LABORATORIJSKA VEŽBA BR. 5

Komunikaciono okruženje

CILJ VEŽBE

- Upoznavanje sa protokolima za uspostavljanje veze sa Web-om
- Upoznavanje sa tehnologijom 6LoWPAN
- Upoznavanje sa HTTP protokolom
- Upoznavanje sa MQTT protokolom
- Upoznavanje sa CoAP protokolom
- Testiranje slanja HTTP zahteva Web serveru
- Komunikacija sa MQTT serverom (broker)
- Konkretni aplikativni primeri korišćenja MQTT posrednika
- Donošenje zaključaka na osnovu urađenih testiranja

POTREBNA OPREMA

- Računar sa instaliranim razvojnim softverom za Arduino
- Arduino UNO + Mega 3 komplet
- WiFi modul za povezivanje ESP8288
- Komplet alata za montažu

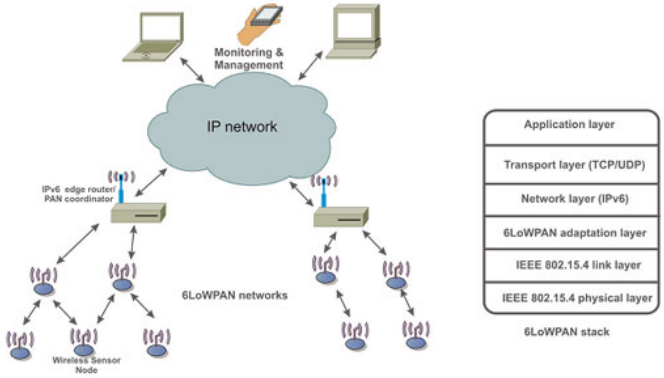
TEORIJSKE OSNOVE

U prethodnoj vežbi upoznali smo se sa načinima povezivanja Arduina sa Internerom koristeći Ethernet i WiFi modul ESP8266. Ova vežba treba da pomogne studentima da se upoznaju sa tehnologijom koja se koristi kod povezivanja uređaja sa osiromašenim resursima (kao što su bežični senzorski čvorovi) sa Internetom kao i odgovarajućim protokolima koji se u toj vezi koriste za slanje i primanje podataka. Protokol je dogovoreni strukturirani format podataka koji se koristi za mrežnu komunikaciju između dva računara. On tačno definiše šta treba poslati i primiti i koje akcije treba preduzeti na osnovu primljenih podataka.

1. TEHNOLOGIJA 6LoWPAN

6LoWPAN je otvoreni standard definisan u RFC6282 od strane Internet Engineering Task Force (IETF), tela koje je definisalo puno otvorenih standarda koji se koriste na Internetu, kao što su UDP, TCP i HTTP. U početku 6LoWPAN je bio zamišljen da podrži standard IEEE 802.15.4 za bežične mreže male snage u 2.4-GHz području, da bi sada prerastao u standard koji se adaptirao i može da se koristi i u raznim drugim mrežnim tehnologijama kao što su Sub-1GHz low-power RF, Bluetooth® Smart, power line control (PLC) i low-power Wi-Fi®. Zasniva se na već dobro poznatom standardu IEEE 802.15.4(LoWPAN) koji je primenjen na uređaje koji imaju jako limitirane resurse kakvi su bežični SČ-ovi. Jedan od osnovnih preduslova koje treba da izvrši svaki SČ je minimalna potrošnja električne energije kako bi se postigao što veći životni vek tj. nesmetani rad aplikacije u BSM . Kako je komunikacija u većini slučajeva najveći potrošać električne energije, sasvim je jasno da je potrebno da se ona svede na najmanju moguću meru.

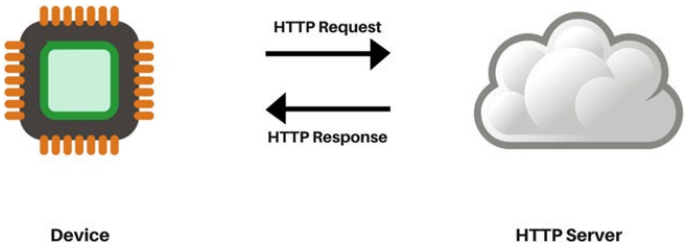
Razvijeno je puno različitih tehnologija za bežičnu komunikaciju između SČ-ova koje omogućavaju pristup Internetu ali je sasvim jasno da tehnologije koje se zasnivaju na TCP/IP skupu protokola imaju prednost. Kako većina protokola iz TCP/IP skupa predstavljaju jako zahtevne protokole, kod kojih su samo zaglavlja dosta velika (IPv4-24 bajta, IPv6-40 bajta, UDP-8 bajta, TCP-24 bajta), jasno je da bi puna primena ovih protokola u BSM-a bio potpuni promašaj sa gledišta energetske isplativosti i efikasnosti. Za samo 2-3 bajta korisnih podataka (*payload data*), kakva je obično veličina podataka koji se prenose kroz BSM, morali bi da prenosimo 30-tak bajtova, a samim tim i potrošimo veliku količinu električne energije, a to indirektno znači da bi životni vek tih SČ-ova bio sveden na samo nekoliko dana. Tehnologija 6LoWPAN je ovaj zahtev stavila kao primarni cilj koji mora da se ispuni.



