Isplativost primene WebSocket protokola u Bežičnim Senzorskim Mrežama

Miloš Kosanović, Mirko Kosanović
Savremene računarske tehnologije
Visoka tehnička škola strukovnih studija
Niš, Srbija
milos.kosanovic@vtsnis.edu.rs
mirko.kosanovic@vtsnis.edu.rs

***Sadržaj* – Internet polako prerasta iz mreže računara u ogromnu mrežu koja povezuje i veliki broj malih inteligentnih stvari (*Internet of Things*). Jedan od osnovnih problema sa kojim se susrećemo kod njih je da zbog jako limitiranih resursa, koje oni poseduju, nisu u mogućnosti da obavljaju neke složene zadatke u vidu interaktivne komunikacije sa složenijim mrežama i prezentacije tih podataka u *real time* režimu. Sa druge strane imamo jednu jako rastuću tehnologiju, Internet, koja ima mogućnost da ogromnu količinu podataka bez problema distribuira bilo kome, bilo kada i bilo gde. Mogućnost stvaranja mreže malih, gotovo nevidljivih senzorskih čvorova, koji će neprekidno prikupljati informacije iz realnog sveta i trenutno te informacije prikazivati velikom broju klijenata širom sveta, omogućava neslućene primene ove tehnologije. Ovaj rad istražuje mogućnosti implementacije jedne nove tehnologije, WebSocket protokola, koji bi trebalo da smanji komunikacioni saobraćaj između izvora i destinacije, omogući real-time detekciju promena, tj. smanji kašnjenje u prenosu podataka i samim tim omogući duži životni vek takve aplikacije. Izvršena je kritička analiza primene WebSocket protokola sa gledišta razmenjenih paketa kao i veličine headera u tim paketima, u odnosu na neka druga rešenja tj. protokole za *real time* komunikaciju.**

Ključne riječi - WebSocket, HTTP, senzorski čvor; veličina paketa; kašnjenje

I. UVOD

Danas je Internet postao sastavni deo našeg života jer gotovo da ne postoji ni jedan deo ljudske delatnosti koja nije nekim svojim delom povezana sa Internetom. Samim tim, zahtevi koji se postavljaju toj mreži svakog dana sve više i više rastu, od običnog informacionog doba WEB 1.0, pa do današnjih multimedijalnih WEB 2.0 sistema koji uglavnom zahtevaju rad u realnom vremenu kao i punu bidirekcionalnu vezu između korisnika. Većina komunikacije između korisnika Interneta danas se odvija putem standardnog HTTP protokola koji se koristi uslugama TCP i IP protokola. Celokupna komunikacija zasnovana je na standardnom klijent-server modelu gde klijent (aktivna komponenta) postavlja zahtev a server (pasivna komponenta) odgovara na taj zahtev. Sasvim je jasno da u ovakvom modelu server nije u mogućnosti da ispoštuje klasičan *real time* režim rada tj. da pošalje informaciju o nekoj promeni onog trenutka kada se ona i dogodi. Sa druge strane svi ovi protokoli su jako zahtevni u pogledu broja korisnih (*payload* podataka) koji se šalju, jer pored njih šalje se i velika količina kontrolnih podataka (veliki *overhead*). Posledica toga je zahtev za velikom propusnošću mreže kao i pojava kašnjenja kod slanja veće količine podataka. Zadnjih godina razvijene su mnoge tehnologije koje bi trebalo da zadovolje *real time* komunikaciju kao što su *Flash*, *Comet* ili *Ajax* [1]. Međutim, sve ove tehnologije nisu uspele najbolje da reše problem *real time* komunikacija jer su zahtevale od svih klijenata da se u Internet pretraživačima instaliraju programski dodaci (*plug-in*-ovi) dok su sa gledišta servera one bile jako zahtevne za instaliranje. Tek pojavom HTML 5 i predstavljanjem jednog novog protokola, nazvanog WebSocket, dolazi do rešavanja gore pomenutih problema tako da se danas ovaj protokol smatra jednim od onih koji mogu da ostvare potpunu bidirekcionalnu *real time* komunikaciju između klijenta i servera [2].

Ubrzani razvoj WEB tehnologija doprineo je ne da se samo veliki broj podataka prikupi u realnom vremenu (*real time*), i to sa velikog broja geografski različitih lokacija, već i da se te informacije gotovo trenutno prezentuju klijentima. Tu se pre svega misli na nove generacije distribuiranih računarskih tehnologija: *Cluster computing*, *Grid computing* i *Cloud computing*. Sve te osobine bile su idealne da se njihove pogodnosti iskoriste za povezivanje sa jednim distribuiranim izvorom velikog broja podataka kao što su bežične senzorske mreže (BSM). BSM predstavljaju jednu distribuiranu računarsku mrežu koja se sastoji od velikog broja bežičnih senzorskih čvorova (SČ), koji potpuno samostalno mogu da formiraju mrežnu infrastrukturu, putem koje oni prikupljaju, obrađuju i razmenjuju podatke. Oni su proizvoljno raspoređeni u širokom geografskom području i uglavnom imaju samostalno napajanje u vidu malih baterija. Kako su gabariti jednog SČ, a samim tim i kapacitet baterije, jako mali, njen životni vek bio bi ograničen na svega nekoliko dana ako bi svi njegovi delovi radili neprekidno. Imajući u vidu da je promena baterija u većini slučajeva praktično neizvodljiva, proizilazi da je efikasna potrošnja električne energije presudni faktor koji utiče na duži životni vek jednog SČ a samim tim i aplikacije u BSM [3]. Dalja integracija BSM, koja bi omogućila znatno veće mogućnosti, bi bila povezivanje sa mrežom svih mreža, Internetom. Mogućnost stvaranja mreže malih, gotovo nevidljivih SČ-ova, koji će neprekidno prikupljati informacije iz realnog sveta i trenutno te informacije slati na WEB servere, omogućava neslućene primene ove tehnologije. Zato i ne čudi

