Varen komunikacijski posrednik na podlagi WebSocket protokola in arhitektura za neposredno povezovanje interneta stvari s spletnimi aplikacijami

Matevž Hribernik[1]

[1] *Univerza v Ljubljani, Fakulteta za elekrotehniko, Tržaška ul. 25, 1000 Ljubljana*
E-pošta: mh4656@student.uni-lj.si

Secure WebSocket Based Broker and Architecture for Connecting Internet of Things Devices and Web Based Applications

Abstract. *This paper presents an intuitive approach into connecting IoT devices and Web applications using WebSocket protocol and specific application network. Paper introduces the term IoT while WebSocket is explained in detail, with particular focus on protocol usage and connection handshake, using connection and termination. The latter part of this paper presents the usage of WebSocket in creating WebSocket application broker, architecture and communication specific packages structure. This architecture is demonstrated using IoT thermostat. It consists of IoT device and web app that is also an IoT device in its own way. The security aspects of this architecture with all parts of logic and communication architecture are explained and demonstrated.*

Key words: *Internet of things, WebSocket, protocol, web application, connected devices, ESP8266, ad-hoc network*

1 Uvod

V modernem svetu komunikacij in povezanih naprav so posebej zanimive naprave interneta stvari (ang. Internet of things, IoT), ki so povezane v oblačne storitve (ang. Cloud Services) z namenom večje uporabnosti. Koncept IoT ni nov, saj si naprave izmenjujejo podatek med seboj že mnogo let. Dokaj nova je sprememba običajnih predmetov, kot so avtomobili, hišne naprave in stavbe v t.i. pametne, povezane naprave, ki lahko komunicirajo s človekom, sistemi, ali med seboj. Poznavanje komunikacijskih protokolov med temi napravami je izjemnega pomena, saj omogoča transparentno, enostavno in varno komunikacijo med napravami. [1] [2]

Namen tega članka je predlog arhitekture, ki uporablja WebSocket protokol za zagotavljanje kvalitetnega in varnega delovanja aplikacije in hitrega ter enostavnega razvoja novih aplikacij. Poenostavljeno bi lahko rekli, da ta arhitektura možnosti in funkcionalnosti nekaterih drugih protokolov, kot so MQTT (Message Queuing Telemetry Transport) ali XMPP (Extensible Messaging and Presence Protocol), torej konceprt posrednika in naročila, razširi na področje WebSocket protokola. V članku bom naprej predstavil nekaj osnov na področju interneta stvari in osnov delovanja WebSocket protokola. V predzadnjem poglavju bom predstavil delovanje aplikacije, ki deluje na podlagi predlagane arhitekture. Arhitektura zajema aplikacijski posrednik na strežniku, spletne IoT aplikacije in fizične IoT naprave. Predlagana arhitektura je namenjena zagotavljanju neposredne varnosti med spletnimi aplikacijama in IoT napravami, kjer je strežniška aplikacija samo vmesen povezovalni člen brez aplikacijske logike. Sama arhitektura oz. koncept predstavlja novost v svetu IoT, saj večina aplikacij IoT predvideva uporabo oblačnih storitev za shranjevanje in obdelavo podatkov. Teh se predlagana arhitektura želi znebiti.

2 Kaj je Internet stvari (IoT)?

Internet stvari je po definiciji omrežje fizičnih naprav, vozil in drugih »stvari« z vgrajeno elektroniko, ki vsebujejo senzorje ter aktuatorje, in so povezane v omrežje, kar omogoča, da te naprave zbirajo in izmenjajo podatke. Namen IoT je te objekte povezati in jih narediti inteligentne ter bolj sposobne interakcije s človekom in med seboj. IoT ne zajema zgolj naprav, ampak tudi načine komunikacije, nadzorno in informacijsko ravnino ter procesiranje le-teh, kar omogoča različne aplikacije. [1] [2] [3]

Termin IoT pomeni mnogo različnih ekosistemov, komunikacijskih tehnologij in primerov uporabe. Zato je težko razložiti internet stvari kot en sam ekosistem ali arhitekturo. Za posamezne komponente je pomembna logična in aplikacijska povezava med napravami in relacija med posameznimi komponentami. Uporaba in povezava komponent je vedno odvisna od aplikacije.