Slika 1. Tipično povezivanje uređaja u 6LoWPAN mreži

6LoWPAN mreža se povezuje sa drugim IP mrežama kroz jedan ili više rutera (*edge routers*) čiji je zadatak da vrše prosleđivanje IP paketa između različitih mreža (vidi Sliku 1). Povezivanje sa drugim IP mrežama može se obezbediti kroz različite veze-linkove kao što su Ethernet, Wi-Fi ili 3G/4G. Kako standard 6LoWPAN definiše operacije protokola IPv6 preko IEEE 802.15.4 standarda, ruteri moraju takođe da podržavaju IPv6 tranzicioni mehanizam koji omogućava povezivanje 6LoWPAN mreže sa IPv4 mrežama, što je definisano u NAT64 RFK 6146 standardu. To oslobađa SČ-ove u 6LoWPAN mreži da imaju implementiran ovaj mehanizam a istovremeno omogučuju im da nesmetano komuniciraju sa IPv4 mrežama. Generalno možemo reći da je osnovni cilj svake 6LoWPAN mreže da omogući korišćenje standardnih Web servisa, kao što su REST, XML, JSON i drugi, uređajima koji imaju jako ograničene resurse, kao što su SČ-ovi i to u sredinama koje imaju jako veliki procenat loše primljenih poruka tkz. LLNs (*Low-power and Lossy Networks*) mreža.

2. HTTP



Slika br. 2 HTTP protokol (Hyper Text Transfer Protocol)

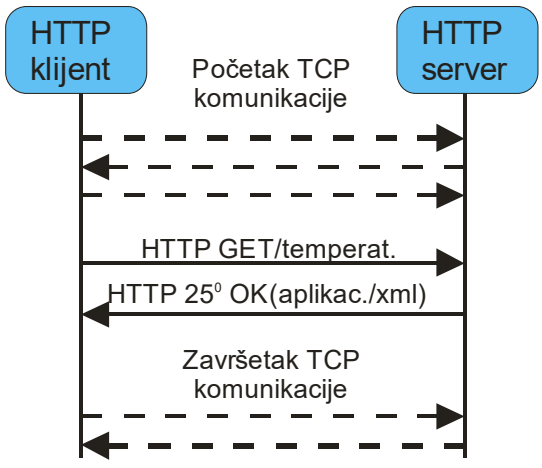
Web koristi HTTP protokol kao osnovni protokol za prenos hiperteksta. HTTP podržava višestruke metode prenosa podataka, ali u ovoj vežbi mi ćemo pisati kod za dve popularne metode, GET i POST. GET i POST metode rade isti posao i njihov kod je vrlo sličan, ali postoji mala razlika u njihovoj implementaciji tj. u formatu koji one koriste. GET ima ograničenje koliko podataka može da prenese u odnosu na POST koji nema nikakvih ograničeja. POST se takođe smatra sigurnijom metodom u odnosu na GET metodu. Na osnovu zahteva aplikacija mi možemo da izaberemo metodu koja će bolje zadovoljiti tj. koja je prilagodnija zahtevima aplikacije. Na slici br.2 prikazana je standardna komunikacija koja se odvija između uređaja i HTTP server.

3. CoAP aplikacioni protokol

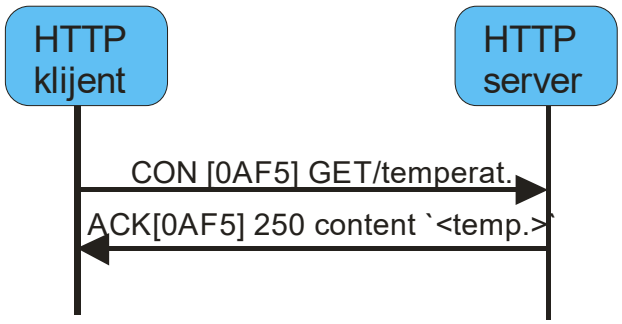
Integracija REST arhitekture u okviru BSM nije nimalo jednostavan zadatak iz proste činjenice da se ovde radi o jako limitiranim resursima kojima raspolažu SČ u okviru te mreže. Tipični SČ napajaju se putem baterije, strogo limitiranog izvora energije, poseduju nekoliko kB memorije i procesore koji imaju ograničene tj. smanjene računske mogućnosti. Samim tim, svaka direktna primena originalnih protokola iz TCP/IP steka je potpuno neprimenljiva jer bi na taj način životni vek svih SČ u mreži bio jako kratak. Zato je razvijen poseban protokol od strane radne grupe IETF koji je nazvan CoAP (*Constrained Application Protocol*) sa ciljem da to bude osnovni Web transportni protokol koji bi se primenjivao u BSM i koji bi zamenio HTTP protokol. CoAP nastoji da primeni potpuno isti sistem prenosa podataka kao i HTTP protokol ali sa znatno manjim zahtevima za resursima. On podržava jedan deo HTTP funkcija, ali i proširuje taj set sa sopstvenim funkcijama kako bi pojednostavio komunikaciju između dva senzorska čvora, tj. omogućio tkz. M2M (*Machine to Machine*) komunikaciju. Na taj način usluge koje nude Web servisi dobijaju na značaju jer sada svaki SČ može da ih koristi i učestvuje u njihovom proširivanju.

Osnovna namera CoAP protokola je da obezbedi generički Web protokol kojim će SČ moći da komuniciraju. On je veoma sličan HTTP protokolu, čija je detaljnija komunikacija prikazana na slici br. 3, ali njegov cilj nije da jednostavno kompresuje HTTP pakete, već da realizuje jedan smanjeni set sistemskih poruka koje će omogućiti M2M komunikaciju. U tu svrhu CoAP koristi sledeće četiri poruke: CON (*Confirmable*), NON (*Non-Confirmable*), ACK (*Acknowledgment*) i RST (*Reset*). Mehanizam rada ovog protokola je sličan klijent-server modelu komunikacije jer klijent uvek zahteva neki servis od servera. Nakon primljenog zahteva server odgovara uključujući u paket jedinstveni broj poruke (ID) koji je dobio sa zahtevom (vidi Sliku 4). Posmatrajući slike 3 i 4 može se videti koliko je CoAP protokol uprošćen jer on koristi samo dva paketa kod normalne komunikacije za razliku od HTTP protokola kome su potrebna 7 paketa. Posebna prednost CoAP protokola je ta da su potrebni minimalni resursi koje treba da poseduje svaki SČ da bi on mogao da se implementira: 10 kB RAM i 100 kB memorijskog prostora za programski kod (RFC 7228).