mišljenje mnogih naučnika koji tvrde da će ova tehnologija u budućnosti u potpunosti izmeniti čovekov život i doprineti da on bude mnogo ugodniji i bolji [4].

Nakon uvoda u problem u poglavlju 2, prikazane su osnovne karakteristike razmene paketa između SČ-ova u BSM. U poglavlju 3 objašnjene su tri osnovne tehnike koje se trenutno koriste u WEB komunikacijama za razmenu podataka u *real time* režimu rada. U poglavlju 4 dat je detaljniji opis jedne nove tehnike koja obećava da će u potpunosti omogućiti rad u *real time* režimu velikog broja malih uređaja od kojih se neće zahtevati veliki softverski i hardverski resursi. U poglavlju 5 date su neke analize isplativosti primene WebSocket protokola u BSM, u odnosu na standardne HTTP tehnike. Poglavlje 6 zaključuje ovaj rad.

II. KARAKTERISTIKE RAZMENE PAKETA KOD BSM

Prenos paketa u BSM razlikuje se od prenosa paketa u standardnim, kako žičanim tako i bežičnim mrežama, po mnogim osobinama. Sigurno da su ograničeni resursi ovih mreža jedan od glavnih problema koji utiču da se standardni protokoli iz TCP/IP skupa ne mogu direktno primenjivati. Kao primer može da nam posluži podatak da je energija koja se potroši za prenos samo 1 bita informacije preko komunikacionog kanala, jednaka energiji koja se potroši za izvršavanje više od 1000 CPU instrukcija [5]. Shodno tome, ako uspemo da smanjimo potrošnju komunikacionog kanala za samo 1-2 %, uspećemo da uštedimo energiju koja je po veličini znatno veća od potrošnje pri svim ostalim aktivnostima SČ. Uzimajući ovu činjenicu u obzir, očigledno je da se najveće uštede u potrošnji električne energije mogu postići primenom energetski efikasnih komunikacionih protokola. Međutim postoje i neke druge specifičnosti koje utiču na taj prenos a koje moramo imati u vidu kada biramo protokol koji ćemo koristiti u BSM kao i šta on mora da ispuni da bi to bio. Te specifičnosti se pre svega odnose na:

a) ***Mehanizam za kontrolu gustine saobraćaja*** - kod BSM najveća gustina saobraćaja se dešava oko *sink*-a i okolnih senzorskih čvorova, jer se tu sakupljaju podaci koji dolaze od senzorskih čvorova iz BSM-a (*upstream*). Za uspešno rešavanje ovog problema potrebno je razviti tri mehanizma i to: za efektivnu detekciju zagušenja u saobraćaju, za izbegavanje zagušenja kao i za kontrolu gustine saobraćaja u BSM-u. Sva tri mehanizma zasnivaju svoja rešenja na kontroli veličine bafera u senzorskim čvorovima, kao i na kontroli frekvencije/brzine slanja podataka (ACC-*Active Congestion Control*).

b) ***Mehanizam za kontrolu sigurnog prenosa paketa*** – siguran prenos podataka kod BSM-a u većini slučajeva može imati drugačije značenje nego što je kod tradicionalnih žičanih veza gde je potrebno garantovati ispravan prenos svakog paketa. Za neke aplikacije kod BSM dovoljno je primiti samo jedan ispravan paket od jednog senzorskog čvora iz nekog regiona, a ne od svih jer većinom oni daju istu informaciju. To nam daje veću slobodu u razvoju transportnih protokola za BSM. Međutim u nekim aplikacijama, kao što su prenos sistemskih podataka, promena ili izmena programa u senzorskim čvorovima ili kod prenosa više podataka (više paketa iste poruke) moramo obezbediti potpuni sigurni prenos. Za rešavanje ovog problema bolje je koristiti mehanizam *hop-to-hop* (HBH) od tradicionalnog *end-to-end* (ETE) mehanizma jer se na taj način smanjuje gustina saobraćaja a samim tim i ušteda potrošnje električne energije je veća. Pored toga ovde nam stoje na raspolaganju ACK i NACK mehanizmi potvrde i negacije prijema ispravnih poruka. Njihovom kombinacijom moguće je na efektan način signalizirati gubitak pojedinih paketa ili ispravnost njihovog prijema.

