SOME/IP- skalabilni servisno orijentirani posrednički softver preko internet protokola

Burušić, Ana

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije

SOME/IP - Scalable service - Oriented Middleware over IP

Diplomski rad

Ana Burušić

Osijek, 2021. godine

1. UVOD

Automatizacija, sve više različitih mogućnosti koje nam trenutna vozila nude te pojava autonomije pokreću automobilsku industriju naprijed. Svakodnevnim istraživanjima i napretkom stvara se potreba za boljim, bržim, sigurnijim, jeftinijim, učinkovitijim, najčešće unaprijeđenim postojećim alatima koji su već standard i dobro poznati bilo u istoj ili u drugim industrijama. Tako iEthernet koji se godinama unazad nije široko primjenjivao u automobilskoj industriji ali je dobro poznat s više od 30 godina upotrebe u LAN mrežnim tehnologijama.

Povezivanje i komunikacija svih elektroničkih kontrolnih jedinica u automobilu, čiji je broju porastu, imaju potrebu za sve većom širinom pojasa te zahtijevaju ekonomičnost i fleksibilnost te mali trošak implementacije. SOME/IP (Scalable service-Oriented MiddlewarE over IP) nudi te beneficije jer koristi Ethernet i UDP. Dizajniran je u BMW grupi 2011. godine i kompatibilan sa AUTOSAR[1] standardom, a 2013. godine izlazi prva serija automobila s Ethernetom kao sustavnom mrežom [1]. SOME/IP je posrednički softver (engl. *middleware*) za kontrolu komunikacije unutar automobila.

U radu će se osim opisa samog protokola prikazati i opisati izrađena računalna aplikacija koja koristi SOME/IP za komunikaciju. Radi se o poslužitelj (engl. *server*) / klijent (engl. *client*) instancama aplikacije koje međusobno mogu razmjenjivati SOME/IP poruke. Iako SOME/IP može koristiti i UDP (engl. *User Datagram Protocol*) i TCP (engl. *Transmission Control Protocol*) na transportnom sloju, u aplikaciji je korišten UDP jer je jednostavniji i brži. Nema provjera isporučenih, pristiglih poruka i praćenja prijenosa što u ovom slučaju i nije važno. Izrađena je jedna aplikacija s dvosmjernom komunikacijom, odnosno ista aplikacija i šalje i prima poruke ali zasebno pokrenuta na jednom ili više računala u istoj mreži. Svrha aplikacije je prenijeti poruku koristeći SOME/IP protokol . Može se poslati slika, zvuk ili tekst iz jedne pokrenute aplikacije koji se trebaju vidjeti/reproducirati na drugoj pokrenutoj aplikaciji. Omogućava korištenje *Service Discovery* poruka, simulaciju događaja i cikličko slanje obavijesti. Moguće je odrediti vrijeme trajanja jedne ponude/potražnje ili pretplate čitajući vrijednost u polju pristigle poruke koje je specifikacijom određeno da nosi tu informaciju.

[1] AUTOSAR je skraćenica za AUTomotive Open System ARchitecture. Osnovan je 2003. godine kao globalno razvojno partnerstvo zainteresiranih strana u automobilskoj industriji. Cilj im je uspostaviti i stvoriti otvorenu i standardiziranu softversku arhitekturu za ECU (Electronic Control Units) u automobilima. Većina najpoznatijih automobilskih proizvođača su dio ove grupe. Izvor: https://en.wikipedia.org/wiki/AUTOSAR

2. SOME/IP

SOME/IP se oslanja na Ethernet i TCP/IP obitelj protokola. Pruža servisno orijentiranu komunikaciju preko mreže i pruža širok spektar značajki središnje programske podrške[2]:

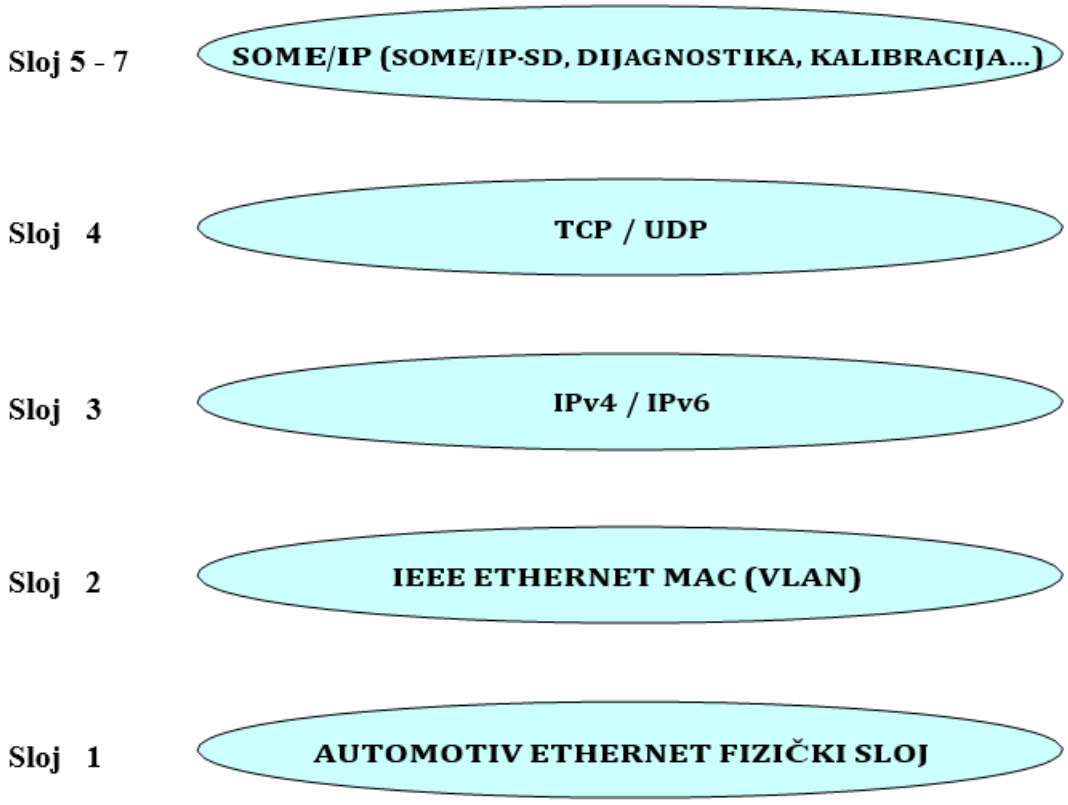
- Serijalizaciju – pretvara objekte ili strukture podataka iz *on wire* reprezentacije i obratno,odnosno prilagođava ih transportu i pohrani
- Udaljeni poziv funkcija *(Remote Process Call (RPC))* – implementira udaljeni poziv funkcija
- Pronalazak usluge *(Service Discovery)* – dinamički pronalazi funkcionalnosti i konfiguriranjihov pristup
- Objava / pretplata (*Publish / Subscribe*) – dinamički konfigurira koji podaci su potrebni i koji će biti poslani klijentu
- Segmentacija UDP poruka – omogućava prijenos velikih SOME/IP poruka koristeći UDP bez da je potrebna fragmentacija

Može biti implementiran na različitim operativnim sustavima, čak i na ugradbenim uređajima bez operativnog sustava. Koristi se za lokalnu klijent/poslužitelj komunikaciju. SOME/IP dopušta aplikacijama da komuniciraju, a formati paketa su određeni specifikacijom servisa. Servis nudi servisne instance na kojima su implementirana servisna sučelja. Klijent koristi te servisne instance. Da bi servisna instanca mogla ponuditi ili pronaći uslugu neki uvjeti moraju biti zadovoljeni. Osim toga, pokretanje ovakvog sustava može zahtijevati vanjske senzore i aktuatore [2]. Od drugih *Middleware*-a se izdvaja zbog slijedećeg [3]:

- Može biti implementiran u AUTOSAR
- Usporediv je i sličan MOST-u (engl. *Media Oriented Systems Transport*)
- Omogućava brzo pokretanje i start cijelog sustava
- Ima jednostavnu serijalizaciju pa podržava i male upravljačke jedinice

SOME/IP je servisno orijentirana komunikacija za razliku od signalno orijentirane. Više je fleksibilnija i dinamičnija u smislu dodavanja ili migracija servisa, ne nudi fiksne IP adrese, a dodavanje i uključivanje novih klijenata je jednostavno. Ima sve benefite Etherneta, odnosno širokopojasnost, veličinu okvira i limitirano korištenje *Broadcast*-a. Definiran je *on-wire* formatom (strukturirani podaci), a mrežne adrese klijenta i poslužitelja nisu statički definirane [6].

Osnovni tipovi podataka koji su omogućeni su unsigned i signed int u rasponu od 8/16/32/64 bita,floating point 32/64 bita, enum, bool, bit polja, strukture i unije, multidimenzionalna polja. Na slici 1. je prikazano kako je SOME/IP raspoređen po slojevima ISO-OSI modela.



Slika 1. SOME/IP po slojevima ISO-OSI modela

Prednosti koje SOME/IP nudi su funkcionalnosti koje ne gubi uklapajući se u postojeći sustav, ima veliku brzinu prijenosa podataka i to može obavljati *unicast*-om, velika je dostupnost njegovih servisa te ima dinamičko IP adresiranje što pruža lako održavanje i fleksibilnost. Osim toga, ima veliku propusnost, malo kašnjenje i toleranciju na gubitak paketa. Ono što bi se moglo ubrajati u nedostatke SOME/IP-a je što ima kompleksniji izračun, samim time i arhitekturu, veći zahtjev za memorijom te manje predvidivo vrijeme izračuna. Iz svih navedenih razloga je pogodan za informacijsko-zabavne sustave i pomoći vozaču sa visokim zahtjevima za prijenos podataka [2].

2.1. Povijest SOME/IP

Kada se počeo razvijati Ethernet ua automobilsku upotrebu gledalo se da se iskoriste postojeća *middleware* rješenja. Kako navode autori knjige „Automotiv Ethernet“ [10], razmatrale su se solucije poput Etch[2], način serijalizacije kakav ima Google Protocol Buffers[3], ili Bonjour[4] za Service Discovery. Svaka od ideja bi morala biti prerađena i prilagođena upotrebi. Nadalje, AUTOASAR pruža mnogo softverskih modula koji uključuju neke od funkcija *middlewarea* i konfiguriraju se uz pomoć zasebnih alata. Da bi se izbjegle nekompatibilnosti, korištenje već postojećih *middleware* rješenja trebalo bi zaobići AUTOSAR module u potpunosti ili osigurati korištenje istih tipova podataka i particioniranje tog postojećeg *middleware*-a tako da on može biti integriran u AUTOSAR. Licenciranje postojećih rješenja nije išlo baš kako je planirano. Iako je bilo dozvoljeno provođenje testiranog i ispitanog rješenja, ključni patenti potrebni za prilagodbu nisu bili dopušteni. Zaštićeni su i u vlasništvu velikih IT tvrtki, s nepoznatim posljedicama. U teoriji, moguće je bilo napraviti jedno od rješenja koristeći AUTOSAR ali ga nije bilo moguće kombinirati s licenciranim proizvodima. Iz tih razloga je odlučeno razvijati novo rješenje. Da bi se izbjegli problemi s licenciranjima, a uzela u obzir IPR (engl. *Intellectual Property Rights*) situacija, istovremeno je objavljena nova verzija rješenja kao state-of-the-art[5]. Naravno, razvijeno je na način da bude upotrebljiv sa AUTOSAR sustavima, odnosno dio njegovog sustava od verzije 4.1. Osim toga, SOME/IP se nudi i kao biblioteka GENIVI [6] sustava [10].

---

[2] Apache Etch Middleware je *open source* program tvrtke Apache Source Foundation. Licenciran je i koristi se kao temeljni protokol komunikacije koji povezuje pametne telefone s jedinicama u automobilu te pruža dvosmjernu komunikaciju na temelju poruka. Neovisan je o platformi, jeziku, *framework*-u te podržava nekoliko programskih jezika. Izvor: https://www.bmw-carit.de/open-source/etch.php

[3] Google je razvio međuspremnike protokola i pružio generator koda za različite programske jezike te ih drži pod *open source* licencom. Puno se koriste se za pohranu i razmjenu svih vrsta strukturiranih podataka. Spremnici protokola je metoda serijalizacije strukturiranih podataka. Izvor: https://en.wikipedia.org/wiki/Protocol_Buffers

[4] Bonjour je implementacija tvrtke Apple, skup tehnologija koja između ostalog uključuje i *Service Discovery.* Koristi se za lociranje uređaja na lokalnoj mreži poput pisača ili drugih računala. Izvor: https://en.wikipedia.org/wiki/Bonjour_(software)

[5] State-of-the-art se odnosi na postignuća u tehničkom ili znanstvenom području u određenom vremenu kad je to postignuće bilo najsuvremenije koristeći i najsuvremenije tehnologije i metode. Ovaj izraz označava i da se radi o pravu na patent i odgovornosti za odštetu. Izvor: https://en.wikipedia.org/wiki/State_of_the_art

[6] GENIVI Aliance je neprofitni savez automobilske industrije, otvorena razvojna zajednica koja zajednički proizvodi automobilske komponente, standardne API-je i razvojnu platformu za *Infotainment*. Savez je osnovan od strane

nekoliko najvećih automobilskih i IT tvrtki u svijetu.

2.2. Usporedba s drugim protokolima

Klasični sustavi temeljeni na sabirnici su signalno orijentirani i tu pošiljatelj odlučuje trebaju li se podaci prenijeti, neovisno o zahtjevu klijenta. Tu je i znatna količina podataka koji nisuzatraženi, a putuju mrežom i zauzimaju je. SOME/IP je *middleware* koji ima drugačiji pristup, odnosno servisno orijentiran je. U suprotnosti od signalno orijentiranih, ovakvi sustavi funkcioniraju i isporučuju podatke jedino na zahtjev klijenta, a ne kada pošiljatelj smatra da bi trebao isporučiti podatak te na taj način povećava kvalitativnu brzinu prijenosa [11]. Kada se spominju informacijsko-zabavni sustavi tu je promet sinkroni, miješani su audio i video podaci i može se usporediti s MOST-om koji se već duži niz godina upotrebljava u multimedijskoj mreži unutar automobila i do uvođenja Etherneta bio je jedina tehnologija u mreži vozila koja podržava pristup temeljen na uslugama. Informacijsko-zabavni sustav je bio prvi koji je trebao složenije sučelje koje može transportirati liste, podržavati složenije tipove podataka, pristupati bazi podatakai slično, a da je pritom orijentirano na usluge. Također, MOST podržava udaljene pozive metoda (RPC). SOME/IP je u ovom slučaju standardniji i brži upravo zbog Etherneta, ima podjednaku otpornost na smetnje ali koristi kabele koji su preskupi za automobilsku industriju i ne podnose najbolje vibracije i koroziju. Postoje čvrsti konektori koji su također skupi. Rješenje je Ethernet putem optičkih vlakana. Jedna od bitnih razlika za napomenuti je činjenica da je Ethernet asinkron dok je MOST sinkron [6]. MOST nudi brzinu od 100 do 150 Mb/s ali i vlasničko licenciranje, ograničen pristup hardveru te teške koaksijalne kabele ili jako osjetljiva optička vlakna podložna oštećenjima što je ograničilo njegovo tržište [9]. Gledano u prošlost, MOST je nudio brži prijenos podataka nego što je trebalo, a prva serijska proizvodnja BMW automobila s MOST tehnologijomje bila 2001. godine. U knjizi [10] je još navedeno da je u početku smatran neprikladnim i skup, no ipak je svoju prvu upotrebu našao u sinkronoj audio komunikaciji.

Još jedna od tehnologija za informacijsko-zabavne sustave je BroadR-Reach koja smanjuje troškove povezivanja u automobilu i samu težinu kablova. Sa brzine od 100 Mbit/s, 1 Gb/s, sada i sa 10 Gb/s posebno pogoduje ADAS algoritmima i obradi slike s kamera. Tehnologija BroadR- Reach je standardni Ethernet fizički sloj koji omogućava da više sustava i uređaja istodobno pristupaju informacijama preko jednog kabela (*Full-duplex*) s jednostrukom upletenom paricom (UTP). Uveden je kao IEEE standard i licenciran od tvrtke Broadcom Corporation. BroadR-Reach je isplativiji od MOST-a i niskonaponskog diferencijalnog signaliziranja (skr. LVDS) najviše zbog troškova u kabliranju, ali LVDS se izuzev toga koristio više kod analognih kamera spojenim koaksijalnim kabelima [8]. Treba uzeti u obzir i bitnu razliku da je MOST tehnologija sabirnice, aBroadR-Reach i SOME/IP dio Etherneta.

Osim za informacijsko-zabavne sustave, kao što je već spomenuto, SOME/IP se koristi za sigurnosne kamere koje imaju kontinuirani *streaming* i trebaju velike brzine prijenosa podataka, dijagnostiku i *flashing* koji zahtijevaju sučelja za vanjske alate i visoku propusnost, kontrolne funkcije ADASA-a koji ima vremenski osjetljivu komunikaciju, velike i male *payloade.* CAN i LIN ne mogu konkurirati navedenim tehnologijama zbog jako malih brzina koje nude, LIN ima i manju propusnost, a FlexRay koji je brži od njih, skuplji i koristi se u vremenski kritičnim aplikacijama. Ostatkom vozila uglavnom dominira CAN gdje se informacije puštaju kanalom i ovise o mehanizmu implementiranom u prijemnik hoće li se te informacije i kako obrađivati. Osim brzine, CAN ima mali *payload* od 8 *byte*-ova i ne dopušta opsežne informacije u zaglavlju što ga sprječava u korištenju udaljenih poziva metoda (RPC) ili otkrivanjima usluga (*Service Discovery*) pa niti ne može zadovoljavati komunikacijske zahtjeve inovativnih domena poput pomoći vozaču. SOME/IP je manje pogodan za sigurnosno kritične sustave ali može zamijeniti FlexRay u intenzivnim, ne-sigurnosnim kritičnim aplikacijama. Osim toga, nije pogodan ni za sustave u stvarnom vremenu poput kontrole motora [2, 5, 7].

Postoji još jedna važna stvar koju pruža SOME/IP-SD. Jedan od kompleksnijih zadataka u dizajnu sustava automobila je njegovo pokretanje svih sustava prilikom paljenja automobila (engl. *start-up*). Svaka upravljačka jedinica ima drugačiji start-up, neke se pokrenu jako brzo dok nekima treba mnogo vremena. Pojedini se pokrenu i kada napon padne ispod 3.5V, a nekima nije ni 8V dovoljno i to znači da se ne startaju sve upravljačke jedinice u isto vrijeme. Bez *Service Discovery*-a moraju postojati čvrsti limiti koji određuju vremensku točku kad su sve funkcije koje se očekuju zasigurno dostupne. To se određuje prema ECU kojemu je potrebno najviše vremena za pokretanje. Suprotno tome, korištenjem SD svaki ECU može obavijestiti kada je spreman i sve bi funkcije bile ranije dostupne, odnosno sve bi se odvijalo brže. To znatno pojednostavljuje proces. Osim toga, tijekom pokretanja cijelog sustava prekidači mogu naučiti tablice adresiranja direktno kroz SD poruku [10, str.228]. Također, mnoge usluge i mogućnosti se daju kupcu na biranje kod kupovine automobila i ovise o cijeni samog, a i mnogi proizvođači nude izradu automobila individualno po zahtjevu kupca. Veći broj opcija znači i veći broj kombinacija već postojećih opcija. Bez SD svaka upravljačka jedinica mora biti statički konfigurirana poštujući dostupnost funkcija i opcija koje nude druge susjedne upravljačke jedinice. Koristeći SD, upravljačka jedinica može samostalnoutvrditi koje usluge se nude od koje upravljačke jedinice u automobilu bez zahtijevanja posebnih kombinacija i pre-konfiguracija. To također pojednostavljuje proces, što kompleksniji zahtjevi u autu to je veća prednost koju nudi SD [10,str.228].

Ipak, SOME/IP pruža brzinu i propusnost za prijenos podataka koji su veliki i „teški“, podaci koji brinu o sigurnosti vozača, brzine koje CAN i FlexRay ne mogu pružiti, a kada se u obzir uzme rastuća industrija autonomnih vozila, uviđaju se prednosti koje automobilski Ethernet s ovakvim protokolom pruža. Dostupan je od mnogih pružatelja usluga, nudi skalabilnost kod umrežavanja i isplativ je zbog korištenja UTP kablova. Svime time nudi veliki potencijal u budućnosti. Tvrtka odgovorna za uvođenje trenutnog automobilskog standarda BroadR-Reach, Broadcom, procjenjuju da mogu smanjiti troškove povezivanja za 80%, a težinu kabela za 30% [9].I sada se predviđa da će se u slijedećih 5-10 godina u vozilima i dalje nalaziti heterogene strukture klaster mreže iz već uspostavljenih sustava sabirnica. Osim za navedene sustave, Ethernet će se početi koristiti u drugim domenama i djelomično će zamijeniti ostale sustave sabirnica, prodrijeti će u druga područja primjene u vozilima [1].

2.3. Ethernet

Ethernet ima različite primjene u raznim industrijama i svaka od njih ima dio onog originalnog „IEEE Etherneta“ bez obzira na ostale prilagodbe svojoj upotrebi. Autor knjige „Automotiv Ethernet“ [10] navodi da će automobil biti drugi čvor u telekomunikacijskoj mreži. Fokus takvog Etherneta je na mrežu unutar automobila, odnosno na komunikaciju između različitih ECU. Zbog toga, industrija vozila nastoji što je više moguće iskoristiti već postojeću tehnologiju na svim slojevima protokola. Uzimajući u obzir i da se godišnje proda nekoliko desetaka milijuna automobila od kojih svaki sadrži više desetaka ECU-a koji moraju biti povezani i komunicirati, za očekivati je da je tržište dovoljno veliko i da se poduzimaju koraci razvoja koji će zadovoljavati traženim zahtjevima proizvođača pritom gledajući i na cijenu koja određenu tehnologiju čini atraktivnijom. Tako se razvoj Etherneta za automobile nije mogao zadržati samo na prva dva sloja ISO-OSI modela, nego obuhvaća sve slojeve. Od najvećih motivacija za korištenje Etherneta u automobilima je bila što su protokoli za sve slojeve već bili dostupni i što industrija može birati koje rješenje će gdje i kako primijeniti [10].

Korištenje Etherneta u vozilima ima dva bitna izazova. Prvo, mora podržavati mnoge i različite slučajeve upotrebe (engl. *Use Cases*), a drugo, Ethernet je u suprotnosti s mnogim drugim uobičajenim komunikacijskim rješenjima u automobilu. Nema sabirnicu pa nema klasičnog emitiranja na medij (engl. *Broadcast-Medium*) [3]. Specificiran je u IEE 802.3 i pokriva prva dva sloja ISO-OSI modela. Dinamika razvoja od samog početka uključivala je razvoj upravljačke jedinice, i osim toga, u području fizičkog sloja je bilo potrebno razvijati primopredajnike (engl. *Transceiver*), konektore, vodiče, utikače (engl. *Plug*), te razmotriti ostale slojeve ISO-OSI

referentnog modela za komunikaciju putem Etherneta koji su iznad fizičkog sloja. Naročito se slojevi iznad podatkovnog mogu uvelike razlikovati i na taj način regulirati propusnost te prilagođavati se raznim primjenama u automobilu. Automobilski Ethernet je svakako drugačiji, ima odlike uobičajenog Etherneta ali je prilagođen korištenju u vozilu. Osmišljen je tako da podržava velik raspon slučajeva upotrebe, a ponajviše za pomoć vozaču i informacijsko-zabavni sustav (engl. *Infotainment*). To je veliki i heterogeni skup zahtjeva i kontrole gdje *middleware* mora moći pružiti rješenje [3]. Osim toga, neophodno je da učinkovita Ethernet mreža u vozilu koristi komunikaciju direktno iz jedne točke prema drugoj (engl. *Unicast*) iako u određenim slučajevima koristi i način komunikacije gdje se podaci istovremeno šalju na više odredišta (engl. *Multicast*). Za razliku od slanja paketa bezuvjetno svima (engl. *Broadcast*), odnosno Ethernet mreže koja koristi ekskluzivni *Broadcast,* kod SOME/IP se opterećenje smanjuje te se povećava opseg komunikacije u mreži. *Multicast*-om iz jedne točke paketi mogu stići na više njih ali samo onima koji ih trebaju i zatraže. Prve od primjena SOME/IP su kamere, pomoć vozaču, udobnost i zabava. I prije njega i drugih kombiniranih mehanizama, na tržištu su dostupni proizvodi i rješenja za povezivanje Etherneta na sustave u stvarnom vremenu kod kojih je fokus bio isključivo na UDP i TCP/IP putem Etherneta. Trenutno je i dalje fokus na njima i slojevima iznad njih no SOME/IP nadopunjuje tada postojeće mogućnosti i mehanizam serijalizacije za upotrebu u komunikaciji temeljenoj na uslugama te uspostavlja vezu između UDP ili TCP/IP i aplikacijskog sloja [5].

Kako svi sudionici u mreži dijele fizički prijenosni medij, može doći do kašnjenja u prijenosu podataka. To kašnjenje se može smanjiti spajanjem od točke do točke pomoću prekidača (engl. *switches*). U radu [4] iz 2013. godine se navode dva osnovna koncepta za komunikaciju putem Etherneta koji se mogu koristiti u AUTOSAR-u i FIBEX-u[7]. Opisan je način korištenja SOME/IP i njegove mogućnosti. Klasična komunikacija na temelju signala poput CAN-a, također se svodi na komunikaciju između kontrolnih jedinica koje komuniciraju preko Etherneta, ali za to i PDU [8] (engl. *Protocol Data Unit*) mora imati tu mogućnost. PDU-ovi se mogu slati putem takozvanih *socket*-a koji se apstrahiraju s IP adrese i priključaka (engl. *port*). Ethernet sa svojim širokim pojasom propusnosti nudi više mogućnosti od prenošenja jednostavnih podataka putem signala. U AUTOSAR-u ili

[7] FIBEX je skraćenica za Field Bus Exchange Format. Razvio ga je ASAM konzorcij kao standardizirani format koji se koristi za mreže u vozilima. Automobilska industrija ga koristi za jednostavnu razmjenu podataka, a ima proširivost potrebnu za razne mrežne protokole. Izvor: https://en.wikipedia.org/wiki/Fibex

[8] Protocol Data Unit u telekomunikacijama je jedinica podataka koja se prenosi između entiteta mreže. Sastoji se od kontrolnih informacija i korisničkih podataka. Vrsta i način razmjene podataka ovisi o slojevima protokola pa je primjerice PDU u TCP prijenosu nazvan segment, u UDP-u je *datagram* kao podatkovna jedinica, a u nižim slojevima se naziva paketom. Izvor: https://en.wikipedia.org/wiki/Protocol_data_unit

FIBEX-u se mogu specificirati složena servisna sučelja koja sadrže metode i događaje i na taj način se mogu prenijeti strukturirane i složene informacije za razliku od jednostavnih signala. Nudi mogućnost korištenja generičkih vrsta podataka za parametre metoda i događaja. Upravljačka jedinica nudi funkcionalnu jedinicu s takozvanim servisnim sučeljem, dok druga upravljačka jedinica može koristiti ta ponuđena servisna sučelja i pozivati metode s prvog upravljačkog uređaja. Korisnik usluga, odnosno upravljačka jedinica koja koristi usluge druge upravljačke jedinice nazivamo klijentom (engl. *Client*), a onu upravljačku jedinicu koja nudi i pruža usluge nazivamo poslužiteljem (engl. *Server*). Ako klijent želi postaviti upit, mora pozvati metodu instance servisnog sučelja s potrebnim parametrima i tada se taj zahtjev šalje poslužitelju. Poslužitelj zaprima zahtjev, pregleda ga, obradi i izračuna odgovor, zatim šalje parametre odgovoraklijentu. Za drugi način prijenosa podataka navode „Fire&Forget“ metodu gdje klijent ne očekuje odgovor poslužitelja. Postoji i varijanta gdje poslužitelj dostavlja potrebne podatke klijentu bez da klijent to svaki put izričito zatraži. To se postiže takozvanim događajima (engl. *Events*). Da bi klijent primio te podatke mora se prvo pretplatiti (eng. *Subscribe*) na događaj koji treba. Na taj način se događaji ne šalju svima kao što je to slučaj sa klasičnim sustavom sabirnica, primjerice CAN-om, nego samo onima koji su se pretplatili na taj događaj. Za dinamičku upotrebu servisa s njihovim metodama i događajima koristi se *Service Discovery* (skr. SD). Ethernet podržava učinkovitu *unicast* komunikaciju. Prilikom adresiranja, odnosno komunikacije s prijemnikom zauzima se samo potrebna širina pojasa na Ethernet vezi. To omogućava veću količinu moguće komunikacije u cjelokupnom sustavu i smanjuje neželjenu i nepotrebnu komunikaciju koju bi ECU morao filtrirati. Za *unicast* komunikaciju pošiljatelj mora biti siguran da poruku koju šalje primateljmože primiti. Ako je primatelj nepoznat *switch*-evima u trenutku slanja, poruke se gomilaju i stvaraju gužvu na sabirnici i tada bi nestala prednost *unicast*-a. Iz tog razloga je 2012. godine SOME/IP proširen sa *Service Discovery* (SOME/IP-SD) gdje je i u AUTOSAR-u kao SWS *Service Discovery* standardiziran. SD ima zadatak o najavi i dostupnosti (IP adresa i *port*) servisne instance te mehanizam registriranja na grupu događaja (*Publish/Subscribe*). Za provedbu tih zadataka SOME/IP-SD servisne instance redovito komuniciraju i prenose različite unose (engl. *Entry*) koje su dio poruka. Na taj način je omogućena učinkovita unicast komunikacija na Ethernetu, a istovremeno umanjena količina cikličkih poruka budući da su komunikacijski partneri eksplicitno informirani o dostupnosti usluga [3, str.2-3].

2.4. Dinamičko vs. Statičko adresiranje

Današnji sustavi imaju promjenjiv broj čvorova i putova u mreži. Da bi sustav bio fleksibilan, jedna od ključnih stvari je da ima dinamičko dodjeljivanje IP adresa. Na mrežu u automobilu se može gledati kao zatvorenu, unutarnju, i iako broj aktivnih čvorova u mreži može varirati, njihov maksimalan broj je poznat i limitiran već unaprijed. Za razliku od ostalih IT mreža, auto može više puta u danu biti upaljen i ugašen, a brzo pokretanje cijelog sustava pri paljenju je veoma bitno i daje naslutiti da je statička IP adresa pravilan izbor. No, u knjizi autora K. Matheus i T. Königseder [10, str. 221] se navode slučajevi upotrebe dinamičkog dodjeljivanja IP adrese koji se koriste u vozilu.

Kod statičkog dodjeljivanja, IP adrese su pridružene svakom ECU još tokom razvoja. ECU koji pripadaju istoj klasi imaju iste IP adrese gledajući na automobile u koje su ugrađeni. Kako se adrese očito ponavljaju, izabrane su iz nekog određenog opsega adresa koje mogu biti izabrane samo za njih, ali i ne moraju. U tom slučaju kod statičkog adresiranja je vrlo važno da dva ECU iz iste klase ne budu ugrađena u isti automobil.

Postoji i pseudo-dinamičko adresiranje. To je slučaj kada se ECU isporuči bez IP adrese i ona mu se dodjeljuje prilikom ugrađivanja u vozilo. Jednom kada se dodijeli onda je statična, odnosno ne može se više mijenjati. Takav slučaj postoji kada je više identičnih uređaja unutar vozila, poput kamera za *surround view*. Potpuno ista kamera se montira na različite lokacije u vozilu pa se one isporučuju bez IP adrese. To smanjuje troškove logistike i pohrane. Ova procedura je standardizirana s ISO 17215.

Dinamičko adresiranje je potrebno kada automobil komunicira s vanjskim svijetom ili vanjskim komponentama, primjerice tokom provođenja dijagnostike ili povezivanja na Internet. U tom slučaju nije više moguće koristiti interne adrese, ECU je vozilo direktno povezao s vanjskim svijetom i koristi IP adresu koju mu je vanjski DHCP (engl. Dynamic Host Configuration Protocol) dodijelio. Mogućnost da se drugi ECU unutar tog auta spoje kroz isti „ECU *port*“ je jedino na način da se implementira prijevod dinamične/statične adrese u mreži [10].