Problem pouzdanosti slanja kod CoAP protokola (koristi se nesiguran UDP protokol), je rešen korišćenjem *stop-and-wait* protokola, ugradnjom mehanizma za detekciju duplih poruka (isti ID) kao i već pomenutim slanjem *multicast* paketa. CoAP protokol je projektovan tako da koristi minimalna sredstva, i u SČ i na mreži. Umesto zahtevnog TCP protokola, koristi se UDP protokol. Fiksni *header* od 4 B i opcija kompaktnog kodiranja omogućava formiranje malih poruka koje izazivaju malo ili nimalo fragmentacije na link sloju. Od ostalih karakteristika CoAP protokola istakli bi još: podrška za slanje *unicast* i *multicast* paketa, mogućnost asinhronog prijema i slanja poruka, veoma mali *overhead*, podrška za proksi servis, ugrađene mogućnosti za keširanje poruka, proširivo i prilagodljivo zaglavlje, jednostavno pronalaženje resursa putem URI i td.



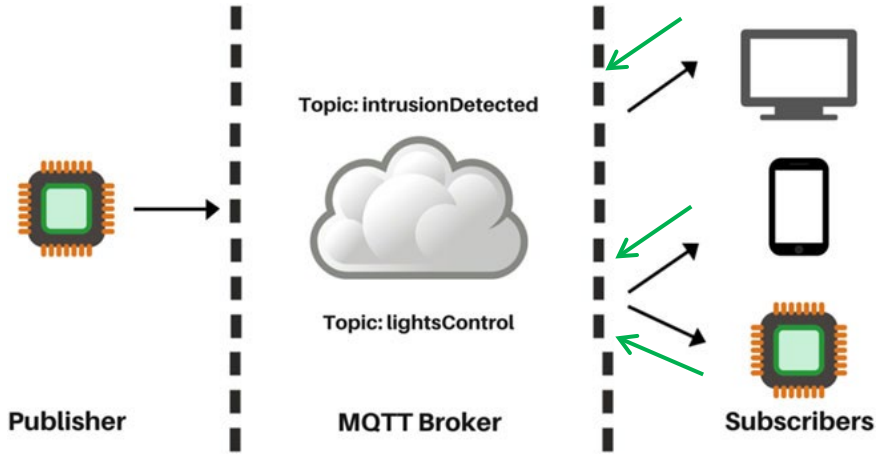
Slika br. 2. HTTP komunikacija



Slika br. 3. CoAP komunikacija

4. **MQTT (*Message Queue Telemetry Transport*) protokol**

Protokol MQTT (*Message Queue Telemetry Transport*) predstavlja otvoren, jednostavan i lagan ptotokol za komunikaciju između dva hardverska uređaja (*machine-to-machine*) koji za svoju realizaciju ne zahteva neke jake resurse (lightweight protocol). Razvili su ga Andy StanfordClark (IBM) i Arlen Nipper (Eurotech) 1999. godine. Osnovni cilj im je bio da razviju protokol koji bi omogućio efikasnu brzinu prenosa podataka uz malu potrošnju el.energije i manju cenu celokupnog potrebnog Sistema za njegovu realizaciju. Razvijeni protokol MQTT bazira se na objavi-pretplati (*publisher-subscriber*) arhitekturi što ovom protokolu omogućava potpuno asihronu komunikaciju. On se pokazao lakšim i ekonomičnijim od protokola HTTP (*HyperText Transfer Protocol*) koji se zasnivao na arhitekturi zahteva i odgovora. Model objavi-pretplati upravljan je događajima i omogućava klijentima da objavljuju poruke bez brige o njihovoj konačnoj destinaciji. Konačnu destinaciju objavljene poruke određuje posrednik MQTT (*MQTT broker*) koji na osnovu ranije dobijenih klijentskih pretplata prosleđuje poruke i tako oslobađa klijente od slanja zahteva za porukama od interesa, za razliku od protokola HTTP kod koga klijenti moraju tražiti informacije koje žele da dobiju od određenih servera. Protokol MQTT odlikuje i mala veličina fiksnog zaglavlja (*fixed header*) od svega 2 bajta što dodatno smanjuje mrežno opterećenje. Prema članku Stepena Nicholasa protokol MQTT nudi uštedu baterije na mobilnim uređajima od oko 4.1% u jednom danu uz održavanje stabilne konekcije naspram protokola HTTP. Iako inicijalno spajanje MQTT klijenta na server troši više baterije jer se dodatno šalje i univerzalni identifikator klijenta, održavanje trajne konekcije uz primanje i objavljivanje poruka pokazalo se ekonomičnijim za protokol MQTT zbog toga što HTTP puno češće obavlja operaciju spajanja koja utiće na povećanu potrošnju el.energije tj. na životni vek baterije (vidi sliku br. 5). Navedene karakteristike čine ga idealnim za korišćenje u ograničenim okruženjima i mrežama niske propusnosti sa ograničenim mogućnostima obrade, malim kapacitetom memorije i visokom latencijom što su tipične karakteristike senzorskih čvorova u bežičnim senzorskim mrežama.