c) ***Jednostavnost u inicijalnom, početnom prenosu*** – većina aplikacija u BSM je reaktivnog tipa gde senzorski čvorovi predstavljaju pasivne aktere nekog događaja. Oni jednostavno posmatraju okolinu očekujući da se dogodi neki događaj, i reaguju tek na neku promenu ili prozivku od nadređenog senzorskog čvora. Sve te promene mogu se smestiti u svega nekoliko paketa koji se šalju nadređenom senzorskom čvoru. Zato je potrebno da inicijalni prenos kod uspostavljanja veze bude što kraći i jednostavniji.

d) ***Mali broj ponovljenih slanja paketa (retransmission)***– vrlo bitna karakteristika koja mora da bude zadovoljena kako bi se izbeglo nepotrebno trošenje električne energije. Ako i dođe do neophodnosti ponovnog slanja paketa treba voditi računa da što manji broj senzorskih čvorova bude uključen u taj prenos. U tom smislu sigurno je da HBH prenos ima prednostu u odnosu na ETE, jer su samo dva senzorska čvora uključena u ponovljeno slanje paketa. Mana ovakvog prenosa je da on ne može da nam garantuje pouzdan prenos na nivou cele poruke od predajnika do prijemnika, što sa druge strane ETE u potpunosti garantuje.

e) ***Mali header u odnosu na payload podatke*** – u većini aplikacija, komunikacija u BSM-u se odvija po mehanizmu HBH. Svaki od senzorskih čvorova pored osnovne funkcije, detekcije nekog događaja, treba da zadovolji i funkciju rutera kod preusmeravanja velikog broja paketa koji kroz njega prolaze. Kako se u tim paketima najčešće nalazi veoma mali broj *payload* podataka od velikog je značaja da *header* tih paketa smanji na najmanju moguću meru kako bi se smanjila ukupna veličina paketa [6].

f) ***Svi senzorski čvorovi moraju da imaju ravnopravan tretman u komunikaciji*** – kako se radi o mreži sa velikim brojem senzora koji nadgledaju neki region, uvek postoji mogućnost da neki od senzorskih čvorova bude zaspostavljen u komunikaciji. Iz tog razloga informacije koje doaze do sinka mogu biti pogrešne jer nisu kompletne, pa samim tim mogu da dovedu do pogrešnih odluka. Zato treba obezbediti da svi senzori u BSM-u budu ravnopravno tretirani, kako bi se obezbedila pravovremena i realna informacija o promenama u nadgledanom regionu.

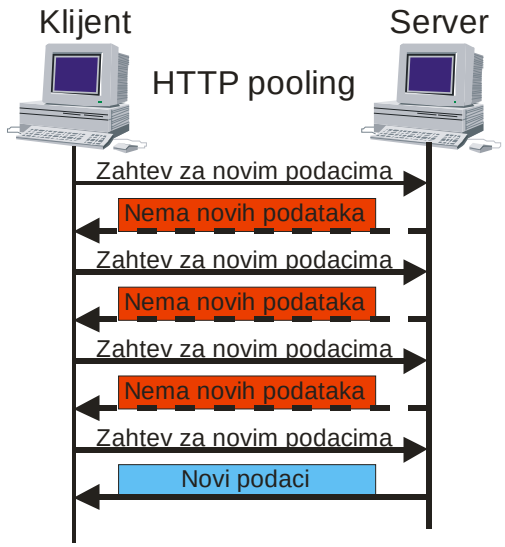
g) ***Saradnja sa susednim čvorovima*** – poželjno je da postoji međusobna komunikacija između susednih slojeva tj. sa mrežnim slojem. Ako ta komunikacija postoji tada *routing* protokol sa nižeg sloja može da obavesti transportni protokol o nekim problemima koji su se javili u komunikaciji (na primer da obavesti da je gubljenje paketa zbog prekida nekog puta-*route failure* a ne zbog pojačanog saobračaja) [6].

Sagledavajući sve gore izložene specifičnosti, idealno bi bilo da sve one budi ispunjene kod primene protokola u BSM. Međutim to je samo idealan slučaj koji je teško izvodljiv, pa

čemo mi definisati tri osnovna cilja koji moraju da budu ispunjeni a to su: mali *header* u odnosu na *payload* podatke, smanjena potrošnja električne energije svakog SČ kao i laka implementacija tj. minimalni potrebni resursi za realizaciju protokola.

