jedan.

- Uvedeni su dodatni mehanizmi za privremeno čuvanje podatka. [7]
- Pregovaranje sadržaja razmjene, uključujući jezik, kodiranje i vrstu sadržaja. Klijent i poslužitelj mogu dogovoriti kakav sadržaj razmijeniti.
- Zaglavlje "Host", omogućilo je hostanje različitih domena s iste IP adrese te omogućila ko-lokaciju poslužitelja.

Najveća promjena koju donosi HTTP/1.1 je da umjesto slanja zahtjeva i odgovora preko osnovnog TCP/IP stoga, kreiran je dodatni enkriptirani sloj prijenosa na iznad toga, SSL. SSL 1.0 nikada nije bio objavljen javnosti, ali SSL 2.0 i njegov nasljednik SSL 3.0 omogućili su stvaranje web stranica za e-trgovinu. Da bi to postigli, enkriptirali su i zajamčili autentičnost poruka razmijenjenih između poslužitelja i klijenta. SSL je konačno standardiziran i postao je TLS. U istom vremenskom razdoblju postalo je jasno da je potrebna enkriptirana transportni sloj. Web više nije bio većinom akademska mreža, već je postao okruženje u kojem potencijalni zlonamjerni korisnici žele uhvatit i iskoristi privatne podatke. Kako su aplikacije izgrađene preko HTTP-a postajale moćnije i zahtijevale pristup privatnim informacijama poput adresara, e-pošte i lokacije korisnika, TLS je postao potreban izvan uporabe e-trgovine. [8]

Iako HTTP/1.1 uvodi brojna poboljšanja u odnosu na prethodnu inačicu, idalje ne rješava problem head-of-line blockinga, čiji nastanak i posljedice, već spomenuto, opisani u posljednjem potpoglavlju.

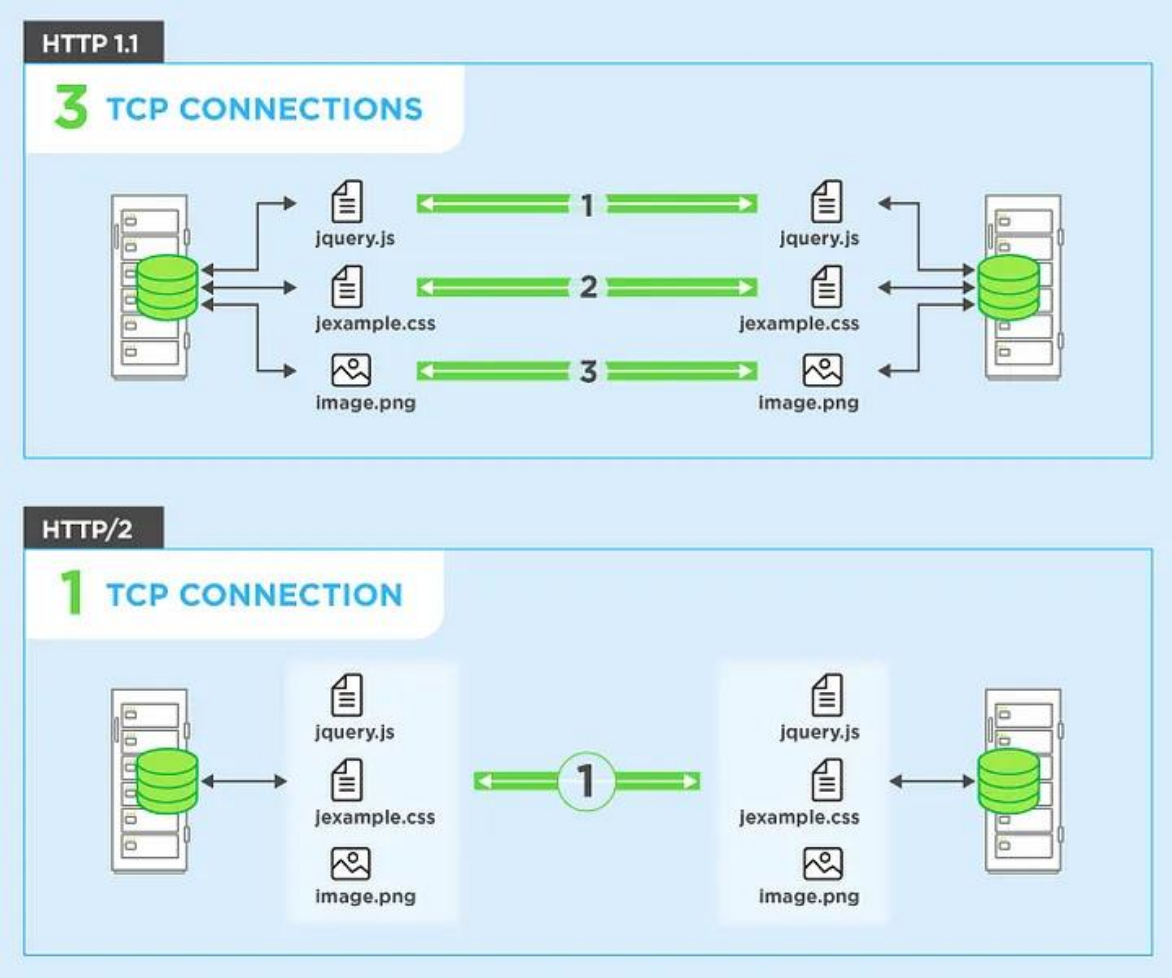
3.4.4. HTTP/2

Razvojem samog interneta te aplikacija i servisa koji se koriste njime, mnogo više podataka prenosilo se putem značajno više HTTP zahtjeva, što je stvaralo veću složenost i opterećenje za HTTP/1.1 veze. Uzimajući u obzir ovo povećanje složenosti, uvedena je inačica HTTP/2 kao alternativni način razmjene podatak između klijenata i poslužitelja te time skratilo vrijeme odziva. [9]

HTTP/2 inačica protkola HTTP u odnosu na HTTP/1 ima sljedeće razlike: [9]

- Binarni protokol umjesto tekstualnog protokola. Ne može se ručno čitati i stvarati. Unatoč tome, omogućava implementaciju poboljšanih tehnika optimizacije.
- Multipleksirani protokol. Paralelni zahtjevi mogu se izvršiti preko iste veze, uklanjajući ograničenja prethodnih inačica protkola HTTP.

- Kompresira zaglavlja. Često su slična među skupom zahtjeva, stoga, kompresijom uklanja dupliciranje i količinu podataka koji se prenose.
- Omogućava poslužitelju da popuni podatke u predmemoriji klijenta putem mehanizma nazvanog "server push".



Sl. 3.8 Usporedba TCP veze HTTP/1.1 i HTTP/2 [21]

Na Sl. 3.8 se može vidjeti kako multipleksiranjem na jednoj TCP vezi omogućuje korištenje iste za slanje HTTP zahtjeva, u odnosu na HTTP1.1 koji bi za isti rezultat otvorio tri različite TCP veze.

Iako je u tom pogledu HTTP riješio problem head-of-line blockinga koji je imao HTTP/1.1 i ranije inačice, i dalje nije riješen problem head-of-line blocking na samoj TCP vezi.

Taj problem rješava sljedeća inačica protokola HTTP opisana u sljedećem potpoglavlju, koristeći protokol UDP umjesto TCP-a na transportnom sloju.

3.4.5. HTTP/3

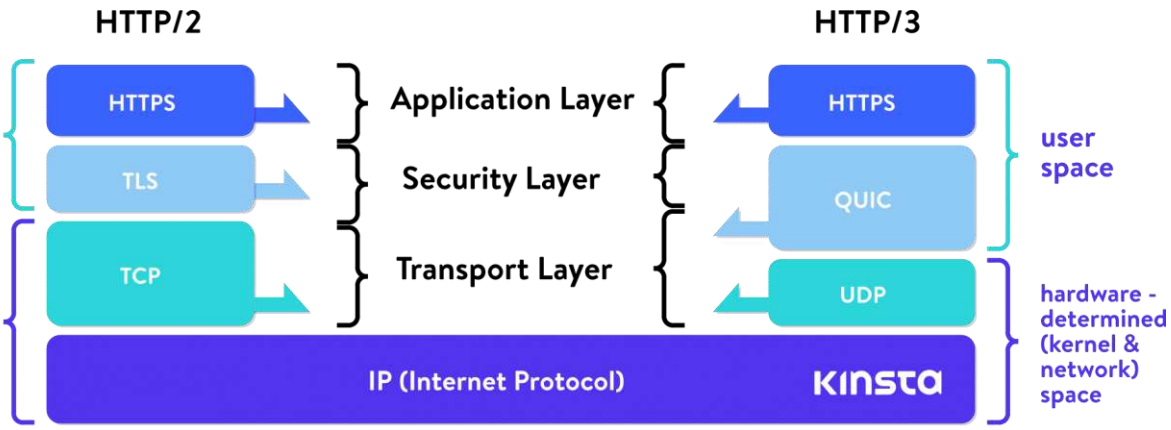
HTTP/3 je najnovija inačica protokola HTTP koja koristi transportni protokol QUIC, punog naziva Quick UDP Internet Connections, na transportnom sloju. Razvijen je kako bi riješio neka od ograničenja i problema s performansama HTTP/2 preko TCP-a. [10]

Karakteristike protokola HTTP/3 su sljedeće: [10]

- HTTP/3 koristi transportni protokol QUIC umjesto TCP-a. QUIC radi preko protkola UDP, punog naziva User Datagram Protocol, te pruža nekoliko prednosti u odnosu na TCP, uključujući brže uspostavljanje veze, poboljšanu kontrolu zagušenja i unaprijeđene sigurnosne značajke.
- QUIC pruža multipleksiranje temeljeno na tokovima, omogućavajući prijenos više tokova podataka istovremeno preko jedne veze. To omogućuje učinkovitiju uporabu resursa i smanjeno kašenje. Osim toga, QUIC uključuje mehanizme kontrole protoka kako bi regulirao protok podataka između klijenta i poslužitelja, sprječavajući jednu stranu da preplavi drugu podacima. Multipleksiranje je izrađeno po uzoru na multipleksiranje TCP veze u inačici HTTP/2.
- Postupak otkrivanja HTTP/3 krajnje točke detaljno je opisan u navedenoj literaturi. Obično uključuje DNS rezoluciju radi dobivanja IP adrese poslužitelja koji hosta HTTP/3 krajnju točku. Nakon što klijent dobije IP adresu poslužitelja, uspostavlja se QUIC veza s tom krajnjom točkom.
- HTTP/3 slaže poruke u okvire po uzoru na inačicu HTTP/2-u. HTTP zahtjevi i odgovori se razlažu i pakiraju u manje manje jedinice nazvane okviri, koje se zatim prenose preko QUIC veze. Ova značajka optimizira prijenos HTTP poruka i omogućuje kompresije zaglavlja i prioritetizaciju okvira.

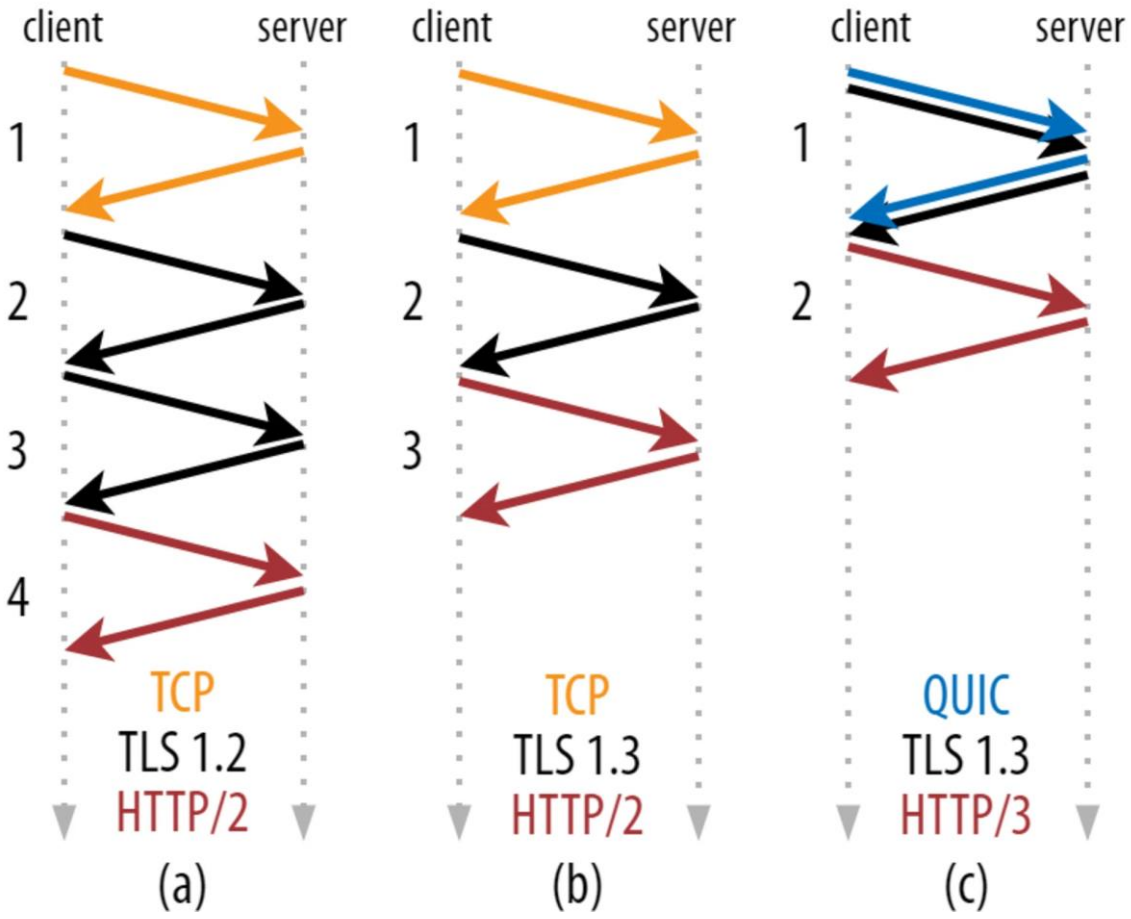
HTTP/3 kombinira poznate semantike HTTP-a s poboljšanim performansama i sigurnosnim značajkama transportnog protokola QUIC. Iskorištavanjem mogućnosti QUIC-a, HTTP/3 pruža brže, pouzdanije iskustvo korištenja te smanjiti kašnjenja u komunikaciji.