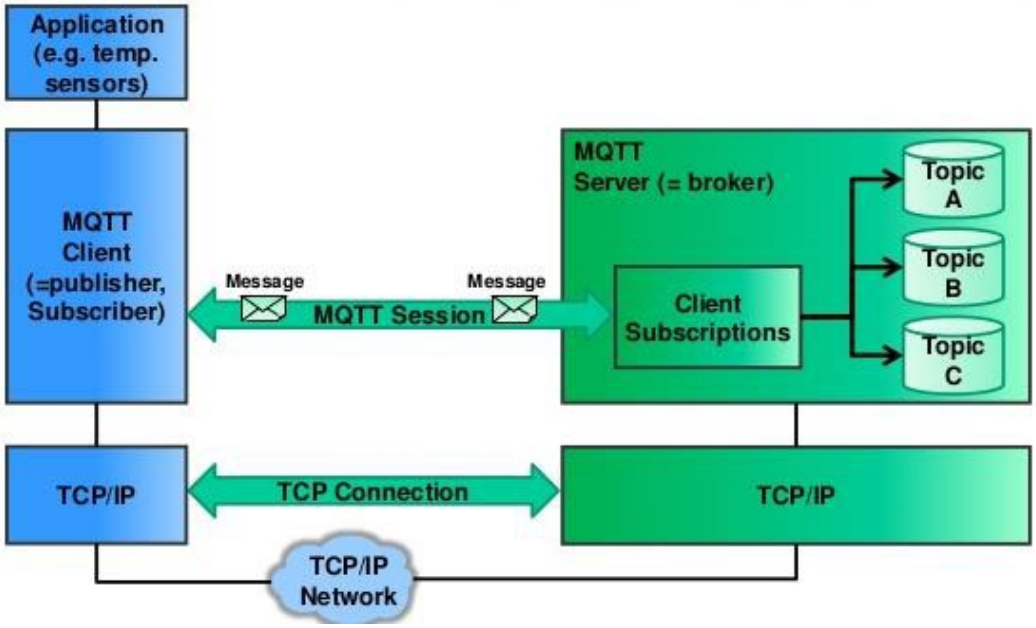


Slika br. 5 Komunikacija kod MQTT protokola

Komunikacija između klijenata kod protokola MQTT ostvarena je modelom objavi-pretplati uz središnju ulogu posrednika MQTT koji prati pretplate klijenata i delegira poruke u zavisnosti od tih pretplata. Kada klijent šalje (publish) poruku posredniku mora uz poruku da navede temu (topic) na koju želi da objaviti poruku. Tema posredniku predstavlja glavnu informaciju za usmeravanje poruke prema krajnjim klijentima. Sa druge strane kada se klijenti pretplaćuju (subscribe) moraju navesti temu na koju se žele da se pretplate kako bi posrednik znao isporučiti sve poruke čija se tema podudara s temama na koje je klijent pretplaćen. Kod ovakvog načina komunikacije nije potrebno da se klijenti međusobno poznaju jer oni međusobno komuniciraju samo preko odabranih tema čime se uklanja zavisnost između proizvođača (*producer*) i potrošača (*consumer*) informacija. Sve to povećava skalabilnost sistema tj. efikasnost i upotrebljivost sistema bez obzira na značajno povećanje zainteresovanih klijenata kako proizvođača (*producer*) tako i potrošača (*consumer*).

5. **MQTT konekcija**

Početak svake komunikacije između MQTT klijenata predstavlja njihovo spajanje na MQTT posrednika koji predstavlja središte komunikacije i nosi odgovornost za primanje svih poruka, njihovo filtriranje i prosleđivanje svim pretplaćenim klijentima. Prilikom spajanja sa posrednikom klijenti moraju navesti adresu posrednika i port 1883 koji je rezervisan isključivo za komunikaciju korišćenjem protokola MQTT. Kako je MQTT baziran nad TCP/IP transportnim i mrežnim slojem (slika 6) svaki klijent prilikom spajanja na posrednika otvara trajnu TCP vezu iznad koje se otvara MQTT sesija za objavljivanje i primanje poruka.



Slika br. 6 Prilaz stvarne veze kod MQTT protokola

Samu konekciju uvek inicira klijent koji šalje *Connect* poruku posredniku MQTT na koju on odgovara ***Connack*** porukom sa statusnim kodom koji daje informaciju o uspešnosti spajanja klijenta (slika 4) [5]. Nakon spajanja veza se održava trajno sve dok klijent ne odluči da je prekine ili se konekcija ne izgubi.

Ovakva vrsta konekcije omogućava da klijenti koji razmenjuju poruke budu potpuno nezavisni jedan od drugog. Ta njihova nezavisnost može se posmatrati kroz sledeće tri dimenzije:

1. Prostorna dimenzija – nije potrebno da se pošiljalac i primalac podataka međusobno poznaju (npr. prema IP adresi i portu)
2. Vremenska dimenzija – pošiljalac i primalac podataka ne moraju da budu aktivni u isto vreme
3. Sinhronizacijska dimenzija – operacije pošiljaoca i primaoca nisu zaustavljene tokom objave i primanja poruka (klijenti rade asinhrono nezavisno jedan od drugog)

1. Programski kod (Arduino) za realizaciju HTTP protokola

Da bi smo mogli da testiramo HTTP protokol potrebno je prvo iskoristiti programski kod iz prethodne vežbe kako bi uspostavili konekciju sa Internetom putem WiFi modula. Sledeći korak u pisanju koda odnosi se na definisanje promenljivih, konstanti i funkcija koje ćemo koristiti za slanje podataka na server koristeći HTTP protokol. Kao što je prikazano na slici Listing-u 1 prvo definišemo adresu i port servera kome pristupa naš Arduino uređaj i šalje podatke. Za potrebe ove vežbe možete koristiti adresu www.httpbin.org, koji je otvoreni javno dostupan test server koji jednostavno odgovara na sve HTTP zahteve koji mu se upute. U budućim vašim projektima vi ćete koristiti servere sa drugim adresama sve u zavisnosti od aplikacije koju želite da uradite.

Listing 1. Adresa I port servera

```
char server [] = {"www.httpbin.org"};
int port = 80;
```

Funkcija doHttpGet () data u listingu 2 objedinjuje sve detalje koji su potrebni da se pripremi zahtev za GET metodu, tj. da se povežemo sa serverom i pošaljemo zahtev. Korišćenjem client.connect u okviru koga navodimo adresu server i port preko koga se povezujemo (server, port) pokušavamo da se konektujemo na željeni server. Ako uspemo da ostvarimo konekciju, onda pripremamo zahtev. U zahtevu koji koristi GET metod, podaci se šalju kao deo URL adrese u format ime/format, na primer:

http://www.httpbin.org/get?temperatureSensor= 85 & metric = F.