Jedrne komponente IoT:

- **Senzorji oz. krmilniki** so majhne povezane naprave, navadno mikrokrmilniške, lahko tudi bolj napredne, in so tisto kar daje ime »stvari« v internetu stvari.
- **Naprave prehoda** so tiste, ki zbirajo in pošiljajo oz. usmerjajo podatke skozi podatkovno omrežje.
- **Omrežje** je fizična arhitektura, ki vse te naprave povezuje.
- **Analiza in prevajanje** podatkov je sloj, ki se ukvarja izključno s skrbjo za podatke.

- **Aplikacijske storitve** skrbijo za režijo, nadzor in predstavitev zbranih podatkov in možnost vpliva nanje.

3 WebSocket

WebSocket (WS) protokol je prvič predstavljen s strani Web Hypertext Application Technology Working Group (WHATWG) in je del HTML 5 standardizacije [4][5]. Leta 2011 je standardizacijo prevzel IETF [6].

Protokol tako omogoča, da lahko veliko spletnih aplikacij postane razširljivih in delujejo v realnem času, saj je WebSocket vgrajen v moderne brskalnike, in se lahko uporablja hkrati s HTTP. Te aplikacije tako postanejo enostavnejše in hitrejše kot prvotne HTTP aplikacije.

3.1 Osnove delovanja WebSocket

WS specifikacija je napisana tako, da želi zamenjati zastarele tehnologije HTTP, kot sta polling ali long polling, ko gre za dvostransko komunikacijo. Protokol dobro deluje z obstoječo spletno infrastrukturo, npr. posredniki, požarne pregrade, filtri in postopki identifikacije. S tem postaja uporaba pollinga nepotrebna in čedalje več aplikacij se poslužuje WS tehnologij.

Za zagotavljanje popolne združljivosti WS omogoča uporabo TCP vrat 80 in 443. Ko je vzpostavljena WS povezava, obe končni napravi izmenjujeta sporočila v katerikoli smeri, brez potrebe po režiji ali dogovorov o pošiljanju. Tako strežnik vzdržuje z odjemalcem zgolj eno TCP povezavo za izmenjavo sporočil v realnem času, z zelo malo redundantnimi podatki. WS podatkovno sporočilo doda zgolj 2 B dejanskim podatkom, kar je mnogo manj kot 871 B (velikost tipične glave HTTP). Manjšanje števila povezav in redundantnih glav zmanjša obremenitev strežnika in kompleksnost aplikacij v realnem času. Ker WS deluje na podlagi vzpostavljanja TCP tunelov, je mogoče vzpostaviti WS povezavo tudi skozi posrednike. Ker je vzpostavitev povezave WS podobna HTTP, vsebuje tudi možnost varne povezave preko SSL. [5]

WS je neodvisen protokol nad TCP in sloni na principu, da mora biti minimalno okvirjanja med prenosom podatkov. Edina režija, ki mora biti prisotna, je ločevanje med vsebino v obliki teksta v Unicode in binarnimi sporočili. Kot omenjeno, sta za to potrebna zgolj 2 B. Takšna oblika predvideva, da bo aplikacija sama poskrbela za ostale metapodatke, ki jih potrebuje. WS je torej ustvarjen tako, da je čim podobnejši surovemu TCP. Hkrati so imeli ustvarjalci v mislih vse omejitve spletnih funkcionalnosti, kar posledično pomeni, da si WS strežnik deli TCP vrata s HTTP strežnikom, kar omogoča zelo široko uporabo v modernih spletnih aplikacijah.

Tako kot je mogoče vzpostaviti HTTP povezavo preko TLS/SSL in temu pravimo HTTPs, je enaka nadgradnja mogoča tudi za WS preko TLS/SSL. Takšna komunikacija je namesto s HTTPs zahtevo začeta s pomočjo HTTPS zahteve. Sam protokol WS, tako kot HTTP, ni varen in šifriran. Takšen postane šele, ko na transportnem nivoju uporabimo varen povezovalni protokol.

3.2 Vzpostavljanje povezave in komunikacija

Protokol vsebuje dva glavna dela: rokovanje vzpostavitve in prenos podatkov. WebSocket povezave so navadno dogovorjene z uporabo HTTP. Namen tega je združljivost s starejšimi odjemalci.