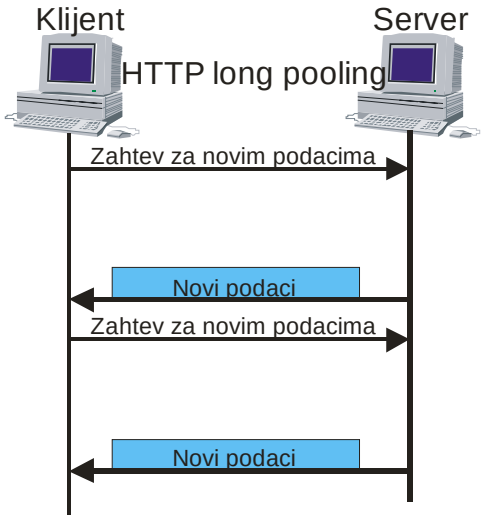
III. TEHNIKE ZA OSTVARIVANJE REAL TIME KONEKCIJE

U *real time* komunikacijama jedan od najvažnijih faktora je kašnjenje (*latency*). Ovaj faktor još više dolazi do izražaja u višeskokovitim topologijama koje su primarne topologije u kojima BSM rade. Osnovna uloga srednjeg sloja (*gateway* ili *proxy* servera) u ovakvim topologijama je da omogući razmenu informacija između različitih servisa i senzorskih čvorova. Standardni protokol koji se na aplikativnom nivou primenjuje za razmenu informacija na Internetu, HTTP protokol, zbog većeg broja paketa koje razmenjuje prilikom uspostave i raskidanja veze kao i veličine tih paketa (veliki *overhead*), nije pogodan za korišćenje u BSM. Njegovo korišćenje sa jedne strane izazvalo bi jako opterećenje svih senzorskih čvorova, tako da bi njihov životni vek bio jako ograničen tj. kratak, a sa druge strane uneo bi velika kašnjenja u isporuci paketa. Sve to potpuno bi eliminisalo ovaj protokol, i on ne bi mogao da se koristi u *real time* aplikacijama u BSM. Zato je bilo neophodno izvršiti neke izmene u okviru HTTP protokola sve u cilju njegovog pojednostavljenja i eliminisanja njegovih prethodno izloženih loših karakteristika. Tako je razvijeno nekoliko tehnika koje su svaka na svoj način donosile nešto novo na ovom polju, ali su sve one parcijalno rešavale ove probleme. Neke od najpoznatijih razvijenih tehnika su HTTP prozivanje (HTTP *pooling*), HTTP dugo pozivanje (HTTP *long pooling*) i HTTP *streaming*.



Slika 1. HTTP *pooling*

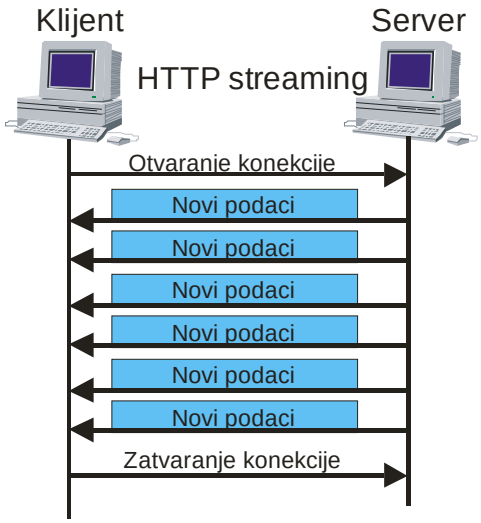
HTTP *pooling* (Sl. 1), predstavlja jedan od prvih i najjednostavnijih načina koji su primenjeni na WEB-u, kako bi omogućili *real time* komunikaciju bez uvođenja nekih specijalnih zahteva, kako na strani klijenta tako i na strani servera. Osnovna ideja ove tehnike sastoji se u tome da klijent vrši povremeno prozivanje servera putem standardnog HTTP porta 80. Bez obzira da li postoji novi podatak ili ne server uvek odgovara na pristigli zahtev. Velike prednosti ovog rešenja su da je njegova implementacija veoma jednostavna kao i da ne zahteva ispunjenje nekih posebnih dodatnih uslova. Međutim, postoje ozbiljni nedostaci ovog rešenja koji se ogledaju u nemogućnosti da se predvidi trenutak promene stanja (informacije) koja treba da se pošalje, pa dolazi do određenog kašnjenja kod slanja podataka jer ono direktno zavisi od trenutka kada je došao zahtev a ne kada se promena stvarno dogodila. Iz tog razloga moguće je da dođe do slanja praznih poruka pa učestala frekvencija zahteva, koja može da smanji kašnjenje kod detekcije promene, može da proizvede znatno jači saobraćaj, pa samim tim i veće opterećenje servera i njegovu veću potrošnju. Ova tehnika takođe nije dala rešenje ni za povećani *overhead* (veličina paketa je velika u odnosu na *payload* podatke koji se šalju) koji je karakterističan za HTTP protokol. Generalno se može reći da HTTP *pooling* daje dobre rezultate kada se prate neka sinhrona događanja kod kojih se period promena unapred zna, a da kod asinhronih događaja daje jako loše rezultate [7].