2.5. Kibernetička sigurnost Etherneta

Sigurnost se još uvijek temelji najviše na zaštiti sabirnice od komprimiranih upravljačkih jedinica, no fizička arhitektura, virtualna segmentacija, specifičnosti mreže i nove opcije i mogućnosti koje pruža izazivaju nove probleme. Sigurnost automobilskog Etherneta mora uključivati više od detekcije, odbacivanja i preusmjeravanja zlonamjernih signala. Ethernet i njegova proširenja se bave upravljanjem mrežnim resursima i nude razne scenarije napada poput neiskorištenih *port*-ova, *MAC spoofing*-a[9] , iskorištavanje propusnosti mreže, do sofisticiranijih poput TCP „otmica“ i VLAN skakanja. Oni zahtijevaju razmatranje sigurnosti još u fazi projektiranja mrežne arhitekture. Ključni problemi koji su se događali sa sabirnicama, primjerice CAN-om, je bila njegova mogućnost da obrađuje vrlo velike količine podataka za sustave poput prepoznavanja i izbjegavanjapješaka te činjenica da elektroničke kontrolne jedinice (ECU) nemaju autentifikaciju kod komunikacije jedna s drugom i to je jedan od razloga da konzorcij automobilskih proizvođača definira i implementira automobilski Ethernet. Uz sve već spomenute i nabrojane prednosti ove mreže ipak je ranjiva i podložna *cyber*-napadima. Među prvim pretpostavkama je da veliki dio programskog koda kopiran i preuzet iz postojećih Ethernet sustava, što zbog skraćenih rokova izrade, što zbog već postojećih mehanizama koji dobro rade i prošli su testiranja i provjere te nema potrebe niti vremena izmišljati nove. Za kôd koji se ne mijenja uvijek postoje mogućnosti ranjivostikoje do sad nisu otkrivene i neće biti pronađene i ispravljene, a hakerima je već poznat način rada i arhitekture.

Svakako se sigurnost Etherneta u automobilu mora razlikovati od klasične sigurnosti i biti posebno dizajnirana za tu uporabu [17]. Trendovi automobilske industrije poput autonomne vožnje i pomoći vozaču zahtijevaju velike propusne širine zbog velikog protoka podataka i jako ovise o dobroj komunikaciji. Brzina prijenosa podataka je sve veća i postavlja se pitanje kako postići sigurnost i pouzdanost u očekivanom vremenu čak i kod prijenosa vrlo velikih količina podataka poput video signala. Kako je Ethernet decentraliziran, time se jedna ranjiva točka centralnog entiteta rješava no nastaju drugi problemi. Svi imaju jednake povlastice i mrežnih putova je puno s istim pravima, tada je teško otkriti i zabraniti pristup neovlaštenim članovima mreže. Uspostavljenasu rješenja poput virtualnih mreža, VLAN-ova za podjelu mrežnog prometa te je moguće označiti Ethernet *port*-ove i okvire neovisno o VLAN-u i to je regulirano IEEE 802.1Q standardom. To i dalje ne rješava problem komunikacije s neželjenim uređajima i zbog toga se uvodi kriptografska provjera autentičnosti i enkripcija. Koriste se klasična IT rješenja usmjerena na gornje slojeve protokola te šifriranje IP paketa na sloju 3 i Ethernet okvira na sloju 2. Postoje i razna pravila filtriranja, vatrozidi

[9] MAC spoofing je tehnika za promjenu tvornički dodijeljene MAC adrese. U pravilu se MAC adresa ne može promijeniti no mnogi upravljački programi dopuštaju promjenu, a ut to postoje i alati koji mogu zavarati OS i maskiratipravu adresu. Uglavnom podrazumijeva promjenu identiteta računala iz određenog razloga. Izvor: https://en.wikipedia.org/wiki/MAC_spoofing

koji kontroliraju koji se paketi kojim putovima kreću… Sustavi za otkrivanje provale (*Intrusion detection systems* - IDS) i sustavi za zaštitu od neželjenih upada u komunikaciju (*Intrusion detection systems* - IPS) se temelje na navedenom te osim toga mogu analizirati i procijeniti promet sve do najdublje analize paketa uključujući *payload*. Mnogo je zajedničkog u mrežnim arhitekturama IT svijeta i modernim automobilima no načini i ciljevi zaštite se mnogo razlikuju, a najveća razlika koja ih razdvaja je rad u stvarnom vremenu što se do sad od Etherneta nije zahtijevalo, a osim toga ECU ima ograničenu računalnu snagu pa se i to mora uzeti u obzir. Korištenjem vatrozida ili moćnih IDS/IPS sustava moguće je strogo razdvojiti IP adrese sa različitim zahtjevima za zaštitu i preciznije ih nadzirati za razliku od klasičnih sustava automobilskih sabirnica. Osim toga, autori „*Ethernet Security*“ [18] navode da sa sigurnosnim protokolom za TLS (*Transport Layer Security*) treba biti oprezan jer prijeti da će izazvati sukobe sazahtjevima u realnom vremenu pa se treba upotrebljavati samo u vremenski nekritičnim situacijama. Isti ti zahtjevi su prepreka i u korištenju kriptografskim potpisima temeljenim na asimetričnim procesima. Kako bi se zaštitila autentičnost paketa, za to je AUTOSAR 4.2 2014. godine objavio SecOC (*Secure On-Board Communication*). Specifikacija je toliko fleksibilna da je prikladna za Ethernet/IP komunikaciju što znači i za SOME/IP ali i za različite vremenski osjetljivemreže uključujući i prijenos videa AVB (*Audio Video Bridging*) koji je razvijen za prijenos vremenski kritičnih podataka. SecOC definira vlastite mehanizme za upravljanje rezervacijom mrežnih resursa, sinkronizacijom vremenskih signala i prioritetima. Najnovija inačica je podržavalai šifriranje prenesenih podataka s vrlo niskim hardverskim zahtjevima [18].

Evolucija postojećih funkcionalnosti za udobnost, pomoć i sigurnost vozača te integracija dodatnih značajki zahtijevaju visoko složene automobilske sustave s naprednim dijelovima hardvera i softvera. Oslanja se na povećanje računalne snage te na veliku razmjenu podataka između senzora, aktuatora i uređaja za obradu podataka. Autori „*Automotive Ethernet: Security opportunity or challenge?“* [19] predstavljaju razlike i potencijale sigurnosnih mehanizama izmeđuautomobilskih i uobičajenih Ethernet mreža. Sigurnost na koju se oni fokusiraju je fizički, mrežni, podatkovni i transportni sloj, IPv4, IPv6, TCP i UDP. Različiti protokoli i sabirnice dovode do heterogenih internih mreža u kojima je potrebno prilagođavati i pretvarati protokole da bi se omogućila međusobna razmjena podataka. Sve dosadašnje tehnologije sabirnica se temelje na arbitražnim mehanizmima i vremenskim intervalima za pristup sabirnici. Ethernet dodaje tehnologiju koja se temelji na paketu i sustavu od točke do točke (engl. *point-to-point*).

Glavna razlika između automobilske Ethernet mreže i one klasične je u hijerarhijskoj strukturi i kombinaciji različitih topologija sabirnica. Fizički sloj je odgovoran za prijenos digitalnih informacija putem analognih signala. Odabir odgovarajućeg transportnog medija ovisi o zahtjevima za širinu pojasa, jednosmjernoj ili dvosmjernoj komunikaciji, elektromagnetskim utjecajima, ekonomičnosti, udaljenosti koju treba savladati. Slično je i kod izbora elektroničkih kontrolnih

jedinica koje moraju osim funkcionalnosti biti dizajnirane i za podnošenje vlage i vibracije. Osim toga, bitan je i operacijski sustav koji se primjenjuje. Komunikacija unutar Etherneta definirana je zaglavljem protokola i s obzirom na sve moguće atribute, frekvencije poruka i omjer TCP i UDP veze pružaju parametre za razvoj sigurnosnih mehanizama za automobilske Ethernet mreže. Sigurnost vozila ovisi o ispravnosti podatka, njegovoj brzini, odnosno točnom vremenu i istinitosti razmijenjenih informacija između senzora, aktuatora i ECU-a pa zaštita od manipulacije ima prednost nad zaštitom od prisluškivanja. Nakon puštanje proizvoda, u ovom slučaju protokola na tržište, malo je vjerojatno da će se raditi izmjene i ažuriranje osim u domeni zabavnih i informativnih sustava [19]. Objavljena specifikacija je bitna za komunikaciju u mreži, a nakon njene objave promjene su malo vjerojatne. Korisniku je omogućena promjena *payloada*-a putem HMI (*Human Machine Interfaces*) ali ne može mijenjati postojeće strukture protokola ili uvesti vlastite protokole. Interakcija putnika putem upravljačkih sučelja izričito je ograničena. Uvođenje kompletne enkripcije podataka bi uzrokovalo veliku upotrebu resursa što bi dovodilo do kašnjenja koji u mnogim slučajevima nisu prihvatljivi. Primjena kodova za provjeru autentičnosti poruka (MAC) se čini učinkovitijim u tom slučaju [19].

Ethernet je mnogima dobro poznata tehnologija, opsežno opisana i dostupna pa je broj potencijalnih napadača puno veći za razliku od nekih drugih tehnologija i sabirnica. Resursi u automobilu su ograničeni pa su ograničeni i sigurnosni uređaji i mehanizmi koji se mogu upotrebljavati. Ethernet zbog svojih prednosti koje nudi se širi u automobilskom svijetu stječe pozornost i potencijalnih napadača, a mrežne komponente su grupirane u vozilu i pristup im je jednostavan. Povećanje propusnosti ipak omogućava mehanizam za provjeru autentičnosti i šifriranja poput MAC-a, a ažuriranja se mogu brže provoditi. Komunikacijski putovi unutar mreže se mogu ograničiti korištenjem prekidača (engl. *switch*) u kombinaciji sa ranije spomenutom specifikacijom IEEE 802.1Q VLAN-a i kvalitete usluge (QoS). Sigurnosni mehanizmi za fizički sloj se ne razmatraju zbog velikog broja primopredajnika (engl. *transciver*), različitih kablova, proširenja i brzina prijenosa ali je prihvatljivo uvođenje jednostavne enkripcije kako bi se spriječilo dodavanje novih ili zamjena postojećih uređaja. Ograničen dizajn ECU-a i poznavanje svih mrežnih atributa i parametara omogućava korištenje meta[10] informacija za otkrivanje neobičnog ponašanja ili zlonamjernog uređaja u mreži vozila. Netipično ponašanje sustava ili mreže se može otkriti i mapiranjem određenih identificiranih stanja sustava u tokove razmjene podataka [19].

[10] Meta podaci u digitalnom smislu objašnjavaju „strukturirane podatke koji opisuju, objašnjavaju, lociraju ili na neki drugi način omogućavaju lakše upravljanje resursima.“ Oni opisuju karakteristike nekog izvora u digitalnom obliku. Izvor: https://hr.wikipedia.org/wiki/Metapodatci

2.6. SOME/IP Middleware

Middleware je posrednički softver koji povezuje zahtjeve u mreži od strane klijenta, aplikacije, sa pozadinskim podacima koje klijent traži, primjerice iz baze podataka. Zahtjevi koji se temelje na mreži su zapravo pokušaj interakcije sa krajnjim podacima gdje *middleware* ima bitnu ulogu i može imati razvijenu logiku koja na temelju IP adrese ili nekog od ulaznih parametara vraća rezultat. Osim toga, preuzima opterećenje i kontrolu pristupa podacima na sebe. U ISO-OSI modelu, on je na višim slojevima. Upravlja transportom kompleksnih podataka, poruka te udaljenim pozivom metoda (RPC) između komponenti. Nedostatak njegove primjene je u njegovoj veličini i opterećenju ali povećanjem performansi ostalih sustava to je zanemareno. Osim toga, u sve kompleksnijim sustavima postaje važno ugodno rukovanje podacima i povećanje kvalitete. Još jedna od prednosti korištenja posredničkog softvera je poboljšanje distribuiranog razvoja na različitim modulima i sama njihova stabilnost. Količina softvera i podataka u današnjim automobilima je ogromna i povećava se. Oni mogu koristiti razne procese s jednog ECU-a ali i povezivati više njih s više različitih ECU-a. Oni mogu imati i različitu arhitekturu, različite operacijske sustave… Jednostavno puštanje poruku u mrežu s pretpostavkom da će završiti na dobrom mjestu u ovakvom kompleksnom sustavu više nije prihvatljivo [10].

Middleware SOME/IP je razvijen za upotrebu u vozilu i uključuje serijalizaciju, razmjenu poruka, otkrivanje servisa te objavljivanje i pretplatu na usluge. Skalabilan je i može se koristiti na malim upravljačkim jedinicama poput kamere, AUTOSAR ili *infotainment* upravljačkim jedinicama. Najvažnije je da je podržan od različitih implementacija i vodećih alata i pružatelja usluga [3].

2.7. Serijalizacija i deserijalizacija podataka

Glavni od zadataka *middleware*-a je serijalizacija i deserijalizacija između internih i eksternih ECU-a, te razmjena poruka. Kada se podaci pohranjuju ili prenose preko mreže, može doći do situacije da će ih čitati uređaj koji ima drugačiju arhitekturu, operacijski sustav ili broj bitova koje koristi za adresiranje i slično. Mehanizam kojim su se podaci prenijeli i pohranili mora biti dosljedan daljnjem čitanju i prijenosu tih podataka kako bi oni ostali valjani. Serijalizacija je proces pretvorbe složenih tipova i struktura podataka u niz binarnih brojeva koji se prenose mrežom i pohranjuju. Obrnuti postupak čitanja tih podataka i vračanje u smislene tipove i strukture u kakvima su bili se naziva deserijalizacija. SOME/IP poruka može imati jedan ali i više parametara koji se njome prenose. Primjerice, nekoliko ECU-a može biti zainteresirano za podatako temperaturi motora i okretajima u minuti. ECU u motoru prenosi te vrijednosti ciklički SOME/IP događajem pa će te podatke dobiti svi pretplaćeni ECU-i. Događaj u tom slučaju ima 3 parametra, temperaturu, okretaje u minuti i smjer [15]. Ako taj ECU iz nekog razloga želi prenijeti još jedan parametar, pridružiti će

ga ostalima prethodno navedenima u *payload*-u što znači da se duljina serijaliziranog *payload*-a promijenila. Deserijalizator mora znati točnu duljinu i položaj tih parametara te se stoga mora uskladiti način razumijevanja između serijalizatora i deserijalizatora. Također, serijalizator mora obavijestiti i o promjenama koje korisnik napravi. Jedno od rješenja koje se koristi u automobilskoj industriji za uspostavljanje „razumijevanja“ između njih je koristeći datoteku baze podataka, XML ili ARXML formata koja sadrži konfiguraciju svakog ECU-a. Ta datoteka može imati i tisuće parametara koji se ažuriraju kod bilo kakve promjene u ECU-u. Trenutno se koriste dvije datoteke, jedna je od AUTOSAR-a, a jedna je za sabirnice (FIBEX) od ASAM-a. Bitna informacija prilikom serijalizacije je jesu li podaci osnovnog ili složenog tipa i jesuli promjenjive ili fiksne duljine. Parametre koji se čitaju iz datoteke je potrebno prvo raščlaniti na odgovarajući način (engl. *parse*) da bi se dobile strukture podataka koje su konfiguracijski parametri potrebni za serijalizaciju. Ti parametri mogu biti lista objekata od kojih svaki sadrži potrebne parametre za serijalizaciju koji se odnose na podatke svakog pojedinog korisničkog ulaza. Znači da bi broj objekata u toj listi bio jednak broju korisničkih ulaza. Kod serijalizacije i deserijalizacije je bitan raspored *byte*-ova, tip podatka i poravnanje. Ako su uključeni nizovi ondaje bitna njihova dimenzija, dužina tog polja, odnosno broj *byte*-ova koji se prenose, a taj podatak nosi polje ispred njih, zatim podatak radi li se o fiksnoj ili varijabilnoj veličini te maksimum i minimum duljine ukoliko je ona varijabilna. Isto je i u slučaju prijenosa *string* podatka samo umjesto dimenzije je potrebna encoding informacija. Modul koji raščlanjuje podatke iz baze te sami serijalizator, zajedno čine SOME/IP modul za serijalizaciju. Nakon toga se poruka prenosi putem Etherneta i SOME/IP transportnog protokola, odnosno koristeći UDP ili TCP [15].

SOME/IP nudi najjednostavnije rješenje za serijalizaciju podataka bez složenih pretvorbi. Definiran je točan položaj svih struktura podataka u PDU te se mora odrediti način usklađivanja memorije. Usklađivanje se koristi za poravnavanje početka podataka umetanjem obložnih elemenata (engl. *padding*) kako bi se osiguralo da podaci počinju na određenim memorijskim adresama, no postoje i arhitekture procesora koji pristupa podacima na efikasniji način. Poravnanje se uvijek računa od početka SOME/IP poruke. Iza podataka fiksne duljine ne smije biti obložnih elemenata, a ako se oni iz nekog razloga ipak moraju umetnuti onda to mora biti izričito naglašeno u definiciji tipa podatka. Serijalizacijom se ne pokušavaju automatski poravnati podaci ali moraju biti usklađeni kako je određeno specifikacijom [12]. Podaci za poravnanje su višekratnici 8, 16 ili 32 bita. Redoslijed *byte*-ova je određen konfiguracijom, jedino se za *boolean* vrijednost uzima najniži bit, a ostali se ignoriraju. Ako je umetanje obložnih podataka potrebno onda je dizajner sučelja odgovoran za to jer SOME/IP implementacija to neće automatski odraditi [12].

Tipovi podataka koje podržava se lako uklapaju (tablica 1.), primjerice uint8, kako je predstavljen na mreži, tako je i interno. To je u suprotnosti s rješenjima kao što su primjerice Google

međuspremnici protokola. Vrijednosti su prenesene na temelju varijable, komprimirani je prikaz i za razliku od CAN-a, može se koristiti za izuzetno složene protokole. CAN okvir ima vrlo ograničenu veličinu podataka od 8 byte-ova kao i veličinu pojasa koju onda mora nadoknađivati. SOME/IP može transportirati vrlo velike pakete, a Ethernet ovisno o sustavu pruža puno veću pojasnu propusnost. Analogno CAN-u, pored pojedinačnih podataka, veće količine se mogu prenijeti pomoću transportnog protokola (TCP) [11].

AUTOSAR određuje da se sva polja moraju prenositi zajedno s „poljem duljine“ koje se nalazi na početku polja koje se prenosi. Ono pomaže dodijeliti odgovarajuću veličinu međuspremnika tijekom deserijalizacije. Tip polja veličine može biti uint8, uint16 ili uint32 i time je i određena duljina polja iza njega.

Nizovi (eng. *strings*) se poput drugih tipova podataka prenose kroz mrežu i pohranjuju u slijedu bitova. Najrašireniji formati kodiranja su ASCII (*American Standard Cod for Information Interchange*) i *Unicode*. Primjena *Unicode*-a omogućava kompatibilnost softverskih platformi, uređaja i aplikacija. Taj standard dodjeljuje jedinstveni broj za sve znakove različitih jezika svijeta. Kada bi svakom znaku i slovu svakog jezika na svijetu dodijelili jedan broj, to bi zauzimalo jako puno memorije u odnosu na trenutnu izvedbu, zbog čega i jesu globalno uvedeni formati kodiranja. SOME/IP podržava UTF-8, UTF-16LE (*Little Endian*) i UTF16BE (*Big Endian*) oblik *Unicode*-a. UTF-8 koristi jednu do četiri 8 bitne sekvence za kodiranje svih znakova definiranih *Unicode*-om. Kada se koristi samo jedan *byte*, prvih sedam bitova prezentiraju ASCII znakove što znači da je kompatibilan s njim. Dva byte-a su za slijedećih 1920 znakova, tri već uključuju kineske, japanske i korejske znakove, a četiri za sve ostalo uključujući razne fontove, simbole i slično. UTF-16 se koristi za kodiranje varijabilnih duljina koristeći jednu ili dvije 16 bitne sekvence [15].

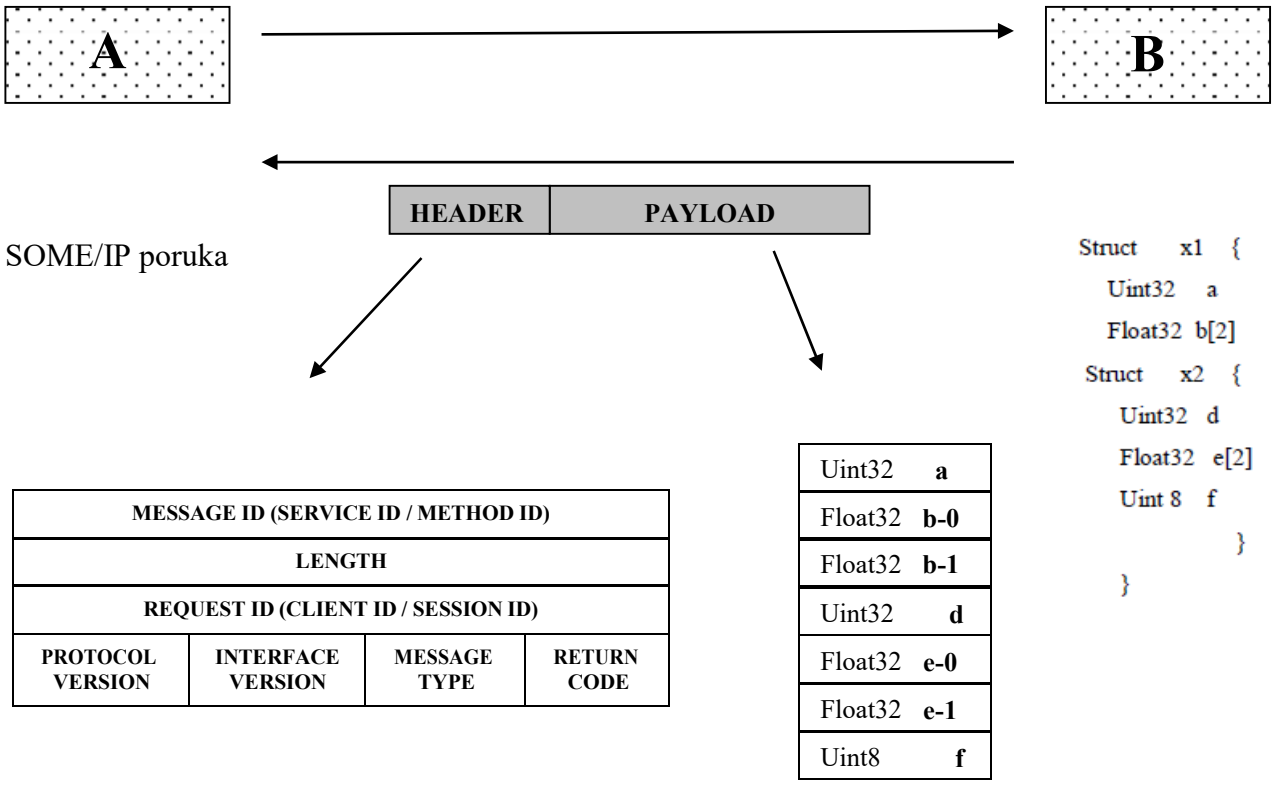
Može se dogoditi da čvorovi ne koriste isti operacijski sustav i komunikacija između njih bi bila nerazumljiva. Serijalizacijom se analiziraju strukture podatkovnih jedinica RPC PDU u AUTOSAR-u i pretvara ih u niz bajtova spremnih za prijenos. Kad drugi čvor primi taj niz bajtova, SOME/IP protokol provodi deserijalizaciju da bi vratio nazad u odgovarajuću strukturu za potrebni OS [11].

TIP	OPIS	VELIČINA (BIT)	NAPOMENA
boolean	True/False vrijednost	8	False (0), True (1)
uint8	Pozitivni cijeli broj	8	
uint16	Pozitivni cijeli broj	16	
uint32	Pozitivni cijeli broj	32	
sint8	Pozitivni ili negativni cijeli broj	8	
sint16	Pozitivni ili negativni cijeli broj	16	
sint32	Pozitivni ili negativni cijeli broj	32	
float32	Broj s pomičnim zarezom	32	IEEE 754 binary32 (mala preciznost)
float64	Broj s pomičnim zarezom	64	IEEE 754 binary32 (dvostruka preciznost)

Tablica 1. Tipovi podataka podržani u SOME/IP

3. SOME/IP poruka

Uzmimo za pretpostavku da se na uređaju B nalazi servis koji nudi funkciju koja se poziva s uređaja A porukom kako je prikazano na slici 2.



Slika 2. Primjer strukture SOME/IP poruke u komunikaciji između dva uređaja

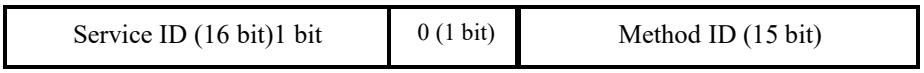
SOME/IP poruka se sastoji od zaglavlja (engl. *header*) i takozvanog korisnog tereta (engl. *payload*). Zaglavlje sadrži osnovne i važne informacije o samoj poruci dok je u polju *payloada* struktura poruke koja se prenosi. Sva specifikacija protokola je određena AUTOSAR standardom koji je isključivo osmišljena za upotrebu u automobilskoj industriji [12] i kao takva je navedena i u ovom radu.

Zaglavlje se sastoji od podataka koji su navedeni na slici 2, a detaljnije su objašnjeni u slijedećem dijelu te će često spominjati kroz cijeli daljnji tekst.

3.1. Zaglavlje

Message ID (ID poruke) 32 bit – identifikator koji se koristi za prepoznavanje RPC poziva za metodu ili događaj. Dodjela tog ID-a ovisi isključivo o klijentu ali on mora biti jedinstven za cijeli

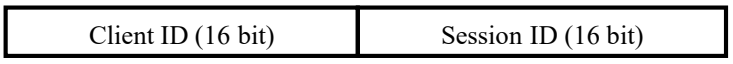
sustav (auto). *Message* ID se sastoji od *Service* ID (ID servisa/usluge) i *Method* ID (ID metode), odnosno od 2^{16} servisa i 2^{15} metoda. ID servisa je također unikatan identifikator za svaki servis. ID metode se dijeli u rangu od 0-32767 ako se radi o metodi, od 32768-65535 ako se radi o događaju. Na slici ispod (Slika 3.) je prikazano polje *Message* ID.



Slika 3.Sadržaj polja Message ID iz zaglavlja SOME/IP poruke

Length (Duljina) 32 bit – sadrži duljinu *Payloada* u *byte*-ovima uključujući i duljinu polja odmah iza ovoga, odnosno duljinu od *Request* ID pa do kraja SOME/IP poruke. Kako se izražava u *byte*- ovima, potrebno je preračunati bitove u *byte*-ove. Veličina poruke ovisi o transportnom protokolu, no za primjer izračuna uzmimo UDP poruku koja je maksimalne veličine 1400 *byte*. Request ID sadrži 32 bita, *Protocol Version*, *Interface Version*, *Message Type* i *Return Code* svaki po 8 bita što je još 32 bita. Ako ih zbrojimo dobijemo 64 bita što je zapravo 8 *byte*-ova. Kad od ukupne veličine UDP poruke oduzmemo spomenuta polja, ostane nam 1392 *byte*-a za transport poruke, tj. za *Payload*. Veće poruke se prenose TCP protokolom, no postoji način prijenosa većih poruka putem UDP paketa što će biti u jednom od narednih poglavlja objašnjeno.

Request ID (ID zahtjeva) 32 bit – omogućava da poslužitelj i klijent mogu razlikovati više paralelnih, istovremenih upotreba iste metode, događaja ili polja. Mora biti jedinstven u cijelom automobilu i jedinstven za pojedinu pretplatu, odnosno poslužitelj - pretplatnik kombinaciju. Kada poslužitelj odgovara na poruku, on samo kopira ovaj ID iz poruke zahtjeva u poruku odgovora koji vraća. Taj ID se ne smije ponovno upotrijebiti dok ne stigne odgovor ili dok ne istekne vrijeme očekivanja odgovora. *Request* ID se sastoji od *Client* ID i *Session* ID polja kako je prikazano na slijedećoj slici (Slika 4.):



Slika 4. Sadržaj polja Request ID iz zaglavlja SOME/IP poruke

Implementator ECU-a može definirati ID-eve kako želi, poslužitelju to nije važno jer on samokopira kompletni *Request* ID u odgovor.

Client ID (ID klijenta) sadrži 16 bita i jedinstveni je identifikator koji označava klijenta unutar ECU. Pomoću njega poslužitelj može razlikovati pozive istih metoda različitih klijenata. *Client* ID ima prefiks od 8 bitova pa za sami ID ostane još 8 bitova.

Session ID (ID sesije) je također unikatni identifikator od 16 bitova kojeg izabire klijent prilikom svakog poziva. Ovaj ID omogućava da klijent može raspoznati više istih metoda koje je pozvao. *Request* / *Response* metode koriste *Session* ID. Nakon svakog poziva on se inkrementalno povećava, a kad dosegne 0xFFFF kreće ispočetka. Klijent će zanemariti odgovor ako ID odgovora ne odgovara ID-u zahtjeva.

Protocol Version (Verzija protokola) 8 bit – sadržava verziju protokola

Interface Version (Verzija sučelja) 8 bit – označava glavnu verziju servisnog sučelja

Message Type (Tip poruke) 8 bit – označava kojeg je tipa poruka. Služi da bi se razlikovale poruke,odnosno koju funkciju ima poslana poruka. Specificirani su slijedeći tipovi:

0x00 REQUEST – Poslani zahtjev očekuje odgovor (makar prazan)

0x01 REQUEST_NO_RETURN – „*Fire&Forget*“ zahtjev

0x02 NOTIFICATION – Zahtjev za obavijest (engl. *Notification*) gdje se ne očekuje odgovor

0x80 RESPONSE – Poruka odgovora

0x81 ERROR – Odgovor koji sadrži grešku

0x20 TP_REQUEST – TP zahtjev koji očekuje odgovor (makar prazan)

0x21 TP_REQUEST_NO_RETURN – TP „*Fire&Forget*“ zahtjev

0x22 TP_NOTIFICATION – TP zahtjev koji ne očekuje odgovor

0x23 TP_RESPONSE – TP odgovor

0x24 TP_ERROR – TP odgovor koji sadrži grešku

Jedan od klasičnih zahtjeva poput poruke tipa 0x00 bi trebao dobiti odgovor 0x80 bez greške. Ako ipak dođe do greške stići će poruka tipa 0x81. TP označava da se radi o velikim porukama u UDP paketima, tj. koristi se SOME/IP-TP (*Transport Protocol*) i takve poruke u *Message* ID polju sadrže TP *Flag*.

Return Code (Kôd oznaka povratka) 8 bit – koristi se za obavijest je li zahtjev uspješno obrađen. Svakom od prethodno nabrojanih tipova poruka pripada nekakva oznaka koja obavještava je li ta poruka uspješno obrađena. Sukladno tome, svaki tip bi trebao imati oznaku da je poruka dobro prošla osim poruke koja sadrži grešku (0x81). Uz nju se šalje ovaj povratni kôd koji ukazuje na moguću grešku koja se dogodila. Sa gledišta SOME/IP-a, poruka *response* sa return kôdom koji nije 0x00, odnosno došlo je do greške, i dalje ima tip poruke 0x80 što znači nekakav odgovor, ne pogrešku i

tako se i tretira. Poruka pogreške ne može biti poslana za događaje, obavijesti i „*Fire&Forget*“ metodu. Za *Request/Response* metodu poruka pogreške će se prepisati preko cijelog SOME/IP zaglavlja, samo će *payload* ostaviti onakav kakav je, a u tip poruke i povratni kôd će upisati odgovarajuću vrijednost. U nastavku je popis tih povratnih kodova koji idu uz određeni tip poruke te popis greški koje poruka ERROR i RESPONSE može sadržavati:

REQUEST 0x00 (E_OK)
REQUEST_NO_RETURN 0x00 (E_OK)
NOTIFICATION 0x00 (E_OK)
RESPONSE - Može imati isti odgovor kao i ERROR poruka, odnosno one su jedine koje se vraćaju kao odgovor od poslužitelja i mogu imati *Return Code* postavljen u kôd greške. Ako poruka pogreške sadrži dulji opis, prenosi se u *payloadu* ERROR poruke i šalje umjesto RESPONSE odgovora.
ERROR

- Ne smije biti 0x00 (E_OK)
- E_NOT_OK Nepoznata greška 0x01
- E_UNKNOWN_SERVICE Nepoznat servis ID 0x02
- E_UNKNOWN_METHOD Nepoznat ID metode 0x03
- E_NOT_READY ID servisa i metode su poznati ali aplikacija nije pokrenuta 0x04
- E_NOT_REACHABLE Sustav na kojemu je servis se ne može dohvatiti 0x05
- E_TIMEOUT Isteklo vrijeme (čekanja odgovora, obrade podataka ili sl.) 0x06
- E_WRONG_PROTOCOL_VERSION Verzija protokola nije podržana 0x07
- E_WRONG_INTERFACE_VERSION Ne podudara se verzija sučelja 0x08
- E_MALFORMED_MESSAGE Greška prilikom deserijalizacije *payloada* 0x09
- E_WRONG_MESSAGE_TYPE Neočekivani tip poruke (primjerice, REQUEST_NO_RETURN za REQUEST poruku) 0x0A
- RESERVED Rezervirano za generičke SOME/IP greške. Te su greškespecificirane u budućim verzijama 0x0B – 0x1F
- RESERVED Rezervirano za pogreške metoda i servisa. Određene suspecifikacijom sučelja 0x20 – 0x5E

Payload (Korisni teret) Varijabilne duljine – veličina ovog polja ovisi o transportnom protokolu. ZaUDP ta duljina iznosi od 0 do 1400 byte-ova te poruke nisu fragmentirane. U tom paketu može biti više poruka ali jedna poruka ne može biti veća nego što je veličina jednog paketa. Veće poruke se prenose preko TCP transportnog protokola. Ukoliko dođe do pogreške sinkronizacije u TCP-u, specifikacija SOME/IP omogućava čarobne kolačiće da bi se ponovno pronašao početak slijedeće poruke. U polju *payload* se transportiraju zapravo sadržaji poruke, serijalizirane metode ili parametri događaja.