QUIC je novi transportni protokol koji pruža niz naprednih značajki. Iako je prvotno dizajniran za upotrebu s HTTP-om, pruža mogućnosti koje se mogu koristiti s mnogo širim spektrom aplikacija. QUIC je enkapsuliran u UDP-u. Inačica 1 QUIC-a integrira TLS 1.3 kako bi šifrirala sve podatke o teretu i većinu kontrolnih informacija. [14]



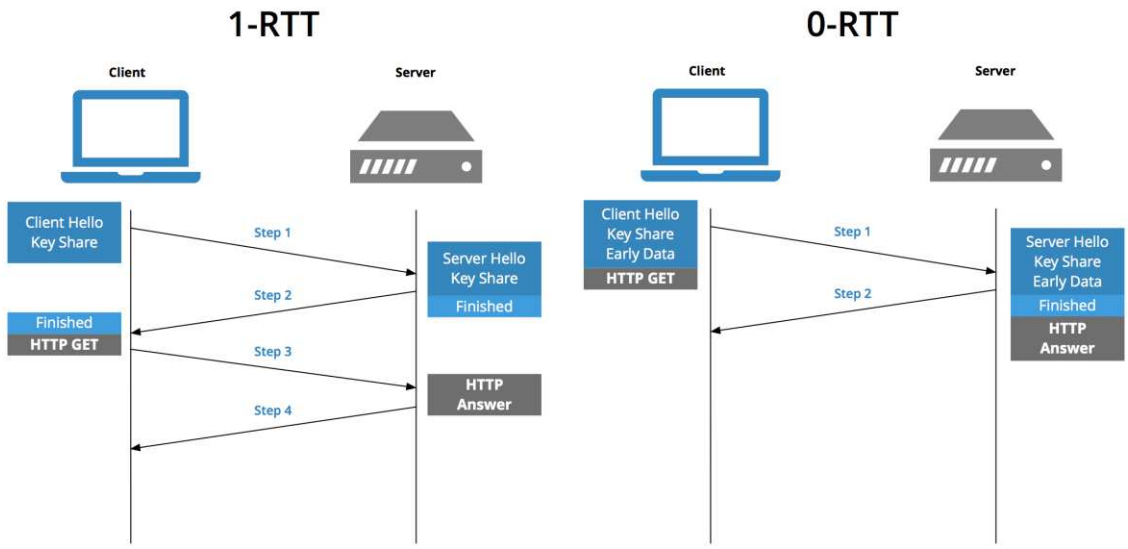
Sl. 3.9 Usporedba HTTP/2 i HTTP/3 stoga [22]

Na Sl. 3.9 se može vidjeti usporedba stoga inačica HTTP/2 i HTTP/3. Na slici je vidljivo da je u HTTP/3 inačici QUIC protocol i na sigurnosnom i na transportnom sloju jer u sebi integrira protokol TLS, kako je prethodno spomenuto, i da je na transportnom sloju korišten protokol UDP u odnosu na TCP koji je korišten u HTTP/2 inačici.



Sl. 3.10 Usporedba uspostavljanja veze HTTP/2 i HTTP/3 [23]

Na prethodnoj Sl. 3.10 se može usporediti uspostavljanje veze protokolima HTTP/2 i HTTP/3. Na prvom dijagramu se može vidjeti uspostavljanje veze korištenjem protokola HTTP/2 s TLS 1.2. Nakon uspostavljanja TCP veze u jednom obilasku (1-RTT), potrebno je odraditi TLS rukovanje koje se sastoji od 2 obilaska mreže (2-RTT) između klijenta i poslužitelja te tek nakon toga uspostavi HTTP/2 veza, ukupno u 3 obilaska mreže (3-RTT). Na drugom dijagramu je prikazano uspostavljanje veze korištenjem protokola HTTP/2 s TLS 1.3. Najbitnija razlika u odnosu na prethodni je korištenje novije inačice TLS-a koja uspostavlja rukovanje u samo jednom obilasku mreže (1-RTT) pa je ukupan broj obilazaka mreže pri uspostavi HTTP/2 veze 2 (2-RTT). Treći dijagram na slici prikazuje uspostavljanje HTTP/3 veze i možemo uočiti da je veza uspostavljena već u jednom obilasku mreže (1rtt) jer protokol QUIC u sebi ima integriran TLS 1.3 pa je uspostava HTTP/3 za jedan obilazak mreže kraća. Dodatno se može napomenuti da protokol TLS 1.3 omogućava privremeno spremanje podataka o enkripciji, stoga podržava ponovno uspostavu, u razumno kratkom roku nakon prekida prethodne veze, bez obilazak mreže (0-RTT). To je prikazano na Sl. 3.11.



Sl. 3.11 Usporedba 1-RTT i 0-RTT uspostavljanja veze [24]

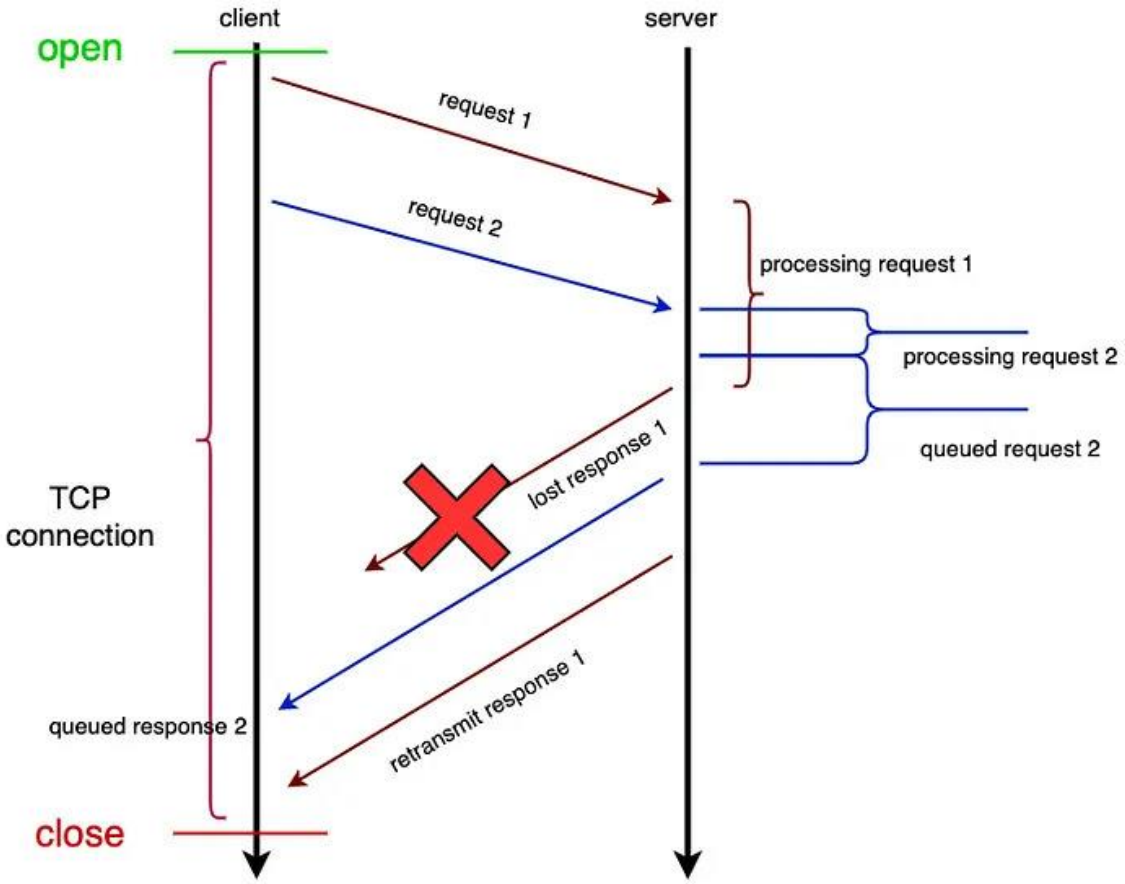
Dakle, korištenjem QUIC transportnog protkola koji radi preko protkola UDP te koristeći multipleksiranje veze i stavljanjem poruka u okvire po uzoru na HTTP/2, HTTP/3 protkol zaobilazi problem head-of-line blockinga koji je uzrokovao smanjenje performansi sustava pri korištenju prethodnih inačica HTTP. [10, 14]

3.4.6. Head-of-line blocking

Head-of-line blocking, odnosno zastoj prvoga u redu, skraćeno HOL, je problem koji nastaje kada zbog prethodnog zahtjeva ili paketa svi nadolazeći paketi također kasne u isporuci. Ovaj problem je inherentan svim protokolima koji koriste TCP na transportnom sloju pa tako i protokolu HTTP. [15]

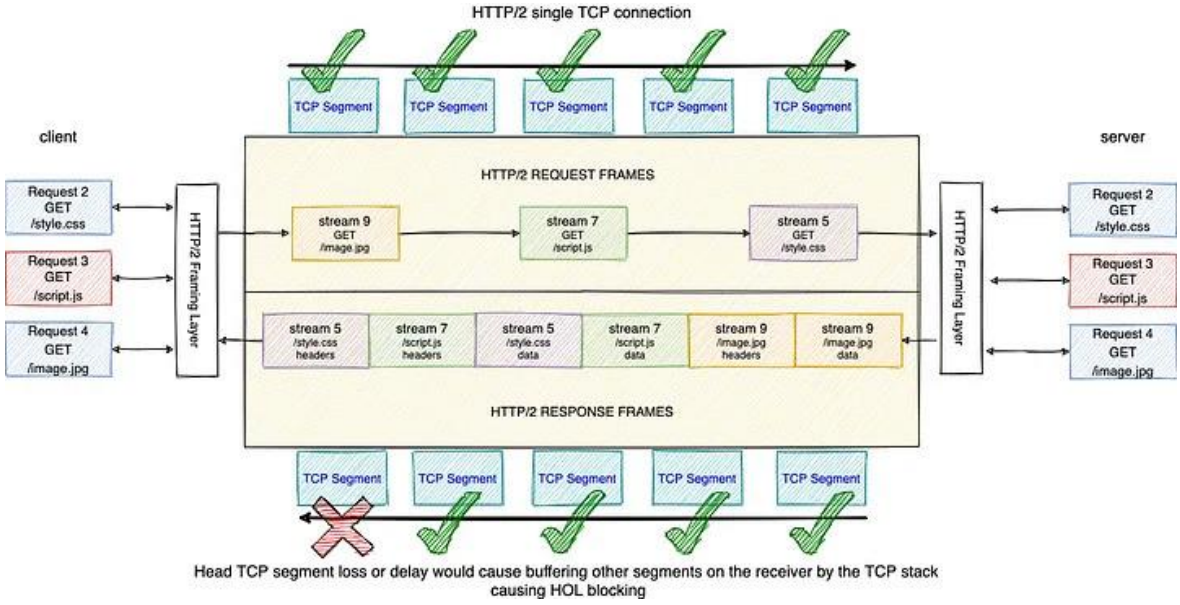
U prvim inačicama HTTP-a, HTTP/0.9 i HTTP/1, HOL problem bi se očitovao tako što bi svi zahtjevi nakon onog koji je trenutno u obradi čekali dok se isti ne izvrši neovisno jesu li međusobno ovisni ili ne.

Kod protokola HTTP/1.1 HOL problem se očituje kod korištenja tehnike pipelininga, korištenja iste TCP veze za slanje novih zahtjeva iako prethodni nije dobio odgovor. Na Sl. 3.12 se može vidjeti kako gubljenjem „response 1“ „response 2“ mora čekati dok prethodni odgovor ne stigne, u ovom slučaju dok ne završi njegova retransmisija.



Sl. 3.12 HTTP/1.1 HOL blocking [25]

Protokolom HTTP/2 se riješio problem HOL blockinga na aplikacijskom sloju protokola HTTP/2, ali je problem HOL blockinga idalje ostao na transportnom sloju jer isti još uvijek koristi TCP kao transportni protkol, tj. zbog načina rada protokola TCP, u slučaju gubitka jednog TCP okvira, svi nadolazeći okviri moraju čekati dolazak istog. Na Sl. 3.13 je prikazan HOL blocking na transportnom sloju protokola TCP.



Sl. 3.13 TCP HOL blocking [26]

3.4.7. Zaključak o protokolu HTTP u Internetu stvari

Na osnovu općenitog opisa protokola HTTP te raspisanih razlika u inačicama istog, može se uočiti kako je protokol HTTP, koji bi se koristio na slojevima iznad drugog sloja povezivosti IoTWF modela arhitekture Interneta stvari, prilično jednostavno moguće integrirati s protokolom CoAP, koji bi se koristio na najnižem sloju, sloju fizičkih uređaja i kontrolera. Međutim inačice 0.9 i 1 ne zadovoljavaju ponajprije uvjet sigurnosti Interneta stvari, dok inačice 1.1 i 2 iako zadovoljavaju sigurnosni aspekt Interneta stvari zbog mogućnosti nastanka head-of-line blockinga i korištenja TCP veze na transportnom sloju također nisu uvijek prikladno rješenje. HTTP/3 koristi protokol QUIC preko UDP-a na transportnom sloju čime zaobilazi problem head-of-line blockinga, a TLS je sastavni dio istog što daje sigurnost komunikaciji, što ga čini mogućim rješenjem za komunikaciju u Internetu stvari pa je u nastavku ovoga rada izabran kao komunikacijski protokol za implementacijski dio, kako bi se moglo analizom ostvarene komunikacije korištenjem prototipa potvrditi je li HTTP/3 prikladan protokol za korištenje u Internetu stvari.

4. Imeplementacija komunikacije uređaja IoT korištenjem protokola HTTP3

U ovom poglavlju je opisana implementacija HTTP/3 prototipa koji u komunikaciji klijenta i poslužitelja koristi protokol HTTP/3.

4.1. HTTP3 Poslužitelj

Za potrebu implementacije HTTP/3 poslužitelja korišten je NGINX softver otvorenog koda koji može služiti za posluživanje weba, obrnuti proxy, predmemoriranje, uravnoteženje opterećenja, strujanje medija i još mnogo toga. Počeo je kao web poslužitelj dizajniran za maksimalnu performansu i stabilnost. Osim svojih sposobnosti poslužitelja HTTP-a, NGINX može funkcionirati i kao proxy poslužitelj za e-poštu (IMAP, POP3 i SMTP) te kao obrnuti proxy i uravnotežitelj opterećenja za poslužitelje HTTP, TCP i UDP. [27]

NGINX je podržan na više različitih distribucija Linux platforme što se može vidjeti na [28], a također se može koristiti i na UNIX platformi te u ograničenoj formi i na Windows distribucijama. U [28] se može vidjeti da je NGINX podržan i na cloud platformama poput Amazon AWS-a, Microsoft Azure, Google Cloud Engine te IBM LinuxONE, ali zbog potrebe za korištenjem protokola HTTP/3 je poslužitelj ovog prototipa pokrenut na osobnom računalu s operacijskim sustavom Ubuntu 22.04 LTS, jer navedene cloud platforme, što je utvrđeno ispitivanjem korištenja istih, ne podržavaju komunikaciju QUIC protkolom te se komunikacija s ovom implementacijom HTTP/3 poslužitelja vrati na protokol HTTP/1.1 ili HTTP/2 ako je isti omogućen, što se može definirati u konfiguraciji NGINX poslužitelja koja će biti prikazana u nastavku. [27]

Korištena je inačica NGINX 1.25.3 koja se može preuzeti s poveznice [29] te je potrebno izgradit iz izvornog koda aplikaciju NGINX koja služi za pokretanje samog poslužitelja.