Primer pokazuje da će se poslati dva parametra, prvi je vrednost 85 koja označava očitani parameter sa temperaturnog senzora i drugi je metrički sa vrednošću F koji označava tip temperaturne skale. Konačno, ovakav HTTP zahtev prosleđujemo serveru koristeći client.println () metodu. Ova metoda će poslati komande na adresirani server preko mreže od koga ćemo kasnije dobiti odgovor.

Listing 2. HTTP GET zahtev

```
void doHttpGet()
{
// Prepare data or parameters that need to be posted to server
String requestData = "requestVar=test";
// Check if a connection to server:port was made
if (client.connect(server, port))
{
Serial.println("[INFO] Server Connected - HTTP GET Started");
// Make HTTP GET request
client.println("GET /get?" + requestData + " HTTP/1.1");
client.println("Host: " + String(server));
client.println("Connection: close");
client.println();
Serial.println("[INFO] HTTP GET Completed");
}
else
{
Serial.println("[ERROR] Connection Failed");
}
Serial.println("-----------------------------------------------");
}
```

Prikazani kod šalje HTTP GET zahtev, ali kao što je ranije pomenuto, ima ograničenje dužine, tako da ako želite da izbegnete ovo ograničenje, treba koristiti HTTP POST zahtev. Funkcija doHttpPost () data je u sledećem listingu 3. Ovaj kod sadrži sve detalje priprema zahteva za POST metodom, povezivanje sa serverom i slanje zahteva. Pokušajte da se povežete na server koristeći client.connect (server, port) metodu kao u prethodnom primeru. Do sada je kod sličan HTTP GET zahtevu. Ako se uspostavi veza potrebno je pripremiti POST zahtev. U zahtevu koji koristi metodu POST, podaci se takođe šalju u formatu para ime/vrednost, ali to je deo zahteva. Kao što možete videti u listingu 3, slanje HTTP POST zahteva zahteva dodatne informacije o zaglavlju. Konačno, prosledite HTTP zahtev serveru koristeći client.println () metodu kao i u prethodnom primeru.

Listing 3. HTTP POST zahtev

```
void doHttpPost()
{
// Prepare data or parameters that need to be posted to server
String requestData = "requestData={\"requestVar:test\"}";
// Check if a connection to server:port was made
if (client.connect(server, port))
{
Serial.println("[INFO] Server Connected - HTTP POST Started");
// Make HTTP POST request
client.println("POST /post HTTP/1.1");
client.println("Host: " + String(server));
client.println("User-Agent: Arduino/1.0");
client.println("Connection: close");
```

```
client.println("Content-Type: application/x-www-form-urlencoded;");
client.print("Content-Length: ");
client.println(requestData.length());
client.println();
client.println(requestData);
Serial.println("[INFO] HTTP POST Completed");
}
else
{
// Connection to server:port failed
Serial.println("[ERROR] Connection Failed");
}
Serial.println("-----------------------------------------------");
}
```

I na kraju potrebno je samo još zadati standardne funkcije setup() I loop() kako bi ovaj Arduino kod profunkcionisao. Kao što pokazuje kod na listingu 4, funkcija setup () inicijalizuje serijski port, povezuje se sa Internetom, a zatim vrši ili HTTP GET zahtev pozivanjem doHttpGet () ili HTTP POST zahtev pozivanjem funkcije doHttpPost ().

Listing 4. Programski kod za standardnu funkciju—setup()

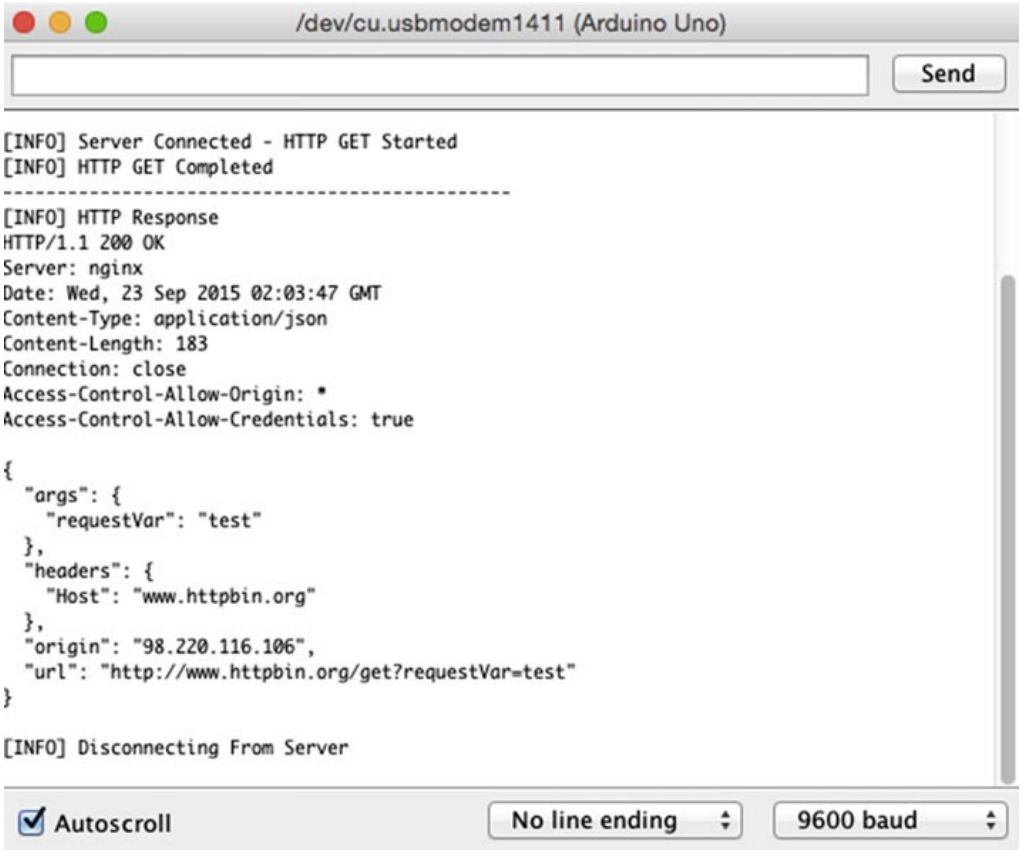
```
void setup()
{
// Initialize serial port
Serial.begin(9600);
// Connect Arduino to internet
connectToInternet();
// Make HTTP GET request
doHttpGet();
}
```