3.2. Vrste poruka i način komunikacije

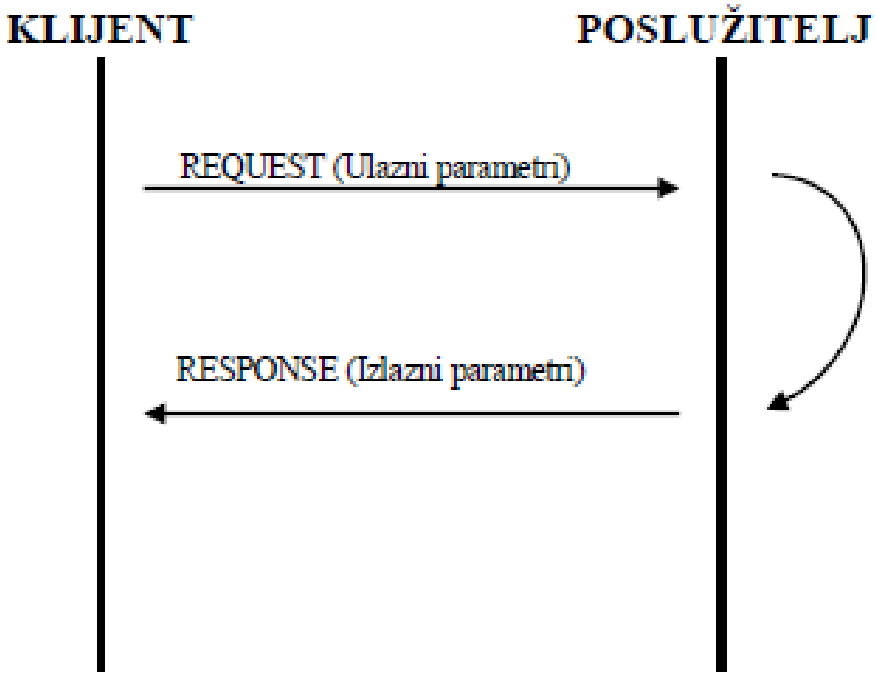
SOME/IP podržava različite komunikacijske principe. Sva komunikacija koja se odvija između klijenta i poslužitelja smatra se uslugom, a SOME/IP nudi slijedeće:

- *Request / Response* metoda je metoda s povratnim odgovorom. To je zahtjev od klijenta do poslužitelja na koji će poslužitelj poslati odgovor ili grešku
- *Fire&Forget* je metoda bez odgovora. Klijent upućuje zahtjev na koji poslužitelj neće odgovoriti
- *Event* je događaj na koji se klijent može pretplatiti i ovom slučaju poslužitelj šalje specifičnu poruku, obavijest tim klijentima koji su se pretplatili na njega, odnosno to je događaj gdje je poruka upućena određenim prethodno zabilježenim klijentima i šalje se ciklički ili na promjenu samog događaja
- *Field* je logički prikaz svojstva sa strane poslužitelja za koji mogu postojati metode upita (engl. *getter*), promjene (engl. *setter*) i obavijesti (engl. *notification*) za nekakvo svojstvo ili status.
- *Eventgroup* je logičko grupiranje polja i događaja

Isti servis se može upotrijebiti više puta i to se naziva instancom usluge. SOME/IP mapira odgovarajuće podatke za sve dijelove usluge na SOME/IP poruke kako bi se one mogle prenijeti transportnim protokolom.

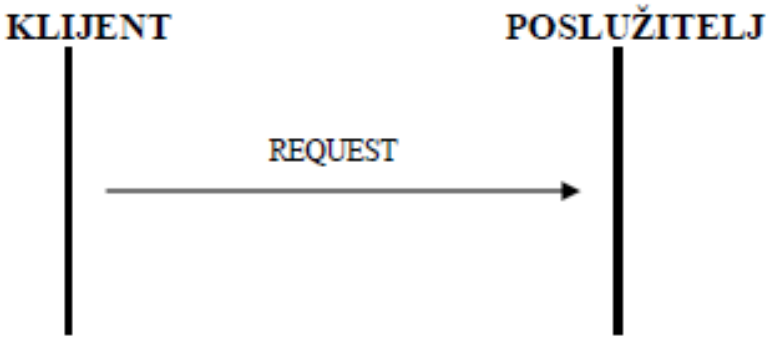
Request / Response (Slika 5.) je od najčešćih načina komunikacije. Klasični postupak gdje klijent šalje poruku zahtjeva na koju poslužitelj odgovara. Za tu poruku klijent mora konstruirati *payload* koji može sadržavati parametre potrebne metodi, postaviti ID poruke ovisan o metodi koju zove, odrediti veličinu u polju *length* te opcionalno postaviti *request ID* u broj koji je unikatan za klijenta.*Message Type* postavlja u REQUEST, *Return Code* u 0x00 te odabire *Protocol* i *Interface Version.* Za konstruiranje *payloada* sav unos ili argumenti metode se serijaliziraju prema redoslijedu argumenata unutar potpisa metode. Poslužitelj konstruira zaglavlje odgovora prema zaglavlju

klijentovog zahtjeva i još na to dodaje svoj *payload*, polje *length* postavlja u 8 *byte*-ova + nova duljina *payloada*. *Message ID* i *Request ID* uzima iz zahtjeva i kopira, tip poruke postavlja u RESPONSE ili ERROR. Poslužitelj neće slati poruku odgovora sa određenim *Request ID* sve dok poruka zahtjeva ne bude pristigla, a isto tako će klijent ignorirati poruku odgovora bez obzira na postavljen *Request ID* ako odgovarajući zahtjev još nije poslan u potpunosti [12]. Klijent ovo obavlja takozvanim udaljenim pozivom (RPC) koji označava početak metode na kontaktiranom poslužitelju.



Slika 5. Request / Response metoda

Fire&Forget (Slika 6.) se može usporediti s prethodnom metodom, samo što u ovom slučaju nema odgovora poslužitelja i klijent ga niti ne očekuje. Postavke zaglavlja i *payloada* su skoro iste, razlika je u tipu poruke koji je ovdje REQUEST_NO_RETURN i ovdje klijent postavlja polje *length* na isti način kako je u prethodnoj metodi poslužitelj postavljao, 8 *byte*-ova + duljina *payloada*. *Fire&Forget* ne bi smio vratiti nikakvu poruku pa ni pogrešku [12].



Slika 6. Fire&Forget metoda

Obavijesti (engl. *notifications*) opisuju opći koncept *Publish / Subscribe*, odnosno pretplata/objava. *Event* sam po sebi nije poruka nego događaj kojim poslužitelj šalje obavijest ciklički ili u određenom slučaju samo ako se klijent na to pretplatio. Isto vrijedi i za polja te imamo *Event Notifications* i *Field Notifications*.

Poslužitelj objavi uslugu na koju se klijent pretplaćuje. U određenim slučajevima poslužitelj će poslati klijentu događaj koji može biti primjerice ažurirana vrijednost ili događaj koji se dogodio. Obavijest se koristi samo za prijenos te vrijednosti, a mehanizme objave i pretplate provodi *Service Discovery* i o tome će biti više u slijedećem poglavlju. Za poruke obavijesti klijent mora postaviti zaglavlje i *payload* poput prethodnih gdje postavlja polje duljine u 8 *byte*-ova + duljina *payloada*, ID poruke ovisi o događaju koji će poslužitelj slati, *Request ID* i *Return Code* u 0x00 te *Protocol* i *Interface Version* [12].

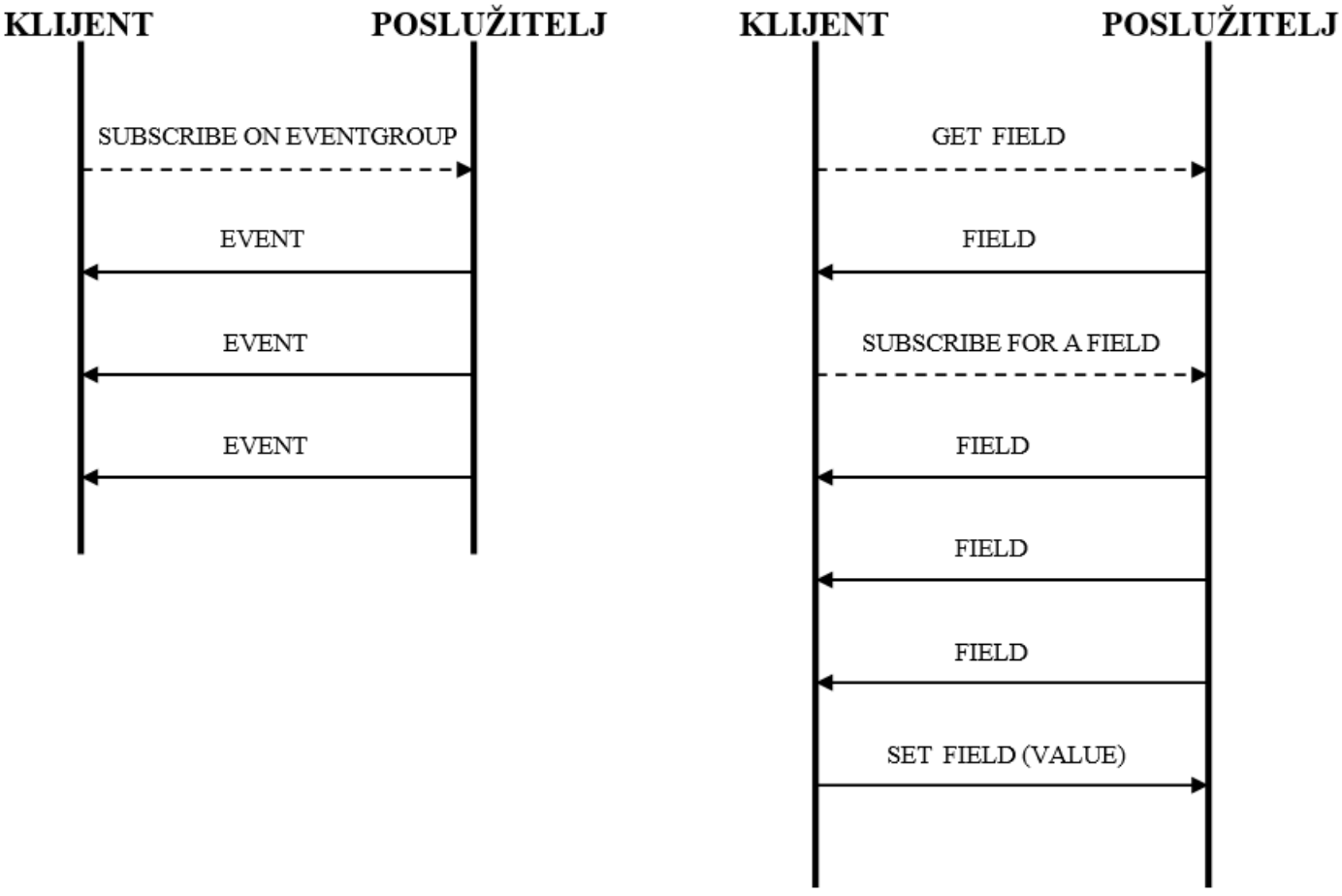
Payload poruke obavijesti sadrže serijalizirane podatke događaja. Kad postoji više pretplaćenih klijenata na istom ECU, sistem upravlja replikacijom obavijesti određenim redoslijedom radi spremanja prijenosa na komunikacijski medij. To je posebno važno kada se poruke prenose *multicast.*

To slanje obavijesti se može odvijati na tri načina. Prvo je cikličko ažuriranje kada se šalje ažurirana vrijednost u nekakvom fiksnom vremenskom intervalu, primjerice svakih 100 ms. Drugi način je slanje vrijednosti na njenu promjenu, odnosno podatak se pošalje samo ako dođe do njegove promjene, npr. kada se otvore vrata na automobilu. Treći način se naziva epsilon promjena. Ažuriranje se šalje samo kada je razlika zadnje vrijednosti veća od određenog epsilona. Taj je koncept adaptivan, predviđa se na temelju povijesti i ažuriranje se provodi samo kada je razlika između te prošle vrijednosti sadašnje veća od postavljenog epsilona [12]. Za primjer može biti porast temperature u kabini vozila.

Polja reprezentiraju status, svojstvo (engl. *property*) i sadrže nekakvu valjanu vrijednost.

Kada se korisnik pretplati na polje automatski dobije njegovu vrijednost kao inicijalni događaj. Polje je kombinacija *gettera, settera* i obavijesti (engl. *notifier*), i ako ih nema onda polje niti ne postoji. *Getter* polja je zapravo *Request / Response* metoda koja ima prazan *payload* u poruci zahtjeva od klijenta, a popunjen s vrijednosti u poruci odgovora. *Setter* polja je isto *Request / Response* metoda samo obrnuta od prethodne gdje klijent prenosi željenu vrijednost u polju *payload*, onu koju želi postaviti u polje i ako to uspije, ta postavljena vrijednost se vraća u poruci odgovora. Kako je prethodno navedeno, obavijest se koristi kod objavljivanja i pretplate gdje se vrijednost polja šalje automatski u slučaju da se klijent na to pretplatio ili na promjenu vrijednosti upolju, događaj, slijedeći njegova pravila [12].

Glavna razlika između obavijesti polja (engl. *Field Notifications*) i obavijesti događaja (engl. *Event Notifications*) je ta da su polja dostupna dokle god je i sustav, a događaj samo onda kada se dogodi [10]. Osim toga, obavijesti događaja više djeluju kao *Fire&Forget* , dok obavijesti polja sadrže podatke koje se odnose na prethodne događaje i zato imaju *get-set* metode. Zajedničko im je što se oboje oslanjaju na događaje, no obavijest o događaju samo pruža podatke bez ikakve veze sa povijesti tog podatka, dok obavijest o polju daje mogućnost klijentu za čitanje i pisanje čak i bez pretplate [16]. Na slici 7. je prikazan način komunikacije s poljima i događajima.



Slika 7. Komunikacija poljima i događajima

Instance istog servisa se prepoznaju po *Instance* ID i podržane su jednako bez obzira nalaze li se na jednom ili različitim ECU. Različiti servisi mogu dijeliti zajednički *port* protokola na transportnom sloju ali više instanci istog servisa na istom ECU moraju slušati različite *port*-ove. Kada se *Instance* ID koristi za *Service Discovery* ne nalazi se u zaglavlju. Servisna instanca se može prepoznati kombinacijom njenog ID-a sa IP adresom, transportnim protokolom ili *port*-om.

3.3. Odabir transportnog protokola

SOME/IP podržava *User Datagram Protocol (UDP)* i *Transmission Control Protocol (TCP).* Dok je UDP prilično siromašan transportni protokol i podržava samo najvažnije značajke, TCP nudi mogućnost postizanja pouzdane komunikacije. Ne samo da obrađuje bitne pogreške negoi segmentaciju, gubitke, dupliciranja, upravlja redoslijedima i zagušenjem mreže. Unutar automobila mnoge aplikacije zahtijevaju vrlo kratko vrijeme reakcije. Tome pak bolje odgovara UDP jer i sama aplikacija može podnijeti malo vjerojatan slučaj pogreške. Primjerice, u slučajevima kada se koriste ciklički podaci često je najbolje pričekati slijedeći podatak nego pokušavati popraviti trenutni. Njegov najveći nedostatak je što ne radi segmentaciju pa je u mogućnosti prenositi samo manje pakete podataka. TCP se koristi za prijenos vrlo velikih podataka (>1400 byte), a u slučaju pogreške nema zahtjeva za latencijom, odnosno nema postavljenu granicu kašnjenja od primjerice nekoliko milisekundi, dok UDP ima zahtjev od <100 ms. Ako želimo prenositi velike podatke koristeći UDP te trebamo čvrste zahtjeve za latenciju, onda ga koristimo zajedno sa SOME/IP-TP. Mogu se pokušati koristiti nekakvi vanjski transporti ili mehanizmi prijenosa (*Network File System, APIX link, 1722…*) ako su prikladniji u tom slučaju.

Transportni protokol je određen specifikacijom sučelja za svaku poruku. Metode, događajii polja obično koriste samo jedan protokol [12]. Ako poslužitelj pokreće različite instance istog servisa, poruke koje pripadaju određenom servisu su dodijeljene portu transportnog protokola na strani poslužitelja. Sve veze transportnog protokola moraju podržavati više od jedne SOME/IP poruke (UDP paket ili TCP segment). SOME/IP na strani primatelja mora biti u stanju zaprimati ne posložene, ne poravnate poruke. Kod transporta više poruka poravnanje može biti zajamčeno samo ako je duljina svakog *payload*-a višekratnik od veličine poravnanja (npr. 32 bita).

SOME/IP ne ograničava upotrebu fragmentacije UDP-a, a format zaglavlja omogućava transport više poruka u jednom UDP paketu. Kraj poruke se onda ne identificira pomoću *length* polja nego se pomoću njega određuje postoje li dodatne poruke u UDP paketu. Svaki *payload* ima svoje zaglavlje [12].

TCP veza se dosta temelji na UDP ali glavna razlika je što može prenositi mnogo veće poruke i koristi robusne značajke TCP-a. Kako bi se smanjilo kašnjenje i vrijeme reakcije potrebno

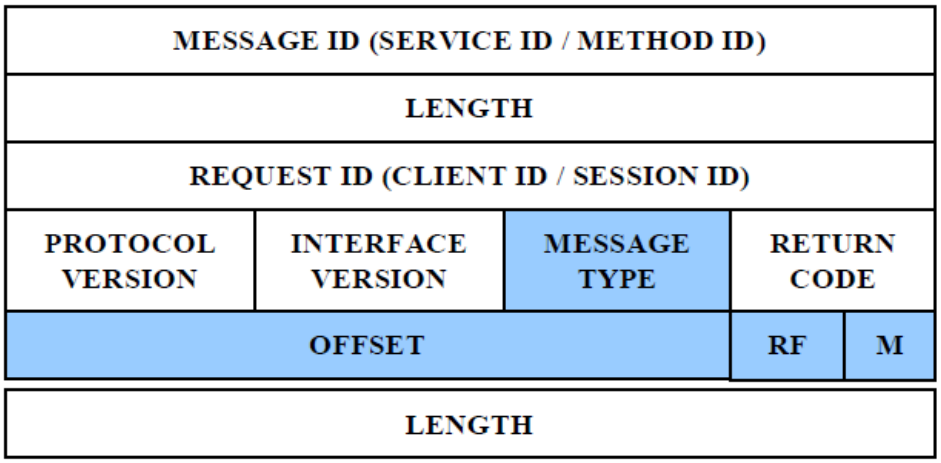
je isključiti Nagleov[11] algoritam. Kada se izgubi TCP veza, neriješeni zahtjevi se tretiraju kao zakašnjeli. Budući da TCP upravlja pouzdanošću, dodatna sredstva za osiguravanje pouzdanosti nisu potrebna. Klijent i poslužitelj trebaju koristiti jednu TCP vezu između sebe za sve metode, događaje i obavijesti, odnosno koristi se jedna TCP veza po servisu. Vezu otvara i započinje klijent kada poziva metodu ili pokušava primiti obavijest (poput telefonskog poziva), a odgovoran je i za ponovno uspostavljanje kada god ona ne uspije. Također, klijent zatvara vezu kada mu više nije potrebna. Obavezan je i zatvoriti vezu kada niti jedna usluga od onih koji koristi više nije dostupna,istekla je ili obustavljena. Poslužitelj neće zatvoriti TCP vezu nakon zaustavljanja servisa jer ostavlja klijentu vremena za obradu kontrolnih podataka te da on nakon toga zatvori sam vezu. U slučaju kada bi poslužitelj zaustavio vezu prije nego je klijent prepoznao da mu ona više nije potrebna on bi pokušao ponovno uspostaviti tu vezu. Zbog svega navedenog TCP je sporiji i veći ali nudi mehanizme provjere greški i ponovno slanje izgubljenih paketa što UDP nema ali je zato jednostavniji, brži i učinkovitiji.

Kako bi alati za testiranje mogli raspoznati granice poruka koje se prenose TCP vezom, u SOME/IP poruku se unose dodatne poruke „čarobnih kolačića“ (engl. *Magic Cookie Message*). Oni su korisni i ako dođe do pogreške sinkronizacije jer se mogu pronaći krajevi poruke.

3.4. SOME/IP-TP

UDP može prenositi poruke koje stanu samo u jedan IP paket. Ako se šalju veće poruke koristi se SOME/IP-TP (*Transport Protocol*). Jedna takva velika poruka se naziva „original“, a njeni dijelovi segmenti. TCP se koristi samo u slučaju jako velikih poruka, većih od 1400 bytes i ako nema zahtjeva za kašnjenjem. Poruke koje koriste TP moraju aktivirati *Session Handling, Session* ID mora biti unikatan za svaku originalnu poruku. Svi TP segmenti moraju sadržavati taj isti ID kako bi se znalo kojem originalu pripadaju te osim njega u *Message Type* imaju posebnu zastavicu postavljenu u 1 [12]. TP zaglavlje se razlikuje od običnog zaglavlja SOME/IP poruke u dijelu iznad *payloada*, odnosno nakon klasičnog SOME/IP zaglavlja gdje je umetnuto polje Offset od 28 bita, tri rezervirane zastavice (engl. *Reserved Flag*) tj. 3 bita i jedna zastavica (engl. *More Segment Flag)* koja označava je li taj segment posljednji u nizu (Slika 8.).

[11] Nagleov algoritam se koristi za smanjenje broja paketa u mreži, odnosno smanjenje zagušenja i poboljšanje učinkovitosti. Funkcionira na način da skuplja i kombinira male poruke i šalje ih zajedno umjesto jednu po jednu. Izvor: https://en.wikipedia.org/wiki/Nagle%27s_algorithm



Slika 8. Zaglavlje SOME/IP-TP poruke

Polje *Offset* prenosi prvih 28 bita tipa uint32, a zadnja 4 bita se uvijek interpretiraju kao 0. to znači da to polje može samo prenijeti vrijednosti koje su višekratnici 16 *byte*-ova. U polje *Offset* se postavlja pomak u *byte*-ovima od prenesenog segmenta u izvornoj, originalnoj poruci. Rezervirane zastavice pošiljatelj će postaviti u 0, a primatelj će ih ignorirati i neće ih niti provjeravati. Zastavica M se postavlja u 1 za sve segmente osim zadnjeg, on je 0. Duljina segmenta mora odgovarati poravnanju slijedećeg segmenta uzimajući u obzir *Offset* polje. Kako su UDP poruke ograničene na 1400 *byte*-a maksimalna duljina segmenta koji je pravilno poravnat je 1392 *byte*-a + 8 *byte*-a od zaglavlja. Zbog zaglavlja TP poruke koja ima dodatnih 32 bita oni se dodaju prethodnom izračunu pa jedna TP poruka ima 1404 *byte*-ova (Slika 9.). Ostatak zaglavlja koji se ne odnosi na TP poruku se popunjava kao sve ostale SOME/IP poruke [12].

Za primjer uzmimo da poruka koju šaljemo ima 5888 *byte*-ova kojih želimo prenijeti u *payloadu*. Polje *length* sadrži duljinu od 5896 *byte*-ova jer se ubraja i veličina zaglavlja. Ako veličinu poruke koju šaljemo podijelimo sa veličinom koja stane u jedan *payload*, dođemo do izračuna da nam treba pet segmenata da bi prenijeli poruku. Kako u sami *payload* zapravo stane 1392 *byte* kad oduzmemo zaglavlje, znači da nam trebaju četiri puna segmenta i jedan sa 320 byte. U slijedećoj tablici je primjer:

SEGMENT	LENGTH (bytes)	MESSAGE TYPE (TP-FLAG)	OFFSET VALUE	MORE SEGMENT FLAG
1.	8 + 4 + 1392 = 1404	'1'	0	1
2.	8 + 4 + 1392 = 1404	'1'	87	1
3.	8 + 4 + 1392 = 1404	'1'	174	1
4.	8 + 4 + 1392 = 1404	'1'	261	1
5.	8 + 4 + 320 = 332	'1'	348	0

Tablica 2. Primjer sadržaja SOME/IP-TP zaglavlja po segmentima

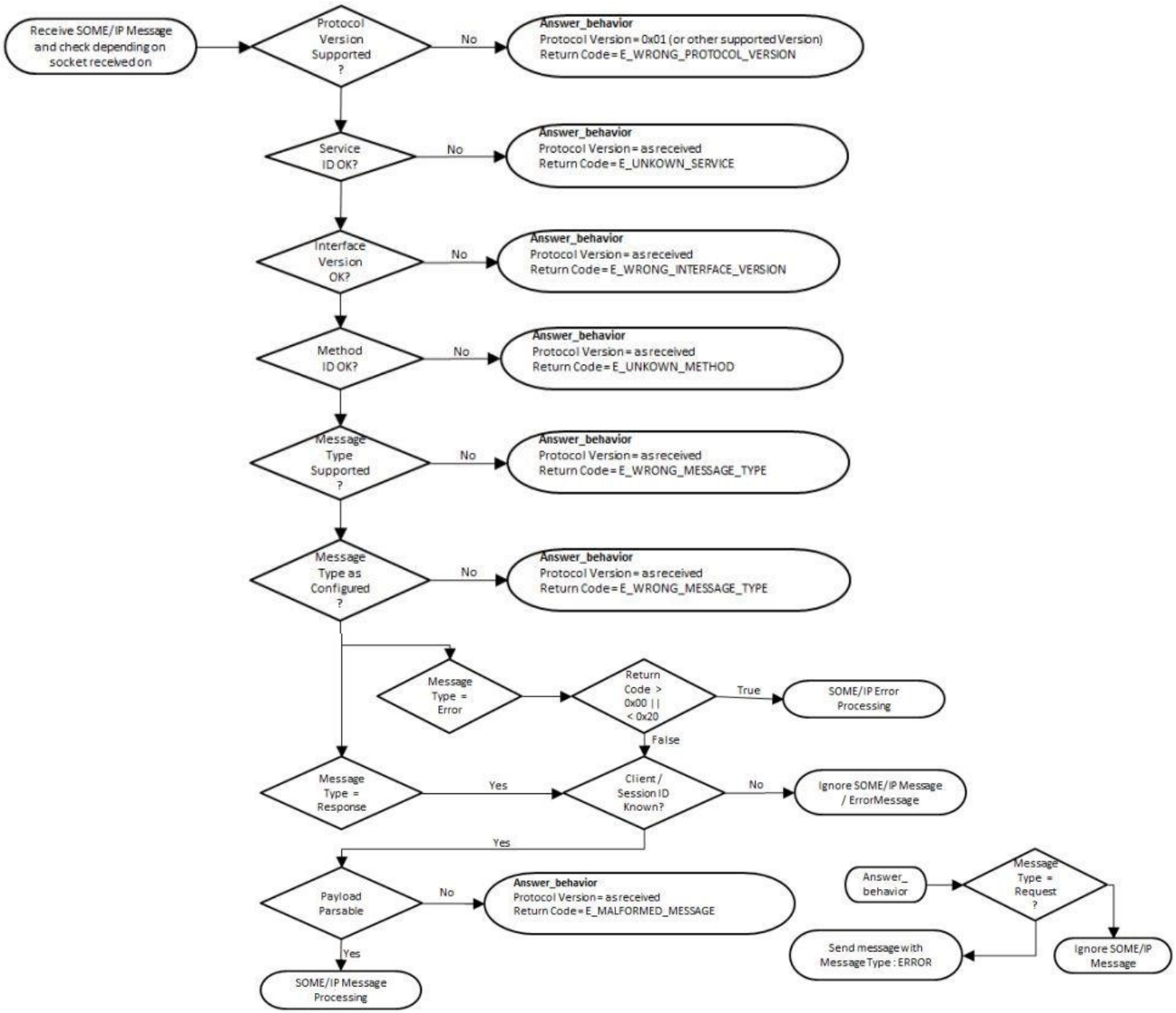
Offset polje je u jedinici od 16 byte-ova kako smo prethodno spomenuli pa ako veličinu od 1392 byte podijelimo sa 16 dobiti ćemo vrijednost 87 koja predstavlja zauzeće tog segmenta. Krenuvši od početka, prvi segment nema nikakvog odmaka od samog sebe. Drugi je od njega odmaknut za 87 jer toliko i zauzima sami segment pa počinje tek na tom mjestu, a treći onda već duplo odmaknut pa dobijemo vrijednost Offset-a od 174.

Slanje i primanje ovakvih, ali i ostalih SOME/IP poruka sa strane pošiljatelja i primatelja ima slijedeće specifikacije.

Pošiljatelj segmentira samo poruke koje su konfigurirane za segmentiranje te ih šalje uzlaznim redoslijedom. Segmentira ih na način da svi segmenti sa M zastavicom postavljenom u 1 imaju istu veličinu. Pokušati će iskoristiti maksimalnu veličinu polja dozvoljenu specifikacijom i neće slati duple ili poruke koje se preklapaju. Primatelj će uvijek provjeravati vrijednosti zaglavlja i ta polja se moraju podudarati inače će poruka biti zanemarena. Podržava sastavljanje više poruka s istim ID poruke ali s različitim pošiljateljima (*Client ID, Sender IP, Sender Port*) i pritom se koristi sa *Session* ID da bi otkrio slijedeću poruku za sastavljanje. To se određuje konfiguracijom i ovisno je o veličini međuspremnika te se svakako obavlja prije deserijalizacije. Novo spajanje poruka započinje s primitkom poruke koja ima drugačiji *Session* ID, a stari segmenti koji nisu uspješno sastavljeni mogu biti odbačeni. Trebao bi sastavljati poruke po veličini svog spremnika (engl. *buffer*), a višak preskočiti. Samo ispravno sastavljene poruke će se proslijediti u aplikaciju. Implementacijom se mora osigurati da svi primljeni *byte*-ovi budu ispravni i dobro sastavljeni. Brojne preklapanja, ne dupliciranih *byte*-ova i usporedba s duljinom može biti valjana provjera. Također, segmenti se mogu sastavljati bez obzira jesu li primljeni uzlaznim ili silaznim redoslijedom. Ako se segmenti sastavljaju u veliku ne segmentiranu poruku, *Message Type* se prilagođava, a TP zastavica resetira u 0. Primatelj bi trebao limitirati resurse koji podržavaju sastavljanje pogrešno raspodijeljenih segmenata u redoslijedu, primjerice da su najmanje tri udaljena mjesta u redoslijedu od ispravnog dopušteni. Ako otkrije da segmenti nedostaju ili se prekoračilo vrijeme čekanja, ako je slijedeća poruka već pristigla ili redoslijed pobrkanih segmenata veći od dozvoljenog, primatelj će otkazati desegmentaciju originalne poruke. Uređivanje segmenata različitih poruka (različit *Message ID*) je moguće u različitim spremnicima ali ispreplitanje različitih poruka ili različitih ID sesije u istom spremniku nije dozvoljeno. Primatelj mora moći ispravno sastaviti preklapajuće ili duplicirane segmente, a ako su oni različiti od onih već zapisanih u spremniku može se otkazati sastavljanje. Ta se opcija može isključiti konfiguracijom. Primatelj mora biti u stanju otkriti i obraditi očigledne pogreške što znači da pogreške u spremniku moraju biti spriječene još tijekom zaprimanja poruke [12].