Uz osnovni izvorni kod, poslužitelj je proširen modulom NCHAN kako bi se podržala arhitektura objavi/pretplati, čije detalje se može pronaći na poveznici iz literature [30].

U nastavku je opisan postupak izgradnje iz izvorno koda, izvođenje konfiguracijske skripte, izgradnja, konfiguriranje te pokretanje NGINX poslužitelja na operacijskom sustavu Linux.

Radi lakšeg izvođenja koraka u nastavku, bez problema oko prava izvođenja naredbi te pristupanju datotekama, preporuka je prvo izvršiti naredbu `sudo` s parametrom `-i` kako bi se korisniku dozvolio pristup `/root` direktoriju operacijskog sustava za čije izvođenje je potrebno unijet superuser lozinku.

Prije same izgradnje NGINX-a iz izvornog koda potrebno je pokrenuti sljedeću naredbu:

```
apt-get install build-essential zlib1g zlib1g-dev libssl-dev
libpcre3 libpcre3-dev git
```

Kôd 4.1 Naredba za instaliranje potrebnih za izgradnju NGINX-a iz izvornog koda

Da bi se preuzeo izvorni kod NGINX poslužitelja potrebno je koristiti naredbu `wget` s parametrom [29], odnosno poveznicom za preuzimanje izvornog koda NGINX-a.

```
wget http://nginx.org/download/nginx-1.25.3.tar.gz
```

Kôd 4.2 Nareba za preuzimanje NGINX izvornog koda

Zatim je potrebno naredbom `tar` s parametrima `-xvfz` te nazivom prethodno preuzete tar datoteke raspakirati tar datoteku u mapu s istim nazivom.

```
tar -xvfz nginx-1.25.3.tar.gz
```

Kôd 4.3 Naredba za raspakiranje tar datoteke

Nakon raspakiranja je potrebno se premjestiti naredbom `cd` u novonastalu mapu.

U novonastaloj mapi je potrebno naredbom `git clone` i parametrom [31], poveznicom na GitHub repozitorij u kojem se nalazi modul NCHAN, preuzeti izvorni kod modula.

```
git clone https://github.com/slact/nchan.git
```

Kôd 4.4 Git naredba za preuzimanje izvornog koda modula NCHAN

Nakon preuzimanja potrebnih izvornih kodova potrebno je pokrenuti `configure` skriptu pozicioniranjem u mapu u kojoj je raspakiran izvorni kod NGINX-a.

```
./configure --sbin-path=/usr/bin/nginx --conf-
path=/etc/nginx/nginx.conf --error-log-
path=/var/log/nginx/error.log --http-log-
path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid
--add-module=/root/nginx-1.25.3/nchan --with-pcre --with-
debug --with-http_v2_module --with-http_ssl_module --with-
```

```
http_v3_module --with-cc-opt="-I../libressl/build/include" --
with-ld-opt="-L../libressl/build/lib"
```

Kôd 4.5 Naredba za pokretanje „configure“ skripte s potrebnim parametrima

Po izvršenju skripte `configure` potrebno je izvršiti naredbe `make` te `make install` kako bi se izradila NGINX aplikacija te kako bi se mogao pokrenut NGINX poslužitelj.

Nginx poslužitelj se pokreće naredbom `nginx` bez parametara, a dodavanjem parametra `-s` na istu se može poslužitelju poslati signale, od kojih su za dalju implementaciju najkorisniji signali `stop` za zaustavljanje istog te signal `reload` za učitavanje promjena.

```
nginx -s stop
```

Kôd 4.6 Naredba za slanje signala za zaustavljanje poslužitelja

NGINX aplikacija također ima i parametar `-t` za naredbu `nginx` kojom provjerava sintaksu napisane konfiguracije i izrazito je koristan pri ispravljanju grešaka pri konfiguriranju poslužitelja.

Nakon izgradnje i instaliranja NGINX poslužitelja, potrebno je konfigurirati poslužitelj.

```
worker_processes auto;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    access_log /var/log/nginx/diplomski.xyz/access.log;
    error_log  /var/log/nginx/diplomski.xyz/error.log;

    sendfile on;

    include /etc/nginx/conf.d/*.conf;
}
```

Kôd 4.7 Konfiguracija nginx.conf

Kôd 4.7 predstavlja konfiguraciju sadržanu u datoteci `/etc/nginx/nginx.conf`. Da bi se mogla koristit ova konfiguracija potrebno je napraviti mape `/etc/nginx/conf.d/` i `/var/log/nginx/diplomski.xyz` naredbom `mkdir` s

parametrom naziva mape kako bi se u njih moglo spremiti potrebne resurse. U prvoj mapi je sadržana HTTP/3 konfiguracija koja je dana u nastavku, dok se u drugu mapu po ovoj konfiguraciji spremaju zapisi pristupa i pogrešaka.

```
server {
    listen 80 default_server;
    return 307 https://$host/$request_uri;
}

server {
        listen 443 quic reuseport default_server;
        listen 443 ssl;
        quic_retry on;
        ssl_early_data on;
        http3 on;
        http3_hq on;

        server_name www.diplomski.xyz;

        ssl_certificate    /etc/ssl/server.crt;
        ssl_certificate_key    /etc/ssl/server.key;

  location / {
     alias /usr/local/nginx/html/;
     try_files index.html =404;
     add_header alt-svc 'h3=":$server_port"; ma=864000';
     add_header x-quic 'h3';
     add_header cache-control 'no-cache,no-store';
  }

  location = /sub {
     nchan_subscriber;
     nchan_channel_id $arg_id;
     add_header alt-svc 'h3=":$server_port"; ma=864000';
     add_header x-quic 'h3';
  }

  location = /pub {
      nchan_publisher;
      nchan_channel_id $arg_id;
      add_header alt-svc 'h3=":$server_port"; ma=864000';
```

```
        add_header x-quic 'h3';
    }
}
```

Kôd 4.8 Konfiguracija http3.conf

Kôd 4.8 prikazuje konfiguraciju poslužitelja kojom se omogućuje korištenje protokola HTTP/3 u komunikaciji s klijentom.

U prvom bloku koda `server` postavljeno je preusmjeravanje na priključnici 80 na HTTPS koji je definiran u sljedećem `server` bloku. Svrha ovog bloka je samo preusmjeriti vezu klijenta na sigurni kanal.

U drugom `server` bloku je naredbama `listen` postavljeno slušanje prometa protokola QUIC i SSL na priključnici 443. Naredba `reuseport` potrebna kako bi se naznačilo da su protokol QUIC i SSL na istoj priključnici, inače bi javilo sintaksnu grešku da je priključnica već iskorištena. Iako je protokol definiran kao `ssl` koristi se TLS 1.3. Na sljedećim linija su navedeni omogućeni mehanizmi protkola HTTP/3 i QUIC.

`server_name` predstavlja naziv poslužitelja koji može biti IP adresa ili naziv domene, u ovoj konfiguraciji je to naziv domene `www.diplomski.xyz` koja je korištena jer za uspješan rad protokola HTTP/3 na NGINX poslužitelju mora biti uspješno obavljeno TLS rukovanje, a s nazivom domene je lakše izgenerirati certifikate te dodati po potrebi CA certifikat izdavatelja certifikata jer je CA certifikat potreban na strani klijenta što je objašnjeno u sljedećem potpoglavlju.

Linije s naredbama `ssl_certificate` i `ssl_certificate_key` se postavljaju rute do certifikata i ključa certifikata u svrhu potpisivanja zahtjeva i odgovora u sigurnom kanalu na priključnici 443.

Blokovi `location` služe za definiranje resursa na lokacijama definiranih znakovnim nizom prije početka bloka. Na lokaciji / se nalazi početna stranica NGINX poslužitelja i služi samo za provjeru vrste veze s poslužiteljem iz pretražitelja.

Lokacija `/sub` služi za pretplatu na temu s nazivom sadržanim u upitnom nizu URI-ja zahtjeva, na primjer `id=1`, predstavlja temu s nazivom 1, a naredbe `add_header` služe za dodavanje zaglavlja HTTP odgovora poslužitelja kojima klijentu koji nije prethodno koristio protokol HTTP/3 obznanio mogućnost korištenja HTTP/3 na priključnici 443.

Na lokaciji `/pub` se na analogan način prijavljuje objavljivač na temu određenu iz upitnog znakovnog niza na isti način kao i za blok pretplaćivanja.

U nastavku je dana kratka uputa za izdavanje samopotpisanih SSL certifikata po uzoru na uputu na poveznici [32].

Za izdavanje samopotpisanih SSL certifikata potrebno je najprije stvoriti CA certifikat (Certificate Authority ili izdavatelja certifikata), koji je potreban kako bi se moglo izdat certifikat samog NGINX poslužitelja.

Za izdavanje certifikata se može koristit alat OpenSSL [33] kojim se mogu izgenerirati ključevi za potpis i sami certifikati.

Pregledom [32] se može pročitati detaljna uputa za izdavanje samopotpisanih SSL certifikata, a za potrebe ovog rada u nastavku je dana sažeta uputa po koracima koje je potrebno napraviti.

Dakle, korištenjem OpenSSL alata potrebno je izdati CA certifikat i njegov ključ koji služi za izdavanje certifikata poslužitelja, zatim je potrebno izdati ključ i certifikat za poslužitelja koje je potrebno smjestiti na poslužitelju na lokacijama koje su navedene u HTTP/3 konfiguraciji koja se može pronaći ranije u ovom poglavlju.

CA certifikat je potrebno dodati na klijentskoj strani, što je opisano u sljedećem potpoglavlju, a isti se može dodati u pouzdane CA na bilo kojem računalu kako bi se u pretraživaču moglo povezati preko domene na kojoj je poslužitelj HTTP/3 vezom.

Certifikati moraju biti izgenerirani pravilno kako bi se mogla uspostaviti HTTP/3 veza, inače se veza s prethodno opisanim poslužiteljem vrati na HTTP/1.1 vezu.

Za dobivanje vlastite domene za poslužitelj, najjednostavniji i najpovoljniji način je bio kupovanjem domene preko [33], na čijoj konzoli se može u DNS postavkama domene postaviti IP adresa na kojoj je dostupan poslužitelj.

Najveći izazov pri implementaciji HTTP/3 poslužitelja je bilo pronaći prikladno rješenje za pokretanje poslužitelja te ga učiniti dostupnim preko interneta. Cloud servisi Amazon EC2, Google Cloud Engine i sl. ne podržavaju komunikaciju protokolom QUIC te se veza s poslužiteljem pokrenutim u tim okruženjima vrati na HTTP/1.1 vezu.

Kako bi poslužitelj bio pokrenut na osobnom računalu, potrebno je postaviti u NAT postavkama usmjerivača prosljeđivanje prometa s odgovarajuće priključnice na lokalnu IP

adresu računala na kojem je pokrenut poslužitelj. Time poslužitelj postaje dostupan preko WAN adrese usmjerivača i proslijeđene priključnice.

Dodatna poteškoća pri izradi prototipa je što pružatelji internetski usluga u Republici Hrvatskoj ne dopuštaju posluživanje QUIC prometa izvan lokalne mreže, stoga je prethodno opisani poslužitelj pokrenut u Bosni i Hercegovini, gdje to nije slučaj.

4.2. HTTP3 Klijent

Za implementaciju klijenta je iskorišten HTTP/3 klijent napisan u programskom jeziku Python koristeći se pokaznim primjerom HTTP/3 klijenta s poveznice [35].

U ovom potpoglavlju će biti prikazani isječci programskog koda klijenta koji se razlikuju od pokaznog primjera [35] jer prikaz cijelog izvornog koda nije potreban. Najvažnija Python biblioteka korištena pri izradi skripte klijenta je biblioteka aioquic [38].

Druga značajna korištena biblioteka je `uvloop` za pisanje konkurentnog koda jer su u skripti korištene korutine pošto je potrebno čekati odgovore poslužitelja na zahtjeve. [39]

Kôd 4.8 prikazan u nastavku čini `main` funkciju Python skripte koja se izvršava na klijentu. Ulazni argumenti funkcije su `QuicConfiguration` instanca koja predstavlja konfiguraciju QUIC veze prema poslužitelju, lista poveznica sadržanih u listi `urls`, znakovni niz `data` koji sadrži putanju do podatka koji će kasnije biti poslani u POST zahtjevu, `include` booleov argument koji služi kao zastavica funkcije `process_http_pushes` kako bi se, ovisno o njenoj vrijednosti, zapisala zaglavlja u izlaznu datoteku u kojoj se sprema sadržaj odgovora, koja je sadržana u mapi određenoj argumentom `output_dir`. Argument `zero_rtt` služi kao zastavica `connect` funkcije kojom se omogućuje uspostavljanje veze u 0RTT, čije značenje je opisano u prethodnom poglavlju.

Nakon prvotnog raščlanjivanja poveznice `urls[0]` i postavljanje priključnice `port` poziva se asinkrona funkcija `connect` s pridruženom varijablom `client` kako bi se moglo koristiti istu koristit u bloku u nastavku. Najprije je potrebno varijablu `client` pretvoriti u tip `HttpClient` kako bi se moglo koristiti metode klase `HttpClient`. U varijabli `coros` su sadržane instance korutine `post_to_topic` te se izvođenjem linije `await asyncio.gather(*coros)` čeka završetak izvođenja pokrenutih korutina.

Nakon izvođenja korutine funkcija `process_http_pushes` obrađuje događaje koje je zabilježio klijent tijekom izvođenja. Kôd 4.10 u nastavku prikazuje programski kod funkcije `process_http_pushes`.