Odjemalec najprej pošlje GET zahtevo z željo po nadgradnji »Upgrade« te povezave. HTTP Connection Upgrade je v glavi HTTP spročila in predlga uporabo WebSocket preko želenih vrat, navadno 80 ali 443. Prava vzpostavitev sledi, ko se protokol zamenja iz HTTP v WebSocket. [7]

Strežnik sprejeme HTTP zahtevo in če podpira WS, in se strinja z menjavo protokolov, ta sproži nadgradnjo s HTTP sporočilom. Katera koli statusna koda drugačna kot 101 pomeni, da bo nadaljevanje sporočanja potekalo kot HTTP. Če je statusna koda enaka 101, bo nadgradnja izvedena. Po odgovoru strežnika je HTTP povezava prekinjena in zamenjana z WS preko iste TCP/IP povezave.

3.3 Prenos podatkov

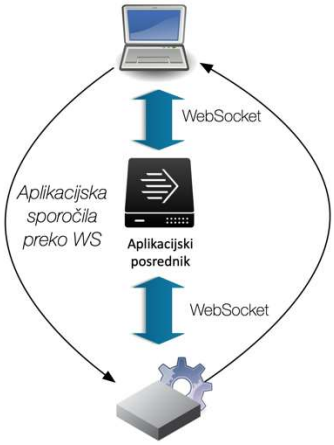
Ko sta strežnik in odjemalec izmenjala pozdravna sporočila, se lahko začne prenos podatkov. Kdorkoli (strežnik ali odjemalec) lahko neodvisno od enega komunicira z drugim. Ravno v tem je bistvo dvostranske komunikacij, ki jo omogoča WebSocket. Besedilo ali binarni podatki se lahko prenašajo kot vsebina v okvirjih. Ti okvirji imajo minimalno glavno z 2 B obvezne vsebine. To sta polji opcode in dolžina. Če je sprejeta opcode, ki je nepoznana, je obvezna prekinitev povezave.

Poleg podatkovnih okvirjev so v uporabi tudi nadzorni okvirji. Ti imajo pomembno vlogo pri vzdrževanju WS povezave. Po definiciji poznamo tri nadzorne okvirje: Close (0x8), Ping (0x9) in Pong (0xA). Vsak od teh ima edinstveno opcode. Close okvirji so namenjeni zapiranju komunikacije med napravama, Ping in Pong pa sta namenjena preverjanju in vzdreževanju povezave. Tako predstavljata »keepalive« in »heartbeat« funkcionalnost.

4 Predlagana arhitektura

V tem članku predstavljam, kako je mogoče s tehnologijami HTML5 specifikacije izvesti varno in enostavno arhitekturo, ki deluje preko vseh omrežnih naprav in hkrati omogoča neposredno komunikacijo med odjemalci. Tako lahko rečemo, da gre za omrežje naprav na L7 ISO/OSI modela. Predlagana arhitektura je tako lahko stalno ali začasno (ad hoc) aplikacijsko omrežje, preko katerega lahko hkrati komunicira več neodvisnih aplikacij.

Arhitektura na sliki 1, zahteva uporabo aplikacijskega posrednik (ang. broker) in IoT naprav, ki so lahko spletne aplikacije ali mikrokrmilniške naprava. S stališča posrednika so vse naprave enakovredne in upošteva uporabljene aplikacijske naslove (ID) za posredovanje sporočil med eno in drugo napravo. Sama arhitektura in posrednik ne zahtevata, da so podatki, ki jih sporočila vsebujejo, šifrirani. Za voljo demonstracije in hkrati dobre prakse sem v demonstracijskem okolju predstavil možnost, pri kateri se podatki v aplikacijsem sporočilu šifrirajo z algoritmom AES.



Slika 1 Predlagana arhitektura aplikacijskega posrednik in omrežja enakovrednih klientov.

4.1 Aplikacijski posrednik in protokol

Glavna komponenta predlagane arhitekture je posrednik. Namen tega je posredovanje sporočila poslanega od ene naprave do druge naprave.

Vsaka naprava, ki želi komunicirati v tem aplikacijskem omrežju zato potrebuje svoj lasten ID, s katerim se predstavi posredniku. Ostale naprave, s katerimi želi komunicirati, ga morajo poznati. Če bodo padatki kriptirani mora imeti geslo AES enako drugi napravi, s katero bo komuniciral.