3.5. Rukovanje pogreškama

Pogreške se mogu ispravljati u samoj aplikaciji ili komunikacijskom sloju i iz tog razloga SOME/IP podržava dva mehanizma. Koristi povratne kodove u zaglavlju (engl. *Return Codes*) poruka odgovora na metode ili dopušta korištenje izričitih poruka o grešci. O konfiguraciji ovisi koji način će se koristiti. Povratni kodovi se koriste za prijenos informacije o pogrešci u aplikaciju itreba uzeti u obzir da kada se koriste u metodi *Request/Response* tretiraju se kao odgovor, ne kao pogreška, barem sa strane SOME/IP jer u zaglavlju za tip poruke i dalje stoji 0x80 što znači da je ona odgovor koji prenosi u ovom slučaju opis greške. Ako je potrebno prenijeti detaljnije informacije o pogrešci koristi se takozvana izričita ili eksplicitna poruka o grešci (engl. *Error Message*) koja je tip poruke 0x81 i njen *payload* se popunjava sa detaljima pogreške (engl. *exception string*) te se šalje umjesto poruke odgovora. To se može koristiti za obradu svih ostalih pogrešaka koje se mogu pojaviti na poslužitelju ali i komunikacijskom mediju i posredničkim komponentama (primjerice *switch*). Bez obzira, sve poruke imaju polje povratnog koda u svom zaglavlju ali samo poruka pogreške (0x81) i poruka odgovora (0x80) mogu imati povratni kod koji predstavlja grešku, odnosno da je različit od 0x00. U slučaju *Request/Response* metode, poruka pogreške se kopira preko zaglavlja (*Message ID, Request ID, Interface ID*) ali ne i preko payloada,a *Message Type* i *Return Code* se postavlja na odgovarajuće vrijednosti. AUTOSAR preporučuje daizričite poruke za pogrešku imaju varijabilnu duljinu polja za opis iznimke koja je nastala te da se kreira unija posebnih izuzetaka bez polja. Unija pruža fleksibilnost za dodavanje novih iznimki u budućnosti, a ljudski čitljiv opis pogreške koji se prenosi olakšava testiranje i uklanjanje tih greški. Rukovanje pogreškama ovisi o vrsti primljene poruke. Za poruke primljene preko UDP-a provjerava se veličina njegovog datagrama koja mora biti najmanje 16 *Byte*-ova jer je to minimalna veličina SOME/IP poruke. Polje duljine mora biti manje ili jednako preostalim *byte*-ovima u *payload*-u. Ako jedna od ove dvije pogreške ne uspije javiti će se greška [12]. Svaka SOME/IP poruka će proći osnovnu provjeru polja kako je prikazano na Slici 9.



Slika 9. Provjera polja zaglavlja i rukovanje pogreškama
(Izvor: AUTOSAR, SOME/IP Protocol Specification, str. 45)

To ipak ne pokriva pravo rukovanje pogreškama samo obavijest porukom. Ako je neka od pogreški pronađena tijekom zaprimanja poruke putem TCP-a primatelj će provjeriti TCP vezu i resetirati ju ako je to potrebno. Provjera TCP veze može uključivati primaju li se svi podaci, primjerice ostale grupe događaja, slanjem čarobnih kolačića i čekanje potvrde (TCP ACK) ili ponovno uspostavljanje veze.

Pri prijenosu RPC postoje različite semantike pouzdanosti:

- „Maybe“ – poruka je možda došla do komunikacijskog partnera

- „At least once“ – poruka je barem jednom stigla do komunikacijskog partnera
- „Exactly once“ – poruka je stigla točno jednom do komunikacijskog partnera

Kad se ovi izrazi koriste u *Request/Response* zahtjevu, onda se odnose na oboje. Iako različite implementacije mogu implementirati različite pristupe, SOME/IP trenutno postiže „*Maybe*“ kad se koristi UDP veza, a „*Exactly once*“ kad se koristi TCP. Daljnje postupanje s pogreškama je prepušteno aplikaciji. Za pouzdanost „*Maybe*“ dovoljan je jedan vremenski prekid u korištenju *Request/Response* metode u kombinaciji s UDP transportnim protokolom i SOME/IP će se javiti signalom E_TIMEOUT klijentu. Za „*Exactly once*“ pouzdanost najčešće se koristi TCP jer jamči pouzdaniju komunikaciju [12].

4. SOME/IP – SD

Dva najvažnija dijela SOME/IP protokola su *Service Discovery* (skr. SD) i *Publish / Subscribe*. SD omogućava svakom ECU da dinamički pronalazi različite funkcionalnosti među ostalim ECU i konfigurira pristup, odnosno pretplaćuje se. Na taj način novi čvorovi koji ulaze u mrežu mogu prisustvovati komunikaciji bez da unaprijed znaju konfiguraciju i koji čvorovi nude koje usluge. Čvorovi mogu koristiti RPC i na taj način pozivati funkcionalnosti i metode.

SOME/IP-SD se koristi za lociranje servisne instance, detekciju je li servisna instanca pokrenuta i implementira rukovanje *Publish/Subscribe* metoda. Zapravo, služi za pronalazak i ponudu dostupnih servisa u automobilu koristeći IP *multicast* i SOME/IP-SD poruke. Podržava samo IP komunikaciju ali za razliku od samog SOME/IP koristi jedino UDP i ovisan je o drugim slojevima protokola. TCP podržava samo *unicast* način slanja, pa slanje *multicast*-om mora ići preko UDP-a. Slijedi pojašnjenje pojedinih akronima koji će se spominjati dalje u radu, a možda nisu u nekom od prethodnih poglavlja opisani [13,14]:

- Parametri – izlazni, ulazni ili ulazno/izlazni argumenti metode ili događaja
- *Remote Procedure Call* (RPC) – poziv metode s jednog ECU prema drugom koji se prenosi koristeći poruke
- Obavijest događaja (engl. *Notification Event*) – poruka obavijesti o polju. Takva poruka se ne razlikuje od poruke događaja.
- Obavijest (engl. *Notifier*) – poruka s novom vrijednosti prilikom promjene polja
- Servis (engl. *Service*) – logička kombinacija metoda, događaja i polja s time da servis može biti i prazan bez da sadržava ijedno navedeno
- Grupa događaja (engl. *Eventgroup*) – logičko grupiranje događaja i njihovih obavijesti unutar servisa omogućavajući njihovo korištenje pretplatama na njih
- Sučelje servisa (engl. *Service Interface*) – formalna specifikacija servisa zajedno s metodama, događajima i poljima
- Servisna instanca (engl. *Service Instance*) – implementacija sučelja servisa sastrane softvera. Može ih biti više u vozilu i više na jednom ECU.
- Poslužitelj (engl. *Server*) – ECU koji nudi instancu neke usluge se naziva poslužiteljem
- Klijent (engl. *Client*) – ECU koji potražuje i koristi ponuđenu uslugu nazivamo klijentom
- Ulaz uslužne instance poslužitelja (engl. *Server-Service-Instance-Entry*) –

konfiguracija i potrebni podaci instance koju nudi lokalni ECU

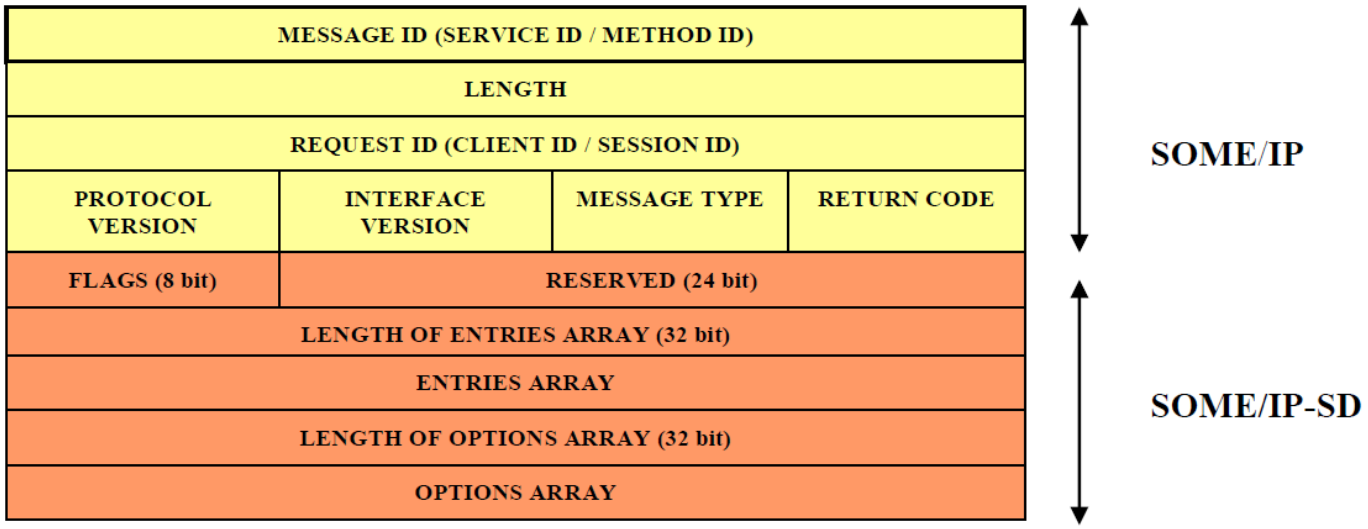
- Ulaz uslužne instance klijenta (engl. *Client-Service-Instance-Entry*) – konfiguracija i potrebni podaci uslužne instance koju nudi drugi ECU
- Objavljivanje grupe događaja (engl. *Publishing an Eventgroup*) – koristiti SOME/IP-SD za ponudu grupe događaja drugim ECU-ima
- Pretplata na grupu događaja (engl. *Subscribing an Eventgroup*) – zahtijevanjegrupe događaja koristeći SOME/IP-SD poruku od ECU koji ju nudi

Stanje u kojemu je servisna instanca i informacije poput IP adrese, *port*-a i protokola spadaju u drugo. Glavni zadatak *Service Discovery* modula je upravljanje dostupnošću funkcionalnih entiteta koje zovemo servisi u komunikaciji unutar automobila te kontrola slanja poruka događaja. Na taj način se poruke događaja šalju samo primateljima koji to zahtijevaju. Uz *Service Discovery* različiti ECU mogu nuditi različite servisne instance unutar mreže vozila te prestati nuditi one koje su do sada nudili. Kasniji pronalazak takve instance će biti bez odgovora. Servisne instance su pojedinačne implementacije servisa koje su definirane njenim sučeljem. Pod „pronaći“ uslugu se podrazumijeva operacija identificiranja dostupne servisne instance i njene lokacije. Osim toga, *Service Discovery* ima mogućnost kontrolirati slanje posebnih poruka koje zovemo događaji. Onisu grupirani u grupe događaja koje se mogu uključivati ili isključivati u objavi/pretplati, odnosno *Service Discovery* odlučuje o slanju i primanju događaja iz skupine događaja.

ECU mora biti jedan od dvije različite vrste servisa. Može biti poslužitelj (engl. *Server Services*) gdje nudi lokalno smještene instance servisa prema ostatku vozila i tada se smatra poslužiteljem za tu servisnu instancu. Druga opcija je da bude klijent (engl. *Client Services*) gdje onda koristi te servisne instance poslužitelja ponuđene od nekog drugog ECU unutar vozila. U tom slučaju se smatra klijentom te servisne instance. *Service Discovery* modul na strani poslužitelja ima ulogu nuditi lokalne servise ako su dostupni, tj. nudi servis koji je spreman i dostupan trenutno na ECU. Na isti način povlači ponudu nazad ako servis više nije dostupan. Na strani klijenta *Service Discovery* ima drugačiju ulogu. Osluškuje i pronalazi ponude ovisno o konfiguraciji spremljenoj u kratkoj, nestabilnoj (*volatile*) memoriji. Osluškuje i ponude koje se prestaju nuditi (*Stop Offer*) i šalje poruke za pronalazak ovisno o stanju ECU-a. U slučaju *Publish/Subscribe* odnosa, *Service Discovery* jednog ECU-a (klijent sa *ConsumedEventgroup*) je zainteresiran za dohvaćanje nekih podataka s drugog ECU-a (poslužitelj sa *EventHandler*). Pretplata je definirana eksplicitno u SD (*Service Discovery*) poruci. Na ponuđenu servisnu instancu klijent se može pretplatiti putem *SubscribeEventgroup*, a poslužitelj će koristiti tu pretplatu za registraciju klijenta kao zainteresiranu stranu za određene informacije specificirane pretplatom i započeti će slati te informacije sve do nekog

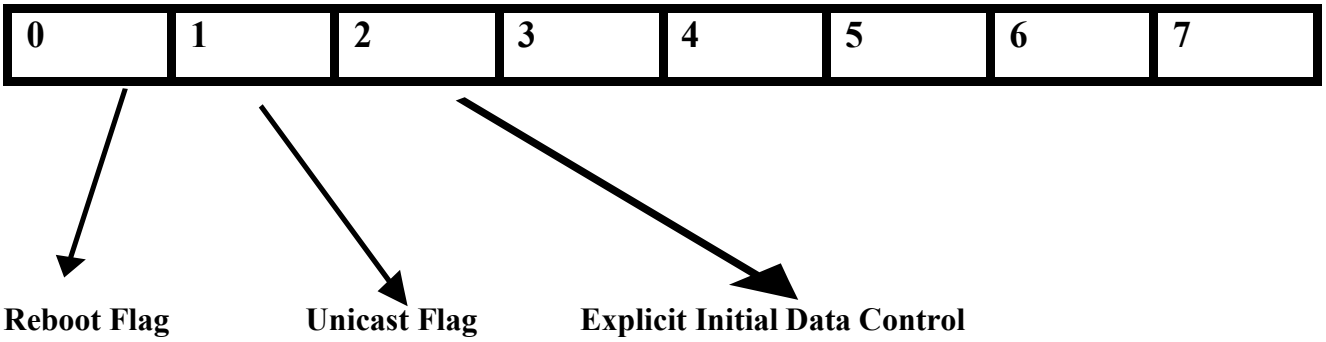
određenog događaja ili isteka vremena. Podržava slanje poruka događaja prema više klijenata koristeći *multicast* poruke umjesto *unicast* pojedinačnih prema svakom klijentu zasebno [13,14].

Format SD poruke je prikazan na slici 10. Zaglavlje započinje SOME/IP zaglavljem i tu se početna polja ponavljaju. U SD porukama ID servisa se postavlja u 0xFFFF, a ID metode u 0x8100. U polje duljine se upisuje duljina koja započinje prvim bajtom nakon njega te sve do zadnjeg bajta SOME/IP-SD poruke. *Request* ID se sastoji od ID klijenta i ID sesije kao i u klasičnojSOME/IP poruci ali se ID klijenta postavlja u 0x0000 budući da postoji samo jedna instanca *Service Discovery*-ja. Ako je prefiks tog ID konfiguriran onda se i on umeće. ID klijenta se ne koristi za SD uslugu, ali ID sesije se koristi za otkrivanje resetiranja neke druge SD instance u vozilu kako bi se ispravo lokalno stanje SD modula. Nakon njegove inicijalizacije, ID sesije porukeposlane od ECU-a je 0x0001 i povećava se nakon svakog poziva određene funkcije. Pohranjuje se odvojeno za *unicast* i *multicast* što znači da prva poruka koja se pošalje na *multicast* adresu ima sesijski ID 0x0001 isto kao i prva *unicast* poruka koja se pošalje. Kada ID sesije dođe do svoje maksimalne vrijednosti od 0xFFFF kreće ispočetka, odnosno slijedeći inkrement će biti opet 0x0001, ali nikada 0. Polje za verziju sučelja i protokola je statički postavljeno na 0x01, a tip poruke je uvijek isti jer se koriste samo poruke događaja pa se to polje postavlja u 0x02. Slično tome, polje povratnog koda je uvijek 0x00 jer se ne primjenjuje za SD poruke [13,14].



Slika 10. Prikaz zaglavlja SOME/IP-SD poruke

S poljem zastavice (engl. *Flags*) započinje SD zaglavlje. Koristi se kao javna i globalna informacija o *Service Discovery*-u što uključuje trenutno stanje od zadnjeg pokretanja te je li u mogućnosti primati *unicast* poruke [13,14]. Detaljniji prikaz polja je na slijedećoj slici (Slika 11.):



Slika 11. Detaljan prikaz polja zastavice iz SD zaglavlja poruke

Zastavica ponovnog pokretanja (engl. *Reboot Flag*) se postavlja u „1“ za sve poruke nakon ponovnog pokretanja sve dok ID sesije ne prođe cijeli krug u svom ciklusu, kad krene ispočetka tada se postavlja u „0“. *Service Discovery* će pratiti posljednju komunikaciju i vrijednost partnerovog *Session* ID-a neovisno radi li se o *unicast* ili *multicast* ali primljena vrijednost preko *multicast* neće ažurirati *unicast* i obratno, odnosno informacije za *Reboot Flag* i *Session* ID se čuvaju odvojeno. Svaki odnos pošiljatelj-primatelj čuva te informacije odvojeno za sebe (tj. za adresu izvora i adresu odredišta). Postoje odvojeni brojači za slanje i primanje poruka, jedan za multicast slanje, i odvojeni za svako *unicast* slanje. Kod zaprimanja poruka postoji odvojeni brojač za svaku *multicast* i za svaku *unicast* primljenu poruku. Detekcija ponovnog pokretanja se obavlja na klasični način usporedbe starog i novog stanja zastavice slanjem uzastopnih SD poruka na slijedeći način [13,14]:

- Zastavica se mijenja iz „0“ u „1“
- ID sesije se ne povećava, a zastavica pokretanja je u 1

U slučaju ponovnog pokretanja poslužitelja čiji servis koristi neki od klijenata, on će se ponašati kao da je zaprimio prekid pretplate (*StopSubscribeEventgroup*), a klijent na način kao da je zaprimio prekid ponude usluge (*StopOffer*).

Unicast zastavica je drugi najveći bit po redu i ukoliko je postavljena u „1“ označava da taj ECU podržava primanje *unicast* poruka. Treći bit ima naziv *Explicit Initial Data Control Flag* i označava podržava li taj ECU eksplicitnu početnu kontrolu podataka. ECU prepoznaje i poštuje tu zastavicu unutar grupe događaja. Mogućnost odabira potječe iz starijih verzija i tu je zbog kompatibilnosti no u novijim verzijama ova zastavica je uvijek postavljena u „1“ [13,14].

Reserved polje se trenutno ne koristi i ostavlja se prazno, odnosno svi bitovi su postavljeni u „0“ i koristiti će se u nekim budućim potrebama.

Kad SD nudi ili traži servisnu instancu ili upravlja pretplatama, čini to kroz polje *Entry array* i u nastavku teksta ih nazivamo ulazima. To polje ima svoju podjelu ovisno je li tip 1 ili tip 2,odnosno

je li ulaz za servise ili grupe događaja. SD podržava višestruke ulaze koji se kombiniraju ujednu SD poruku. Ispred tog polja je polje koje sadrži duljinu polja ulaza. Upravljačke jedinice će distribuirati dostupne i potrebne servisne instance i grupe događaja razmjenjujući ulaze u SD porukama [13,14].

Minimalne i osnovne značajke propisane specifikacijom AUTOSAR-a moraju biti uvedene kod implementacije. Ako nisu uključene i primjenjive smatra se da implementacije ne udovoljava osnovnim SOME/IP-SD značajkama. Ulazi koji moraju biti omogućeni su *Find Service*, *Offer Service*, *Stop Offer Service*, *Subscribe Eventgroup*, *Subscribe Eventgroup Ack* i *Subscribe Eventgroup Nack*. Ako je zahtijevan IPv4 ili IPv6 moraju biti uključene *IPv4/IPv6 Endpoint Option*, *IPv4/IPv6 Multicast Option*, *Configuration Option* i *IPv4/IPv6 SD Endpoint Option* [13, str. 69].

4.1. Ulaz za servise (Tip 1)

TYPE (8 bit)	INDEKS 1ST. OPTIONS (8 bit)	INDEKS 2ND OPTIONS (8 bit)	# OF OPT 1	# OF OPT 2
SERVICE ID (16 bit)		INSTANCE ID (16 bit)		
MAJOR VER. (8bit)	TTL (24 bit)			
MINOR VERSION (32 bit)				

Slika 12. Prikaz polja Entry array za servise

Tip 1 je ulaz za servise i sadrži 16 *byte*-ova. Tipovi ulaza koje podržava su servisi 0x00, 0x01, 0x02i 0x03. Prvo polje *Type* označava koji je tip i može biti postavljen u 0x00 ako se koristi za potražnju servisa (*FindService*) ili u 0x01 ako se nudi servis ili prestaje nuditi (*OfferService* ili *StopOfferService*). Polje indeksa prve opcije pokretanja je indeks prve opcije koja će se pokrenuti ovim ulazom i sadržana je u polju *Options array*. Isto vrijedi i za drugu opciju pokretanja. U poljima *Number of Option 1* i *Number of Option 2* su sadržani brojevi opcija koje pojedina opcija koristi. Ako su postavljeni u „0“ smatraju se praznima, odnosno smatra se da nema opcije za pokretanje. Implementacija bi trebala podržavati SD poruke u kojima je duljina opcije za pokretanje postavljena u nulu, a indeks nije. Podržavanje dvije opcije pokretanja nudi mogućnost dodavanja opcije za višestruke SD ulaze, odnosno opcije koje su zajedničke između više SOME/IP-SD ulaza i opcije različite za svaki SD ulaz [13,14].

Service ID i *Instance* ID opisuju ID servisa i servisne instance koji se odnose na ovaj ulaz, ovisno jesu li poslužitelj ili klijent. Ukoliko ta instanca nije jedina, to polje će imati vrijednost 0xFFFF. *Major Version* polje označava glavnu, osnovnu verziju te servisne instance. TTL polje sadrži životni ciklus tog ulaza u sekundama, osim za Stop-ulaze koji imaju TTL u nuli. *Minor Version*

sadržava manju, sporednu verziju servisne instance [13,14]. Kako bi bile moguće migracije, ECU podržava korištenje i nuđenje različitih i nekompatibilnih verzija na istom uređaju ali moraju biti zadovoljeni slijedeći uvjeti. Poslužitelj će uvijek ponuditi jednu instancu usluge glavne (*Major*) verzije, a klijent će tražiti uslugu prema podržanoj glavnoj verziji ili će tražiti 0xFF što podrazumijeva bilo koju verziju. Pretplatiti će se na događaj onakve verzije kakva mu treba. Sviulazi jedne SOME/IP-SD poruke će koristiti isti ID servisa i instance ali različite glavne verzije. Poslužitelj mora demultipleksirati poruke ovisno o *socketu* na koji stižu te prenijeti informacije o verziji, ID poruke i ostalo na ispravan prijemnik [13, str.72-73].

4.2. Ulaz za grupe događaja (Tip 2)

TYPE (8 bit)	INDEKS 1ST. OPTIONS (8 bit)			INDEKS 2ND OPTIONS (8 bit)	# OF OPT 1	# OF OPT 2
SERVICE ID (16 bit)				INSTANCE ID (16 bit)		
MAJOR VER. (8bit)	TTL (24 bit)					
RESERVED (8 bit)	I	R 2	COUNTER	EVENTGROUP ID (16 bit)		

Slika 13. Prikaz polja Entry array za grupe događaja

Kod ulaza za grupe događaja nešto su drugačija zadnja polja. Ovaj ulaz podržava tipove 0x04, 0x05, 0x06 i 0x07. Polje *Type* može imati vrijednost 0x06 ako se odnosi na pretplatu za grupu događaja ili prekid te pretplate (*SubscribeEventgroup* ili *StopSubscribeEventgroup*) te 0x07 ako se ulaz koristi za potvrdu ili odbijanje pretplate (*SubscribeEventgroupAck* ili *SubscribeEventgroupNack*). Za polja opcija se odnosi sve isto kao kod ulaza za servise. ID servisa iinstance se odnosi na servis grupe događaja također ovisno radi li se o ulazu klijenta ili poslužitelja.*Major* i TTL su isti prethodno opisanom ulazu. *Reserved* polje je postavljeno statički u nulu (0x00)i ne koristi se. Slijedeće polje na slici označeno sa I predstavlja *Initial Dana Requested Flag* i zauzima 1 bit. Ako početne podatke šalje poslužitelj, odnosno ako od njega započinje s inicijalnim podacima onda je ta zastavica postavljena uvijek u „1“. Nakon zastavice je još jedno rezervirano polje (*Reserved 2*) koje je također postavljeno u nulu ako nije drugačije specificirano. Brojač (engl. *Counter*) se koristi da bi se razlikovali identični ulazi istog pretplatnika, tj. višestruke pretplate na istu grupu događaja istog klijenta. Ako se ne koristi, to je polje postavljeno u nulu. U polje *Eventgroup* ID se unosi ID grupe događaja na koji se ovaj ulaz odnosi [13,14].

4.3. Polje opcija (*Options Array*)

Nakon polja za ulaze slijedi polje sa duljinom polja opcija te samo polje opcija (engl.

Options Array) koje je ujedno i zadnje polje u SD poruci. Ono sadrži dodatne informacije poput dostupnosti usluge, IP adresu, transporti protokol, broj *port*-a, *host name* i slično [13,14]. Podjela tog polja je prikazana na slijedećoj slici (Slika 14.).

LENGTH (16 bit)	TYPE =0X01 (8bit)	RESERVED (8 bit)
ZERO-TERMINATED CONFIGURATION STRING (32 bit)		

Slika 14. Prikaz podjele polja Options Array

Započinje poljem duljine koje sadrži njegovu duljinu izuzev samog polja duljine i polja *Type* koje slijedi iza njega. Polje *Type* specificira tip opcije i uvijek je postavljen u 0x01. Rezervirano polje kao i svako do sada spomenuto je postavljeno u 0x00.

U samom polju *Zero-terminated Configuration String* se prenose proizvoljni konfiguracijski nizovi. Upotreba ovog polja je limitirana na ulaze tipa 1 bilo kakvog servisa i ulaze tipa 2 samo za ID servisa 0xFFFE. Taj ID označava Non-SOME/IP servise koji se ne identificiraju putem unikatnog 16 bitnog *Service* ID-a ali ima jedinstvenu vrijednost ključa koji je zapisan u polju. ECU koji prima poruku s takvim unosom koristi tu konfiguraciju i ključ kako bi identificirao servis ili grupu događaja. Broj konfiguracijskih stavki koje se mogu prenijeti je od 0 do n. Zapis u konfiguracijskom polju je u DNS TXT i DNS-SD formatu[12] [13,14].

0x0010		**0x01**	**0x00**
[5]	**a**	**1**	**3**
=	**b**	**[3]**	**b**
=	**c**	**[4]**	**9**
e	**=**	**z**	**[0]**

Slika 15. Primjer sadržaja konfiguracijskog polja

Zapis u konfiguracijskom polju započinje s *byte*-om koji označava dužinu, odnosno broj *byte*-ova koji slijede u zapisu iza njega (Slika 15.). Posljednji karakter u nizu tog polja uvijek treba biti nula što označava da iza njega više nema zapisa, tj. broj *byte*-ova koji slijede je nula. Taj niz znakova sadrži ključ i po želji vrijednost , a znak „=“ između ih dijeli. Ako je više od jednog ključa u nizu onda se znakovi ne bi trebali ponavljati i trebao bi biti sadržan barem jedan znak koji nije

[12] DNS-SD koristi DNS TXT zapis za pohranu proizvoljnih imena/vrijednosti kako bi se transportirale dodatne informacije o servisu. Namjera DNS-SD TXT je pružiti mali broj korisnih informacija o usluzi. Izvor: http://www.zeroconf.org/Rendezvous/txtrecords.html

prazan prostor (engl. *Whitespace character*). Znakovi se moraju ispisivati US-ASCII vrijednostima (0x20- 0x7E) izuzev znaka jednakosti (0x3D). Podržan je unos višestrukih ulaza s istim ključem u jednom polju konfiguracije. Znak jednakosti ne smije biti prvi znak u nizu ali ako postoji zapis bez njega onda se cijeli zapis smatra ključem, odnosno da je prisutan ključ bez vrijednosti, a ako je znak jednakosti zadnji u nizu onda se smatra da postoji ključ sa praznom vrijednošću [13,14].

Ulazi podržavaju dvije opcije pokretanja koje mogu pokazivati na istu opciju pomoću različitih ulaza. Sa samo jednom opcijom pokretanja, krajnje točke s različitim konfiguracijskim opcijama neće raditi učinkovito.

4.4. Komunikacija SOME/IP-SD s drugim protokolima (non-SOME/IP)

Osim SOME/IP unutar automobila postoje i drugi protokoli i načini komunikacije, primjerice za dijagnostiku i ažuriranja. Ti protokoli mogu zatražiti informacije i imati potrebu komunicirati sa servisnom instancom ili imati svoju grupu događaja. Te druge protokole nazivamo non-SOME/IP protokolima. Aplikacijski protokol ne koristi SOME/IP ali se objavljuje preko SOME/IP-SD i kao takav ima svoj specijalni *Service* ID postavljen u 0xFFFE i rezerviran je samo za tu upotrebu. Dodaje se konfiguracijska opcija i mora sadržavati najmanje jedan ulaz sa kličem „otherserv“ i podesivu vrijednost koja ne smije biti prazna, a određena je sustavom. *String* „otherserv“ se ne smije koristiti niti u jednoj drugoj SOME/IP usluzi. Za pronalazak, ponudu ili zahtijevanje usluge (*Find, Offer, Request Service*) taj *string* služi kao najava za non-SOME/IP servisnu instancu [13, str.55].

4.5. IPv4 i IPv6 opcije krajnjih točaka

Polja koja transportiraju IP adresu, protokole četvrtog sloja (UDP iliTCP) i broj *port*-a, tj. informacije potrebne za komunikaciju sa servisom se razlikuju u mogućnostima ovisno je li prenose IPv4 ili IPv6 adresu i jesu li *unicast* ili *multicast*. Kad ECU zaprimi poruku koja nudi uslugu, dinamički može konfigurirati *Socket Adaptor* za korištenje te usluge ažurirajući *Socket Connection*. Ova polja opisuju podatke krajnje točke (Slika 16):

LENGHT (16 bit)		TYPE (8 bit)	RESERVED (8bit)
IPv4 (32 bit) ili IPv6 (128 bit)			
RESERVED (8 bit)	L4-PROTO (TCP/UDP)	PORT NUMBER (16 bit)	

Slika 16. Prikaz IPv4/IPv6 polja krajnjih točaka

Svaki ulaz ponude servisa (*OfferService*) mora upućivati na dvije opcije krajnje točke, jednu za UDP i jednu za TCP bez obzira radi li se o IPv4 ili IPv6. Te krajnje točke poslužitelj koristi i to su one točke koje trebaju njegovu uslugu. Krajnje točke poslužitelja koji nudi uslugu se koriste kao izvor događaja, a krajnje točke klijenta koji se pretplaćuje na grupu događaja se koriste za pružanje informacija o adresi i broju *port*-a na koji želi primiti događaje. Različite servisne instance istog servisa na istom ECU koriste različite krajnje točke tako da ih se može razlikovati, no različiti servisi mogu imati jednake krajnje točke. SD će prebrisati preuzetu IP adresu i *port* u krajnjim točkama i multicast opcijama ako se razlikuju od statički konfiguriranih vrijednosti. Polje *Length* se razlikuje ovisno o adresi jer su im duljine različite pa IPv4 ima statički postavljenu veličinu na 0x0009, a IPv6 u 0x0015. Polje *Type* ima vrijednost 0x04 ukoliko se radi o IPv4, a 0x06 ako je IPv6. *Reserved* polja kao i do sada ostaju na 0x00. U polje za adresu se sprema lokalna IP adresa servisa ili grupe događaja. L4-Proto označava protokol četvrte razine ISO/OSI modela i može imati vrijednost 0x06 za TCP prijenos ili 0x11 ako je UDP prijenos. Polje *Port Number* sadrži broj *port*-a [13,14].

Multicast polja se razlikuju od prethodnog samo u tome što su adresa i *port* višestruki (*multicast*), a na transportnom sloju je podržan samo UDP. I polja dužine ostaju ista ali se mijenjaju *Type* polja pa je tako za IPv4 *multicast* vrijednost 0x14, a za IPv6 *multicast* 0x16. *Multicast* opcije koristi poslužitelj da bi najavio adresu, protokol četvrtog sloja i broj *port*-a na koji se šalju *multicast* događaji i obavijesti i te poruke će biti referencirane porukama prihvaćanja pretplate na grupu događaja.