Kako u ovoj vežbi nije potrebno da se uradi neka obrada poslatih podataka na strani servera, dodaćete kod za čitanje odgovora od servera do loop() funkcije. Test server koji smo koristili u ovoj vežbi jednostavno vraća (echo) sve informacije koje smo naveli u zahtevu koji mu je upućen. Na taj način mi ćemo primljene podatke proslediti na serijski monitor kako bi mogli da vidimo da li nam kod funkcioniše. Kao što je navedeno u listingu 5, proverite da li postoje bajtovi koji su dostupni za čitanje od ViFiClient, pročitajte sve dostupne bajtove i odštampajte ih u prozoru Serial Monitor. Kada su svi bajtovi pročitani i odštampani, potrebno je zaustaviti rad klijenta.

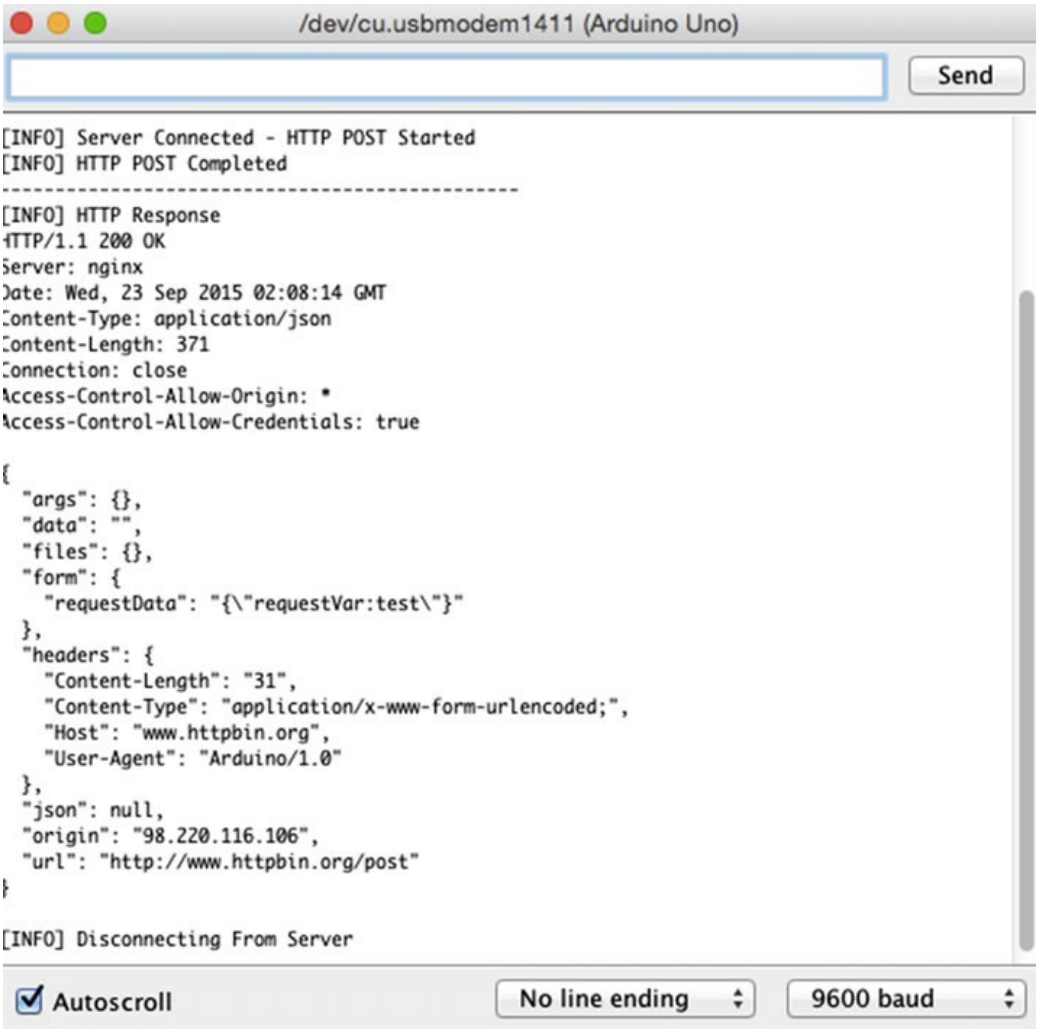
Listing 5. Programski kod za standardnu funkciju —loop()

```
void loop()
{
if (client.available())
{
Serial.println("[INFO] HTTP Response");
}
// Read available incoming bytes from the server and print
while (client.available())
{
char c = client.read();
Serial.write(c);
}
// If the server:port has disconnected, then stop the client
if (!client.connected())
{
Serial.println();
Serial.println("[INFO] Disconnecting From Server");
client.stop();
}
}
```

Ako ste sve dobro uradili u prozoru Serial monitor trebalo bi da dobijete informacije koje su prikazane na slikama br.7 i 8.