Slika 2. HTTP long *pooling*

Da bi se omogućila stalna komunikacija između klijenta i servera predložena je nova tehnika nazvana HTTP *long pooling* (Sl. 2). Princip rada ove tehnike sastoji se u tome da klijent uspostavlja vezu sa serverom i drži tu vezu stalno otvorenom sve dok ne dođe do detekcije novog podataka ili dok ne prođe neko određeno vreme koje je unapred definisano. Nakon prijema novog podatka ili isteka definisanog vremena, veza se opet odmah otvara tj. klijent šalje novi zahtev za novim podatkom i opet se čeka definisani period ili dok ne dođe do nove detekcije promene. Na ovaj način izbegnute su loše karakteristike prethodne tehnike oko velikih kašnjenja u detekciji događaja, ali je osnovni problem ove tehnike bio u tome što server nije najbolje mogao da zadovolji veći broj istovremenih zahteva različitih klijenata. Naime, ukoliko je bilo više klijenata koji su uputili istovremeni HTTP zahtev jednom serveru, on je morao da pamti sve te njihove zahteve. To je u znatnoj meri opterećivalo kako njegovu memoriju tako i njegove procesorske kapacitete. Globalno, problem velikog *overhead*-a u odnosu zaokruženu komunikaciju ovde je delimično rešen, jer je broj paketa koji se razmenjuju u toj komunikaciji smanjen. Međutim taj problem je i dalje ostao na nivou pojedinačnih paketa [8].

Još jedan pokušaj da se obezbedi *real time* komunikacija bila je tehnika HTTP *streaming*-a (Sl. 3) i predstavlja najbolje rešenje koje koristi HTTP protokole u potpunosti. Međutim to je i dalje jednosmerna komunikacija (*half-duplex*). Ovde je klijent upućivao samo jedan zahtev prema serveru i taj zahtev je stalno važio tj. veza je bila uvek otvorena.

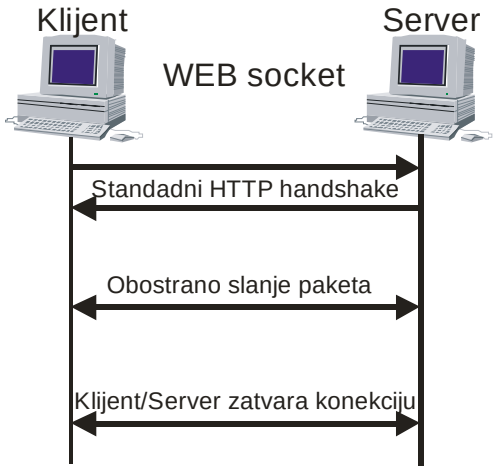


Slika 3. HTTP *streaming*

Osnovni problem koji se ovde javio je to što server nikada nije zatvarao tu vezu, dok se kompletan prenos podataka nije završio. Ukoliko je bilo potrebno da se usluži više klijenata, tada je server morao da sa svakim novim klijentom otvori posebnu, novu konekciju. To je značilo da ako više klijenata istovremeno pošalje zahtev za konekcijom oni su morali da se baferuju sve dok se prethodni klijent-server zahtev ne razreši tj. prethodna konekcija zatvori kako bi se nova otvorila. To baferovanje poruka, koje su zahtevale konekciju sa serverom, se radilo među-mrežnim uređajima kao što su *proxy* serveri ili *firewall*-ovi. To je dodatno otežavalo njihov rad jer su oni morali da čuvaju te poruke sve dok se ne otvori nova konekcija. To je predstavljalo ozbiljan problem jer se kod nekih klijenata javljalo veliko kašnjenje u komunikaciji koje je dovodilo do pada performansi *real time* aplikacije. Ova tehnika pokazala se kao dobro rešenje za aplikacije kod kojih je potrebno brzo i konstantno menjanje podataka koje one prikazuju (jednosmerna komunikacija), kao što su različiti informacioni displeji ili praćenje trenutne lokacije nekog pokretnog objekta. Kada su u pitanju neke interaktivne aplikacije sa više korisnika, ili kada je potreba za komunikacijom obostrana, ova tehnika se suočava sa istim problemima kao i HTTP *long pooling* i ne daje zadovoljavajuće rezultate [7, 8].

IV. WebSocket protokol

U prethodnom poglavlju objasnili smo samo neke osnovne tehnike koje su bile preduslov za razvoj velikog broja rešenja za *real time* aplikacije na WEB-u kao što su *Ajax,* JSON, *Comet, WebRTC* i druge. Međutim sve ove tehnike nisu rešavale glavni problem komunikacije kod BSM-a a to je veliki broj paketa koji su potrebni kod uspostavljanja i raskidanja veza a samim tim i veliki broj prenetih bajtova. Pored toga svi protokoli koji su se koristili kod ovih rešenja imali su veliki *overhead* u paketima. To se posebno odnosi na jako zahtevan HTTP protokol, koji je osnovni protokol u gotovo svim predloženim rešenjima. Imajući to u vidu, jasno je da ni jedno od njih ne odgovara kao prihvatljivo rešenje za primenu u BSM. Kako je potreba povezivanja BSM sa WEB-om stalno rasla, bilo je neophodno da se razviju neka nova rešenja. Jedna od tehnika koja se zadnjih godina pojavila i koja nudi zadovoljavajuća rešenja za probleme koji su izloženi u podpoglavlju 2 je tehnika WebSocket-a.