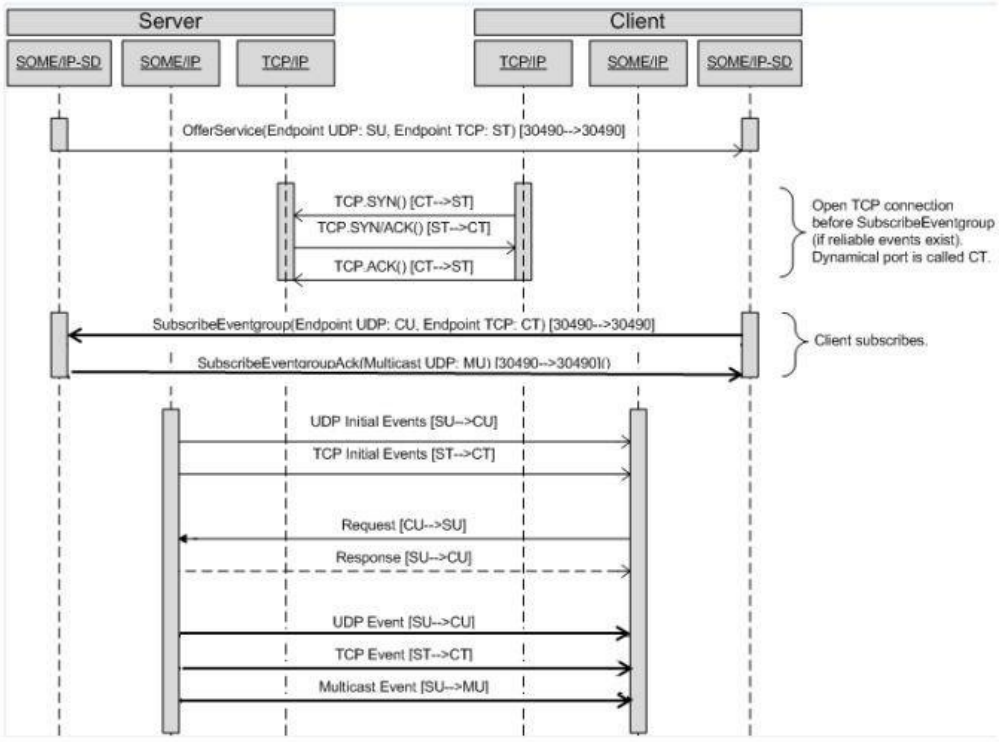
Postoje i SD *Endpoint Option* koje prenose krajnje točke, IP adresu i *port* od pošiljatelja SD poruke. Koriste se za identifikaciju SOME/IP-SD instance čak i u slučajevima kada se IP adresaili broj *port*-a ne mogu koristiti. Takav slučaj bi bio otkrivanje *proxy*[13] servisa na jednom ECU koji upravlja *multicast* prometom za drugi ECU. Svaka SD poruka mora sadržavati IPv4 ili IPv6 polje krajnjih točaka ovisno kako se prenosi i to bi trebala biti prva opcija u *Options Array* ako postoji. Ako se primi više opcija krajnjih točaka samo se prva uzima u obzir, ostale se ignoriraju, a niti jedan SD ulaz ne odnosi se na krajnju točku. Polje *Type* IPv4 SD krajnje točke ima vrijednost 0x24,a IPv6

[13] *Proxy* je posrednički poslužitelj, odnosno računalo koje stoji između klijenta i glavnog poslužitelja. Najčešće se koristi za uporabu interneta. Izvor: https://hr.wikipedia.org/wiki/Proxy

SD vrijednost 0x26. Koriste *unicast* adrese i ti podaci su važni kod odgovora na SD poruku,poput odgovora ponude nakon poruke potražnje, pretplate nakon poruke ponude ili poruka prihvaćanja pretplate [13,14].

Servisne krajnje točke, primjerice za ponudu servisa, označavaju IP adresu i broj *port*-a usluge koja je dostupna na poslužitelju ili IP adresu i broj *port*-a s koje servisna instanca šalje događaje. Događaji se neće slati niti sa jedne druge krajnje točke osim onih navedenih u opcijama ulaza ponude.

Krajnje točke grupa događaja se također upotrebljavaju za slanje *unicast* UDP ili TCP događaja određene servisne instance pa je navedena adresa i *port* u ulazu za pretplatu na grupu događaja klijentova. TCP događaji se prenose preko TCP veze koju je klijent otvorio prije nego je poslao ulaz zahtjeva za pretplatu. Inicijalni, početni događaji se šalju *unicast* od poslužitelja prema klijentu. Potvrda pretplate za grupu događaja (engl. *Subscribe Eventgroup Ack*) će imati najviše jednu *multicast* opciju za IPv4 ili IPv6, a za transportni protokol će biti postavljen UDP. Klijent će čim prije otvoriti krajnju točku navedenu u *multicast* opciji potvrde za pretplatu da ne bi propustio niti jedan *multicast* događaj. Primjer je na slijedećoj slici preuzetoj iz Autosar-ove SOME/IP Service Discovery Protocol Specification [13].



Slika 17. Primjer komunikacije poslužitelj/klijent s različitim krajnjim točkama i multicast opcijom (Izvor: AUTOSAR, SOME/IP Service Discovery Protocol Specification [13, str.41]

Poslužitelj nudi servisnu instancu sa UDP krajnjim točkama (SU) i TCP krajnjim točkama (ST) nakon čega klijent otvara TCP komunikaciju. Klijent šalje ulaz za pretplatom na grupu događaja (*SubscribeEventgroup*) sa UDP (CU) i TCP (CT) krajnjim točkama što poslužitelj potvrđuje slanjem *SubscribeEventgroupAck* ulaza na *multicast* adresu. Nakon toga se šalju inicijalni događaji UDP protokolom. Klijent poziva metodu i poslužitelj mu odgovara te šalje događaj putem UDP veze, zatim TCP i na kraju na *multicast* adresu [13, str. 40].

4.6. Servisni ulazi (*Service Entry*)

Ulaz za pronalazak servisa (engl. *Find Service Entry*) je tip ulaza koji se koristi za pronalazak servisne instance i biti će poslan samo ako je poznato trenutno stanje servisa, tj. nije zaprimljena ponuda. Polja ulaza bi trebala biti postavljena na slijedeći način: *Type* na 0x00 što označava da je to *FindService*, *Service* ID je ID servisa kojeg se traži, Instance ID je 0xFFFF ako želimo da se vrati bilo koja instance usluge, a ukoliko je potrebna neka određena onda će biti njen ID. To znači da ako stigne poruka potražnje s ID instancom 0xFFFF na nju se odgovara instancom koju servis nudi. Primjerice, ECU1 nudi uslugu 0x1111 s ID instance 0xabab. ECU2 traži uslugu 0x1111 s ID instance 0xFFFF na koju ECU1 odgovara prethodnom ponudom. Ako želimo da nam se vraćaju svi servisi neovisno o verziji, onda u *Major Version* treba postaviti 0xFF. Kada bi postavili neku drugačiju vrijednost onda bi nam se vraćali samo servisi s tom određenom verzijom koju smo definirali. Isto vrijedi i za *Minor Version* koja bi trebala biti postavljena u 0xFFFF FFFF. Različite glavne verzije sučelja nisu kompatibilne, no različite *Minor* jesu. Očekivano je u realnoj situaciji da je glavna, *Major* verzija definirana na klijentovoj strani i da će samo takvu i potraživati, ali očekivano je i da će *Minor* verzija biti postavljena u 0xFFFF FFFF i da će prihvaćati bilo kakve *Minor* verzije sučelja. TTL se postavlja na životni vijek ovog ulaza i nakon toga se smatra da višene postoji. U slučaju da se postavi na 0xFFFFFF smatrati će se validnim i slijedeći krug, odnosno nakon resetiranja. Vrijednost 0x000000 se ne postavlja jer to označava stopiranje servisa (*Stop Offer Service Entry*). Pošiljatelj nije obvezan upisivati krajnje točke u ulaz, a primatelj će ih ignorirati ako budu upisane [13,14].

Ulaz za ponudu servisa (engl. *Offer Service Entry*) služi da bi se ponudila usluga drugim komunikacijskim partnerima. Ovaj ulaz za tip poruke ima 0x01 što označava da je to *OfferService*,a ID servisa je ID onog servisa koji se nudi, isto kao ID instance. Nije dopuštena vrijednost ID servisa 0xFFFF jer to označava razne slučajeve, a nuditi se može jedna usluga. Sve vrijednosti u poljima se odnose na uslugu koju nudi. Na isti način kao u prethodno objašnjenom ulazu se postavlja i *Major* i *Minor* verzija sučelja te TTL vrijednost. *Offer Service* uvijek mora imati vrijednosti barem jedne od krajnjih točaka, IPv4 ili IPv6 kako bi signalizirao da je usluga dostupna. Za svaki sloj transportnog protokola potrebna je jedna krajnja točka. U slučaju UDP prijenosa taj sepodatak koristi kao adresa i

port izvora događaja i obavijesti i to je adresa na koju klijent može poslati zahtjev za metodom, a u slučaju TCP prijenosa ta adresa i broj *port*-a su potrebni za otvaranje TCP veze u redoslijedu zaprimanja događaja. SOME/IP će dopustiti korištenje UDP i TCP istovremeno ali se mora odrediti konfiguracijom koja poruka ide kojim protokolom, što znači da jedna poruka ne može ići i UDP-om i TCP-om [13,14].

Ulaz za prekid ponude servisa (engl. *Stop Offer Service Entry*) se koristi kada se prekida usluga koja se nudi. Polje *Type* je isto kao i kod ponude ali je glavna razlika u životnom ciklusu koje se postavlja u 0x000000 što znači da je određena ponuda navedena u tom ulazu stopirana [13,14].

4.7. Ulazi za grupe događaja (engl. *Eventgroup Entries*)

Ulaz pretplate na grupu događaja (engl. *Subscribe Eventgroup Entry*) je tip 2 ulaza i koristi ga upravljačka jedinica kako bi se pretplatila na grupu događaja druge upravljačke jedinice. Polje Type se postavlja u vrijednost 0x06 što ukazuje da se radi o pretplati na grupu događaja, a ID servisa, instance, glavna verzija i ID grupe događaja se odnose na grupu na koju se klijent pretplaćuje. U toj situaciji se može dogoditi da postoje dva identična ulaza koja onda razlikujemo koristeći polje *Counter,* bez obzira što su im različite krajnje točke. To polje će se od strane poslužitelja reflektirati na ulaze potvrde i odbijanja. Vrijednost brojača se određuje implementacijom, a ukoliko se ne koristi postavlja se u nulu. Veličina polja od 4 bita limitira korištenje na najviše 16 različitih pretplata po istoj grupi događaja. TTL će sadržavati vrijednost životnog ciklusa pretplate i može biti postavljen u 0xFFFF FFFF ali ne i u 0x0000 0000. Klijent određuje koliko će mu vremena trebati određeni podaci i postavlja TTL. Zastavica za početnim podacima (*Initial Dana Request Flag*) se postavlja u „1“ samo u slučaju ako klijent šalje prvu pretplatu i želi pokrenuti inicijalno slanje početnih događaja. U suprotnom je uvijek postavljena u

„0“. Sva rezervirana polja su uvijek postavljena u „0“. Ulaz pretplate će sadržavati i jednu ili dvije IPv4 ili/i jednu ili dvije IPv6 adrese, jednu za UDP, jednu za TCP [13,14].

Ulaz za prekid pretplate na grupu događaja (engl. *Stop Subscribe Eventgroup*) ima identična polja kao ulaz za pretplatu (niti polje *Type* se ne razlikuje), osim TTL životnog ciklusa koji se postavlja u 0x0000 0000.

Ulaz prihvaćanja pretplate na grupu događaja (engl. *Subscribe Eventgroup Acknowledgement*-ACK) koristi poslužitelj da bi obavijestio klijenta da je prihvatio njegov zahtjev za pretplatom. Od prethodnih ulaza tipa 2 se razlikuje u polju *Type* koje je u ovom slučaju 0x07, odnosno sva ostala polja su ista kao u ulazu pretplate koju se potvrđuje. Ako se odnose na obavijesti i događaje koji se šalju *multicast*-om, polje *multicast* opcija mora biti popunjeno, IPv4 ili IPV6 i broj *port*-a na koje će događaji ili obavijesti biti poslani [13,14].

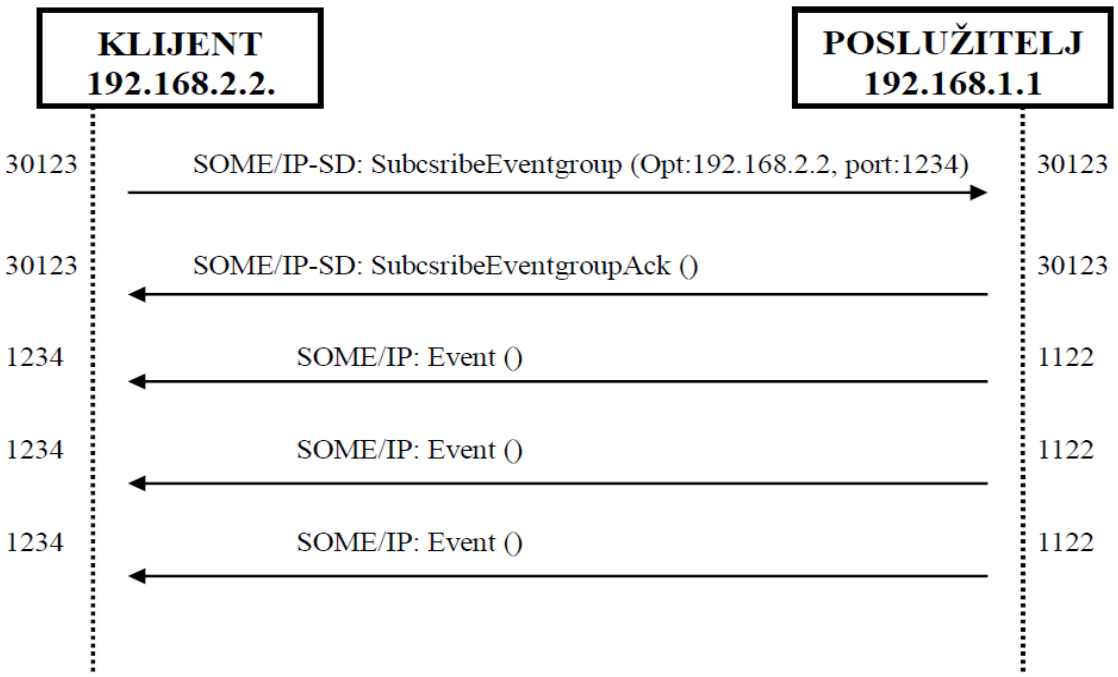
Ulaz odbijanja pretplate na grupu događaja (engl. *Subscribe Eventgroup Negative Acknowledgement*-NACK) je u suprotnosti s prethodnim ulazom, šalje se kada pretplata na određenu grupu događaja nije prihvaćena iz nekog razloga, a mogući razlozi su slijedeći:

- Vrijednosti u poljima su nepoznate
- Klijent nije otvorio TCP vezu
- Problemi s resursima na poslužitelju
- Kombinacija ID-eva je nepoznata ili je došlo do konflikta s krajnjim točkama

Konflikt s krajnjim točkama se može dogoditi u slučaju da su jednake ali im se jedna od vrijednostirazlikuju, adresa ili *port*. Polje *Type* je isto kao i kod ulaza prihvaćanja, a sva ostala polja imajuvrijednosti kao i ulaz koji se odbija. TTL polje je nula. Kad klijent primi NACK kao odgovor na pretplatu za koju je potrebna TCP veza, on ju odmah provjerava i po potrebi resetira. Može sedogoditi situacija da je poslužitelj izgubio TCP vezu, a klijent nije. Provjera uključuje pregledprimaju li se podaci drugih grupa događaja, slanje čarobnih kolačića i čekanje prihvaćanja tenaposljetku ponovno otvaranje veze [13, str.42-45].

4.8. Objavljivanje / Pretplata (engl. *Publish / Subscribe*)

Ovaj način se razlikuje od *Request*/*Response* komunikacije iz razloga što je dovoljno da s klijent jednom pretplati na skup parametara koje zahtjeva od poslužitelja i onda će te informacije zaprimati stalno u određenim ciklusima onoliko dugo koliki je životni ciklus pretplate (TTL). Slanje i primanje tih informacija se nazivaju obavijestima (engl. *Notifications*) i podrazumijevaju događaje i polja (engl. *events and field*). Svi klijenti kojima su potrebni događaji i/ili obavijesti događaja moraju se pretplatiti koristeći SOME/IP-SD poruke [13, str.57]. Primjer interakcije obavijestima je prikazan na slijedećoj slici (Slika 18.):



Slika 18. Primjer interakcije obavijestima između klijenta i poslužitelja

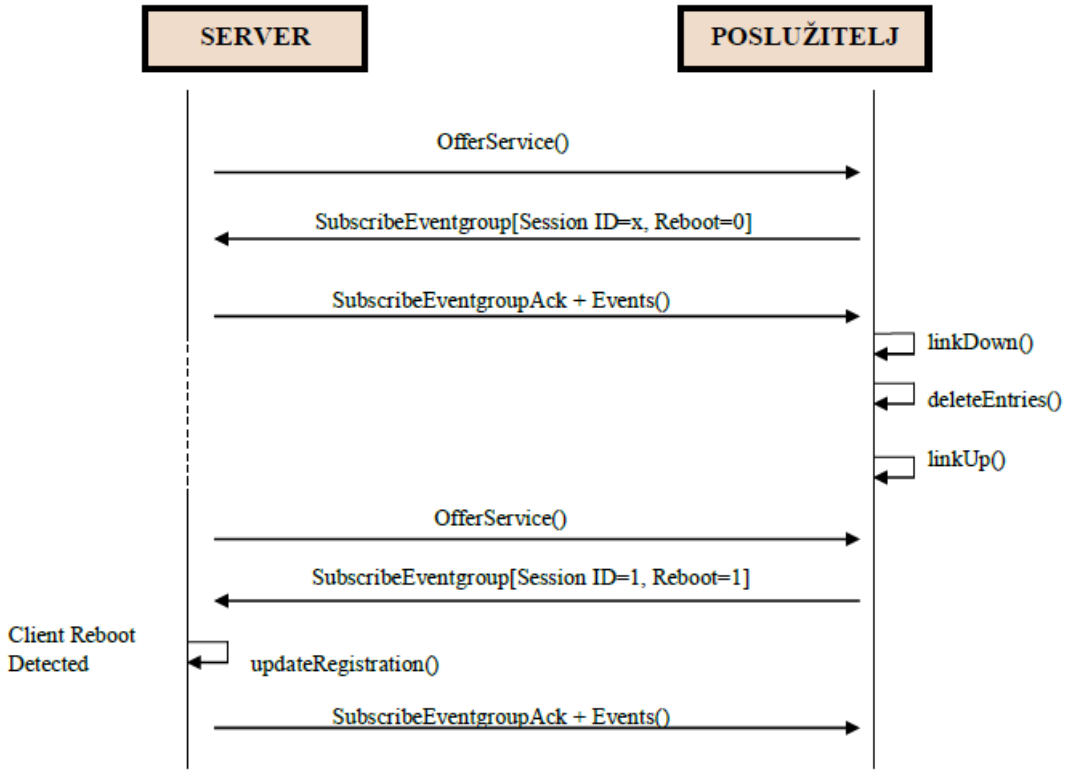
Ovaj način je usporediv ali ne identičan MOST mehanizmu obavijesti. Nakon zaprimanja ulaza prvi korak je identificiranje te servisne instance ili grupe događaja. Ako stigne *FindService* ulaz, ID servisa, instance, *Minor* i *Major* verzija se moraju u potpunosti podudarati konfiguriranim vrijednostima servisnih instanci i njima pridruženim grupama događaja. Izuzetak su u polju upisane „bilo kakve vrijednosti“, odnosno 0xFF za glavnu verziju, 0xFFFFFF za *Minor* verziju, 0xFFFF za ID usluge ili ID instance. Primjerice, ako se potražuje usluga s ID 0x1234, ID instance 0xFFFF, glavnom verzijom 0x05 i *Minor* verzijom 0xFFFFFF, onda se samo provjeravaju podudaranja ID servisa i glavne verzije s lokalnom servisnom instancom i pridruženom grupom događaja kako bi sepronašla usluga koja odgovara traženoj. Ista situacija je i kod zaprimanja ponude usluge (*OfferService*) ili ulaza za grupe događaja (*EventGroup Entry*). U slučaju zaprimanja SD poruka korištenjem *multicast* adrese, te poruke može primiti više upravljačkih jedinica i svi oni mogu odgovarati paralelno na te poruke što bi moglo dovesti do zagušenja i preopterećenja upravljačke jedinice koja je prva poslala poruku. Zbog toga se uvodi odgađanje odgovora [14, str. 63].

Sa SOME/IP ulazom za ponudu usluge (*OfferService*) pošiljatelj nudi slanje obavijesti klijentu i na isti način se obavlja pretplata, a ponuda služi kao okidač (engl. *trigger*). Svaki klijentje dužan odgovoriti na SD ponudu poslužitelja sa SD pretplatom na grupu događaja sve dok je zainteresiran za primanje događaja i obavijesti te grupe. I ako ne želi pretplatu odgovoriti će porukom bez konfiguracije o čekanju, zahtjevu i odgodi, što znači da neće čekati ali će ipak odgovoriti. Ako klijent može otkriti ponovno pokretanje servisa gledajući *reboot* zastavicu, onda može odabrati odgovarati jedino na poruke ponude tog servera kada se ponovno pokrene. Za to bi naravno morao

imati podešen TTL na maksimalnu vrijednost. Ovo djeluje pouzdano čak i kad se izgube poruke na poslužitelju. Ukoliko klijent nema aktivnu pretplatu na grupu događaja, izričito ćezatražiti početne događaje (*Initial Events*) postavljanjem *Initial Dana Requested Flag* na određenu vrijednosti, no ako šalje zahtjev za dodatnu pretplatu na grupu događaja i ako TTL nije istekao zato vrijeme, onda ne zahtjeva početne događaje. Najčešći razlozi za slanje inicijalnih događaja su [13, str.58]:

- Kada klijent nije trenutno pretplaćen na grupu događaja
- Klijent je primijetio da je veza sa poslužiteljem pukla od zadnje pretplate
- Klijent nije zaprimio potvrdu (ACK) od poslužitelja za pretplatu nakon poslanog zahtjeva
- Klijent je detektirao ponovno pokretanje poslužitelja

Može se dogoditi da se klijent pretplati na više grupa događaja koje uključuju jedan ili više identičnih događaja ili polja, i tada poslužitelj neće slati duplicirano niti će to shvatiti kao ponovnu pretplatu pa se neće niti slati inicijalni događaji. Slijedeća slika prikazuje odvijanje komunikacije u slučaju gubitka veze na strani klijenta (Slika 19.) [13, str.59]:



Slika 19. Prikaz komunikacije u slučaju gubitka veze na strani klijenta

I klijent i poslužitelj moraju implementirati detekciju ponovnog pokretanja kako je navedeno u AUTOSAR specifikaciji. Pošiljatelj koji šalje ponudu implicitno kao objavu mora čuvati stanja poruka za pretplatu na tu instancu grupe događaja kako bi znao redoslijed slanja obavijesti i događaja. Klijent se odjavljuje s pretplate tako da šalje SOME/IP-SD poruku pretplate na tu grupu s TTL vrijednosti u 0. Ako dođe do pogreške prilikom slanja događaja ili obavijesti, primjerice greška u TCP vezi ili komunikaciji, SD će na poslužitelju obrisati pretplatu. Klijent je dužan otvoriti i uspostaviti vezu prema poslužitelju prije slanja pretplate ako se usluga nudi preko TCP veze, a klijent zahtijeva prema konfiguraciji primitak skupine događaja putem TCP-a. Nakon što klijent pošalje poruku pretplate, poslužitelj odgovara prihvaćanjem. Klijent će pričekati na potvrdu jedno vrijeme ali ako ona ne stigne prije ponovnog slanja ulaza za pretplatu i ako je zastavica za slanje inicijalnih događaja na poslužitelju postavljena u nulu, klijent šalje ulaz zaustavljanja pretplate i u istoj toj SOME/IP-SD poruci šalje ulaz za ponovnu pretplatu na tu grupu događaja, drugim riječima, klijent u istoj poruci šalje dva ulaza, jedan kojim zaustavlja pretplatu i jedan za ponovno zahtijevanje pretplate. Ako je spomenuta zastavica bila postavljena u „1“ u trenutku prekida, ponovnim slanjem se opet postavlja u „1“. To se ne primjenjuje na ponude koje su odgovor na potražnju usluge. Ako klijent šalje pretplatu na grupu događaja kao reakcija na *unicast* ponudu i odmah nakon toga stigne *multicast* ponuda, prije nego je poslužitelj uspio odgovoriti potvrdom na prethodnu pretplatu, klijent se ne može žaliti o ne primanju prihvaćanja, tj.zaustaviti pretplatu. Ti uvjeti su uvedeni kako bi gubici komunikacije i/ili gubitka paketa trajali što kraće, odnosno da bi se čim prije započelo/nastavilo ponovno razmjenjivanje informacija. SOME/IP podržava automatsko pokretanje slanja inicijalnih događaja. Postoje određeni slučajevi gdje će poslužitelj bez obzira na postavljenu zastavicu ipak poslati početne događaje odmah nakon prihvaćanja pretplate. Klijent će ponoviti pretplatu ako ne primi događaj/obavijest u određenom vremenu neovisno o stanju zastavice. Ako je pretplata već validna i ažurna, neće se slati početni događaji, no ako je primjeno zaustavljanje pretplate i odmah ponovna pretplata to je okidač za slanje početnih događaja ili obavijesti polja [13, str. 59-63]. Ako se inicijalni događaji šalju, šaljuse uvijek nakon prihvaćanja pretplate. U slučaju da se klijent pretplaćuje na različite grupe događaja iste servisne instance koje uključuju isto polje u različitim SOME/IP-SD porukama, poslužitelj će poslati početne događaje za svaku pretplatu odvojeno, no u slučaju da se radi o pretplati na različite grupe događaja u jednoj SOME/IP-SD poruci onda poslužitelj može poslati samo jednom početne događaje za tu poruku, ali ta odluka ovisi o njegovoj arhitekturi. SOME/IP- SD se može automatski prebacivati s *unicast* na *multicast* i obratno ako je pređen odabrani prag broja pretplatnika. Podržati će implicitnu konfiguraciju krajnjih točaka i registraciju pretplatnika, a one se moraju temeljiti na statičkim konfiguracijama i ne koriste SD poruke na mreži.

Može postojati slučaj korištenja usluga baziranih na statičkoj konfiguraciji koji uopće ne

koriste SD i nisu dio SOME/IP-a pa se ne mogu pronalaziti na mreži i pretplaćivati ali se njihove usluge koriste na drugačiji način i njihova konfiguracija i implementacija je specificirana projektom na početku.

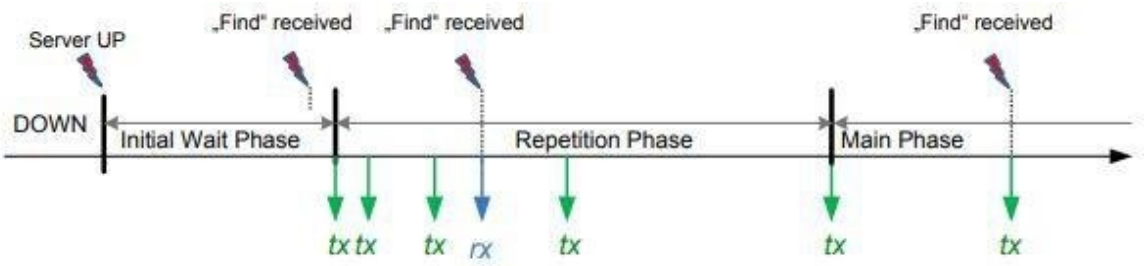
Pretplate (*Subscribe Eventgroup*), prekid pretplate (*Stop Subscribe Eventgroup*), prihvaćanje pretplate (*Subscribe Eventgroup Ack*) i odbijanje (*Subscribe Eventgroup Nack*) se šalju jedino i isključivo *unicast* načinom [13, str.63-66].

4.9. Komunikacija

SOME/IP će smanjiti broj SD poruka pakirajući ulaze zajedno kad god je to moguće.Svaka servisna instanca ili grupa događaja mora imati najmanje tri faze kod slanja ulaza[13, str.45]:

- Inicijalna faza čekanja (engl. *Initial Wait Phase*)
- Faza ponavljanja (engl. *Repetition Phase*)
- Glavna faza (engl. *Main Phase*)

U stvarnosti, potrebno je više od ove tri faze. Prikaz komunikacije između navedenih faza je na slijedećoj slici (Slika 20.):



Slika 20. Prikaz komunikacije između tri osnovne faze
Izvor: AUTOSAR, Specification of Service Discovery [14, str.66]

4.9.1. Inicijalna faza čekanja

Service Discovery će ući u inicijalnu fazu čekanja na klijentovoj strani za servisnu instancu kada je klijent zatražio uslugu *broadcast* porukom (*Find Services*) i kada je uspostavljena veza za tu servisnu instancu. Klijent u početnu fazu čekanja ulazi na zahtjev iz aplikacijskog sloja i ostaje u njoj neko nasumično izabrano vrijeme specificirano u SOME/IP. Tijekom te faze klijent je tih. Ponuda (*Offer Service*) od poslužitelja stiže čim prije, čak i tokom trajanja te faze i kada se pojavi usluga za koju je zainteresiran onda poduzima akciju, pretplaćuje se na taj servis i ulazi u glavnu fazu. Slično, poslužiteljev SD će ući u tu fazu kad je njegova usluga dostupna i kad je veza potrebna za ovu instancu na sučelju uspostavljena. Poslužitelj *broadcast*-a poruku ponude na mrežuda obavijesti o dostupnosti servisa. Može se dogoditi da je veza uspostavljena ali usluga još nije dostupna i obratno. Dostupnost usluge objavljuje aplikacijski sloj i tijekom toga nasumično odabranog vremena poslužitelj ne šalje nikakve poruke. Za razliku od klijenta, poslužitelj će ignorirati svaku pristiglu poruku potražnje servisa za to vrijeme, a i ne može preskočiti fazu ponavljanja[11, str.50].

Nakon ulaska u inicijalnu fazu čekanja, *Service Discovery* će pričekati koliko je određeno u INITIAL_DELAY prije slanja prvih poruka. INITIAL_DELAY se definira kao minimalna i maksimalna odgoda i odabire se slučajnom (engl. *random*) vrijednosti. Ista slučajna vrijednost se koristi i u slučaju više ulaza različitih tipova koji se pakiraju zajedno u jednu poruku kako bi im se smanjio broj te za različite servisne instance koje mogu obuhvatiti zajedno više ponuda. SD može spakirati više ulaza zajedno i kada nije uključena slučajna vrijednost ali se poruke onda šalju istovremeno. Primjerice, moguće je odgovoriti jednom porukom na sve ulaze pretplate za grupu događaja[13, str.45].

4.9.2. Faza ponavljanja

U fazi ponavljanja klijent će poslati unaprijed određen broj poruka potražnje usluge s eksponencijalno povećanim vremenom čekanja između poruka. Ako za to vrijeme primi poruku ponude o usluzi za koju je zainteresiran, zatražiti će pretplatu i preći će u glavnu fazu. Na isti način, nakon ulaska u fazu ponavljanja poslužitelj *broadcast*-a poruku ponude i one se šalju uzastopno s vremenom odgađanja koje se eksponencijalno povećava. Ako u tom vremenu primi poruku potražnje usluge od klijenta, poslužitelj će čekati neko određeno nasumično vrijeme prije nego mu odgovori *unicast* porukom ponude. Poslužitelj ne mora odgovarati na poruke potražnje u fazi ponavljanja. Iz faze ponavljanja prelazi u glavnu fazu nakon što su sve unaprijed definirane poruke poslane[11, str.50].

Već nakon prvog slanja poruke servisna instanca ulazi u fazu ponavljanja. Trajanje faze ponavljanja je određeno s REPETITIONS_BASE_DELAY, i kao što je već spomenuto, nakon svake

poruke poslane unutar te faze ovo vrijeme se udvostručuje. Broj poslanih poruka za to vrijeme je određeno vrijednosti u REPETITIONS_MAX. Slanje ulaza za pronalazak usluge će se zaustaviti ako stigne odgovarajući ulaz ponude i odmah se preskače na treću, glavnu fazu. Isto će sedogoditi ako je REPETITIONS_MAX postavljen u nula[13, str.46]. Poslužitelj nudi usluge koristeći *multicast* adresu definiranu u SD_MULTICAST_IP, isto kao i klijent tokom potražnje usluge samo on to radi u fazi ponavljanja. Čim primi ispravnu i pravovremenu ponudu prestati će jutražiti.