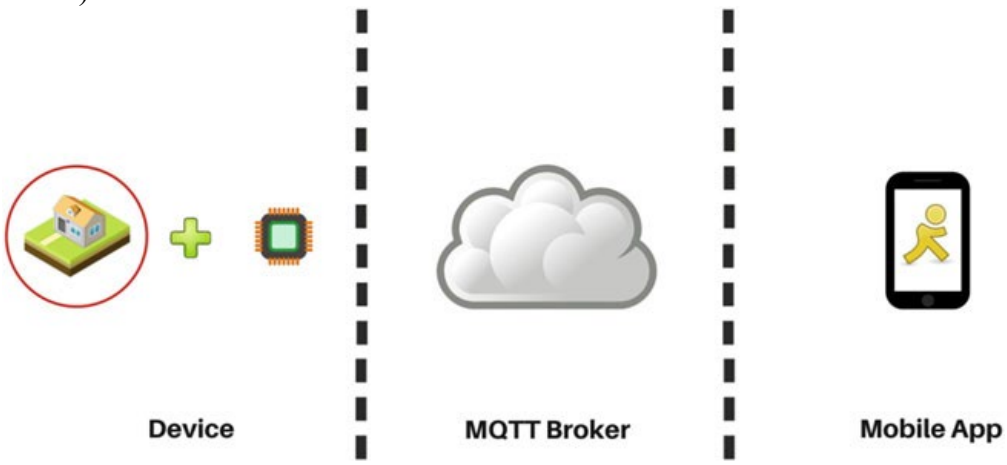
Slika br. 7. *HTTP zahtev:: GET method*



Slika br.8. *HTTP zahtev: POST method*

2. Programski kod (Arduino) za realizaciju MQTT protokola

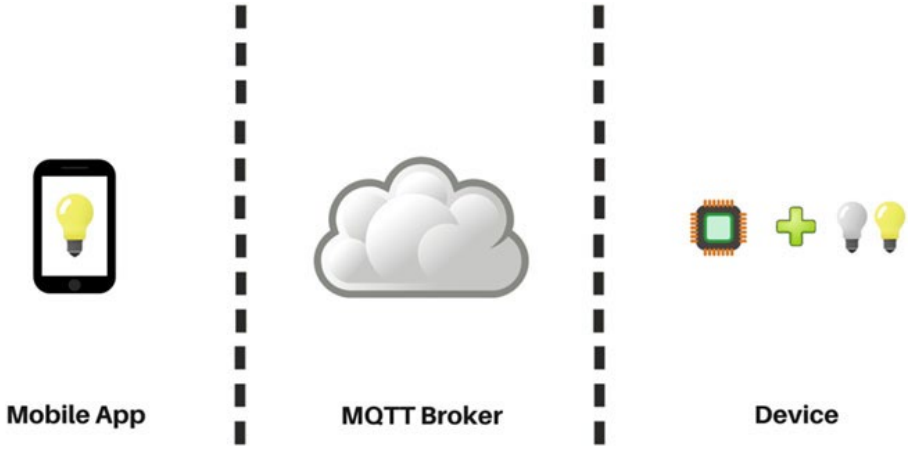
Razumevanje MQTT-a je važno za izgradnju IoT aplikacija, pa pogledajmo njegovu primenu na nekoliko primera kako bi lakše razumeli ovaj protokol. Prikazaćemo prvo primenu na jednostavnom sistem za detekciju narušavanja određenog prostora koji je prikazan na slici br. 9. Sistem se sastoji od tri komponente - senzora pokreta koji detektuju upade i objavljuju podatke, mobilne Android aplikacije koja prima te podatke i upozorava korisnika aplikacije i komponentu koja nas interesuje a to je MQTT posrednika (broker).



Slika br.9. Sistem za nadgledanje prostora

U ovom primeru sensor će predstavljati izdavač (publisher) i slaće poruke prema MQTT posredniku kada god detektuje neku promenu u nadgledanom prostoru. Zadatak MQTT posrednika je da doda ovu poruku u spisak tema koje objavljuje (u ovom slučaju to je tema codifythings/intrusionDetected). Potrebno je da se ranije mobilna aplikacija prijavi na temu codifithings/intrusionDetected kako bi kada god se objavi nova poruka u ovoj temi dobila obaveštenje. Ovo će dovesti do toga da mobilna aplikacija uvek dobije poruku kada god se pojavi nova poruka u okvitu ove teme.

MQTT prtokol može da se koristi i za razvoj mobilnih aplikacija koje služe kao daljinski upravljači različitih vrsta uređaja, kao što je aplikacija za kontrolu osvetljenja. Kao što je prikazano na slici br.10, ovaj system se sastoji od tri komponente, ali u poređenju sa prethodnim primerom redosled prvih dve komponente je obrnut. To znači da je prva komponenta mobilna aplikacija koja omogućava korisniku da uključi ili isključi svetla, druga komponenta je uređaj povezan sa svetlima, a treća komponenta je MQTT posrednik.



Slika br.10. Komponente u okviru Sistema za daljisko upravljanje uređajima

Programski kod

Prikazaćemo sada kako treba napisati programski kod putem koga možemo da koristimo usluge MQTT protokola. Kompletan kod podeljen je u četri dela i to: inicijalizacija spoljnih datoteka, WiFi povezivanje sa Internetom, objavljivanje podataka na MQTT server i prijavljivanje na MQTT server za dobijanje poruka na određenu temu.

1.Inicijalizacija spoljnih datoteka

Prvi deo koda nalazi se u Listingu 6. On uključuje sve potrebne spoljne biblioteke koje su potrebne za pokretanje našeg koda. Za Internet povezivanje, morate uključiti datoteku <WiFi.h> (pod pretpostavkom da koristite WiFi modul) i, za MQTT komunikaciju sa MQTT posrednikom, moramo uključiti datoteku <PubSubClient.h>. Biblioteku <PubSubClient.h> možete dobiti sa adrese:

https://github.com/knolleary/pubsubclient/releases/tag/v2.3

Listing 6. Spoljne biblioteke

```
#include <SPI.h>
#include <WiFi.h>
#include <PubSubClient.h>
```

2.Povezivanje sa Internetom

Drugi deo koda definiše promenljive, konstante i funkcije koje su će se koristiti za povezivanje na Internet. Ovde treba ubaciti kod koji smo koristili u prethodnoj laboratorijskoj vežbi broj 4.