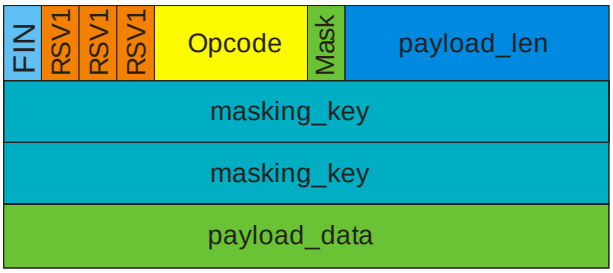


Slika 4. WebSocket komunikacija

WebSocket protokol omogućuje potpunu dvosmernu, *full duplex* vezu, koja je zasnovana na klijent server tehnologiji (Sl. 4). Razvijen je da omogući mehanizam WEB aplikacijama za punu dvosmernu komunikaciju sa serverom a da se pri tome ispoštuju minimalni zahtevi HTTP veze. Osnovna ideja je u tome da se uspostavi *socket* veza između klijenata i servera koristeći HTTP protokol samo u početku uspostavljanja te veze. Nakon toga prelazi se na WebSocket protokol koji omogućava nezavisno i istovremeno slanje poruka u oba smera na ranije formiranoj *socket* vezi. To znači da i klijent i server mogu samostalno inicirati prenos poruke kada oni za to imaju potrebu. Time su izbegnuta kašnjenja u dostavi detektovane promene, omogućena je potpuna *full-dupex* veza, pa su stvoreni uslovi za omogućavanje klasične *real time* komunikacije između klijenta i servera. Samim tim ovaj model ne pripada tradicionalnom modelu zahtev-odgovor, koji je važio kod klasičnog HTTP protokola, jer i klijent i server, u ovom slučaju, se ponašaju kao klijenti jer mogu samostalno da započnu komunikaciju. Raskidanje veze takođe mogu da izvrše oba učesnika u komunikaciji i to kada god oni to žele. Za razliku od TCP protokola, WebSocket protokol pripada grupi protokola koji se zasnivaju na slanju poruka (*message-based*). To znači da se on služi okvirom koji pravi oko svakog podatka koji se šalje, što pojednostavljuje komunikaciju između klijenta i servera. Sada nije potrebno slati dodatnu poruku koja označava da je prenos podataka završen. Pored toga, od serverskog/klijentskog programa, na primer od JavaScript-a, se ne zahteva neka složenost u formiranju ili čitanju poruke koja se šalje ili prima. Kako se zadnjih godina WebSocket protokol intenzivno razvijao, specifikacija njegovog okvira se često menjala. U početku on je bio dosta jednostavan. Verzija protokola HyBi 00 taj okvir ograničava sa dva bajta, početni bajt je 00 a zadnji FF.

Međutim, da bi se otklonili mnogi nedostaci ovakvo koncipiranog jednostavnog okvira, pre svega u bezbednosti podataka koje on prenosi, on je pretrpeo neke izmene tako da je veličina zaglavlja koja je u početku bila samo dva bajta sada povećana na min. šest bajtova. Na Sl. 5 prikazan je jedan takav tipičan okvir WebSocket protokola koji pripada verziji HyBi10 [9, 10]. On se sastoji iz sledećih polja:

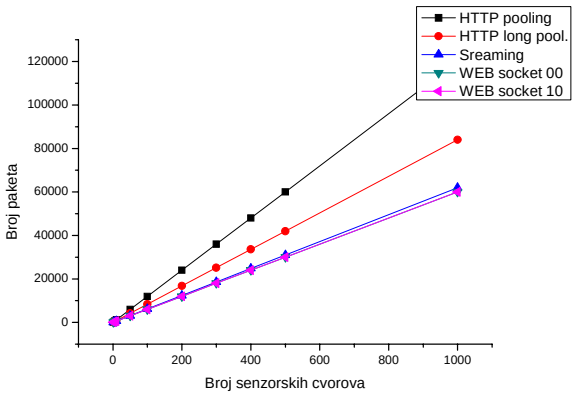
1. **FIN**(1 bit) – ukazuje na zadnji okvir u okviru jedne poruke,
2. **RSV1**, **RSV2**, **RSV3** (1 bit svaki) – rezervisani bitovi za neke buduće funkcije protokola,
3. **Opcode** (4 bita) – definiše tip okvira koji se šalje,
4. **Mask** (1 bit) – ukazuje kada je okvir zaštićen,
5. **Payload_len** (7 bita, 7+16 bita ili 7+ 64 bita) – dužina korisnih (*payload*) podataka,
6. **Masking-key** (32 bita) – podatak kojim se štite payload podaci (XOR sa payload podacima),
7. **Payload podaci** – veličina ovog polja zavisi od specifikacije date u polju ***payload_len,***



Slika 5. WebSocket okvir

Posmatrajući strukturu WebSocket okvira, primećujemo da je on dosta pojednostavljen i da je *overhead* kod njega znatno smanjen na najmanju moguću meru. Naime, minimalna veličina zaglavlja iznosi samo 2-6 bajta što je u odnosu na standardni HTTP protokol, kod koga se ovaj *overhead* obično meri od nekoliko stotina bajtova pa do nekoliko kB, značajno smanjenje [11].