`client._quic.close(error_code=ErrorCode.H3_NO_ERROR)` funkcijom se zatvara QUIC veza s poslužitelje te se njenim izvođenjem završava izvođenje `main` funkcije skripte na poslužitelju.

```
async def main(
    configuration: QuicConfiguration,
    urls: List[str],
    data: Optional[str],
    include: bool,
    output_dir: Optional[str],
    zero_rtt: bool,
) -> None:
    # parse URL
    parsed = urlparse(urls[0])
    assert parsed.scheme in (
        "https"
    ), "Only https:// URLs are supported."
    host = parsed.hostname
    if parsed.port is not None:
        port = parsed.port
    else:
        port = 443

    async with connect(
            host,
            port,
            configuration=configuration,
            create_protocol=HttpClient,
            session_ticket_handler=save_session_ticket,
            wait_connected=not zero_rtt,
    ) as client:
        client = cast(HttpClient, client)
        coros = [
            post_to_topic(
                client,
                urls[0],
                data,
```

```
                        include,
                        output_dir)
                ]
                await asyncio.gather(*coros)
                process_http_pushes(client=client, include=include,
        output_dir=output_dir)
                client._quic.close(error_code=ErrorCode.H3_NO_ERROR)
```

Kôd 4.8 `main` funkcija klijenta

```
async def post_to_topic(client, url, data, include: bool, output_dir:
Optional[str],):
    with open(data, "rb") as image_file:
        image_bytes = image_file.read()
    base64_encoded_image = base64.b64encode(image_bytes)
    start = time.time()
    http_events = await client.post(
        url,
        data=base64_encoded_image,
        headers={ "content-length": str(len(base64_encoded_image)),
            "content-type": "image/png",
            "content-transfer-encoding": "base64"},)
    elapsed = time.time() - start
    octets = 0
    for http_event in http_events:
        if isinstance(http_event, DataReceived):
            octets += len(http_event.data)
            logger.info(http_event.data)
        if isinstance(http_event, HeadersReceived):
            logger.info(http_event.headers)
    logger.info(
        "Response received for %s %s : %d bytes in %.1f s (%.3f Mbps)"
        % ("GET", urlparse(url).path, octets, elapsed, octets * 8 /
elapsed / 1000000))
```

```
    if output_dir is not None:
        output_path = os.path.join(output_dir,
os.path.basename(urlparse(url).path))
        with open(output_path, "wb") as output_file:
            write_response(http_events=http_events, include=include,
output_file=output_file)
```

Kôd 4.9 `post_to_topic` funkcija

```
            def process_http_pushes(
                client: HttpClient,
                include: bool,
                output_dir: Optional[str],
            ) -> None:
                for _, http_events in client.pushes.items():
                    method = ""
                    octets = 0
                    path = ""
                    for http_event in http_events:
                        if isinstance(http_event, DataReceived):
                            octets += len(http_event.data)
                        elif isinstance(http_event, PushPromiseReceived):
                            for header, value in http_event.headers:
                                if header == b":method":
                                    method = value.decode()
                                elif header == b":path":
                                    path = value.decode()
                    logger.info("Push received for %s %s : %s bytes",
            method, path, octets)

                    # output response
                    if output_dir is not None:
                        output_path = os.path.join(output_dir,
            os.path.basename(path) or "index.html")
                        with open(output_path, "wb") as output_file:
                            write_response(http_events=http_events,
            include=include, output_file=output_file)
```

Kôd 4.10 `process_http_pushes` funkcija

Kôd 4.9 prikazuje izvorni kod funkcije za slanje POST zahtjeva na temu na poslužitelju određenu preko url-a, kako je objašnjeno u prethodnom potpoglavlju. Funkcija kao ulazne

argumente prima `client` instancu klijenta, poveznicu na poslužitelj u `url`, putanju do datoteke u znakovnom nizu `data` te booleov argument `include` i znakovni niz `output_dir` koji služe na samom kraju funkcije za zapis odgovora i zaglavlja odgovora u datoteku.

Na početku se izvođenja funkcije se otvara datoteka te kodira u base64 formatu kako bi se PNG datoteka predstavila u formatu znakovnog niza. Zatim se pozivanjem `post` metode klijenta šalje POST zahtjev na poslužitelj s kodiranim nizom i odgovarajućim zaglavljima.

Prije slanja POST zahtjeva u varijabli `start` je zabilježeno vrijeme početka slanja zahtjeva te se u varijabli `elapsed` sprema ukupno vrijeme u milisekundama potrošeno na slanje i primanje odgovora POST zahtjeva u varijabli `http_events`. U nastavku su u `logger`-u u `info` kanalu prolaskom kroz listu http događaja u `http_events` ispisani primljeni podaci, zaglavlja te zapisano za koju metodu i putanju, koje veličine, za koliko vremena s brzinom u megabitima u sekundi je primljen odgovor.

```
python src/client.py --ca-certs cert/ca.pem
https://www.diplomski.xyz/pub?id=2 --verbose --data
'resources/image.png' --output-dir 'output/' -i -q 'log/' -l
'wireshark/log.pcapng'
```

Kôd 4.11

Kôd 4.11 sadrži naredbu za pokretanje skripte klijenta naziva `client.py` s ulaznim argumentima CA certifikata, poveznice na koju će se slati POST zahtjev, argumenta za detaljan ispis u konzoli `–verbose`, putanje do datoteke koja će se slati u zahtjevu, putanje do mape za spremanje odgovora, zastavice `-i` kako bi spremanje odgovora uključivalo zaglavlja, putanje do mape u kojoj će biti spremljen QUIC log te argumenta `-l` s putanjom do datoteke u kojoj će biti stvoren SSL zapisnik za dekripciju snimljenog prometa u alatu Wireshark.

Najvažniji argument je CA certifikat kojim je izdan certifikat poslužitelja kako bi se TLS rukovanje uspješno provelo te samim time omogućilo uspostavljanje HTTP/3 veze između klijenta i poslužitelja. Osim druga dva obvezna argumenta su putanja do datoteke koja se šalje u zahtjevu te poveznica na koju će se slati isti.

4.3. Moguća unapređenja prototipa

Implementacija HTTP/3 prototipa se može proširi uvođenjem mehanizma autorizacije korisnika za pretplaćivanje i objavljivanje na teme, korištenjem autorizacijskih tokena ili sličnog autorizacijskog mehanizama, klijent se može proširiti tako što bi se pretplatio na upravljačku temu kojom se mogu slati upravljačke poruke, npr. za pokretanje i zaustavljanje, poslužitelj se može povezati na bazu podataka kako bi se spremali podaci iz tema u bazu. Klijentska skripta bi također trebala biti napisana tako da šalje stvarne i korisne podatke, a ne samo testne podatke. To bi bila samo neka od mogućih unapređenja

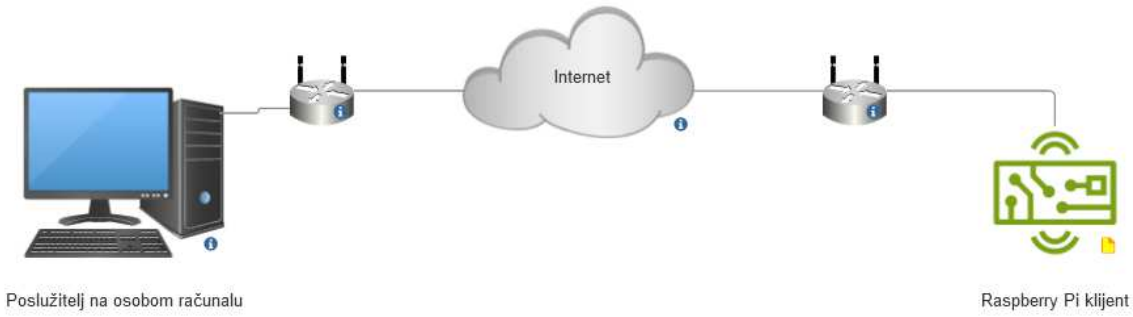
5. Korištena okolina Interneta stvari

U ovom poglavlju je opisana okolina Interneta stvari u kojoj je HTTP/3 prototip iz prethodnog poglavlja pokrenut te analizirana ostvarena komunikacija.

Poslužitelj je pokrenut na osobnom računalu s operacijskim sustavom Ubuntu 22.04 LTS koje je povezano preko na internet preko lokalne mreže u kojoj je, kako je prethodno spomenuto, u NAT postavkama usmjerivača postavljeno prosljeđivanje prometa na priključnicama 80 i 443 na IP adresu poslužitelja u lokalnoj mreži.

Klijent je pokrenut na malom računalu Raspberry Pi, čije su specifikacije dostupne na [36], na kojem je podignut operacijski sustav Ubuntu Core 22 kako bi se na njemu mogla izvršavati Python skripta HTTP/3 klijenta. Klijent je povezan na lokalnu mrežu kao i bilo koji drugi uređaj bez potrebe za posebnom konfiguracijom mreže, kao što je to bio slučaj za poslužitelja.

U nastavku je pojednostavljeni prikaz prethodno opisane okoline Interneta stvari izrađen korištenjem alata Smartdraw [37].



Sl. 5.1 Prikaz korištene okoline stvari [37]

6. Analiza ostvarene komunikacije

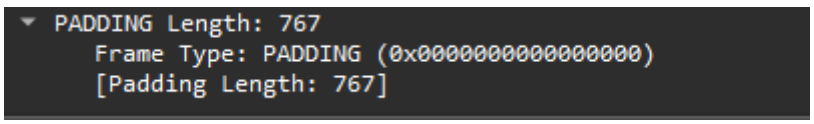
Ovo poglavlje opisuje i prikazuje rezultate analize ostvarene komunikacije korištenjem HTTP3 prototipa u korištenoj okolini Interneta stvari. Podaci za analizu su prikupljeni korištenjem alata Wireshark [40] kojim je snimljen promet između klijenta i poslužitelja. Važno je napomenuti da zbog enkripcije prometa korištenjem protokola QUICje potrebno u TLS postavkama Wiresharka dodati datoteku kojoj je sadržan SSL zapisnik.

6.1. Analiza uspostavljanja veze i slanja paketa

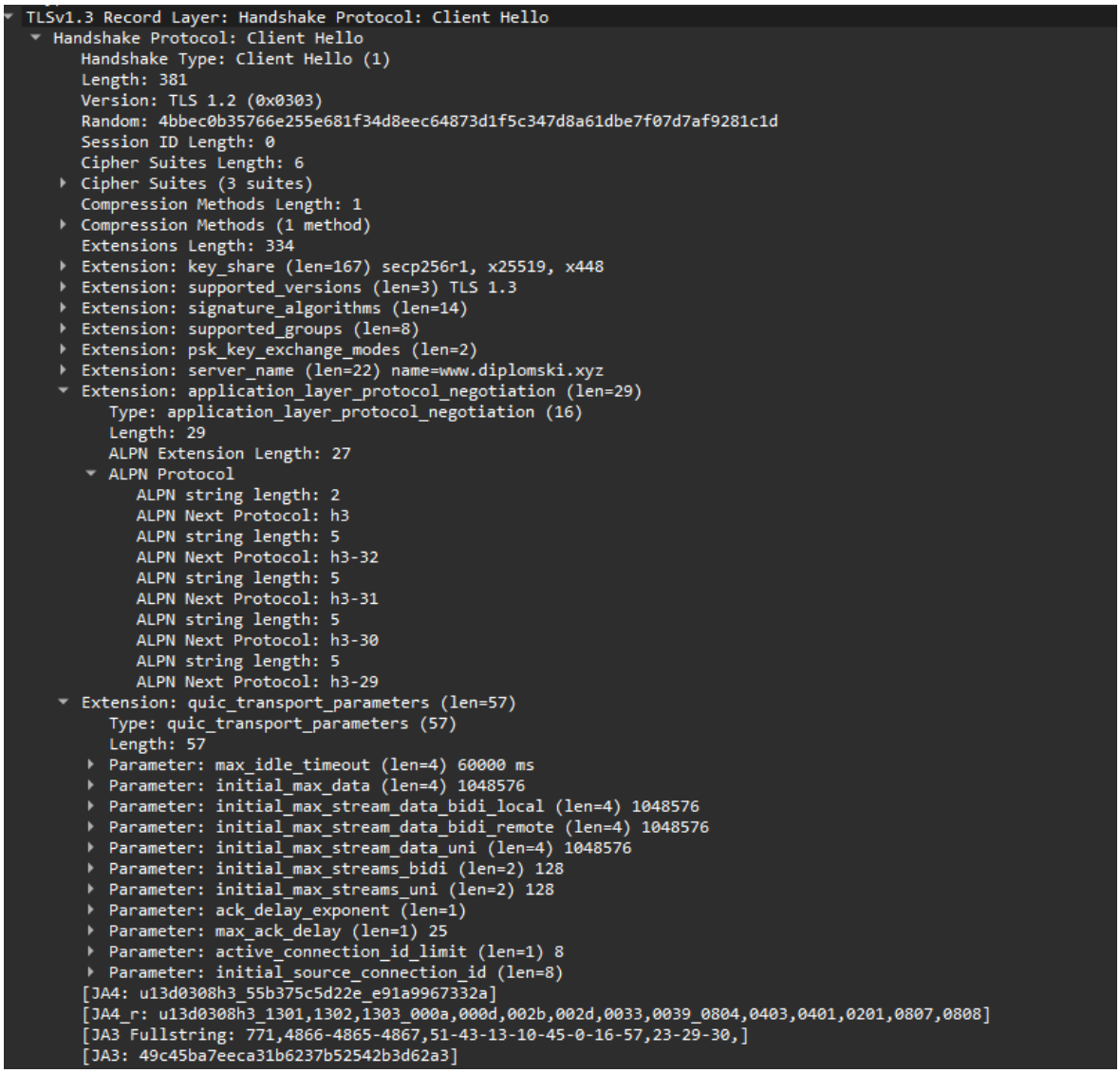
Prvi korak uspostavljanja QUIC veze je slanje `Initial` paketa koji je najveće moguće veličine paketa, 1242 bajta za snimljeni promet u nastavku, te je znatno veći od paketa koji se šalju prilikom TCP rukovanja, koji su veličine oko 70 bajta. Na Sl. 6.2 se može vidjeti prikaz sadržaja rukovanja protokola TLS unutar `Initial` paketa na kojem se može vidjeti poslana `Client Hello` poruka.

Unutar `application_layer_protocol_negotation` bloka su navedeni protokoli na aplikacijskom sloju, `h3`, tj. HTTP/3 u više draftova. Unutar `quic_connection_parametres` bloka se mogu vidjeti parametri QUIC veze gdje su navedeni parametri vrijeme isteka, granice veličina tokova podataka i sl. što se sve može vidjeti na snimci zaslona na Sl. 6.2. Ostali elementi TLS rukovanja se ne razlikuju od TLS rukovanja na TCP vezi.

Slika 6.1 prikazuje nadopunjavanje `Initial` paketa čime se postiže velika veličina početnog paketa koja je obično približna najvećoj mogućoj veličini paketa. Kao što se može vidjeti, paket se nadopunjuje nulama do najveće veličine paketa.

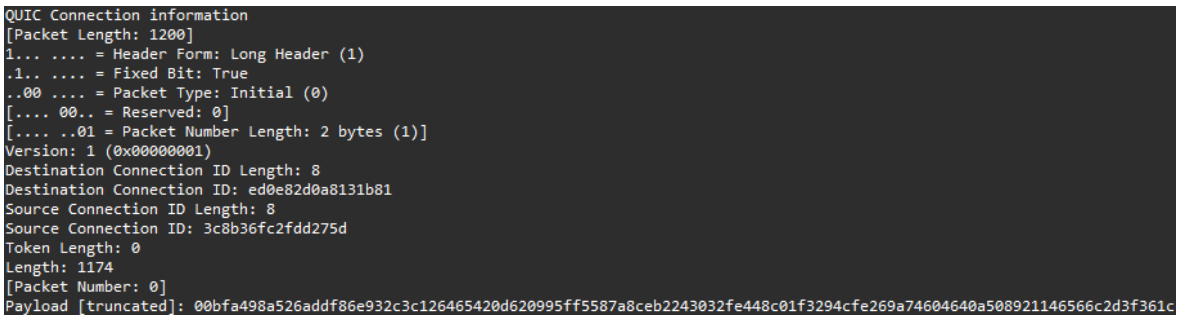


Sl. 6.1 Snimak zaslona s prikazom nadopunjavanja paketa



Sl. 6.2 Snimak zaslona s prikazom sadržaja `Client Hello`

Dodatan zanimljiv dio `Initial` paketa je `QUIC Connection Information`, s elementima `Destination Connection ID` te `Source Connection ID` preko kojih krajnja točka u komunikaciji identificira QUIC vezu na drugoj strani, što omogućuje održavanje veze iako se na slojevima ispod veza izmijenila (IP adrese, UDP port ili sl.).