4.9.3. Glavna faza

Klijent u glavnoj fazi neće slati poruke potražnje samo će odgovoriti na poruku ponude ako je potrebno. Suprotno tome, poslužitelj će ciklički slati poruke ponude i istovremeno odgovarati na pristigle poruke potražnje usluge. Ulaskom u glavnu fazu poslužitelj mora čekati 1·CYCLIC_OFFER_DELAY prije slanja slijedeće poruke određenoj servisnoj instanci. U toj fazi poruke se šalju ciklički kako je određeno u CYCLIC_OFFER_DELAY te nakon svake poslane poruke opet se čeka 1·CYCLIC_OFFER_DELAY[13, str.47].

Primjer slijeda komunikacije u glavnoj fazi:

Inicijalna faza čekanja:

- čekaj iznos slučajne vrijednosti u rangu(INITIAL_DELAY_MIN,_MAX)
- pošalji poruku (ulazi *Find Service* i *Offer Service*)

Faza ponavljanja: (REPETITION_BASE_DELAY=100ms, REPETITION_MAX=2)

- čekaj 2^0·100ms
- pošalji poruku (ulazi *Find Service* i *Offer Service*)
- čekaj 2^1·100ms
- pošalji poruku (ulazi *Find Service* i *Offer Service*)

Glavna faza: (sve dok je poruka aktivna i dok je definiran CYCLIC_OFFER_DELAY)

- čekaj CYCLIC_OFFER_DELAY
- pošalji poruku (ulaz *Offer Service*)

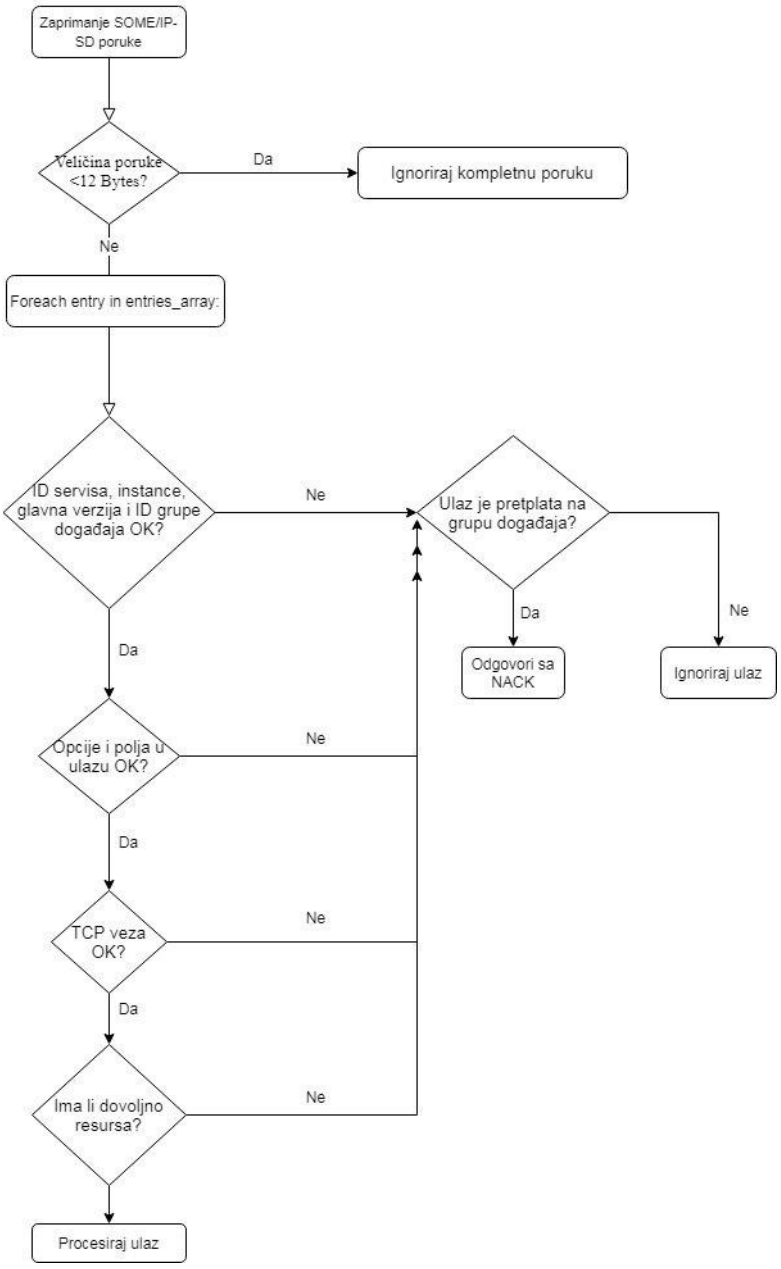
4.9.4. Odgađanje odgovora

Service Discovery će odgoditi odgovor na ulaze koji su primljeni *multicast* SD porukama pomoću konfiguracijske vrijednosti REQUEST_RESPONSE_DELAY. To će se dogodi u dva slučaja. Prvi je slučaj da je primljen ulaz ponude bez obzira bio *multicast* ili *unicast*, nakon što je zaprimljen *multicast* ulaz potražnje servisa. Druga situacija je kada je zaprimljen *unicast* ulaz pretplate na grupu događaja nakon *multicast* poslanog ulaza ponude servisa. Odgoda se ne koristi ako se *unicast* porukom odgovara na *unicast* poruku. REQUEST_RESPONSE_DELAY je određen minimumom i maksimumom i to su zapravo slučajne vrijednosti izabrane u tom rasponu. U osnovi,na sve ulaze za pronalazak usluge (bez obzira na stanje *unicast* zastavice) će biti odgovoreno *unicast* ulazom ponude. Radi optimizacije preporučuje se da u glavnoj fazi na primljene poruke potražnje sa *unicast* zastavicom postavljenom u „1“ odgovori također budu *unicast* ako od zadnje poslane ponude nije prošlo više od 1/2 REQUEST_RESPONSE_DELAY. Ukoliko je prošlo točno toliko ili više odgovara se *multicast* porukom. Na poruke za pronalazak servisa kojima je *Unicast Flag* postavljena u „0“ (*multicast*) odgovara se *multicast* porukom[13, str.47-48].

Klijent može biti u dva moda. U jednom zahtjeva uslugu (engl. *Request*) i šalje poruke u fazi ponavljanja, a u drugom „osluškuje“ (engl. *Listen*) i čeka odgovore na njih. Poslužitelj također radi u dva moda, u modu ponude (engl. *Offer*) kada šalje poruke ponude u glavnoj i fazi ponavljanja i modu „tišine“ (engl. *Silent*) kada će samo odgovarati na poruke [11, str.50]. Kada se servisna instanca poslužitelja nalazi u fazi ponavljanja ili glavnoj fazi i u tom trenutku bude zaustavljena, odmah se šalje ulaz zaustavljanja usluge (*Stop Offer Service entry*). Kada veza pukne u bilo kojoj od tri faze, SD će ponovno ući u inicijalnu fazu čekanja kada se veza ponovno uspostavi i servis bude opet dostupan i ponuđen. Ako veza pukne na klijentovoj strani dogoditi će se isto samo će umjesto ponude usluge nakon ponovnog uspostavljanja veze i inicijalne faze sada klijent opet potraživati uslugu. Kad poslužitelj pošalje ulaz za stopiranje usluge, sve pretplate za usluge te servisne instance će biti izbrisane na njegovoj strani, a isto će se dogoditi i na klijentovoj strani nakon što zaprimi taj ulaz. Nakon toga klijent neće više slati ulaze potražnje servisa nego će čekati ponudu ili promijeniti status. Ukoliko se klijentova servisna instanca nalazi uglavnoj fazi i biva stopirana, SD će poslati ulaz o zaustavljanju na sve pretplaćene grupe događaja. U slučaju da se ugasi cijeli ECU, poruka o zaustavljanju se šalje za sve usluge i grupe događaja [13,str.48-49]. Potvrda o prihvaćanju ili odbijanju pretplate (*Ack* i *Nack*) se šalju koristeći *unicast* adresu.

4.10. Mehanizmi i rukovanje pogreškama i zagušenjima

SOME/IP-SD je dizajniran kao *soft state protocol* što znači da ulazi imaju svoj vijek trajanja i trebaju biti osvježavani kako ostali valjani, no postavljenje TTL vrijednosti na maksimum isključuje ovu mogućnost. Inicijalna faza čekanja je uvedena iz dva razloga, prvenstveno da se izbjegne navala i početna gužva na ECU ali i da bi mogao prikupiti više ulaza u SD poruku. Faza ponavljanja služi da bi se cijeli proces ubrzao, odnosno da bi se ubrzala sinkronizacija klijenta i poslužitelja. Ako klijent malo zakasni s porukom, poslužitelj će ga brzo naći i obratno. Vrijeme između dvije poruke se eksponencijalno povećava da bi se izbjeglo stanje preopterećenja prilikom sinkronizacije. U glavnoj fazi SD pokušava stabilizirati stanje tako da se smanjuje broj paketa potražnje usluge, šalju se samo ponude i to u cikličkom intervalu od primjerice 1s. U velikim sustavima sa puno ECU je vrlo korisno što SOME/IP-SD mora odgađati odgovor na *multicast* ulazejer odgovaranje na poruku sa puno ulaza zahtijeva puno odgovora u isto vrijeme i stvorilo bi gužvui pritisak na ECU koji to zaprima [13, str.52]. Na slici 21. je prikazan način provjere pristigle SOME/IP-SD poruke. Kad poruka pristigne provjerava ispravnost polja kako bi se mogla uspješno procesirati dalje. To uključuje prvo provjeru veličine SD poruke i ukoliko je manja od 12 *Byte*-ova odmah se odbacuje bez daljnje provjere. Zatim se provjerava jesu li poznati *Service* ID, njegova instanca, glavna verzija i ID grupe događaja sa već poznatom glavnom verzijom (vrijedi samo za ulaze grupe događaja). Provjerava se da li je duljina *Entries Array* ispravna. Polje opcija se provjerava je li sintaksno ispravno na način da je duljina polja opcija konzistentna, ako je broj u „Opt1“ polju jednak nuli, indeks prve opcije također mora biti jednak nuli. Isto vrijedi i za opciju 2.Tip opcije mora biti poznat, krajnja točka mora imati validan L4 protokol, a TCP veza se provjerava samo ako je određena konfiguracijom za pretplatu ili grupu događaja. Resursi se provjeravaju zadnji [13, str.54-55].



Slika 21. Način provjere ispravnosti SOME/IP-SD poruke

Zanemaruju se ulazi sa nepoznatim opcijama, suvišnim vrijednostima koje nisu potrebne tom ulazu. Ukoliko pretplata na grupu događaja ima dvije ili više opcija koje su u konfliktu, vraća joj se ulaz *SubscribeEventgroupNack*. Ako primljeni ulaz ne sadrži najmanje konfiguracijske opcije biti će zanemaren ili odgovoren *Nack* ulazom ako se radi o grupi događaja. Nedostajuće informacije *multicast* krajnje točke će se ignorirati od strane klijenta ako je *unicast* komunikacija preko UDP-a (UDP *Endpoint*) ispravno postavljena. U slučaju da su u ulazu dvije različite konfiguracijske opcije spojiti će se, a ako se dogodi sukob jer imaju isto ime onda se ne smiju spajati i tretiraju se odvojeno.

Napomenuti ćemo i faktore koji mogu utjecati na kašnjenje pretplate koje su autori naveli i

simulirali u svojem radu „*Insights on the configuration and performances of SOME/IP Service Discovery*“ [5]. Klijent gubi vrijeme čekajući da servis bude dostupan. Ako servis kasni klijent će se registrirati na prvu ponudu poslanu na kraju inicijalne faze čekanja. Za latenciju su najgori *listen*i *silent* načini rada poslužitelja i klijenta te parametri SOME/IP-SD protokola, no nema smjernica kako ga na najbolji način konfigurirati. Također, navode da se razmak u komunikaciji mjeri u rasponu od mikrosekunde do milisekunde. Zaključeno je da je SOME/IP dinamika dobro razumljiva i analizirana te da su potrebni alati dostupni. Prihvatljivo je za veliki broj parametara ako vremenska ograničenja nisu prekratka, a glavno što utječe na kašnjenje pretplate je vrijeme pripravnosti servisa, njegovo inicijalno vrijeme čekanja i kašnjenje servisnog ciklusa. Kašnjenje u mreži može biti jako značajno te parametri moraju biti na odgovarajući način odabrani. Detaljnija analiza se može pronaći u njihovom radu [20].

4.11. Zahtjevi za alate testiranja i razvoja

Kao prvi korak koji je potrebno poduzeti u arhitekturi vozila je integracija Etherneta s tehnologijama poput CAN, LIN, FlexRay, MOST koje su već dobro poznate i uspostavljene u autoindustriji. Za njih su odavno razvijeni alati, a Ethernet s druge strane ima svoje standardne alate koji ne podržavaju fizički sloj i protokole automobilske industrije pa su potrebni novi alati za razvoj i testiranje što zahtijeva dodatna ulaganja među testnim inženjerima i programerima. Arhitektura mrežnog sustava mora biti u potpunosti jasna jer se više ne implementira sve kao sustav sabirnica nego kao mreža sa više *Full-duplex* veza. Da bi se istovremeno analizirao sav podatkovni promet, pristup svim čvorovima mora biti sinkroniziran, a inženjeri se moraju nositi s problemima i strategijama filtriranja i obrade ogromnih količina podataka kojih je svakim danom sve više u modernim automobilima. Gigabitne brzine prijenosa su već početkom SOME/IP bile na listama zahtjeva. Mora se uzeti u obzir i izbjegavanje utjecaja mjerenja na sustav što kod sabirnica nema [1]. U nastavku su autori „*Neue Werkzeuge für Automotive Ethernet*“ [1] predstavili različite postavke u mjerenjima te u svrhu analize i mjerenja su istaknuli željene načine za smanjenje loših utjecaja te načina testiranja SOME/IP-a.

Jedan od načina analize Ethernet mreže je korištenje dodatnog priključka (engl. *port*) na prekidačima/mrežnim priključcima (engl. *switches*) u sustavu. Taj dodatni prekidač nazivamo *Monitorport*, a sustav ovakve analize prometa *Mirroring*[14]. Svi paketi koji prođu preko priključka

[14] *Port mirroring* se koristi za pregled paketa na *port*-u drugačijem od onog na koji su stigli, odnosno kopiranje paketa s jednog *port*-a na drugi. Može se koristiti za nadzor ulaznog ili izlaznog prometa na jednom ili više sučelja te služi mrežnim inženjerima i administratorima za nadzor i uklanjanje greški. Izvor:

prosljeđuju se na ovaj *port* i na taj način se ostvaruje pristup paketima podataka, a oni nemaju međusobni odnos jedan prema drugome, odnosno nema vremenske oznake što je otežavajuća okolnost u testiranju. Osim toga, na *Monitorport* dolaze samo ispravni paketi pa ne postoji uvid u pogrešne i zaostale. Ako na postojećem mrežnom prekidaču ne postoji dodatni *port* može ga se dodati no on uzrokuje kašnjenje u cijeloj prijenosnoj mreži. U mrežama koje su sinkronizirane protokolom (poput AVB) ovakvo dinamičko kašnjenje će ometati vremensku sinkronizaciju u prijenosu audio i video paketa i za ta mjerenja se koriste uobičajeni alati i sklopke s tog područja. Sa širokopojasnim mrežama poput *BroadR-Reach*-a standardnim u automobilima, neophodna je konverzija medija na standardni *Ethernet* (IEEE 802.3). Poželjno je da se mreža nadzire metodom koja je što više transparentna. Fokus je najviše na analizi i filtriranju neispravnih i zakašnjelih podataka na koje sustav ne može utjecati. To se postiže takozvanom testnom pristupnom točkom (*TAP-Test Access Point*) koja se nalazi na fizičkom sloju kroz koju podatak prođe i ode dalje. Latencija osim što je kratka, može biti i konstantna što je prednost kod analize audio i video sustava. Druga metoda transparentne analize i testiranja je upotreba *switcha* s vremenskom sinkronizacijom. Latencija koja nastaje prilikom prosljeđivanja paketa nadoknađuje se AVB protokolom.

Bez obzira na odabranu metodu, vremenske oznake moraju biti točne da bi analiza podataka bila što točnija i jasnija. Paketi se uzimaju što je bliže moguće fizičkom sloju, a vremenske oznake se moraju sinkronizirati s ostalim sučeljima jer analiza mreže koja se provodi često ne obuhvaća samo jednu Ethernet mrežu. Osim toga, u obzir se moraju uzeti i neaktivna sučelja. Pod testiranje se podrazumijeva i namjerno slanje neispravnih paketa tokom normalne komunikacije s čvorovima. Ti podaci se dobiju izravno iz aplikacije za testiranje, primjerice Vector-ovih alata poput CANoe.IP koji nudi razne mogućnosti, između ostalog generiranje paketa koji izravno na sučelju opterećuju sabirnicu [1].

Posebno kod razvoja pojedinih upravljačkih jedinica, simulacija ostalih sabirnica je prilagodljiva opcija testiranja bilo kojeg scenarija prije nego se ti upravljački uređaji integriraju u stvarnu mrežu i okruženje. Ethernet mreža predstavlja različite zahtjeve na hardver i softver. Kako bi se izbjegla promjena sučelja u različitim postavkama mjerenja, sučelje mora biti fleksibilno kao TAP, konverter ili prekidač s dodatnom funkcionalnošću. Slijedeća svojstva se navode kao poželjna [1, str.5]:

https://en.wikipedia.org/wiki/Port_mirroring

- U najjednostavnijem slučaju kada se sučelje koristi kao TAP, sami TAP može prouzrokovati minimalno i točno određeno kašnjenje
- Sučelje mora biti u mogućnosti vršiti konverzije između svih uobičajenih medija poput BroadR-Reach, Fast Ethernet, Gigabit Ethernet te RTPGE. Time se eliminira potreba za vanjskim konverterima
- Za testnu vožnju mora biti ugrađeno sučelje i ne smije prekidati ili ometati mrežu kada se ne koristi (engl. *Standalone Mode*)
- Od velike važnosti su generatori paketa na softverskoj ili hardverskoj razini jer proces razvoja zahtjeva kontroliranu analizu i simulaciju
- Zajedno sa softverom za simulaciju hardversko sučelje mora omogućiti stvarni medijski pristup jednom ili više čvorova virtualne mreže
- Alat za analizu i simulaciju mora biti u mogućnosti analizirati i manipulirati podacima na svim slojevima i razinama protokola
- Da bi se podržale heterogene mreže, sučelje mora biti u mogućnosti sinkronizirati se sa svim uobičajenim sustavima sabirnica

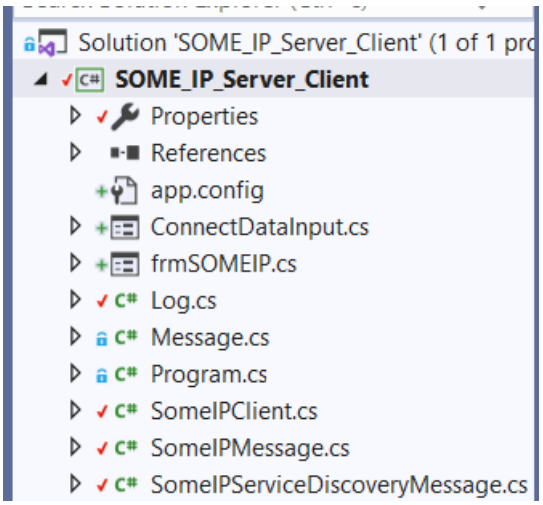
Iz navedenih razloga upotreba moćnih alata za analizu komunikacije u vezi s vanjskim konverterima medija je često nedovoljna. Ti zahtjevi se mogu ispuniti samo sa specijaliziranim hardverom koji je usko povezan sa softverom za analizu i simulaciju. Kombinacija koja se koristi u praksi je Ethernet/CAN sučelje VN 5610 tvrtke Vector zajedno s razvojnim alatom CANoe.IP. Ovo rješenje je najčešće i koriste ga mnogi proizvođači i dobavljači vozila [1, str.5].

5. RAZVOJ APLIKACIJE

U nastavku je opisano kreiranje aplikacije koja prenosi poruke putem SOME/IP protokola. Radi se o istoj, jednoj aplikaciji koja ima dvosmjernu komunikaciju, odnosno pokrećemo istu aplikaciju dva puta kako bi te dvije instance razgovarale međusobno. Izrađena je u objektno-orijentiranom programskom jeziku C#, *Visual Studio* 2019 s *.NET Frameworkom* 4.8 koji podržava metode poputasinkronog zaprimanja UDP paketa. Iako isti *port* jedne aplikacije može i slušati i slati poruke, dvije aplikacije ne mogu imati isti *port* i iz toga razloga prije spajanja i slanja, odnosno zaprimanja poruke, moramo odrediti koji je *port* trenutno pokrenute aplikacije i *port* na koji šaljemo poruku (*engl. EndPoint*). Druga pokrenuta instanca mora imati obrnuto upisane podatke. Aplikacija podržava slanje teksta, zvuka i slike u onoj veličini koju UDP paket može prenijeti i to su paketi s kojima su moguće sve vrste komunikacije definirane specifikacijom. Kreiran je događaj koji korisnik sam može inicirati poput otvaranja vrata na automobilu te funkcija s udaljenim pozivom koja imitira paljenje i gašenje svjetla u kabini automobila. Aplikacija podržava sve metode, načine komunikacije i osnovna pravila kojih se SOME/IP protokol treba držati. U nastavku je opisana izrada, svrha i funkcija pojedine klase te rad i izgled same aplikacije.

5.1. Razvoj aplikacije

Aplikacija se sastoji od dvije forme i šest kreiranih klasa gdje svaka definira sadržaj i izgled poruke, odnosno funkcionalnost SOME/IP protokola ili pomaže u njenom praćenju, definiciji i nadzoru kroz aplikaciju. Kreirana je struktura kompletne poruke koja se može naknadno dorađivati detaljima iz specifikacije. Klase su prikazane na slijedećoj slici (Slika 22.):

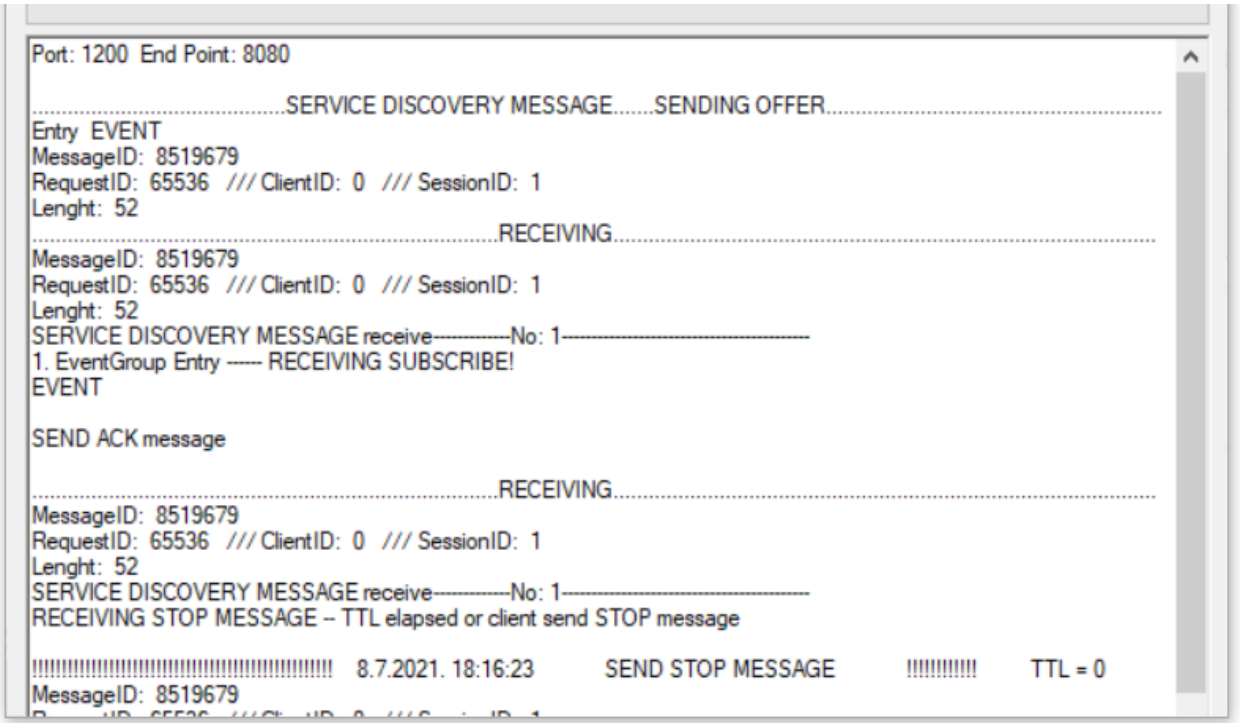


Slika 22. Klase u SOME_IP_Server_Client aplikaciji

5.2. Klasa Log

Klasa Log.cs služi za ispisivanje poruka na samom prozorčiću aplikacije i detalja pristigle i poslane poruke tijekom njenog prijenosa i rada aplikacije. Te poruke olakšavaju praćenje prijenosa podataka jer je u njima vidljivo na koji *port* se šalje i koji je *port* same instance koja šalje poruku te svi detalji važni u identifikaciji poruke. Prikazani su osnovni podaci zaglavlja poruke, njene duljine i obavijesti. Instanca koja šalje poruku će tijekom slanja ispisati obavijest da se poruka šalje i kakve je vrste ta poruka, isto kao i instanca koja „sluša" na drugoj strani nakon što zaprimi poruku.

Sastoji seod dvije funkcije, MessageSent koja prima nekakav tekst i pamti ga, te Log_Txt koji vraća i ispisuje sav taj tekst. Svaka linija teksta koja se ispisuje u aplikaciji na formi je pozvana funkcija MessageSent u određenom dijelu programskog koda i predana joj je vrijednost za ispis. To ju razlikuje od Log_Txt funkcije koja se samo jednom poziva kako bi sve to ispisala. Kako se aplikacija izvršava u više niti, za ispis se u ponekim dijelovima morao koristiti delegat funkcije. Izgled ispisanog loga na formi je prikazan na slijedećoj slici (Slika 23.):



Slika 23. Prikaz ispisa logova na formi aplikacije

5.3. Klasa Message

Klasa Message.cs je gruba, osnovna i jednostavna struktura samog tereta poruke koju će naslijediti ostale klase koje se bave detaljima SOME/IP poruke. U njoj je definirano polje *byte*-ova *Payloada* što je zajedničko za sve poruke te lista mogućih signala i njena klasa koja se može po potrebi definirati. U ovom radu ne koristimo signale u poruci ali su kreirani zbog mogućeg

proširivanja aplikacije poput *Reserved* polja u zaglavljima poruke.

5.4. Klasa SomeIpClient

Ova klasa sadrži sve potrebno za povezivanje, slanje i primanje UDP paketa. Specifična je što kreira događaj (eng. *Event*) i izvodi se asinkrono, odnosno tijekom rada cijele aplikacije ona u drugoj niti „sluša *port*“ i čeka poruke. Dvije najvažnije varijable koje su potrebne za komunikaciju su *port* i krajnja točka na koju se šalje poruka. Klasa se sastoji od jednog konstruktora, dva svojstva (*engl. Property*) i tri funkcije. Konstruktor je definiran tako da kod samog kreiranja instance te klase moramo dodijeliti *port* i krajnju točku koji se odmah koriste za povezivanje. Te podatke unosimo na formi klikom na tipku „Connect“ gdje se prikazuje novi prozor sa poljima za unos. Programski kod te forme će biti opisan naknadno no glavni njegov cilj je prenijeti vrijednost iz polja u svojstva ove klase i omogućiti povezivanje dvije forme, odnosno slušanje i slanje poruke na određeni *port*. U aplikaciji zbog jednostavnosti nije dorađena mogućnost da različita računala u istoj mreži komuniciraju međusobno pa je fiksno dodijeljena lokalna IP adresa no moguće ju je doraditi listom IPv4 i IPv6 krajnjih točaka gdje bi se spremale razne pristigle adrese i time bilo moguće i *multicast* slanje. Kreiran je događaj ReceiveResultsEvent delegirane funkcije SomeIpClientEventHandler koja prima polje *byte*-ova. Event će na svako zaprimanje poruke reagirati i pozivati funkciju za provjeru i prikaz pristigle poruke. Uz događaj se uvijek kreira funkcija koja „podiže“ tajdogađaj. U ovoj klasi se naziva OnDataReceived. Ona sprječava da se aplikacija raspadne u slučaju neispravnog rukovanja događajem i zaprimanjem poruke. Događaj koji pozivamo nazivamo objavom (*engl. Publisher*), a funkciju koja se poziva svaki put na taj događaj nazivamo pretplatom (*engl. Subscriber*) ali nisu povezani s pojmom pretplatnika i poslužitelja iz SOME/IP specifikacije. Funkcija Received je definirana kao asinkrona i u beskonačnoj petlji prima poruke i prosljeđuje događaju. Slanje poruke se odvija preko metode Send koja prima polje *byte*- ova, kreira privremenog, drugog UDP klijenta i šalje tu poruku. Tom klijentu nije važan njegov *port* pa ga nije potrebno niti definirati za razliku onoga koji prima poruku.

```
2 references
public int EndPointPort
{
    get { return endPointPort; }
    set { endPointPort = value; }
}
1 reference
public async Task Receive()
{
    while (true)
    {

        var ReceiveResults = await Client.ReceiveAsync();
            if (ReceiveResults != null)
            {
                OnDataReceived(ReceiveResults.Buffer);
            }
    }
}

1 reference
public void Send(byte[] SOMEIP_Message)
{
    UdpClient tempClient = new UdpClient();
    tempClient.Send(SOMEIP_Message, SOMEIP_Message.Length, EndPoint);
}

1 reference
private void OnDataReceived(byte[] args)
{
    if (ReceiveResultsEvent != null)
    {
        ReceiveResultsEvent(args);
    }
}
```

Slika 24. Prikaz programskog koda SomeIpClient.cs klase

5.5. Klasa SomeIpMessage

Klasa SomeIpMessage je najvažnija klasa jer definira detaljni izgled cijele poruke, naročito specifičnog zaglavlja SOME/IP-a, a nasljeđuje ju i klasa za *Service Discovery* poruke. Ova klasa se koristi kod obične, klasične poruke sa zaglavljem (eng. *header*) i teretom (eng. *Payload*). Sastoji se od četiri enum strukture, sedam svojstava, jednog konstruktora klase i tri funkcije. Ova klasa određuje i raspoređuje sve byte-ove poslane i pristigle SOME/IP poruke. Iz tog razloga je jako važna veličina svakog polja u zaglavlju, a praćena je specifikacijom SOME/IP poruke. Veličina svake varijable je određena tipom podatka koji određuje koliko točno *byte*-a zauzima određeno polje.

Iza deklariranih varijabli se nalaze enum strukture koje sadrže skup vrijednosti povratnog koda, tipa poruke i ID-a (slika, tekst, event…) koji je jedini tu proizvoljne vrijednosti. Povratni kod i tip poruke su praćeni specifikacijom , a svakoj strukturi je unaprijed određen tip podatka.

U svojstvu SomeIPHeader rastavljamo i sastavljamo sva polja koja pripadaju zaglavlju.

```
byte[] tempSetter;
uint Length;    //32
ushort ServiceID, MethodID, ClientID, SessionID;  //16
byte ProtocolVersion, InterfaceVersion, MessageType, ReturnCode;  //8
20 references
public enum SOMEIP_MessageID ...
22 references
public enum SOMEIP_MessageType : byte
{
    REQUEST = 0x00,
    REQUEST_NO_RETURN = 0x01,
    NOTIFICATION = 0x02,
    RESPONSE = 0x80,
    ERROR = 0x81,
    TP_REQUEST = 0x20,
    TP_REQUEST_NO_RETURN = 0x21,
    TP_NOTIFICATION = 0x22,
    TP_RESPONSE = 0x23,
    TP_ERROR = 0x24
}
10 references
public enum SOMEIP_ReturnCode ...
26 references
public enum Rest...
21 references
public uint MessageID
{
    get
    {
        return MessageID = (uint)(MethodID) << 16 | ServiceID;
    }
    set
    {
        ServiceID = (ushort)value;
        MethodID = (ushort)(value >> 16);

    }
```

Slika 25. Prikaz programskog koda SomeIpMessage.cs klase

MessageID i *RequestID* polja su veličine 32 bita, a sastavljena od druge dvije vrijednosti veličine 16 bita. Iz tog razloga su kreirana dva svojstva koja će pristiglu vrijednost rastaviti kod primitka poruke, odnosno sastaviti kod slanja. To se izvodi pomicanjem bitova u lijevo ili u desno. Na primjeru s gornje slike (Slika 25.) se uzima dodijeljena vrijednost od 32 bita i skraćuje u 16 bita, pričemu uzima prvi dio te vrijednosti i sprema u *ServiceID*. Za *MethodID* se ista ta dodijeljena vrijednost pomiče za 16 jer u ovom slučaju trebamo drugih 16 bitova. Skraćujemo ga i dodjeljujemo *MethodID*. Kod sastavljanja tih dviju vrijednosti, povećavamo polje i pomičemo jednu vrijednost na početak te joj pridodajemo drugu. Isti postupak je u svojstvu *RequestID*

GetLength svojstvo koristimo samo kod ispisa loga, da bi mogli doći do podatka o veličini poruke. FullMessagePayload (slika 26.) svojstvo odvaja zaglavlje od sadržaja poruke koristeći DissectFullPayload metodu. U slučaju slanja poruke sastavlja te dvije vrijednosti u kompletnu poruku te poziva funkciju koja postavlja veličinu same poruke. Konstruktor koristimo kod kreiranja nove instance trenutne klase. Ona se kreira kod svakog novog slanja poruke. Zato podaci koje dajemo objektu prilikom kreiranja određuju o kakvoj se poruci radi i ti podaci su zaglavlje poruke. ProtocolVersion i InterfaceVersion postavljamo na fiksno na jedan (slika 26.).