Da bi proverili uspešnost gore izloženih tehnika koje nam pružaju mogućnost konekcije sa SČ-ima u BSM izvršili smo analizu broja poslatih paketa kao i količine poslatih bajtova u *headeru* svakih od navedenih protokola u funkciji broja SČ u mreži. Pretpostavili smo da svaki SČ direktno komunicira sa *gateway* uređajem koji je povezan sa WEB-om preko standardnih protokola iz TCP/IP skupa. Takođe smo usvojili da je prosečan *header* kod HTTP protokola oko 500 B, kod WebSocket HyBi00 2 B, kod WebSocket HyBi10 6 B, definisani period otvorene veze kod HTTP *long pooling*-a 300 s, period otvorene veze kod HTTP *streaming*-a je isti kao i period testiranja i iznosi 3600 s i da se promene na svim senzorima događaju u okviru perioda od 60 s. Analiza je rađena samo za jedan skok (*hop*) kod slanja paketa tj. samo za slanje paketa između SČ-ova i *gateway* uređaja. Više je nego očigledno da se sa povećanjem broja skokova vrši linearno povećanje broja paketa koji se šalju, pa se samim tim i prednosti WebSocket protokola još više dolaze do izražaja

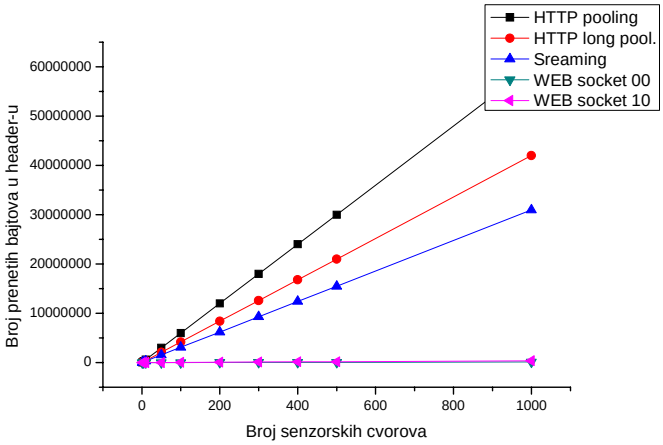


Slika 6. Broj poslatih paketa za različiti broj senzorskih čvorova

Na Sl. 6 prikazani su grafici zavisnost broja paketa u funkciji različitog broja SČ-ova za sve protokole. Posmatrajući grafikone prikazane na Sl. 6 možemo da zaključimo sledeće:

1. broj paketa kod svih protokola linearno raste sa povećanjem broja SČ-ova,
2. protokoli WebSocket HyBi00 i WebSocket HyBi10 imaju potpuno isti broj paketa,
3. ukupan broj paketa koji šalju protokoli iz familije WebSocket-a i HTTP *steaminga* se neznatno razlikuju,
4. najveći broj paketa sa šalje kod protokola HTTP pooling,
5. sa povećanjem broja SČ-ova razlika između ukupnog broja poslatih paketa između protokola je sve veća.

Generalno možemo reći da što se tiče ukupnog broja poslatih paketa WebSocket protokol je uvek u prednosti u odnosu na protokole koji se služe standardnim HTTP protokolom i da jedino HTTP *streaming* protokol može približno da se poredi sa WebSocket protokolom.



Slika 7. Broj bajtova u *header*-u u zavisnosti od broja senzorskih čvorova

Na Sl. 7 prikazani su grafici zavisnosti količine bajtova koji su poslati a nalaze se u *header*-u svakog paketa u funkciji broja SČ-ova za sve protokole. Iz prikazanog grafika možemo izvući sledeće zaključke:

1. kod svih protokola sa porastom broja senzorskih čvorova raste i broj bajtova koji se prenose a nalaze se u *header*-u tj. *overhead* se povećava,

2. protokoli iz familije WebSocket-a imaju znatno manji *overhead* u odnosu na protokole koji se služe HTTP protokolom, pa je samim tim potrošnja elektične energije smanjena,

3. najveći *overhead* je prisutan kod protokola HTTP *pooling*, što je i očekivano,

4. razlika u veličini *overhead*-a se drastično povećava u korist WebSocket protokola kako se broj senzorskih čvorova povećava.

Kako je ušteda energije u jednoj BSM jedan od osnovnih ciljeva kojem teži svaka aplikacija koja se u njoj izvršava, a kako je slanje i prijem paketa jedan od najvećih konzumenata, očigledno je da ako želimo da postignemo što duži životni vek SČ-ova u mreži, a samim tim i aplikacije, potrebno je da tu komunikaciju svedemo na najmanju moguću meru. Na osnovu urađenih analiza i grafika prikazanih na Sl. 6 i Sl. 7 može se zaključiti da je najmanji broj paketa koji se distribuiraju kao i da je najmanji *overhead* upravo kod WebSocket protokola. Imajući u vidu karakteristike prenosa paketa u BSM izložene u podpoglavlju 2 a naročito jednostavnost u inicijalnom, početnom prenosu i mali *header* u odnosu na *payload* podatke, nije teško zaključiti da je WebSocket protokol po svojim osobinama jako dobro rešenje za implementaciju u SČ-ima u okviru jedne WSN *real time* aplikacije.