Sl. 6.3 Snimak zaslona s prikazom `QUIC Connection Information`

U primjeru komunikacije koja je promatrana poslužitelj odgovara `Retry` paketom kako bi poslao klijentu identifikaciju QUIC veze jer za identifikaciju koju je poslao klijent unutar `Destination Connection ID` nije bilo moguće uspostaviti istu.

```
QUIC Connection information
[Packet Length: 117]
1... .... = Header Form: Long Header (1)
..11 .... = Packet Type: Retry (3)
Version: 1 (0x00000001)
Destination Connection ID Length: 8
Destination Connection ID: 3c8b36fc2fdd275d
Source Connection ID Length: 20
Source Connection ID: 0bf11e7b564f26bdadbbfc24289893caa49af30a
Retry Token: 43054effa014d9802a4d2ee2f24b9865bcdeeda9ffe5cc6c0a7ea09684b45bdc3026de89c85f416faae6ca758fdca6c6bc79f547596ab041526e
Retry Integrity Tag: c279f9a0811a6d90d38e2b0bc0ea434a [verified]
```

Sl. 6.4 Snimak zaslona s prikazom informacija veze `Retry` paketa

Na `Retry` paket s poslužitelja klijent ponovno odgovara `Initial` paketom, ali ovaj put je, kako je vidljivo na Sl. 6.5, pravilno uparene vrijednosti identifikacija veze.

```
QUIC Connection information
[Packet Length: 1200]
1... .... = Header Form: Long Header (1)
.1.. .... = Fixed Bit: True
..00 .... = Packet Type: Initial (0)
[.... 00.. = Reserved: 0]
[.... ..01 = Packet Number Length: 2 bytes (1)]
Version: 1 (0x00000001)
Destination Connection ID Length: 20
Destination Connection ID: 0bf11e7b564f26bdadbbfc24289893caa49af30a
Source Connection ID Length: 8
Source Connection ID: 3c8b36fc2fdd275d
Token Length: 66
Token: 43054effa014d9802a4d2ee2f24b9865bcdeeda9ffe5cc6c0a7ea09684b45bdc3026de89c85f416faae6ca758fdca6c6bc79f547596ab041526e3350966
Length: 1095
[Packet Number: 1]
Payload [truncated]: d6d450d18b878d8c243489255e171b9f94af0819858387656d6b4127c70008d22dc2f601d0b1a5e758b725f4994509fbce721fc3886bb
```

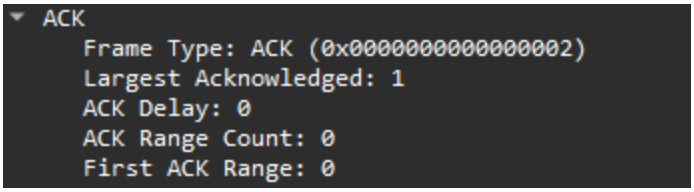
Sl. 6.5 Snimak zaslona s prikazom informacija veze ponovljenog `Initial` paketa

Nakon ponovnog slanja `Initial` paketa s klijenta, poslužitelj odgovara svojim `Initial` paketom u kojem šalje svoj TLS `Server Hello` na Sl. 6.6.

```
TLSv1.3 Record Layer: Handshake Protocol: Server Hello
▾ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 119
    Version: TLS 1.2 (0x0303)
    Random: 55d51a5a4dbf571764dd5b6f8911e85c3f2ad7736971029af64ae0dfebb6b7ab
    Session ID Length: 0
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Compression Method: null (0)
    Extensions Length: 79
  ▾ Extension: supported_versions (len=2) TLS 1.3
      Type: supported_versions (43)
      Length: 2
      Supported Version: TLS 1.3 (0x0304)
  ▾ Extension: key_share (len=69) secp256r1
      Type: key_share (51)
      Length: 69
    ▸ Key Share extension
    [JA3S Fullstring: 771,4866,43-51]
    [JA3S: 15af977ce25de452b96affa2addb1036]
```

Sl. 6.6 Snimak zaslona s prikazom `Server Hello`

Dodatan zanimljiv dio paketa kojim odgovara poslužitelj je `ACK` koji u sebi sadrži primanja prethodnog paketa od klijenta što omogućuje pouzdanost QUIC veze.

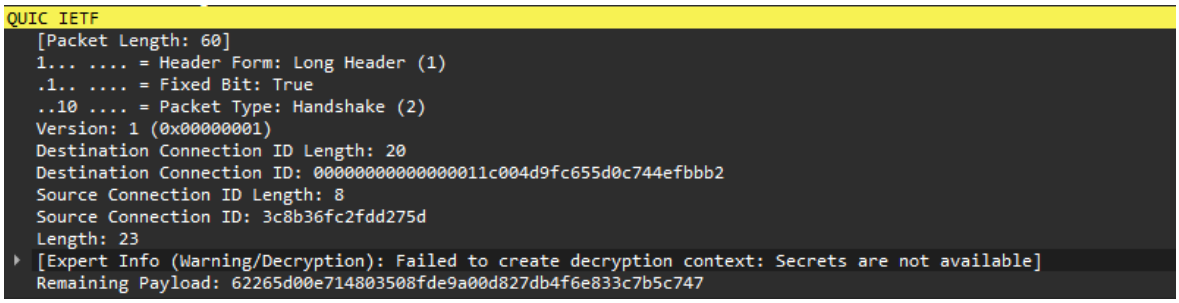


Sl. 6.7 Snimak zaslona s prikazom `ACK` dijela paketa

Na Sl. 6.7 se može vidjeti da server potvrđuje klijentu primanje paketa s brojem 1, a može se vidjeti na Sl. 6.5 da je klijent poslao ponovljeni `Initial` paket s brojem paketa 1.

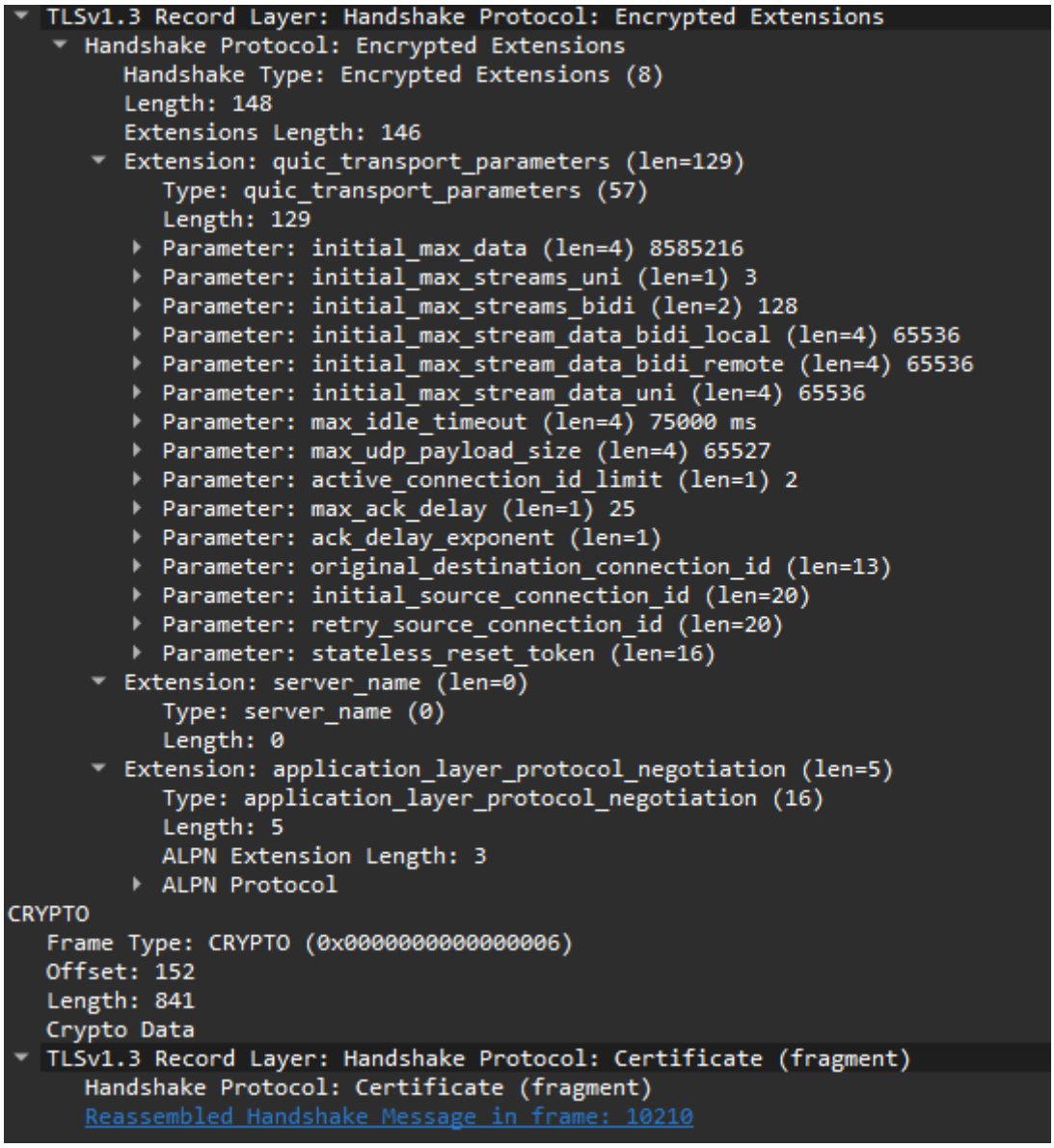
U nastavku se šalju `Handshake` paketi, što se može vidjeti na Sl. 6.8 te Sl. 6.9, međutim zbog enkripcije sadržaja prometa TLS-om ne može se vidjeti potpuni sadržaj paketa.

Sl. 6.8 Snimak zaslona s prikazom paketa rukovanja koji šalje poslužitelj



Sl. 6.9 Snimak zaslona s prikazom paketa rukovanja koji šalje klijent

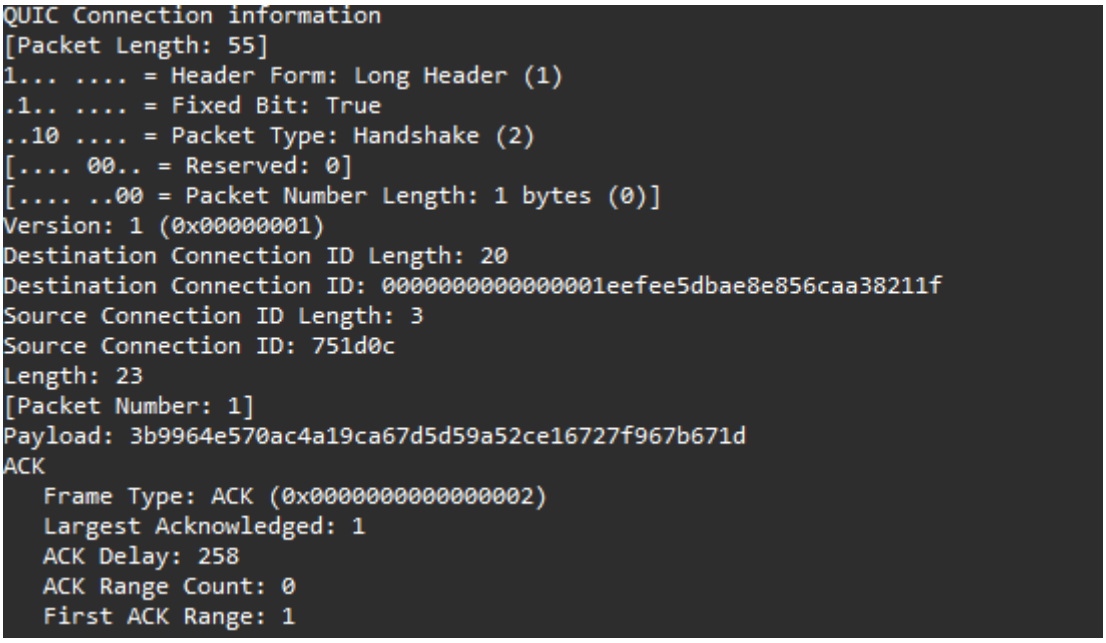
Za potrebu prikaza skrivenog sadržaja `Handshake` paketa u nastavku su prikazani sadržaji isti koji su poslani iz Firefox pretraživača prema poslužitelju za koje alat Wireshark uspješno pročita privremene ključeve enkripcije iz SSL zapisnika. Na Sl. 6.10 se može vidjeti sadržaj paketa rukovanja koji šalje poslužitelj, a to su najprije ekstenzija enkripcija s opisom transportnih parametara te dio certifikata za enkripciju. Sl. 6.14 prikazuje sadržaj drugog paketa rukovanja poslanog s poslužitelja koji sadrži drugi dio certifikata, javni ključ poslužitelja za provjeru enkripcije i poruku kojom je označen kraj uspješnog rukovanja.



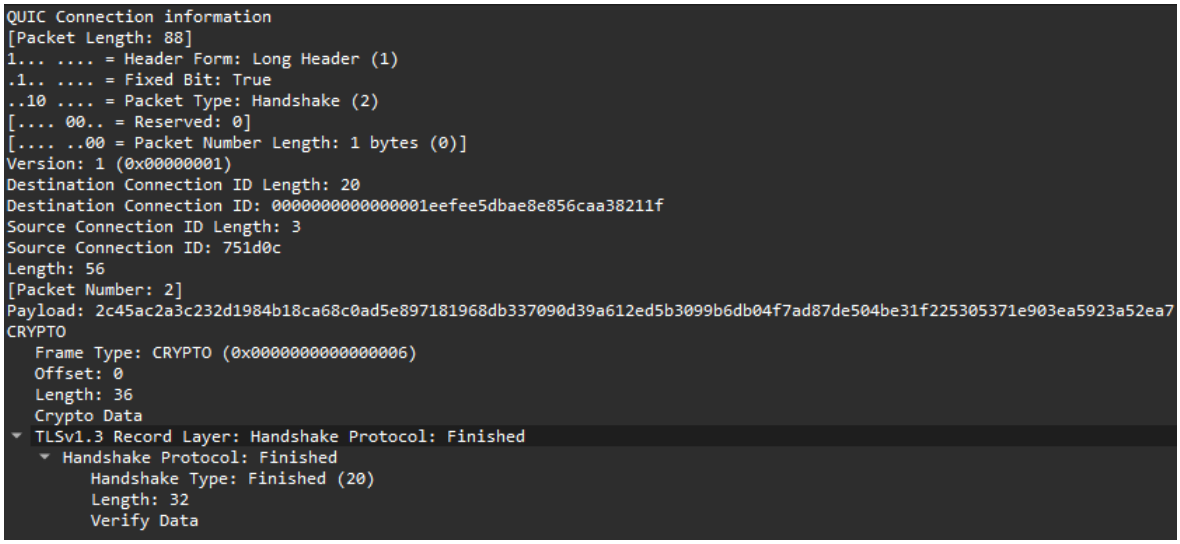
Sl. 6.10 Snimak zaslona s prikazom `CRYPTO` sadržaja paketa rukovanja poslužitelja

Klijent vraća poslužitelju dva `Handshake` paketa kojima samo potvrđuje uspješno primanje paketa rukovanja koje je poslao poslužitelj te na kraju drugog paketa također stavlja poruku kojom obavještava poslužitelja da je protokol rukovanja gotov. Sadržaj prethodno spomenutih klijentskih paketa su prikazani u nastavku na Sl 6.11 te Sl. 6.12.