Vsaka naprava najprej vzpostavi svojo WebSocket (secure) sejo na posrednik, ki je stalna in se vzdržuje po potrebi. Če je omrežje vezano na lokalno omrežje to ni potrebno, če pa je vezano na javno IP arhitekturo je zaradi prevajanja omrežnih naslovov (NAT) vzdrževanje TCP seje potrebno. Na posrdovalniku je vsaka WS seja vezana na ID naprave ali splentne aplikacije. [8]

Ta arhitektura zahteva tri vrste spročil. Prvo je prijava naprave na posrednik *('packet_type':'LOG_ON')*, drugo je potrdilo te prijave (*'packet_type':'ACK_LOG_ON'*). Tretje sporočilo (MSG) pa je namenjeno posredovanju in vsebuje aplikacijske podatke. Vse vrste sporočil so v obliki JSON, saj je ta enostavna za obdelavo in implementacijo.

{'packet_type':'MSG ', //START...
'transmitter':'ID oddajnika',
'receiver': ID sprejemnika, // broadcast, multicast
'theme': ' something', // če multicast
'data': [šifrirani podatki]}

S stališča posrednika je zgornje sporočilo vse, kar potrebuje. Potrebuje vrsto sporočila, da ve kaj z njim storiti, in sprejemni ID za posredovanje tega sporočila naprej. Arhitektura za posrednik predvidevata tako broadcast kot multicast sporočila, čeprav jih demonstracijska aplikacija ne uporablja.

4.2 IoT aplikacija in naprava

Logika aplikacije je glede na podano arhitekturo lahko na eni napravi v tem omrežju, ali več njih. Za demonstracijo primer lahko vzamemo povezani termostat, ki je neposredno krmiljen s strani odjemalca v brskalniku. Logika je razdeljena tako v samo napravo (odločitev o vklopu ogrevanja), kot v aplikacijo v spletnem brskalniku (nastavitev temperature).

Slika 2 Spletni IoT klient za kontrolo termostata.

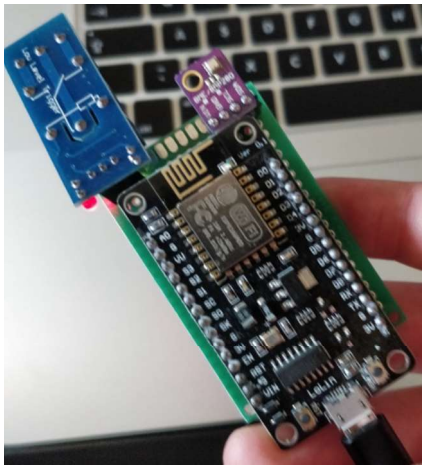
Za predstavitev aspekta HTML5 IoT tudi spletna aplikacija na sliki 2 uporablja senzorje naprave, na kateri deluje. Spletna aplikacija v času povezave odjemalca v primeru premika izračuna, ali se od »doma« (IoT termostata) nahaja manj kot 10 km. V tem primeru vklopi t.i. domačo temperaturo, drugače je vklopljena temperatura »z doma«.

Mikrokrmilniška naprava, na sliki 3 je grajena na platformi ESP8266 (WeMos NodeMCU). To je mikrokrmilnik, ki je pripravljen za povezavo preko Wi-Fi na medmrežje. Na mikrokrmilnik je poleg elektromagnetnega stikala za vklop oz. izklop ogrevanja preko vodila I^2C povezan senzor okolja BME280.

Naprava vedno začne komunikacijo z drugo z uporabo START sporočila (uporabnik zahteva povezavo). Protokol za delovanje aplikacije je zapisan v »data« delu sporočila. Za delovanje demonstracijske aplikacije sem uporabil nekaj enostavnih sporočilnih oblik, saj je oblika odvisna od aplikacije in ne od posrednika.

```
[{  "instruction":"STATE",
    "current_temp":"23.89",
    "home_temp":"25.00",
    "away_temp":"19.00",
    "releY":"1",
    "home":"1",
    "manual":"0"}]
[{  "instruction":"HOME_set",
    "distance":33.850807552794954,
    "home":"true"}]
[{  "instruction":"SET",
    "home_temp":"26.00",
    "away_temp":"19.00",
    "releY":"false"}]
```