V. ZAKLJUČAK

Zadnjih godina vodi se velika polemika oko izbora odgovarajuće tehnologije koja će biti u mogućnosti da prikaže veliku količinu prikupljenih podataka sa različitih izvora, velikom broju klijenata na različitim mestima, i sve to u *real time* režimu rada. Kako se Internet izborio kao osnovni medijum za prikaz tih podataka, a kako su na njemu primarni protokoli iz TCP/IP skupa, logično je da se oni zadrže kako bi se ostvarila kompatibilnost sa velikim brojem uređaja koji ih podržavaju. Sa druge strane primarni cilj kojem se teži kod razvijanja aplikacije za BSM je efikasna energetska potrošnja i primena manje zahtevnih protokola zbog ograničenih resursa SČ-a. WebSocket protokol koji je prikazan u ovom radu upravo je dizajniran da pomiri ove dve suprotnosti i da sa veoma skromnim resursima svakog SČ-a omogući njegovu dvosmernu komunikaciju sa WEB-om. Sa jedne strane on je potpuno kompatibilan sa HTTP protokolom, a sa druge strane on se prilagodio ograničenim resursima SČ. Pored omogućavanja potpuno nezavisne dvosmerne komunikacije, smanjenje broja poruka u komunikaciji kao i smanjenje *overhead*-a u većini razmenjenih poruka, učinio je da se smanji kašnjenje u dostavljanju informacija. Samim tim omogućen je i *real time* režim rada aplikacija u BSM. Smanjenjem intenziteta saobraćaja, smanjena je i potrošnja energije svakog SČ čime smo omogućili znatno duži životni vek aplikacije u BSM. Kako je WebSocket protokol i dalje jako mladi protokol, dalja istraživanja u ovoj oblasti kao i buduća rešenja mogla bi da ga kvalifikuju kao osnovno rešenje za većinu *real time* aplikacija u BSM.

LITERATURA

[1] Shruti M. Rakhunde, Real Time Data Communication over Full Duplex Network Using Websocket, IOSR Journal of Computer Science pp.15-19, http://iosrjournals.org/iosr-jce/papers/ICAET-2014/volume-5/3.pdf

[2] Pimentel, V.; Nickerson, B.G.,"Communicating and Displaying Real-Ti me Data withWebSocket,",Internet Computing, IEEE, vol.16, no.4, pp.4 5,53, July-Aug. 2012

[3] I.F.Akyildiz, T.Melodia, K.R.Chowdhury,''A survey on wireless multimedia sensor networks'', Computer Network (2006), doi:10.1016/j.comnet. 2006.10.002

[4] J. Gubbia, R.Buyyab, S.Marusic, M.Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions", Future Generation Computer Systems 29, 2013, pp.1645–1660

[5] P.Dutta, D.Culler, S.Shenker, ``Procrastination Might Lead to a Longer and More Useful Life``, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.5780 acc. 14.12.2015

[6] M.Kosanović, "Primena standarnih transportnih protokola u bežičnim senzorskim mrežama" , YU INFO 2009, Kopaonik 8-11.03.2009

[7] Upgrading HTTP to WebSocket, http://enterprisewebbook.com/ch8_websockets.html , pos. 25.01.2016

[8] R. Narayana Swamy, Dr. G.Mahadevan, Event Driven Architecture using HTML5 Web Sockets for Wireless Sensor Networks, http://isems.org/images/extraimages/3.pdf , pos. 20.12.2015

[9] What has been changed since WebSocket HyBi 00? https://developers.google.com/web/updates/2011/08/What-s-different-in-the-new-WebSocket-protocol?hl=en , pos. 20.12.2015

[10] The Web protocol draft, May 2010, https://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76

[11] Jasdeep Jaitla, WebSockets vs Rest: Understanding the difference, https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/

ABSTRACT

Internet is gradually developing from a network of the computers into a vast network that connects a large number of the small intelligent things (Internet of Things). One of the main problems we face with them is that Wireless Sensor Networks (WSNs) are not able to perform some complex tasks. One of the task is interactive communication with complex networks and presentation of the data in real time mode, due to very limited resources. On the other hand, we have a strongly growing technology, the Internet, which has the possibility of distributing an enormous amount of the data to anyone, anytime, anywhere, without any problems. The ability to create a network of small, almost invisible sensor nodes, which will continuously collect real-world information and then distribute it to a large number of clients around the world opens unimaginable applications of this technology. This paper explores the possibilities of a new technology, WebSocket protocol, which should reduce the communication traffic between the origin and destination. This technology should enable the real-time detection of changes, i.e. reduce the delay in the transfer of data, and, by doing that, enable prolonged life of such applications. A critical analysis of the WebSocket protocol from the viewpoint of exchanged packets as well as the size of headers in these packages compared to some other solutions i.e. protocols for real time communication is done.

APPLICABILITY OF WEBSOCKET PROTOCOL IN WIRELESS SENSOR NETWORKS

Miloš Kosanović, Mirko Kosanović