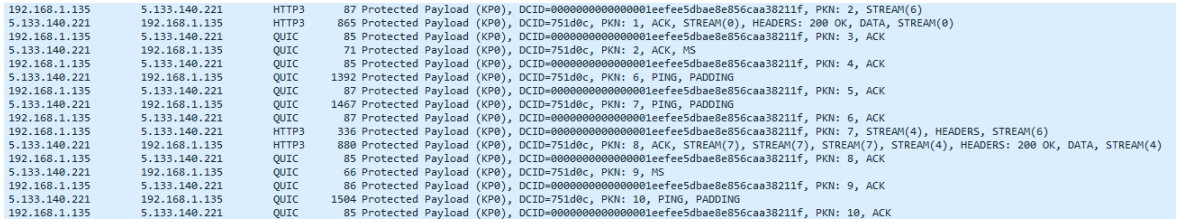
Nakon završenog rukovanja klijent otvara QUIC tokove prema poslužitelju, vidljivo na Sl. 6.15 te na toku `0` šalje `GET` zahtjev, što se može vidjeti na Sl 6.17. Nakon otvaranja QUIC tokova poslužitelj odgovara `New Session Ticket` porukom čiji je sadržaj prikazan na Sl. 6.17. Nakon uspostavljanja QUIC tokova, poslužitelj šalje pakete odabranim tokom te jedan drugome odgovaraju paketima koji sadrže element `ACK` kako bi javili drugoj strani koji je prethodni paket već primljen, što je vidljivo na sažetom prikazu na Sl. 6.13.



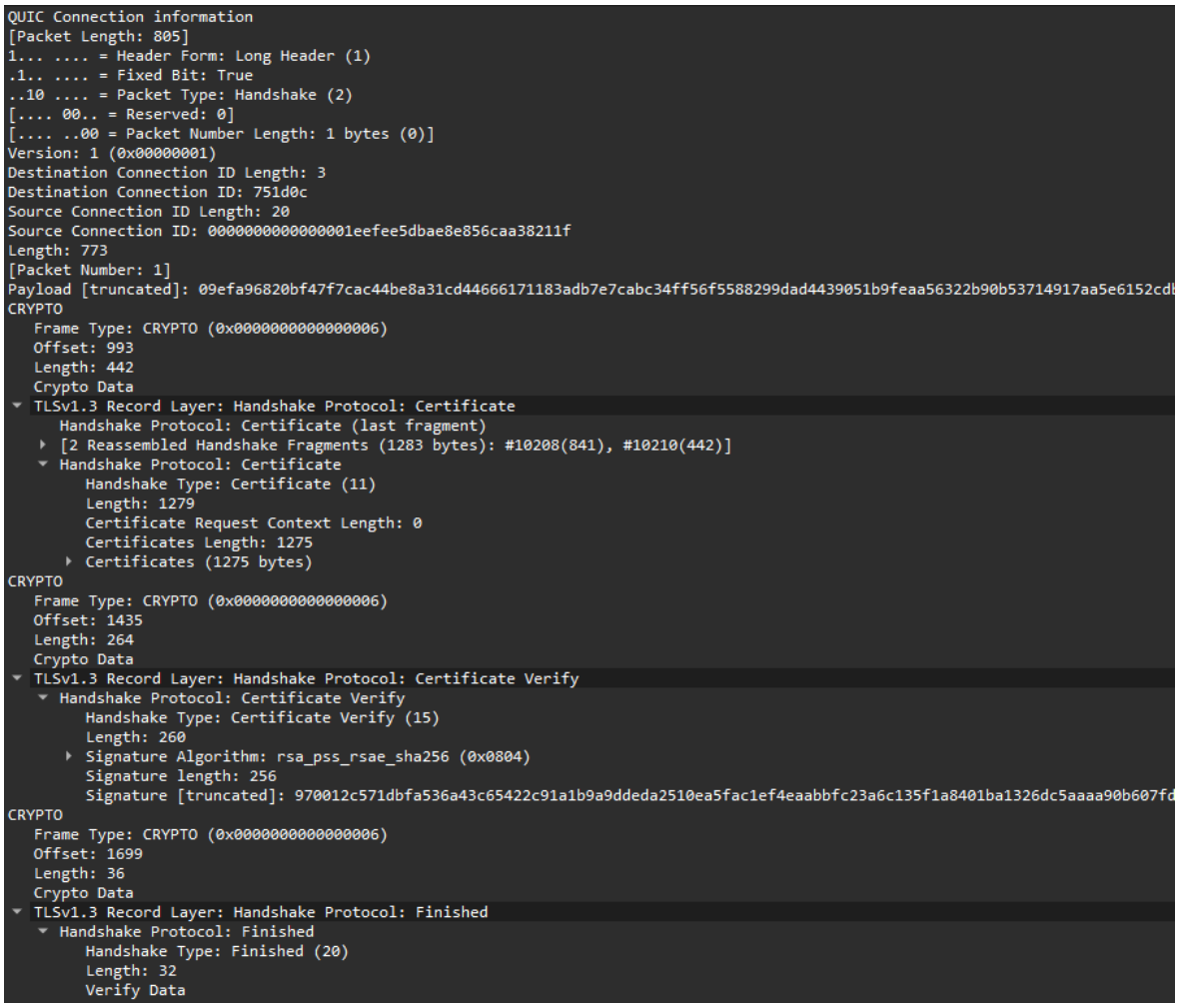
Sl. 6.11 Snimak zaslona s prikazom sadržaja prvog `Handshake` paketa klijenta



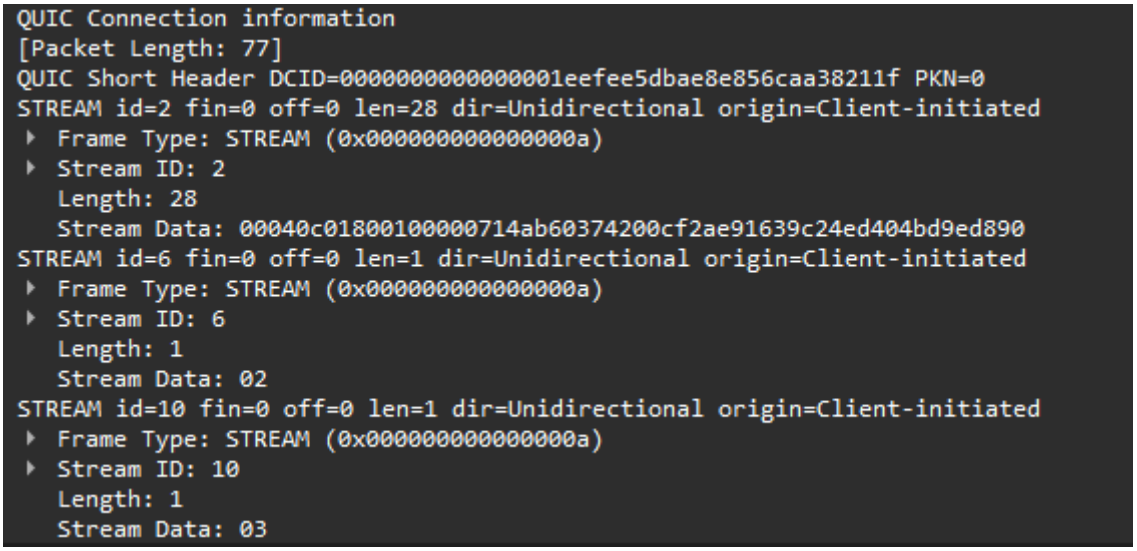
Sl. 6.12 Snimak zaslona s prikazom sadržaja drugog `Handshake` paketa klijenta



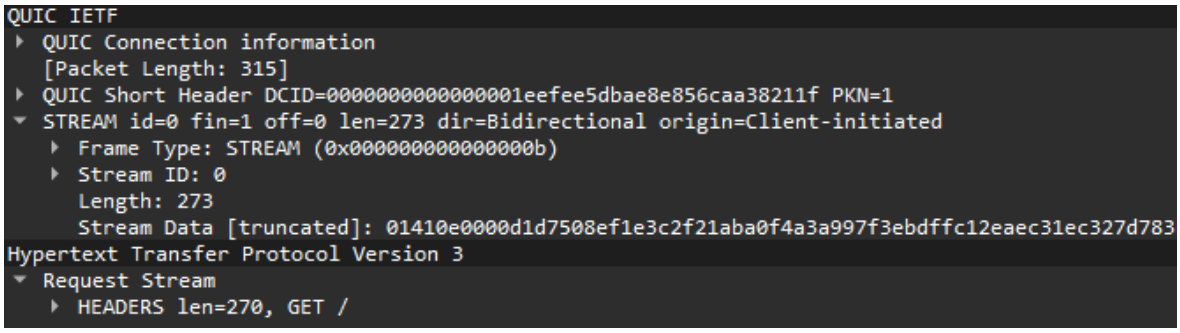
Sl. 6.13 Snimak zaslona sa sažetim prikazom slanja sadržaja



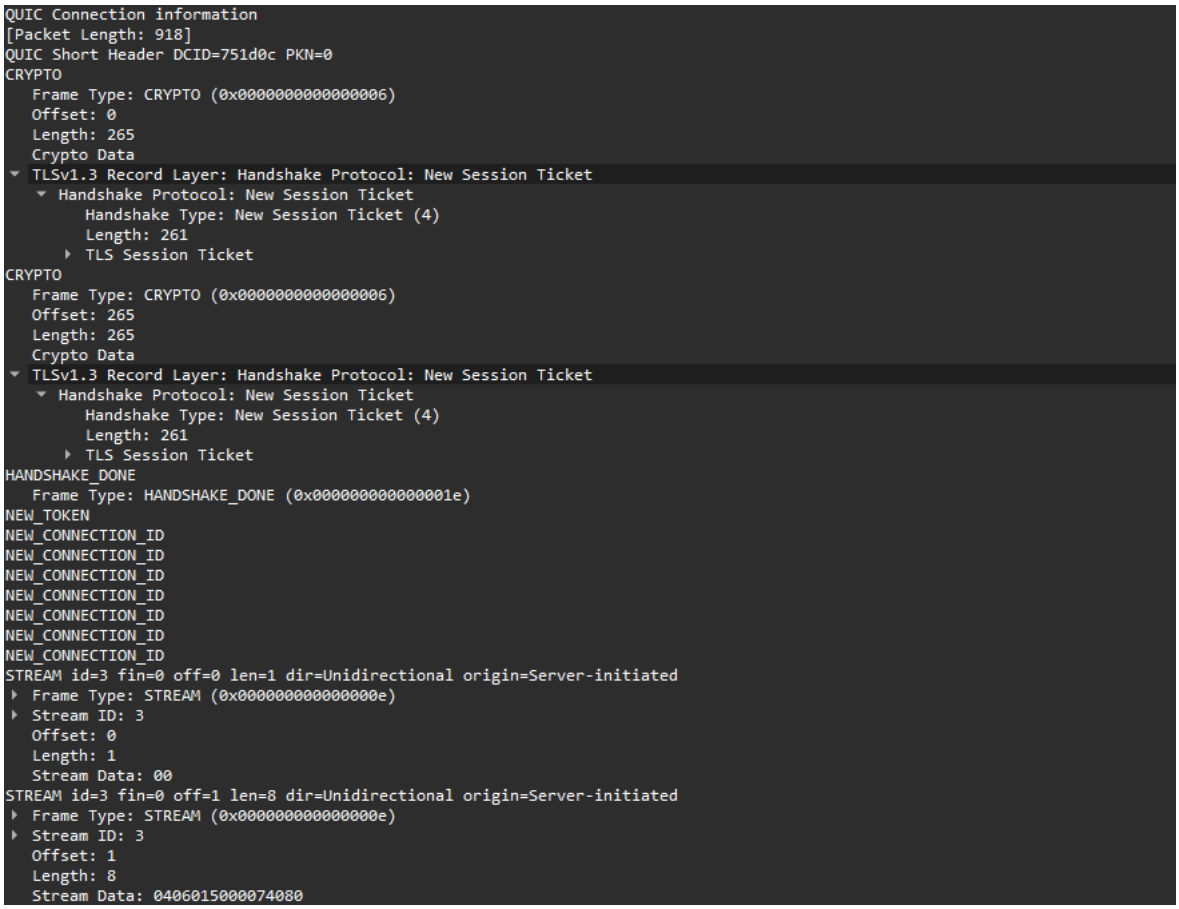
Sl. 6.14 Snimak zaslona s prikazom sadržaja drugog paketa rukovanja poslužitelja



Sl. 6.15 Snimak zaslona s prikazom sadržaja paketa kojim klijent otvara QUIC tokove

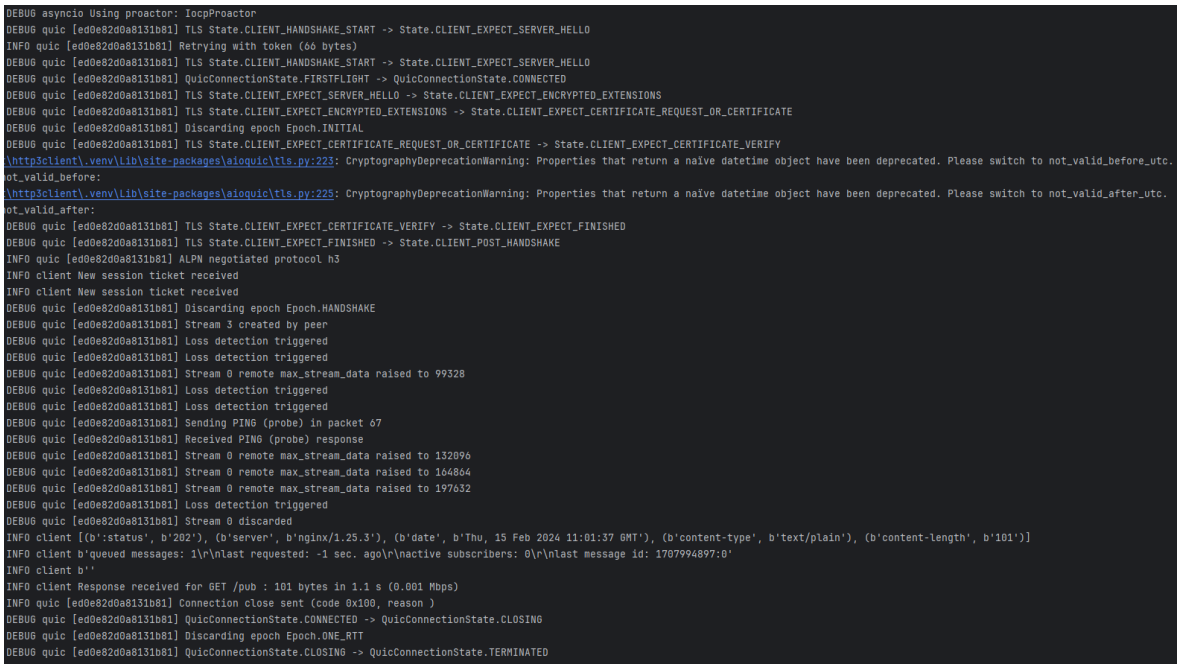


Sl. 6.16 Snimak zaslona s prikazanim slanjem `GET` zahtjeva na toku `0`



Sl. 6.17 Snimak zaslona s prikazom sadržaja paketa s `New Session Ticket`

Iako se korištenjem alata Wireshark ne može dešifrirati sadržaj razmijenjenih poruka korištenjem prototipa klijenta, klijent radi ispis u konzolu na kojem se može vidjeti tijek komunikacije koji je jednak prethodno opisanom tijeku komunikacije. Ispis u konzoli klijenta se može vidjeti u nastavku na Sl. 6.18. U ispisu se može vidjeti tijek komunikacije s naznačenim promjenama stanja QUIC i TLS veze kao i informacija o ispregovaranom protokolu HTTP/3 (`h3`). Ispis također obavještava o novom `session ticket` te prati stanje na toku otkrivajući gubitak paketa te na samom kraju ispisuje zaglavlja i sadržaj odgovora poslužitelja da bi naposljetku zatvorio QUIC vezu i opisao promjene stanja iste.