IoT naprava iz spomina vedno naloži zadnje shranjene podatke. Prav tako shrani podatke WiFi omrežja, ki jih je uporabnik vnesel in se poveže na posrednik. Ko se na posrednik poveže spletna aplikacija, ta pošlje začetno sporočilo START. S tem IoT napravi sporoči, kam želi, da ta vrača aktualne podatke o stanju. Aktualni podatki se pošljejo vsakič, ko pride do kakršne koli spremembe, naj bo to temperatura ali menjava stanja stikala, v obliki sporočila MSG/STATE. To sporočilo se prav tako pošlje po vsaki zahtevi START, da lahko uporabnik vidi začetne oz. shranjene podatke. Sporočilo MSG/SET je namenjeno spreminjaju nastavitev termostata. Sporočilo MSG/HOME_set sporoči IoT termostatu koliko je telefon oddaljen od termostata. Telefon izračuna razdaljo, glede na to se termostat sam odloči, katero temperaturo bo izbral za primerjavo z ambientno. Šele nato glede na ambientno temperaturo odloči, ali bo ogreval prostor ali ne.



Slika 3 IoT naprava - termostat.

Glede na omrežje in aplikacijo lahko rečemo, da gre za protokol, ki nadgrajuje transportnega oz. aplikacijskega, t.j. WebSocket protokol. Predlagana arhitektura je postavljena z namenom zelo enostavnega razvoja in hkrati zelo varne komunikacije, saj so transportna sporočila vsakega klienta šifrirana do posrednika s TLS kot WSS, hkrati pa so tudi aplikacijska »data« sporočila med klienti šifrirana simetrično z AES, kar pomeni, da niti posrednik ne pozna vsebine komunikacije med klientoma.

5 Zaključek

Število IoT naprav narašča. S tem se večajo zahteve za uporabo različnih protokolov in prav tako so pred nami vedno novi varnostni problemi. Vse te je mogoče rešiti in narediti IoT kot robustno platfomo, ki bo poganjala svet v prihodnjih letih. Pri tem si lahko pomagamo s protokolom WebSocket, ki sem ga z varno in učinkovito arhitekturo želel predstaviti v tem članku.

Takšna arhitektura je novost v svetu spletnih komunikacij in pomeni hkrati popolno anonimnosti uporabnih naprav brez možnosti napada na naprave oz. aplikacijsko omrežje. Ne morem reči, da je takšna arhitektura primerna za vse mikrokrmilniške naprave, prav tako ne moremo govoriti, da je energetsko varčna ali brez redundance podatkov. Brez težav lahko potrdim, da je enostavna in učinkovita iz razvojnega vidika. Reši tudi problem IoT aplikacij, ki so povezane na splet. Drugi protokoli, ki delujejo na podoben način procesorsko učinkoviteje, na žalost niso podprti s strani brskalnikov. V tem članku tako rešim specifičen problem IoT aplikacij, ki zahtevajo popolno varnost in hkrati za svoje delovanje potrebujejo brskalnik ter podatkov ne smejo deliti z ostalimi omrežnimi ali strežniškimi napravami. Takšna arhiketura je prihodnost v luči zasebnosti podatkov in hrkati tudi iz vidika dejstva, da je na tržišču prisotnih čedalje več mobilnih naprav oz. računalnikov, ki za svoj operacijski sistem uporabljajo kar brskalnik.

Literatura

[1] A. E. Hakim, Internet of Things (IoT) System Architecture and Technologies, HTTPs://www.researchgate.net/publication/323525875, 2018

[2] Winjit, Demystifying Industrial IoT, HTTP://www.iotsense.io/wp-content/uploads/2017/05/WhitePaper-IIOT-Sense-.pdf, 2017

[3] K. Rose, S. Eldridge, L. Chapin, The Internet of Things: an Overview, HTTPs://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf , 2015

[4] G. L. Muller, HTML5 WebSocket protocol and its application to distributed computing, HTTPs://pdfs.semanticscholar.org/2556/ee58dce37d0e6d b10f3cef4241d67ae8c 090.pdf, 2013

[5] P. Lubbers, F. Greco, HTML5 WebSocket: A Quantum Leap in Scalability for the Web, HTTP://www.websocket.org/quantum.html, 2010

[6] I. Fette, Google, inc, The WebSocket Protocol, (IETF), HTTPs://www.ietf.org/rfc/rfc6455.txt, 2011

[7] T. A. Gamage, HTTP and Websockets: Understanding the capabilities of today's web communication technologies, HTTPs://medium.com/platform-engineer/web-api-design-35df8167460 , 2017

[8] J. Karlsson, Complete Guide To Node Client-Server Communication, HTTPs://medium.com/@joekarlsson/complete-guide-to-node-client-server-communication-b156440c029