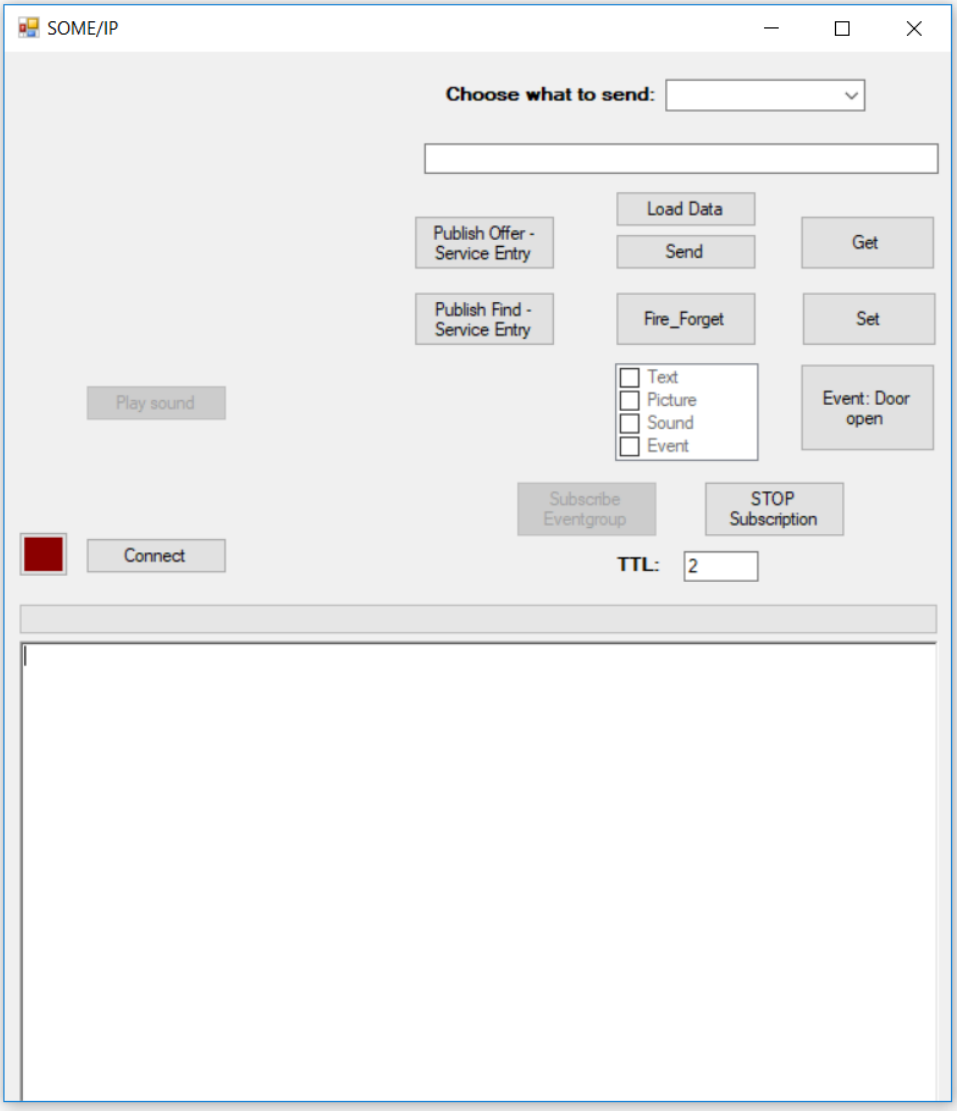
```
153     1 reference
        public byte[] FullMessagePayload
154     {
155         get
156         {
157             SetLength();
158             return SomeIPHeader.Concat(Payload).ToArray();
159         }
160         set
161         {
162             DissectFullPayload(value);
163         }
164     }
165
        4 references
166     public SomeIPMessage(uint Mess_ID, uint Req_ID,byte Mess_Type, byte Ret_Code)
167     {
168         MessageID = Mess_ID;
169         RequestID = Req_ID;
170         MessageType = Mess_Type;
171         ReturnCode = Ret_Code;
172         ProtocolVersion = 1;
173         InterfaceVersion = 1;
174     }
175
        2 references
176     public SomeIPMessage(byte[] Full_MessagePayload)
177     {
178         DissectFullPayload(Full_MessagePayload);
179     }
180
```

Slika 26. Prikaz programskog koda SomeIpMessage.cs klase

5.6. Forma frmSOMEIP

Ovo je osnovna forma aplikacije. Sastoji se od padajućeg izbornika u kojem odabiremo što se šalje, jednog polja za unos teksta kojeg šaljemo ili vrijednosti koju prosljeđujemo pozivom udaljene procedure (eng. *Remote Procedure Call*). U tom polju se prikazuje i primljeni tekst ili ispis putanje slikovne i glazbene datoteke, ovisno o tome što se šalje. Gumb „Load Data“ otvara prozor ukojemu su filtrirane datoteke po njihovim ekstenzijama te odabiremo sliku ili zvuk za slanje. Na gumb „Send“ se poruka šalje i ima sličnu funkciju kao gumb „Get“, no prije slanja moramo uspostaviti komunikaciju i odrediti trenutni i odredišni *port* na gumbu „Connect“. Gumb „Play Sound“ je uvijek onemogućen za klik osim u slučaju da šaljemo ili smo zaprimili zvučnu datoteku. Praznina s lijeve strane forme služi za prikaz slike koja se šalje ili je zaprimljena. Slika će se prikazati prilikom njenog odabira i nestati nakon slanja, te prikazati na formi koja je primila sliku.

Slika 27. Prikaz osnovne forme aplikacije



U polje TTL unosimo vrijeme u sekundama koje označava trajanje jedne poruke. Odnosi se na poruke koje to podržavaju, odnosno *Service Discovery* poruke. Obična poruka odgovora ili obavijesti nema životni vijek i ignorirati će ovo polje. Gumb „Publish Offer“ postavlja zastavicu u vrijednost koju će koristiti funkcija koja ovisno o toj zastavici slaže podatke u polja ulaza za servise i tako označiti da se radi o ponudi. Isto radi i gumb „Publish Find“. Oba gumba nakon odabira omogućavaju izbor vrste servisa u prozorčiću predviđenom za to na sredini forme. Kako specifikacija nalaže, u jednu SD poruku moguće je složiti više ulaza pa je u tom prozorčiću moguće odabrati više poruka koje se nude ili traže. Na istom prozorčiću će se označiti pristigli ponuđeni ili traženi servisi kojima se onda odabirom može upravljati. Gumb „Fire&Forget“ šalje RPC poruku bez odgovora i obavijesti ali je potrebno upisati vrijednosti koje se šalju tim pozivom. „Get“ gumb šalje poruku zahtjeva za određenom vrijednosti polja, a „Set“ šalje istu poruku ali s vrijednosti koju želi postaviti u polje kod poslužitelja.

Ukoliko stigne poruka ponude na čije servise se klijent može pretplatiti omogućiti će se gumb za pretplatu „Subscribe EventGroup“. Ista pretplata ili ponuda i potražnja servisa se može prekinuti klikom na gumb „STOP Subscription“. Svaka pretplata se odvija ciklički odmah nakon slanja potvrdne poruke. Obavijesti stižu svake sekunde dok ne istekne TTL. Pretplata na događaj omogućava simulaciju obavijesti o događaju koji korisnik sam uzrokuje pritiskom na gumb „Event: Door Open“.

Crveni kvadrat na lijevoj strani forme je gumb za ponovno pokretanje aplikacije.

U glavnom dijelu programskog koda te klase su najvažniji dijelovi koji određuju kakve poruke se šalju i kada te što je pristigla poruka i kako ju procesuirati. Klasa ima preko 1300 linija programskog koda. U kodu je puno konvertiranja raznih tipova podataka u byte-ove i obratno kojima aplikacija omogućava prijenos SOME/IP porukom. Izrađene su funkcije koje pretvaraju tekst i zvuk spremnim za slanje u teretu poruke. Na odabir gumba za učitavanjedatoteke otvara se prozor koji filtrira prikazane datoteke ovisno što je izabrano za slanje u padajućem izborniku. Potvrđivanjem izbora i zatvaranjem prozora ispisuje se putanja u polje za tekst na formi te se, ovisno o izboru, prikazuje slika u prostoru predviđenom za to ili se omogućava klik na gumb za reproduciranje zvuka.

Odabirom gumba za povezivanje se otvara nova forma gdje upisujemo trenutni i odredišni *port* te uspostavljamo komunikaciju između dvije instance aplikacije kreirajući objekt klase SomeIPClient. Tu se upisuje prvi log koji se pojavljuje na formi sa informacijama o povezivanju. Pozivamo metodu koja će početi „osluškivati“ pristigle poruke.

Gumb za slanje koristi obrnutu funkciju koja provjerava poruku za slanje i ispisuje njene

osnovne podatke u log na formi. Prosljeđuje kompletnu poruku funkciji za slanje, briše sliku i ispisuje sve zapisane logove do tad na formu aplikacije koja šalje.

Najveća i najvažnija funkcija koja provjerava pristiglu poruku je „checkReceiveMessage“. U ovisnosti o vrijednostima u poljima funkcija procesuira daljnje radnje definirane protokolom. Iz tog razloga postoji mnogo ugniježdenih petlji koja prolaze kroz polja. Suženi prikaz funkcije se nalazi na slijedećoj slici (slika 28.) :

```
public void checkReceiveMessage(byte[] udpPayload)
{
    string messType;
    j = 12;
    uint rnd_ReqID = RandomClientID();
    ImageConverter converter = new ImageConverter();
    SomeIPMessage temp = new SomeIPMessage(udpPayload);
    Log.MessageSent(".....RECEIVING" +
        ".....");
    Log.MessageSent("MessageID:  " + temp.MessageID.ToString());
    Log.MessageSent("RequestID:  " + temp.RequestID.ToString() + "   /// ClientID:  " + temp.Client_ID.ToString() + "   /// SessionID:  " + te
    Log.MessageSent("Lenght:  " + temp.GetLength.ToString());

    if (temp.FullMessagePayload[14] == Convert.ToByte(SomeIPMessage.SOMEIP_MessageType.RESPONSE))
    {
        messType = "Message Type:  RESPONSE 0x80";
        Log.MessageSent(messType.ToString());

        if (temp.MessageID == Convert.ToUInt32(SomeIPMessage.SOMEIP_MessageID.PICTURE))...
        else if (temp.MessageID == Convert.ToUInt32(SomeIPMessage.SOMEIP_MessageID.TEXT))...
        else if (temp.MessageID == Convert.ToUInt32(SomeIPMessage.SOMEIP_MessageID.SOUND))...
        else if (temp.MessageID == Convert.ToUInt32(SomeIPMessage.SOMEIP_MessageID.RPC))...
        else...
    }

    else if (temp.FullMessagePayload[14] == Convert.ToByte(SomeIPMessage.SOMEIP_MessageType.NOTIFICATION))...
    else if (temp.FullMessagePayload[14] == Convert.ToByte(SomeIPMessage.SOMEIP_MessageType.REQUEST))...
    else if (temp.FullMessagePayload[14] == Convert.ToByte(SomeIPMessage.SOMEIP_MessageType.REQUEST_NO_RETURN))...
    else...

    WriteLog();
    ResetCounter();
```

Slika 28. Suženi prikaz programskog koda funkcije „checkReceiveMessage“

Prvo se provjerava polje koje nosi vrijednost tipa poruke. Ako je pristigla poruka odgovor na prije poslanu onda gledamo polje *MessageID* koji govori što je u toj poruci i ovisno o tome prikazuje poruku na formi. Slično je i ako se radi o poruci zahtjeva gdje se onda unutar te petlje na isti način provjerava koji je zahtjev te veličina tereta. Ako je teret prazan radi se o zahtjevu za dohvaćanjem vrijednosti i ta se vrijednost šalje. Ako postoji vrijednost u teretu onda se postavlja tražena vrijednost u polje uz povratni odgovor klijentu. U slučaju da je pristigao zahtjev bez odgovora poziva se procedura i prosljeđuje joj se vrijednost za izvršavanje. Kreirana je jedna moguća procedura za izvršavanje i udaljeno pozivanje koja simulira paljenje svjetla u kabini automobila. SD poruke u tom polju nose vrijednost koja označava obavijest pa je ulaskom u tu petlju potrebno provjeriti je li to SD poruka ili stigla prava obavijest. *Service Discovery* poruka ima fiksnu i jedinstvenu vrijednost u *MessageID* polju po kojoj ju prepoznajemo. Ako se radi o SD poruci prvo provjeravamo TTL vrijeme kako bi znali radi li se o poruci stopiranja pretplate ili

prekida ponude i potražnje određenog servisa. Ukoliko je vrijednost TTL-a pristigle poruke veći od nula onda se petljom provjerava polje koje nosi oznaku je li to ulaz pretplate, ponude, potražnje servisa ili je stigla poruka potvrde pretplate od poslužitelja.

Za svaki servis koji se može ponuditi ili tražiti postoji funkcija koja ima složen ulaz (eng. *Entry*) prilagođen njoj pa je kod slanja takve poruke dovoljno pozvati tu funkciju koja automatski kreira potrebni ulaz. Isto je i za ulaze grupe događaja koji imaju nešto drugačija polja pa su stoga zasebne funkcije (slika 29).

```
2 references
private byte[] PictureEntry()
{
    byte[] service_id = BitConverter.GetBytes((ushort)SomeIPServiceDiscoveryMessage.SOMEIP_ServiceID.PICTURE);
    byte[] instance_id = BitConverter.GetBytes((ushort)SomeIPMessage.Rest.ANY_INSTANCE);

    if (TagForFindOrOffer == 0)
    {
        sdEntry.ServiceEntry = new byte[] { Convert.ToByte(SomeIPServiceDiscoveryMessage.ENTRY_TYPE.OFFER), //Type
                                            0x00, 0x00, 0x00, //No1, No2 options
                                            service_id[0], service_id[1], //service id
                                            instance_id[0], instance_id[1], //instance id
                                            Convert.ToByte(SomeIPMessage.Rest.ANY_MAYOR),   //major version
                                            GetTTLTimeByte()[0], GetTTLTimeByte()[1], GetTTLTimeByte()[2], //TTL
                                            0xFF, 0xFF, 0xFF, 0xFF };   //minor version
    }
    else
    {
        sdEntry.ServiceEntry = new byte[] { Convert.ToByte(SomeIPServiceDiscoveryMessage.ENTRY_TYPE.FIND), //Type
                                            0x00, 0x00, 0x00, //No1, No2 options
                                            service_id[0], service_id[1], //service id
                                            instance_id[0], instance_id[1], //instance id
                                            Convert.ToByte(SomeIPMessage.Rest.ANY_MAYOR),   //major version
                                            GetTTLTimeByte()[0], GetTTLTimeByte()[1], GetTTLTimeByte()[2], //TTL
                                            0xFF, 0xFF, 0xFF, 0xFF };   //minor version
    }

    return sdEntry.ServiceEntry;

}

2 references
private byte[] SoundEntry()...

2 references
private byte[] EventEntry()...
```

Slika 29. Prikaz funkcije za kreiranje servisnog ulaza za poruku koja prenosi sliku

U ovom radu nisu sva polja od važnosti pa se njihove vrijednosti ne dodjeljuju dinamički nego su fiksno postavljena. Radi se o verzijama sučelja i opcijama izvršavanja polja u *OptionsArray* dijelu poruke koji se ne koriste i nisu potrebna ako se aplikacija pokreće samo na jednom računalu. Slične funkcije su i za slanje poruke prekida i potvrde. Razlika je što te dvije funkcije kopiraju pristiglu poruku zamjenjujući određena polja. Funkcija za prekid ponude ili stopiranje pretplate postojeću TTL vrijednost zamjenjuje sa nulama pa se kod provjere pristigle poruke uvijek prvo gleda ta vrijednost, a funkcija za slanje potvrde mijenja vrijednost polja koje

označava pretplatu u vrijednost koja označava potvrdu zahtjeva za tom pretplatom. Na taj način se klijentu vraća identična SD poruka koja ukazuje za koje to servise prestaje pretplata ili koji su to servisi prihvaćeni za pretplatu.

Funkcija „offer_OR_find“ slaže ulaze u jednu SD poruku prije slanja ovisno o zastavici postavljenoj klikom na gumb ponude ili gumb potražnje servisa. U funkciji se provjeravaju označena polja na formi. Takva kreirana poruka se šalje na gumb „Send“.

Obične SOME/IP poruke koje zahtijevaju ili prenose neku vrijednost se prije slanja kreiraju u „CheckSendMessage“ funkciji. Takve poruke imaju samo zaglavlje i teret pa ne koriste funkcije kreiranja ulaza. Imaju zastavicu koja određuje radi li se o *get* ili *set* metodi i u ovisnosti o tome kreiraju teret.

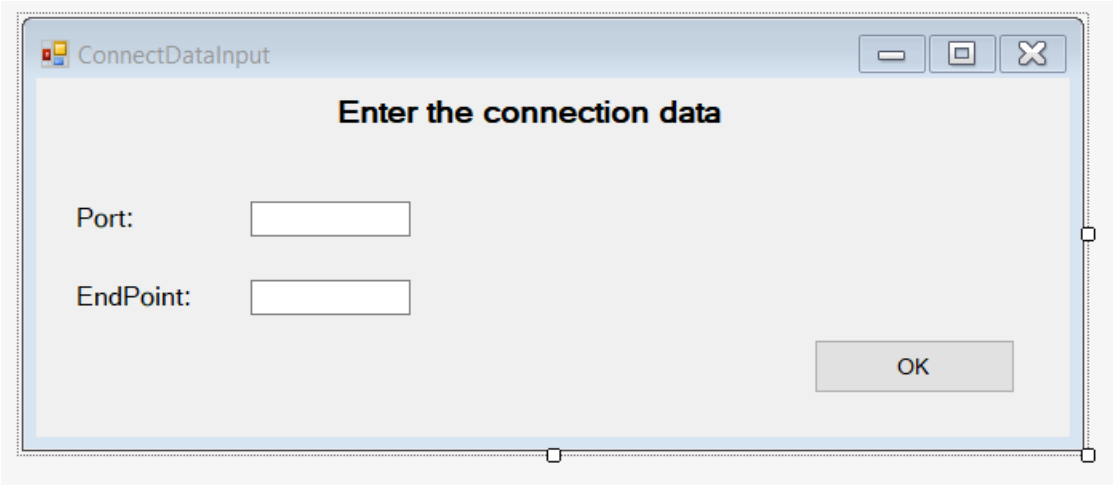
U aplikaciji se koriste dva tajmera. Jedan odbrojava vrijeme pretplate i pokreće se kod slanja cikličkih poruka kako bi poslužitelj slao poruke u vremenu određenom TTL poljem pristigle poruke. Drugi se pokreće kod ponude ili potražnje servise i broji vrijeme trajanja te ponude. Isti tajmer se koristi i pokreće na gumb pretplate i tako odbrojava vrijeme koje je klijent odredio za pretplatu. U slučaju isteka vremena se pokreće događaj za zaustavljanje i prekid primitka ili slanja poruka, resetiranje svih brojača i zastavica te slanje poruke o prekidu.

Kada stigne poruka ponude na prozorčiću ponuđenih servisa budu označeni oni koje poslužitelj nudi. Klijent tako može ukloniti servis koji ne želi i pretplatiti se samo na one koje je tražio ili koji mu trebaju. Funkcija „SubscribeOrStopSubscribe“ slaže SD poruku sa ulazima za pretplatu i šalje ju nazad poslužitelju.

U klasi postoji nekoliko pomoćnih funkcija za pretvorbu tipa podatka, čitanja određenih vrijednosti iz polja i funkcija koja nasumično kreira *ClientID*, spaja ga sa *SessionID* i vraća kao *RequestID* kako se oznake klijenta ne bi ponavljale kod slanja različitih poruka i zahtjeva. *SessionID* se povećava inkrementalno kod slanja svakog novog zahtjeva. Kod SD poruka se fiksno postavlja *ClientID* u nulu jer je tako određeno specifikacijom.

5.7. Forma ConnectDataInput

Forma ConnectDataInput je pomoćna forma na kojoj se ispunjavaju podaci potrebni za komunikaciju dvije instance. Prikazuje se odabirom gumba „Connect“ na glavnoj formi. Nakon ispunjavanja *port*-a i krajnje točke te potvrde i zatvaranja forme na gumb „OK“ aplikacija se povezuje. Podaci su vidljivi u prozoru glavne forme gdje se ispisuju log-ovi.



Slika 30. Prikaz pomoćne forme ConnectDataInput

5.5. Klasa SomeIPServiceDiscoveryMessage

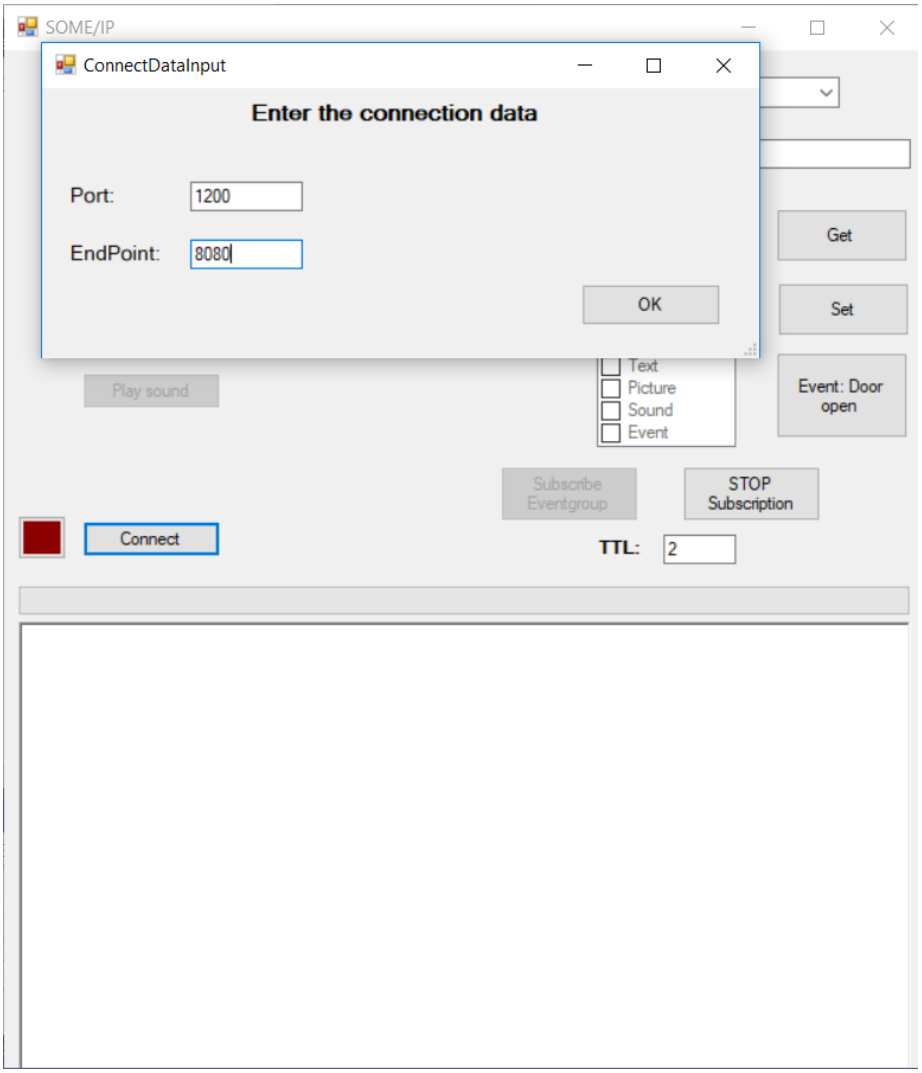
Ova klasa nasljeđuje klasu SomeIPMessage i proširuje poljima kako je navedeno u specifikaciji. Unutar te klase je kreirana klasa ServiceDiscoveryEntry koja se bavi ulazima Service Discovery. Jako je slična SomeIPMessage klasi jer ima svojstva i konstruktore koji obavljaju isto poput onih u SomeIPMessage klasi. U njoj je definiran izgled cijele Service Discovery poruke i ulaza za servise i grupe događaja.

FUNKCIONALNOST APLIKACIJE

U ovom poglavlju je prikazan način rada aplikacije i njene mogućnosti. Kako je do sada navedeno, aplikacija može slati tekst, sliku ili zvuk u veličini UDP paketa. Moguće je simulirati sve načine komunikacije koji su navedeni u specifikaciji protokola. Primjer svakog slanja i primitka poruke s dvije instance aplikacije će biti dalje u radu prikazan slikom.

Prvo moramo pokrenuti aplikaciju dva puta da dobijemo 2 pokrenute instance. Prije nego se poruka izabere i pošalje, potrebno je povezati se, odnosno odrediti *port* instance i krajnju točku na koju šaljemo paket. Ta dva podatka unosimo na pomoćnoj formi koja se prikazuje odabirom gumba „Connect“. To činimo za svaku instancu. Ako želimo da one međusobno komuniciraju onda ih unosimo na način da je port jedne instance krajnja točke druge, odnosno, ta dva podatka moraju biti upisana obrnutim redoslijedom jer ulaz jedne instance je krajnja točka druge instance.

Slika 31. Prikaz forme prilikom unosa podataka za spajanje

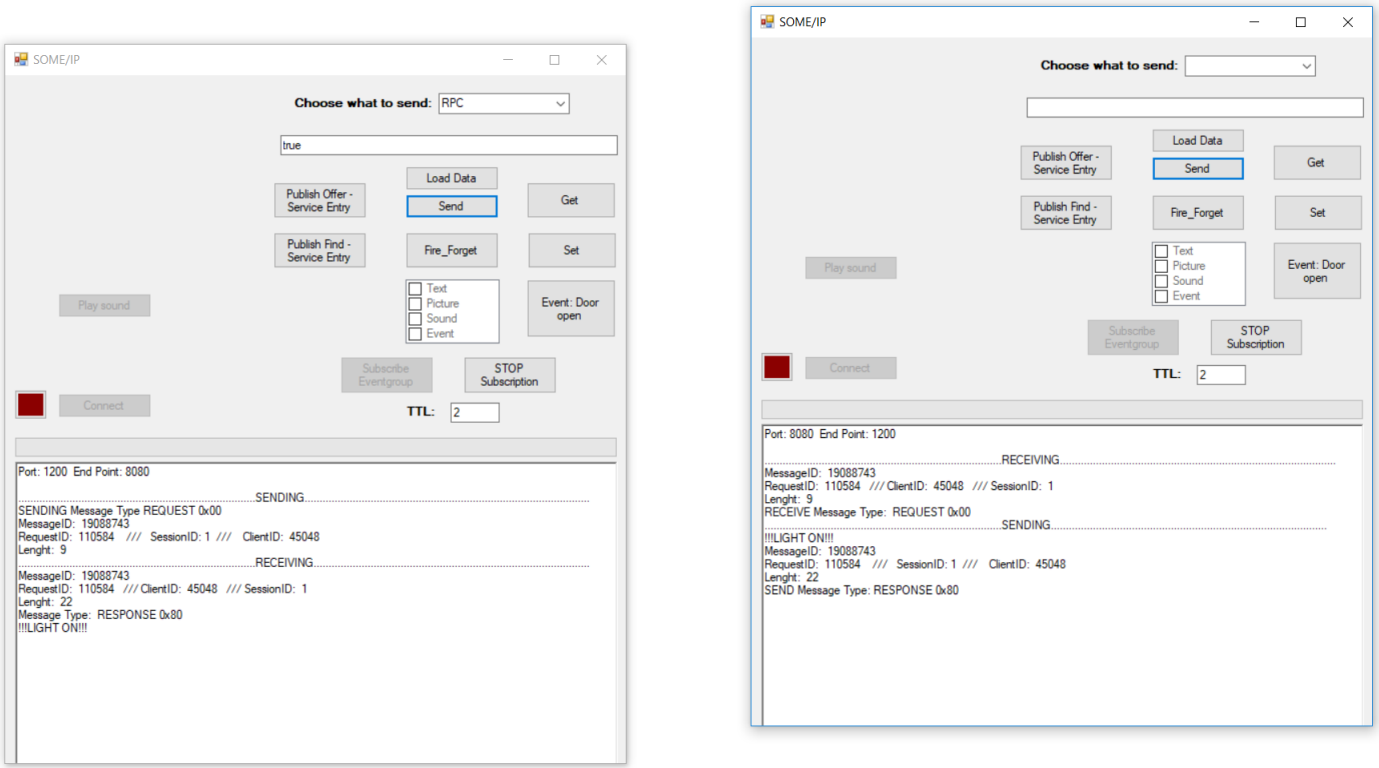


Potvrdom unosa aplikacija je povezana, a pomoćna forma se zatvara. U bijelom prozoru

na dnu forme će se ispisati informacija o portu i krajnjoj točki. Nakon tog koraka odabiremo što ćemo slati.

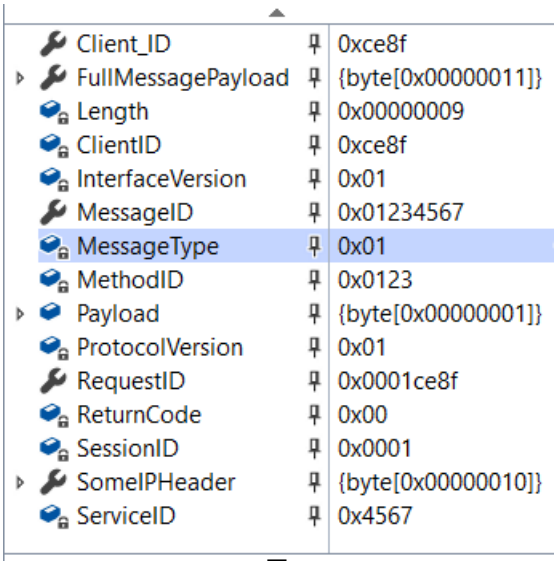
Request&Response način komunikacije možemo simulirati na više načina jer *get* i *set* zahtjevi su takva vrsta komunikacije gdje se upućuje zahtjev i povratno dobije odgovor. No kako se radi o poljima o kojima će biti dalje u radu pojašnjeno simulirati ćemo ovu metodu na pozivu udaljene funkcije poslužitelju koja u odgovoru mora vratiti vrijednost. Funkcija je nazvana RPC i izabiremo ju u padajućem izborniku prije slanja. Kako je već spomenuto u prethodnom poglavlju, funkcija simulira paljenje svijetla u kabini automobila. Tu funkciju treba izvršiti poslužitelj i vratiti rezultat nazad klijentu. U polje za unos teksta unosimo vrijednost „true“ ako želimo da poslužitelj upali svijetlo, odnosno „false“ ako ga želimo ugasiti. Na taj način možemo vidjeti komunikaciju porukama između dva ECU-a u automobilu u slučaju kad se primjerice otvore vrata na automobilu pa jedinica koja time upravlja šalje poruku jedinici koja pali svijetlo.