Sl. 6.18 Snimak zaslona s prikazom ispisa u konzoli klijenta

Iako prikazani ispis na Sl. 6.18 ne daje uvid u pakete komunikacije kao što je to moguće praćenjem paketa alatom Wireshark, ispis u konzoli klijenta je koristan kako bi se moglo u sažetom obliku pratiti komunikaciju klijenta s poslužiteljem pri izvođenju same skripte klijenta.

6.2. Analiza ponovnog uspostavljanja veze

U ovom potpoglavlju je analizirano ponovno uspostavljanje QUIC veze uzrokovanom promjenom na slojevima ispod QUIC-a kako bi se ustanovilo na koji se način ponovno uspostavlja veza između klijenta i poslužitelja. Za potrebe ove analize je u `http3.conf` konfiguraciji na poslužitelju dodan novi `location` blok sadržaja prikazanog u Kôd 6.1 kako bi se `GET` zahtjevom na putanju do tog bloka vratila klijentu veća datoteka pri čemu je lakše za potrebe ove analize moguće simulirati prekid ili promjenu veze.

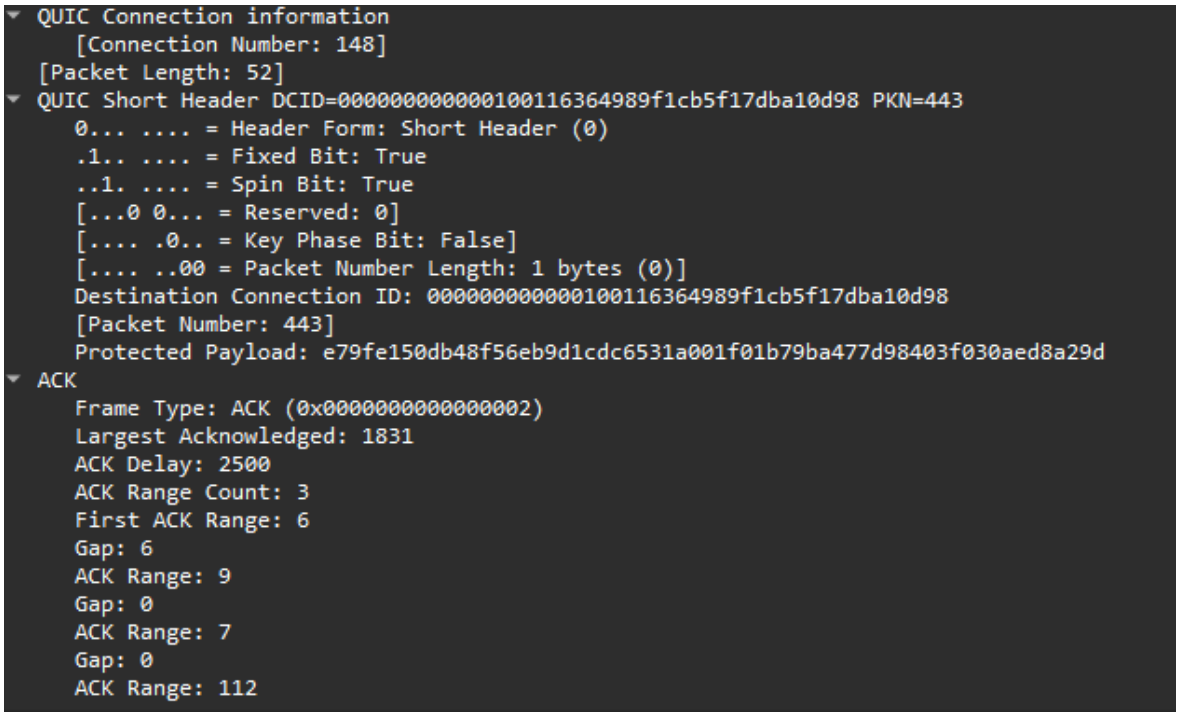
```
location /image {
   alias /usr/local/nginx/resource/;
   try_files image.png =404;
   add_header alt-svc 'h3=":$server_port"; ma=864000';
   add_header x-quic 'h3';
   add_header cache-control 'no-cache,no-store';
}
```

Kôd 6.1 `location` blok za slanje slike

Nakon gubitka veze s poslužiteljem klijent šalje paket `PING` kojim ponovno pokušava pronaći poslužitelja kako preko iste identifikacije QUIC veze pokušao nastaviti prijenos. Sadržaj paketa PING se može vidjeti na Sl 6.19. U paketu možemo vidjeti da klijent traži QUIC vezu s poslužiteljem po `Destination Connection ID` koji je određen pri rukovanju na početku komunikacije. Prije slanja `PING` paketa klijent je poslužitelju u `ACK` poslao potvrdu da je primio paket `1831` što je vidljivo na Sl. 6.20.

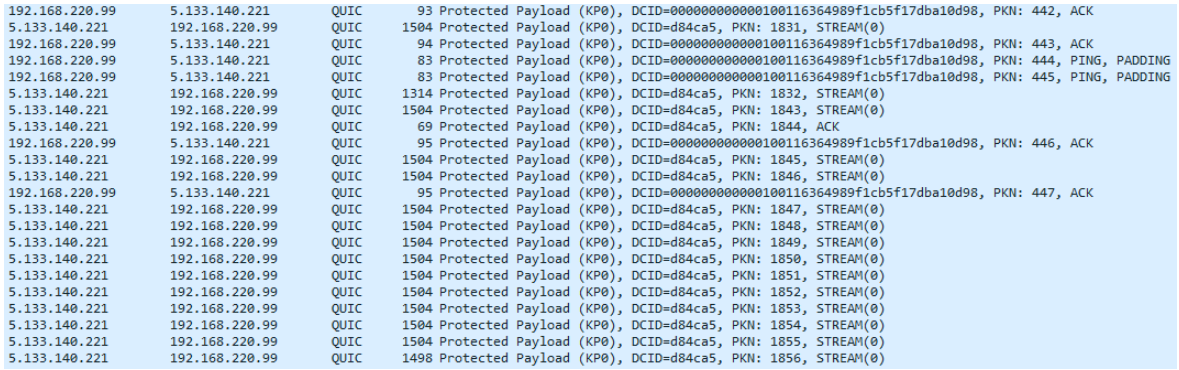
```
▾ QUIC Connection information
    [Connection Number: 148]
  [Packet Length: 41]
▾ QUIC Short Header DCID=0000000000001001116364989f1cb5f17dba10d98 PKN=445
    0... .... = Header Form: Short Header (0)
    .1.. .... = Fixed Bit: True
    ..1. .... = Spin Bit: True
    [...0 0... = Reserved: 0]
    [.... .0.. = Key Phase Bit: False]
    [.... ..00 = Packet Number Length: 1 bytes (0)]
    Destination Connection ID: 0000000000001001116364989f1cb5f17dba10d98
    [Packet Number: 445]
    Protected Payload: 0a9cd8d132935ce080cbc0396ca449cd14c319
▾ PING
    Frame Type: PING (0x0000000000000001)
▾ PADDING Length: 2
    Frame Type: PADDING (0x0000000000000000)
    [Padding Length: 2]
```

Sl. 6.19 Snimak zaslona s prikazom sadržaja `PING` paketa klijenta



Sl. 6.20 Snimak zaslona s prikazom poruke klijenta kojom potvrđuje primanje paketa `1831`

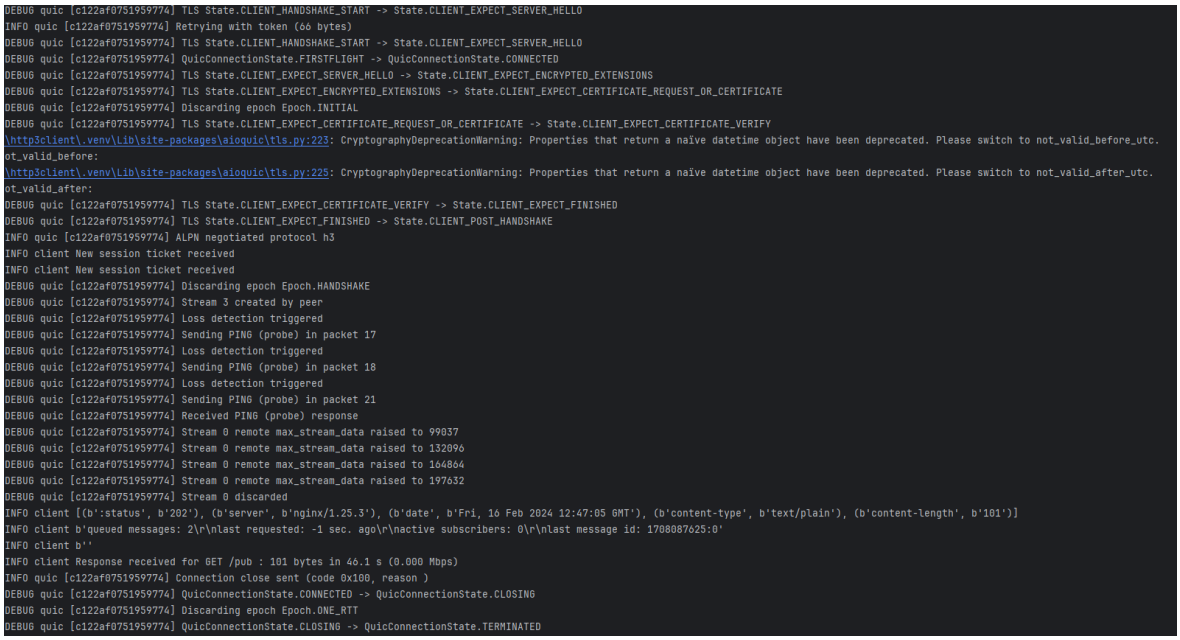
Na Sl. 6.21 se može vidjeti da nakon paketa `PING` od poslužitelja nakon ponovnog uspostavljanja veze dolazi paket s brojem `1832` kojim poslužitelj nastavlja slanje podataka koji je ujedno i prvi paket poslije `1831` za koji je dobio potvrdu da je primljen. Također je važno napomenuti, a to je glavna značajka protokola QUIC, na Sl 6.21 je vidljivo da klijent i poslužitelj nastavljaju komunikaciju preko istih QUIC veza, neovisno što je došlo do prekida u međuvremenu.



Sl. 6.21 Snimak zaslona sa sažetim prikazom slijeda komunikacije pri prekidu veze

Ovom analizom je pokazano da u slučaju prekida veze između klijenta i poslužitelja je moguće nastaviti prijenos QUIC vezom. Ovo čini protokol HTTP/3 koji se oslanja na QUIC pogodnim za korištenje u Internetu stvari jer omogućuje nastavak komunikacije i s uređajima koji zbog svojih ograničenih resursa ne mogu istovremeno komunicirati preko

mreže i raditi obradu podataka ili zbog loših uvjeta mreže gube stalnu vezu s poslužiteljem i nakon navedenih poteškoća, koristeći se identifikacijom prethodno uspostavljene QUIC veze kako bi se nastavila komunikacija preko iste.



Sl. 6.22 Snimak zaslona s prikazom ispisa u konzoli klijenta prilikom ispada veze

Na Sl. 6.22 je prikazan ispis konzole klijenta u kojem se može vidjeti da klijent šalje `PING` paket nakon detekcije gubitka paketa kako bi oporavio QUIC vezu prema poslužitelju.

6.3. Analiza utjecaja promjene veličine sadržaja

Ovo potpoglavlje sadrži analizu ostvarene komunikacije mijenjanjem veličine sadržaja koji se šalju. Ovo potpoglavlje su skupljena prethodna dva potpoglavlja jer provedenom analizom jer se u prikazu u nastavku može vidjeti da tijekom slanja većeg sadržaja se samo povećava mogućnost spontanog prekida i oporavka QUIC veze te su u snimljenoj komunikaciji također prisutni elementi uspostavljanja QUIC veze i slanja paketa.

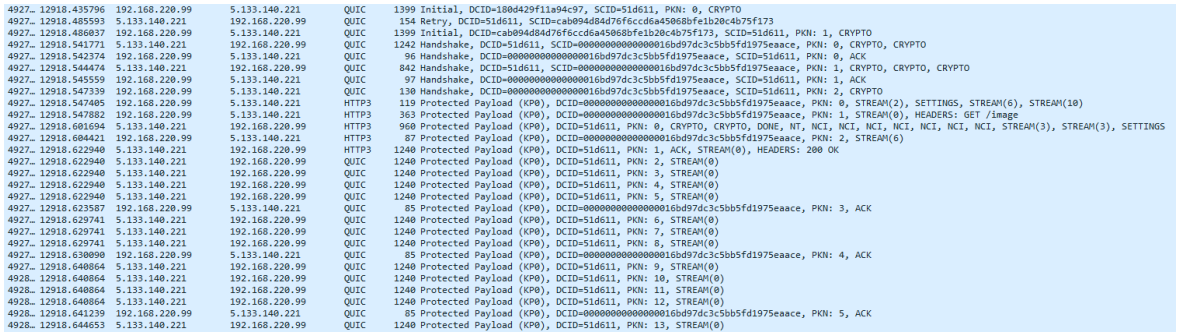
Na osnovu toga se može donijeti zaključak da veličina samog sadržaja nema značajnog utjecaja na pouzdanost protokola HTTP/3, osim što produljuje ukupno vrijeme koje je potrebno za slanje zahtjeva i dobivanje odgovora.