3. Objavljivanje/prijavljivanje podataka sa MQTT servera

Treći deo koda definiše promenljive, konstante i funkcije koje se koriste za objavljivanje i pretplatu na MQTT server/posrednik. Kod objavljuje i pretplaćuje se na podatke za istu temu. Potrebno je definisati adresu i port (podrazumevano je 1883) MQTT posrednika sa kojim želite da povežete vaš Arduino uređaj, kao što je prikazano u listing 7. Promenljiva topic definiše na koju temu na posredniku želite da se prijavite ili da dobijate obaveštenja. Ukoliko nemamte podignut svoj MQTT server, možete koristiti otvoreni dostupan MQTT posrednik iz Eclipse-a Fondacija (www.iot.eclipse.org) ili Moskuitto (www.test.moskuitto.org).

Listing 7. MQTT Setup

```
// IP address of the MQTT broker
char server[] = { "iot.eclipse.org" };
int port = 1883
char topic[] = { "codifythings/testMessage" };
```

Kao što je prikazano na listing 3-8 potrebno je izvršiti inicijalizaciju MQTT klijenta. Takođe funkcija **callback()** enkapsulira sve podatke (*payload* podaci) koje je primila od brokera.

Listing 3-8. MQTT Inicijalizacija i Callback funcija

```
PubSubClient pubSubClient(server, 1883, callback, client);
void callback(char* topic, byte* payload, unsigned int length)
{
// Print payload
String payloadContent = String((char *)payload);
Serial.println("[INFO] Payload: " + payloadContent);
}
```

4.Standardne funkcije

Na kraju, kod u ovom poslednjem odeljku je dat u Listingu 9. Primenjuju se Arduino standardne funkcije setup () i loop (). U funkciji setup (), programski kod inicijalizuje serijski port i povezuje Arduino uređaj sa Internetom. Ako je MQTT posrednik povezan, on će se pretplatiti na codifythings/testMessage temu. Kada se uspešno pretplatite, kod objavljuje novu poruku na temu codifithings/testMessage. Kako su i svi klijenti prijavljeni na istu temu čim se neka nova poruka objavi, funkcija povratnog poziva callback() biće pozvana. Funkcija loop () jednostavno čeka nove poruke od MQTT brokera.

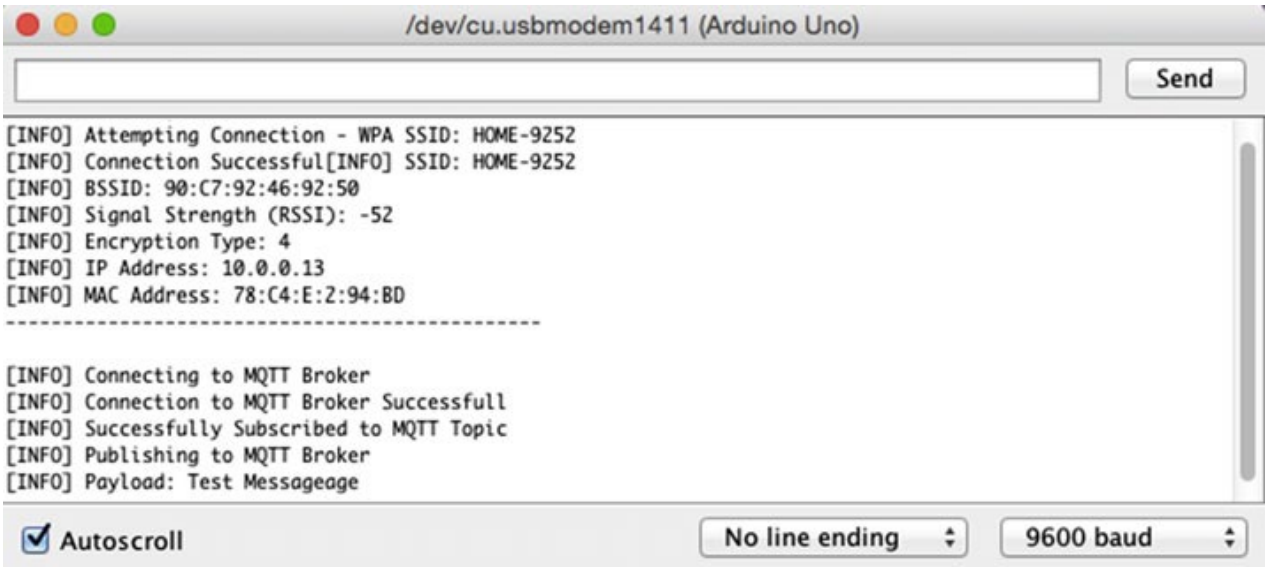
Listing 9. Programski kod za standardne Arduino funkcije

```
void setup()
{
// Initialize serial port
Serial.begin(9600);
```

```
// Connect Arduino to internet
connectToInternet();

//Connect MQTT Broker
Serial.println("[INFO] Connecting to MQTT Broker");
if (pubSubClient.connect( "arduinoClient" ))
{
Serial.println("[INFO] Connection to MQTT Broker Successful");
pubSubClient.subscribe(topic);
Serial.println("[INFO] Successfully Subscribed to MQTT Topic ");
Serial.println("[INFO] Publishing to MQTT Broker");
pubSubClient.publish(topic, "Test Message");
}
else
{
Serial.println("[INFO] Connection to MQTT Broker Failed");
}
}
void loop()
{
// Wait for messages from MQTT broker
pubSubClient.loop();
}
```

Vaš Arduino programski kod je gotov.Da biste testirali aplikaciju, pokrenite program I otvorite Serial Monitor prozir. U njemu bi trebalo da vidite nešto slično kao što je prikazana na slici br. 11.



Slika br. 11. MQTT: Publish/subscribe log poruke