Slika 32. Simulacija Request/Response komunikacije pozivom RPC funkcije



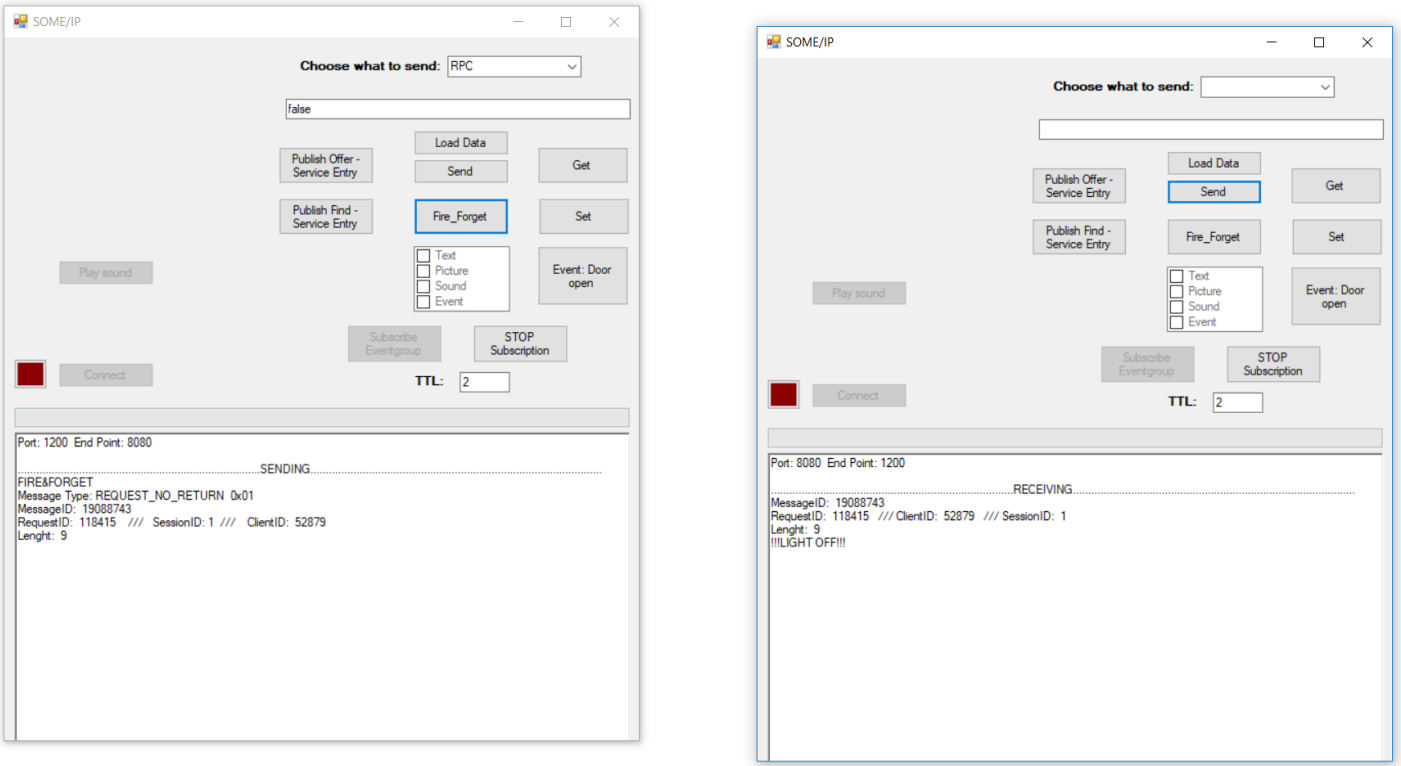
U ispisu u donjem prozoru vidimo da je klijent poslao zahtjev za porukom određenog ID-a što je u ovom slučaju RPC funkcija. RequestID definira pošiljatelj i mora biti isti i jedinstven u svakoj poruci jednog zahtjeva između klijenta i poslužitelja. Ako klijent pošalje novi zahtjev, imati će drugačiji RequestID, odnosno broj sesije će biti uvećan za jedan. Poslužitelju ta polja nisu toliko važna pa on cijeli RequestID kopira i vrati u poruci odgovora.

Fire&Forget je poruka zahtjeva bez odgovora. Možemo pozvati istu funkciju ali će ovaj put polje tipa poruke biti postavljeno u vrijednost koja poslužitelju označava da na tu poruku ne mora odgovoriti. Vrijednosti polja detaljno vidimo u programskom kodu. Zbog lakšeg pregleda i praćenja se ne ispisuju sve vrijednosti u prozor za ispis logova. Na slijedećoj slici (slika 33.) možemo vidjeti polja poruke u trenutku slanja zahtjeva.



Slika 33. Vrijednosti u poruci u trenutku slanja Fire&Forget zahtjeva

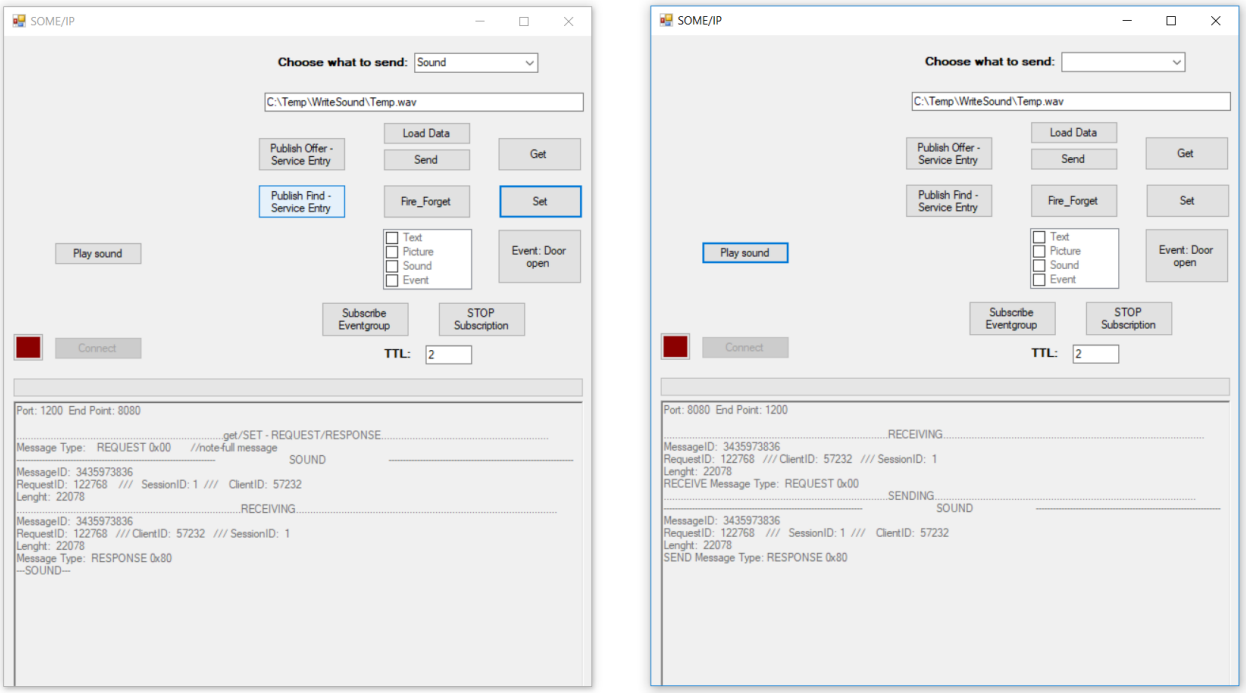
Prikaz *Fire&Forget* zahtjeva u aplikaciji se vidi na slici 34. U specifikaciji se sve vrijednosti prikazuju u heksadecimalnom zapisu pa je takav prikaz podešen i u aplikaciji. U logovima se ispisuju dekadske vrijednosti. Polja na koje treba više obratiti pozornost je veličina zaglavlja koja iznosi točno 16 byte-a. FullMessagePayload pokazuje veličinu kompletne poruke, u ovom slučaju 17 byte-ova. U teretu se prenosi boolean vrijednost i on je velik 1 byte. Length je polje unutar zaglavlja koje se po pravilima protokola računa od tog polja pa do kraja poruke. Length polje nosi zapis duljine 9 byte-ova i ako mu pridružimo njegovu duljinu te duljinu MessageID prije njega koji su veliki po 4 byte-a dobijemo svih 17 byte-ova poruke.



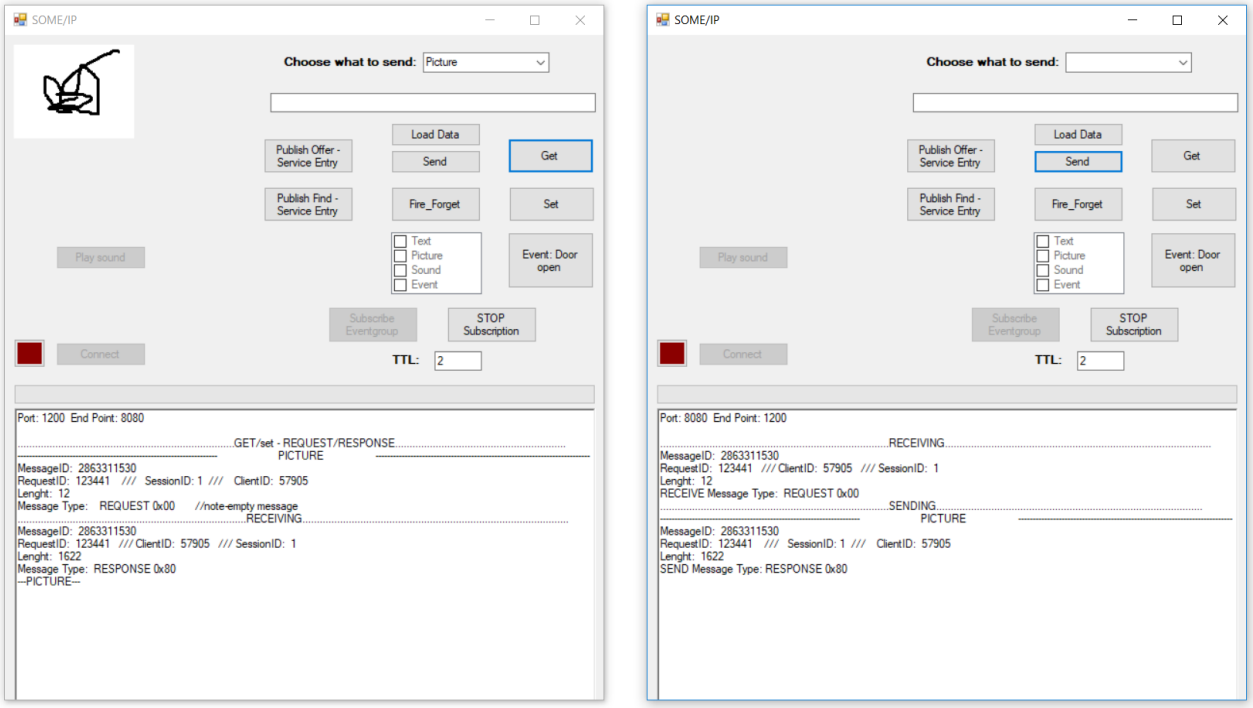
Slika 34. Prikaz Fire&Forget zahtjeva u aplikaciji

Get i *Set* metode koje rade s poljima su vrsta *Request&Response* komunikacije. Specifično je što poslužitelj na tu poruku mora odgovoriti barem greškom ili praznom porukom. Kada klijent upućuje zahtjev (slika 36.) za dohvaćanjem vrijednosti polja, upućuje *MessageID* one vrijednosti koju želi, a tip poruke postavlja u 0x00. Teret ostaje prazan. Na taj način će mu poslužitelj odgovoriti tipom poruke 0x80 što označava odgovor i u teretu poslati traženu vrijednost. Suprotno od toga je zahtjev za postavljanjem (slika 35.), odnosno promjenom vrijednosti polja kod poslužitelja. Sva polja su ista kao kod prethodne poruke ali je u teretu podatak koji želi postaviti u polje. Dakle, kod *Set* zahtjeva je teret pun, a kod *Get* prazan i to se može vidjeti u ispisu. Ukoliko klijent uspije promijeniti zapis u polju poslužitelj ga inicijalno vraća u poruci odgovora. Navedeno je najlakše pratiti duljinom poruke. Na slici 36. se vidi da je duljina poruke 12 byte-ova kod zahtjeva i 1622 byte-a u povratnoj poruci jer je poslužitelj poslao sliku koju je klijent tražio. Isto, na slici 35. se vidi upućen zahtjev od klijenta za promjenom vrijednosti polja zvuka. Nakon uspješne promjene na obje forme je omogućena tipka za reprodukciju zvuka, a duljina poruke ostaje nepromijenjena.

U padajućem izborniku biramo što želimo dohvatiti ili poslati poslužitelju te ovisno o tome odaberemo gumb „Get“ ili „Set“.



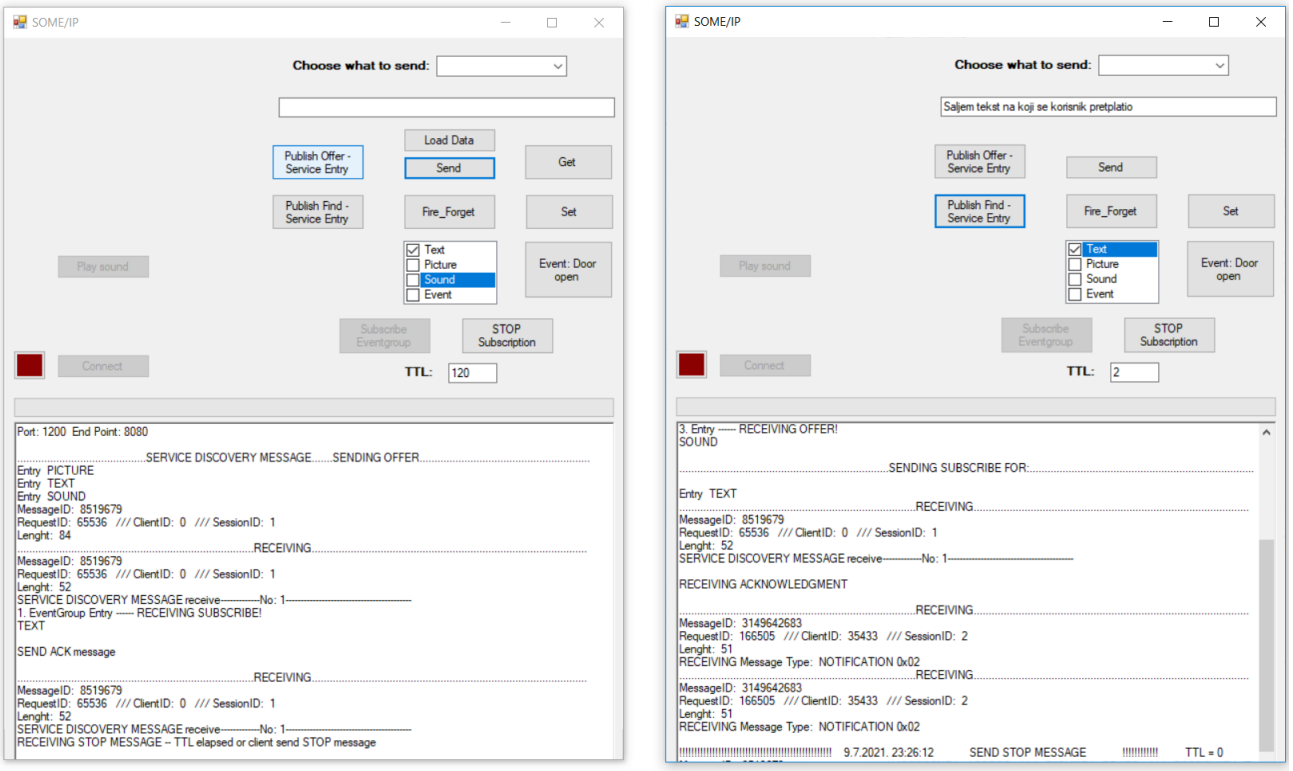
Slika 35. Prikaz upućenog zahtjeva za promjenom zvuka na strani poslužitelja



Slika 36. Prikaz upućenog zahtjeva za dohvaćanjem slike na strani poslužitelja

Da bi se klijent mogao pretplatiti na polje ili događaj, oni moraju biti ponuđeni SD porukom. *Service Discovery* poruka nakon zaglavlja slaže ulaze servisa koje nudi. Svaki ulaz je velik 16 byte-ova.. U aplikaciji je moguće nuditi događaj, sliku, tekst, zvuk i događaj. Za ponudu servisa je potrebno prvo kliknuti na gumb „Publish Offer“ zatim u prozorčiću na sredini aplikacije odabrati servise koji će se nuditi. Na strani klijenta će stići poruka ponude i označiti se ponuđeni servisi u istom prozorčiću te ispisati u logovima. Klijent može odbiti pretplatu ili izabrati hoće li se pretplatiti na neki od servisa. Na slici 37. je

prikazana ponuda nekoliko servisa, tekst, sliku i zvuk te pretplata samo na tekst. U TTL polje je potrebno unijeti vrijeme pretplate, odnosno prilikom slanja ponude određujemo vrijeme trajanja te ponude. Po pravilima protokola, kad vrijeme istekne druga strana je obavezna znati za to. Šalje se poruka prestanka pretplate sa TTL vrijednosti nula za iste servise za koje je do tada klijent bio pretplaćen. Isto je za ponudu i potražnju servisa.



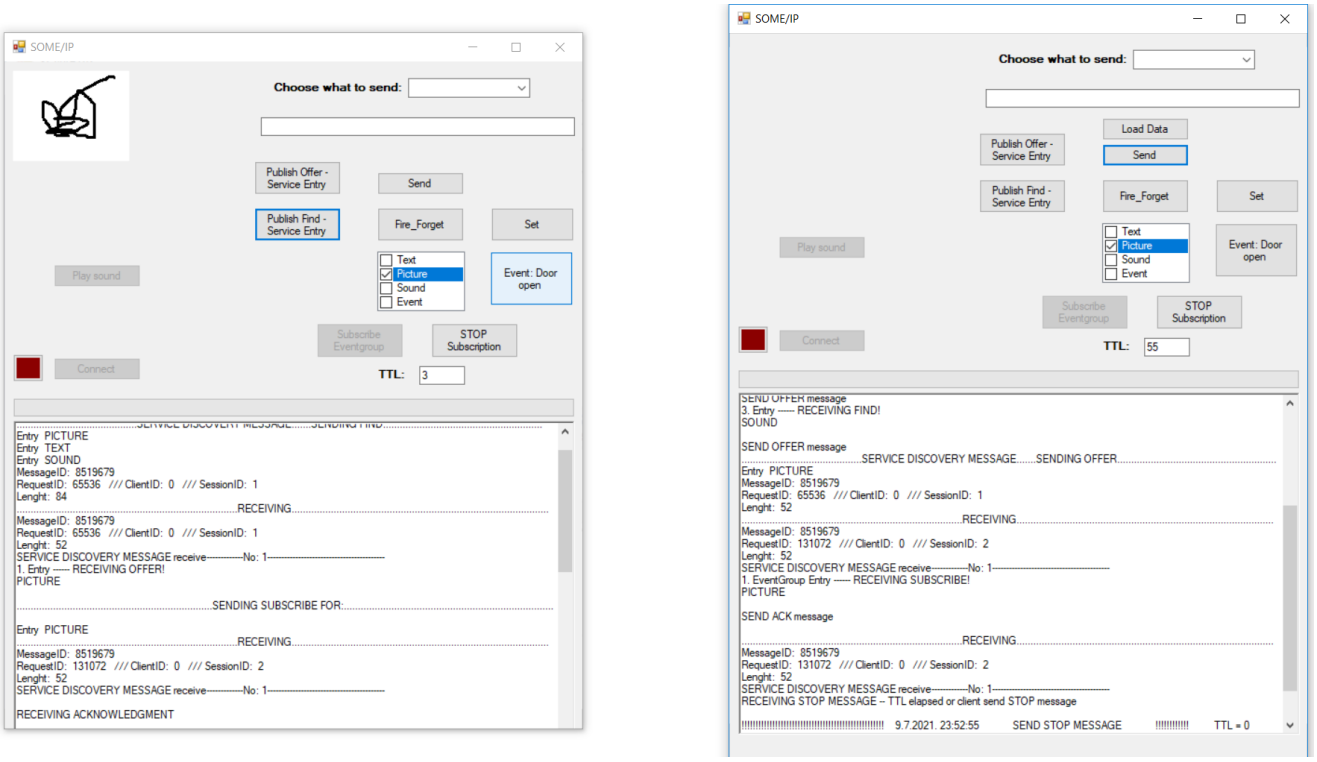
Slika 37. Prikaz ponude servisa i pretplate na jedan od njih

U ispisu logova je vidljivo da je poslužitelj poslao ponudu sa tri servisa u trajanju od 120 sekundi, a klijent istu tu ponudu zaprimio. Izabran je jedan servis, pretplata na tekst. Kako se radi o polju, poslužitelj će ciklički slati vrijednost tog polja svake sekunde. Klijent se pretplatio na 2 sekunde i u logovima su vidljive dvije pristigle obavijesti. Vrijednost je prikazana u polju za unos teksta na vrhu forme. Nakon što su prošle dvije sekunde klijent je poslao poruku o prestanku pretplate što je poslužitelj zaprimio i prestao slati obavijesti. U protokolu se ove obavijesti nazivaju obavijestima polja. Postoje i obavijesti događaja koje će se spomenuti niže u tekstu.

Service Discovery porukama se mogu i tražiti servisi. Ako je klijentu potrebna informacija o temperaturi motora u automobilu on može poslati *multicast* poruku tražeći taj servis. Poruka je ista kao i poruka ponude samo se vrijednost tipa ulaza mijenja. Po pravilima, na poruku potražnje servisa se može odgovoriti jedino tipom ponude servisa i klijent se onda na to može pretplatiti. U praksi bi to za spomenuti primjer bilo kad bi kontrolna ploča tražila informaciju o temperaturi motora. ECU motora može ponuditi tu informaciju svake sekunde sve dok motor radi. ECU koji upravlja hlađenjem nudi tu informaciju samo u slučaju događaja pregrijavanja motora. Kontrolna ploča bira informaciju u skladu sa zahtjevima. U aplikaciji

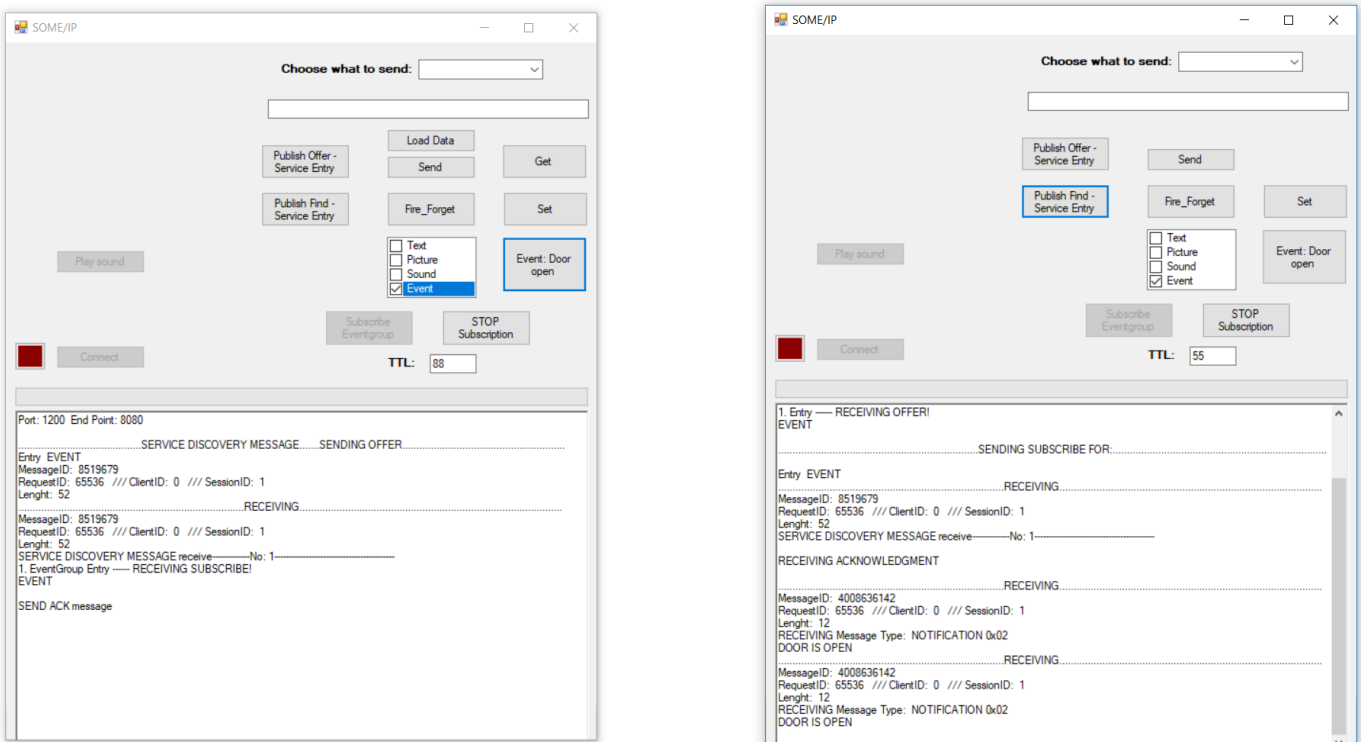
je to napravljeno na način da kada stigne poruka potražnje u logovima se ispiše obavijest o slanju ponude. Moguće je vratiti servise koje klijent nije tražio no on se onda neće pretplatiti. Poruka potražnje servisa dolazi na isti način kao i ponuda i odabirom u prozorčiću servisa poslužitelj vraća ponudu na koju se onda klijent može pretplatiti. Primjer takve komunikacije je prikazan slikom 38.

Slika 38. Prikaz potražnje, ponude i pretplate na servis



Prikazom u logu je vidljivo da klijent potražuje tri servisa o kojima je poslužitelj obaviješten. Poslužitelj vraća ponudu samo na jedan servis, sliku. Kako je taj servis jedan od traženih klijent se pretplaćuje 3 sekunde. Poslužitelj je dužan poslati poruku potvrde za svaki servis za koji je prihvatio pretplatu i u logovima to vidimo. Ta poruka je važna jer obavijesti klijenta da ne mora dalje tražiti ili prihvatiti drugu ponudu.

Slanje obavijesti o događaju je simulacija obavijesti o otvaranju vrata u automobilu. Takva komunikacija je prikazana na slici 39. Klijent se može pretplatiti na događaj, a obavijest mu stiže samo kad se događaj dogodi. U aplikaciji je taj događaj gumb koji šalje obavijest kad se klikne na njega. Prije toga je potrebno ponuditi servis klijentu na način kako je prethodno opisana ponuda servisa.



Slika 39. Poruka obavijesti događaja

6. ZAKLJUČAK

SOME/IP protokol olakšava dodavanje novih funkcija elektroničkom sustavu što je novim, naročito autonomnim vozilima od velike važnosti. Iz dana u dan se stvaraju nove potrebe za razvoj sustava u automobilima te komunikacijom između njih. Uslužno orijentirana arhitektura čini cijelu mrežu vozila dinamičnom i fleksibilnom, a protokol se može implementirati i na ugradbene sustave bez OS-a. Od nastanka do danas SOME/IP se promijenio i razvio nove funkcionalnosti koje ga čine ovakvim kakav je predstavljen u ovom radu.

U aplikaciji izrađenoj kroz ovaj rad je omogućen način svih vrsta komunikacije unutar protokola koje su specificirane od strane AUTOSAR-a. Najvažniji dijelovi protokola su obuhvaćeni uključujući točan izgled poruke, raspored i redoslijed polja unutar nje. Cijeli protokol je implementiran u jednoj aplikaciji čije instance mogu međusobno razmjenjivati poruke. Aplikacija ima sve temelje i osnove za doradu i daljnje razvijanje da bi se uključile sve mogućnosti koje protokol nudi.

SOME/IP protokol ima potencijal promijeniti način razvoja softvera za automobilsku industriju i njegovu integraciju u ECU. Potražnjom za sve većom širinom pojasa i velikim brzinama prijenosa, automobilska industrija počinje uviđati prednosti korištenja Etherneta. Konkurenti su skupi, zahtijevaju vlasničko licenciranje, slabije su istraženi te podložni teškim i skupim kabelima. Njegova ključna prednost je jednostavnost i fleksibilnost. Ojačanjem kibernetičke sigurnosti Ethernet bi mogao biti budućnost V2X povezanosti.

7. LITERATURA

[1] H. Werner Schaal, M. Schwedt, Neue Werkzeuge für Automotive Ethernet, HANSER automotive 03-04/2013, https://www.hanser-automotive.de/zeitschrift/archiv/artikel/flexible-interfaces-und-softwarewerkzeuge-fuer-die-steuergeraeteentwicklung-2478444.html?search.highlight=some&cc.dlstate=true&d=1577049574880

[2] GENIVI, SOME-IP Intro, http://at.projects.genivi.org/

[3] L. Völker, SOME/IP – Die Middleware für Ethernet-basierte Kommunikation, HANSER automotive Networks/2013, https://www.hanser-automotive.de/zeitschrift/archiv/artikel/someip-die-middleware-fuer-ethernet-basierte-kommunikation-676160.html?search.highlight=some

[4] R. Stolpe, B. Müller, Toolunterstützung für Ethernet und SOME/IP, HANSER automobilske mreže / 2013, https://www.hanser-automotive.de/zeitschrift/archiv/artikel/toolunterstuetzung-fuer-ethernet-und-someip-674039.html?search.highlight=some

[5] J. R. Seyler, N. Navet, L. Fejoz, Insights on the configuration and performances of SOME/IP Service Discovery, April 14, 2015, SAE International in United States, ISSN: 1946-4614, e-ISSN: 1946-4622, https://doi.org/10.4271/2015-01-0197, https://www.slideshare.net/NicolasNavet/saesomeipweb

[6] https://en.wikipedia.org/wiki/MOST_Bus, Veljača 2020

[7] https://en.wikipedia.org/wiki/BroadR-Reach, Veljača 2020

[8] D. Malinak, What's the Difference Between BroadR-Reach and 100Base-T1, Svibanj 31, 2018, https://www.electronicdesign.com/markets/automotive/article/21806576/whats-the-difference-between-broadrreach-and-100baset1

[9] Z. Levi, Automotive Ethernet: The Future of In-Car Networking?, Travanj 04, 2018, https://www.electronicdesign.com/markets/automotive/article/21806349/automotive-ethernet-the-future-of-incar-networking

[10] K. Matheus, T. Königseder, Automotive Ethernet, Cambridge University Press, Srpanj 13, 2017, ISBN: 1107183227, 9781107183223

[11] O. Aspestrand, V. Claeson, The fast-lane development of Automotive Ethernet for Autonomous Drive, Master's thesis EX026/2018, Department of Electrical Engineering Chalmers, University of Technology Gothenburg, Sweden 2018 http://publications.lib.chalmers.se/records/fulltext/256211/256211.pdf

[12] AUTOSAR, SOME/IP Protocol Specification, AUTOSAR FO Release 1.0.0, Studeni 30, 2016, Document Identification No: 696,

https://www.autosar.org/fileadmin/user_upload/standards/foundation/10/AUTOSAR_PRS_SOMEIPProtocol.pdf

[13] AUTOSAR, SOME/IP Service Discovery Protocol Specification, AUTOSAR FO Release 1.2.0, Listopad 27, 2017, Document Identification No: 802, https://www.autosar.org/fileadmin/user_upload/standards/foundation/12/AUTOSAR_PRS_SOMEIPServiceDiscoveryProtocol.pdf

[14] AUTOSAR, Specification of Service Discovery, AUTOSAR CP Release 4.3.1, Prosinac 8, 2017, Document Identification No: 802, https://www.autosar.org/fileadmin/user_upload/standards/classic/43/AUTOSAR_SWS_ServiceDiscovery.pdf

[15] https://forums.ni.com/t5/LabVIEW-Automotive-Ethernet/SOME-IP-Serialization/gpm-p/3845830?profile.language=en (Posjećeno 22.2.2020.)

[16] VECTOR E-LEARNING, Introduction to Automotive Ethernet | SOME/IP, https://elearning.vector.com/mod/page/view.php?id=165 (Posjećeno 23.2.2020.)

[17] H. Zambrano, Automotive Ethernet: Cybersecurity Implications, Auto Cybersecurity, Siječanj 28, 2019, https://dellfer.com/automotive-ethernet-cybersecurity/ (Posjećeno 1.3.2020.)

[18] N. Fabritius, R. Jung, J. Holle, Ethernet Security, https://www.etas.com/data/RealTimes_2017/rt_2017_1_05_en.pdf (Posjećeno 1.3.2020.)

[19] C. Corbett, E. Schoch, F. Karg, P. Felix, Automotive Ethernet: Security opportunity or challenge?, Lecture Notes in Informatics (LNI), Gesellschaft fur Informatik, Bonn 2016

[20] J. Seyler, T. Streichert, N. Navet, M. Glaß, J.Teich, Formal Analysis of the Startup Delay of SOME/IP Service Discovery, Design, Automation & Test in Europe (DATE), At Grenoble, 2015/03/10, 10.7873/DATE.2015.0469