U nastavku je na Sl. 6.23 prikazana komunikacija za mali sadržaj, html datoteku od 0.65 kilobajta od početka do kraja te za veliki sadržaj, PNG datoteku od 10 megabajta, Sl 6.24. te Sl. 6.25. Najbitnija razlika je sami broj poslanih paketa, dok bi se detaljnim pregledom slanja PNG datoteke povremeno mogao naći i `PING` paket u slučaju kraćeg prekida mreže tijekom prijenosa iste.

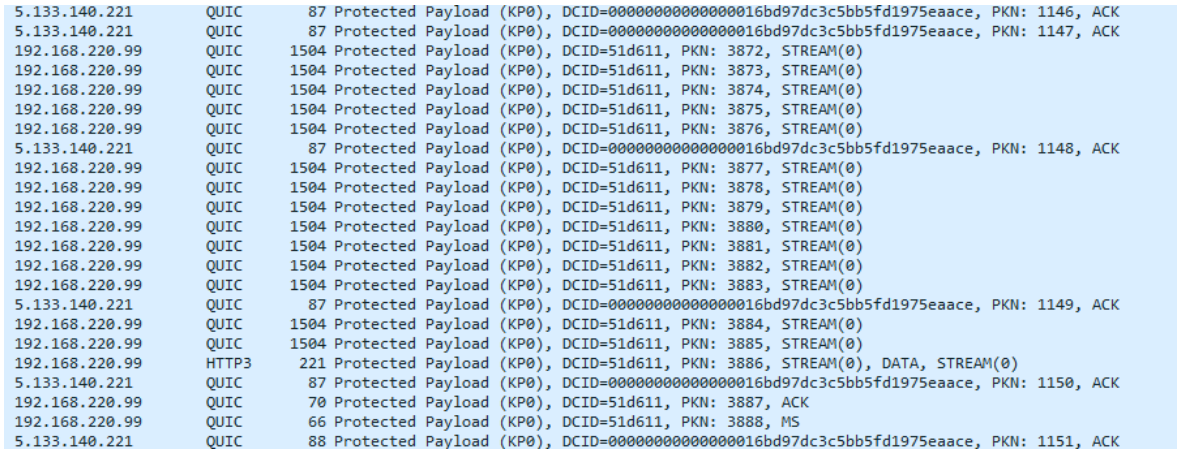
Sl. 6.23 Snimak zaslona s prikazom cjelokupne komunikacije pri slanju male datoteke

Na Sl. 6.23 se može vidjeti da je cjelokupna komunikacija pri prijenosu html datoteke dovršena u 18 paketa, dok u nastavku, na Sl. 6.24, možemo uočiti kako komunikacija u prijenosu velike datoteke započinje na isti način, a odvila se u mnogo više paketa jer se na samom kraju komunikacije, na Sl. 6.25, može vidjeti da je broj posljednjeg paketa koji je poslao poslužitelj `3888`. Pri prijenosu velike datoteke se dogodio poneki prekid mreže te je oporavak veze u sažetom formatu prikazan na Sl 6.26.

Provođenjem analizi opisanih u ovom i prethodna dva potpoglavlja može se doći do zaključka koji je iznesen u poglavlju zaključka ovoga rada.



Sl. 6.24 Snimak zaslona s prikazom početka komunikacije za prijenos velike datoteke



Sl. 6.25 Snimak zaslona s prikazom završetka komunikacije za prijenos velike datoteke

Sl. 6.26 Snimak zaslona na kojem se može vidjeti oporavak QUIC veze paketom `PING`

Zaključak

Analizom ostvarene komunikacije korištenjem prototipa HTTP/3 klijenta i poslužitelja u okolini Interneta stvari zaključak je da se protkol HTTP u inačici 3 može koristiti kao komunikacijska tehnologija u Internetu stvari kao alternativa do sada najčešće korištenim protkolima MQTT i CoAP jer HTTP/3 korištenjem transportnog protokola QUIC rješava problem head-of-line blokiranja zbog kojeg prijašnje inačice HTTP-a, koje koriste transportni protkol TCP, nisu bile pouzdano rješenje za komunikacijsku tehnologiju u Internetu stvari. Korištenjem protokola HTTP3 je ostvarena komunikacija koja je pouzdana u uvjetima ograničenih procesnih i memorijskih resursa te spore i nestabilne veze između klijenta i poslužitelja.

Literatura

[1] Tanenbaum, A. S., Wetherall, D. J. *Computer Networks*. 5. izdanje. London: Pearson, 2013.

[2] Hanes, D, Salgueiro, G., Grossetete, P., Barton, R., Henry, J. *IoT Fundamentals* 1. izdanje. Cisco Press, 2017.

[3] Bormann, C., Hartke, K., Shelby, Z. *RFC7252 The Constrained Application Protocol (CoAP)* IETF, 2014.

[4] Banks, A., Briggs, E., Borgendale, K., Gupta, R. *MQTT Version 5* OASIS Standard, 2019.

[5] Fette, I., Melinkov, A., *RFC6455 The WebSocket Protocol* IETF, 2011.

[6] Fielding, R., Nottingham, M., Reschke, J. *RFC9110 HTTP Semantics* IETF, 2022.

[7] Fielding, R., Nottingham, M., Reschke, J. *RFC9111 HTTP Caching* IETF, 2022.

[8] Fielding, R., Nottingham, M., Reschke, J. *RFC9112 HTTP/1.1* IETF, 2022.

[9] Thomson, M., Benfield, C. *RFC9113 HTTP/2* IETF, 2022.

[10] Bishop, M. *RFC9114 HTTP/3* IETF, 2022.

[11] Freier, A., Karlton, P., Kocher, P. *RFC6101 The Secure Socket Layer (SSL) Protocol Version 3.0* IETF, 2011.

[12] Dierks, T., Rescorla, E. *RFC5246 The Transport Layer Security (TLS) Protocol Verison 1.2* IETF, 2008.

[13] Rescorla, E. *RFC8446 The Transport Layer Security (TLS) Protocol Verison 1.3* IETF, 2018.

[14] Kühlewind, M., Trammell, B. *RFC9308 Applicability of the QUIC Transport Protocol* IETF, 2022.

[15] Lee, G. *Cloud Networking* 1. izadnje Morgan Kaufmann, 2014.

[16] IoT mreža. Poveznica: https://www.c-sharpcorner.com/UploadFile/f88748/internet-of-thingsiot-part-4-network-protocols-and-arc/Images/IoT%20Network.jpg; pristupljeno 2. Veljače 2024..

[17] Prikaz objavi/pretplati arhitekture MQTT . Poveznica: https://mqtt.org/assets/img/mqtt-publish-subscribe.png; pristupljeno 3. veljače 2024.

[18] Razmjena MQTT poruka između klijenata i posrednika. Poveznica: https://upload.wikimedia.org/wikipedia/commons/8/82/MQTT_protocol_example_without_QoS.svg; pristupljeno 3. veljače 2024.

[19] Primjer dohvaćanja web dokumenta protokolom HTTP. Poveznica: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview/fetching_a_page.png; pristupljeno 3. veljače 2024.

[20] Primjer HTTP zahtjeva i odgovora. Poveznica: https://www.researchgate.net/profile/Hussein-

Suleman/publication/267954448/figure/fig1/AS:670019910385671@1536756603995/HTTP-communication-example.png; pristupljeno 3. veljače 2024.

[21] Usporedba TCP veze HTTP/1.1 i HTTP/2. Poveznica: https://miro.medium.com/v2/resize:fit:720/format:webp/0*lY05UTuA-dWCXU-q.png; pristupljeno 3. veljače 2024.

[22] Usporedba HTTP/2 i HTTP/3 stoga. Poveznica: https://kinsta.com/wp-content/uploads/2019/03/http2-stack-vs-http3-stack.png; pristupljeno 4. veljače 2024.

[23] Usporedba uspostavljanja veze HTTP/2 i HTTP/3. Poveznica: https://baptistout.net/upgrade-envoy-http3/connection-setup-1.webp; pristupljeno 4. veljače 2024.

[24] Usporedba 1-RTT i 0-RTT uspostavljanja veze. Poveznica: http://www.fasterize.com/wp-content/uploads/2018/11/0-rtt-vs-1-rtt.png; pristupljeno 4. veljače 2024.

[25] HTTP/1.1 HOL blocking. Poveznica: https://miro.medium.com/v2/resize:fit:720/format:webp/0*DuQAxHXb5jbdkXBz.png; pristupljeno 5. veljače 2024.

[26] TCP HOL blocking. Poveznica: https://miro.medium.com/v2/resize:fit:720/format:webp/0*HLTDshj_2KeQbG5T.png; pristupljeno 5. veljače 2024.

[27] Što je NGINX? Poveznica: https://www.nginx.com/resources/glossary/nginx/; pristupljeno 6. veljače 2025.

[28] Tehnička specifikacija NGINX-a Poveznica: https://docs.nginx.com/nginx/technical-specs/; pristupljeno 6. veljače 2024.

[29] NGINX 1.25.3 Poveznica: http://nginx.org/download/nginx-1.25.3.tar.gz; pritsupljeno 6. veljače 2024.

[30] NCHAN Poveznica: https://nchan.io/; pristupljeno 6. veljače 2024.

[31] NCHAN GitHub repozitorij Poveznica: https://github.com/slact/nchan.git; pritsupljeno 6. veljače 2024.

[32] Uputa za izdavanje samopotpisanih SSL certifikata Poveznica: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-18-04; pristupljeno 7. veljače 2024.

[33] OpenSSL dokumentacija Poveznica: https://www.openssl.org/docs/, pristupljeno 7. veljače 2024.

[34] GoDaddy naslovna stranica Poveznica: https://www.godaddy.com/en-ie; pristupljeno 7. veljače 2024.

[35] Pokazni primjer korištenja aioquic Python biblioteke Poveznica: https://github.com/aiortc/aioquic; pristupljeno 7. veljače 2024.

[36] Raspberry Pi 4 model B specifikacije Poveznica https://www.raspberrypi.com/products/raspberry-pi-4-model-b/; pristupljeno 7. veljače 2024.

[37] Smartdraw alat za crtanje mrežnih dijagrama Poveznica: https://www.smartdraw.com/network-diagram/network-diagram-software.htm; pritsupljeno 7. veljače 2024.

[38] Aioquic dokumentacija Poveznica: https://aioquic.readthedocs.io/en/latest/; pristupljeno 8. veljače 2024.

[39] Uvloop dokumentacija Povenica: https://uvloop.readthedocs.io/; pristupljeno 8. veljače 2024.

[40] Wireshark dokumentacija Poveznica: https://www.wireshark.org/docs/; pristupljeno 8. veljače 2024.

Sažetak

Korištenje protokola HTTP3 u komunikaciji uređaja Interneta stvari

U ovom radu je opisana računalna mreža, OSI i TCP/IP modeli mrežne arhitekture, pojam Internet stvari, komunikacijske tehnologije korištene u internetu stvari, protokoli CoAP, MQTT, WebSocket te protokol HTTP s naglaskom na inačicu HTTP3 koja korištenjem QUIC transportnog protokola umjesto TCP je riješila problem head-of-line blockinga,. Zatim je opisana implementacija prototipa klijenta i poslužitelja koji koriste HTTP3 za komunikaciju, okolina Interneta stvari u kojoj je iskorišten prototip te je vrednovana i komentirana analiza ostvarene komunikacije korištenjem prototipa HTTP3 klijenta i poslužitelja u opisanoj okolini Interneta stvari na temelju koje je donesen zaključak da je HTTP3 prikladna komunikacijska tehnologija u Internetu stvari.

Summary

Using HTTP3 protocol for communication in Internet of things devices

This master's thesis explains what a computer network is, OSI and TCP/IP network architecture models, the concept of the Internet of Things, communication technologies used in the Internet of Things, CoAP, MQTT, WebSocket protocols, and the HTTP protocol with a focus on the HTTP version 3, which, by using the QUIC transport protocol instead of previously used TCP, resolves the issue of head-of-line blocking. This thesis also contains the implementation of the prototype client and server using HTTP3, along with the description of the Internet of Things environment in which the prototype is utilized and the analysis of the achieved communication using the prototype in the described environment. Based on this analysis, the conclusion is drawn that HTTP3 is a viable communication technology in the Internet of Things.

Skraćenice

IoT	*Internet of Things*	Internet stvari
WPAN	*Wide Personal Area Network*	osobna mreža šireg prostora
6LowPAN	*IPv6 over Low-Power Wireless Personal Area Networks*	IPv6 na osobnih mrežama šireg prostora male snage
IP	*Internet Protocol*	internetski protokol
LTE	*Long Term Evolution*	mreža dugoročne evolucije
5G	*Fifth Generation*	peta generacija
WAN	*Wide Area Network*	mreža šireg prostora
OSI	*Open Systems Interconnection*	međupovezanosti otvorenih sustava
ISO	*International Organization For Standardization*	međunarodna organizacija za standardizaciju
TCP	*Transmission Control Protocol*	protokol upravljanja prijenosom
UDP	*User Datagram Protocol*	protokol korisničkih datagrama
ARPANET	*Advanced Research Projects Agency Network*	mreža agencija naprednih istraživačkih projekata
ICMP	*Internet Control Message Protocol*	protokol internetskih kontrolnih poruka
SMTP	*Simple Mail Transfer Protocol*	protokol prijenosa jednostavnih mail-a
DNS	*Domain Name System*	sustav domenskom imena
RTP	*Real-time Transport Protocol*	protokol stvarnovremenog prijenosa
HTTP	*Hyper Text Transfer Protocol*	protokol prijenosa hiper teksta
IT	*Information Technology*	informacijska tehnologija
OT	*Operational Technology*	primjenska tehnologija
IoTWF	*Internet of Things World Forum*	svjetski forum interneta stvari
CoAP	*Constrained Application Protocol*	ograničeni aplikacijski protokol
M2M	*Machine to Machine*	stroj sa strojem
MQTT	*Message Queuing Telemetry Transport*	prijenos telemetrije reda poruka
ASCII	*American Standard Code for Information Interchange*	američki standardni kod za razmjenu informacija
TLS	*Transport Layer Security*	sigurnost transportnog sloja
SSL	*Secure Sockets Layer*	sloj sigurnih priključnica

UTF	*Unicode Transformation Format*	unicode transformacijski format
QUIC	*Quick UDP Internet Connections*	brze internetske UDP veze
RTT	*Round Trip Time*	vrijeme obilaska
HOL	*Head of Line*	prvi u redu
HTML	*Hypertext Markup Language*	hipertekst pisani jezik
PNG	*Portable Network Graphic*	prenosiva mrežna grafika