Zbornik radova Fakulteta tehničkih nauka, Novi Sad

**UDK: 004.41**
**DOI: https://doi.org/10.24867/08BE33Bubnjevic**

# MASTER/SLAVE KOMUNIKACIONI PROTOKOLI ZA SCADA SISTEME

# MASTER/SLAVE COMMUNICATION PROTOCOLS FOR SCADA SYSTEMS

Marko Bubnjević, *Fakultet tehničkih nauka, Novi Sad*

**Oblast – ELEKTROTEHNIKA I RAČUNARSTVO**

**Kratak sadržaj –** *U radu je prezentovano programsko rešenje kojim simuliramo slave stranu VanComm Master/Slave protokola. Implementirano programsko rešenje može se koristiti za simulaciju komunikacije između master i slave strane, gde master strana šalje zahteve za čitanje podataka ili izvršavanje komandi, a slave strana te zahteve obrađuje i izvršava naređenja dobijena od master strane koristeći VanComm protokol.*

**Ključne reči:** *Protokol, SCADA, VanComm Protokol.*

**Abstract** – *The document presents a software solution which is used for simulaton of slave part of VanComm Master/Slave communication protocol. The implemented software solution may be used for simulation of communication between master and slave sides, where master side sends requests for reading of data or command execution, and slave part process and executes requests which came from master side while using VanComm communication protocol.*

**Keywords:** *Protocols, SCADA, VanComm Protocol.*

## 1. UVOD

Komunikacioni protokoli u SCADA sistemima predstavljaju ključnu ulogu prilikom komunikacije između master i slave strane sistema, gde master stranu uglavnom predstavlja aplikacija koja prikuplja podatke sa terena, obrađuje ih i dobijene obrađene podatke prikazuje unutar aplikacije, a slave stranu uglavnom predstavlja RTU koji šalje podatke sa terena, te izvršava komande poslate od strane mastera nad tačkama koje se nalaze "ispod" samog RTU-a.

Posmatraćemo razlike između tri veoma zastupljena protokola (DNP3, IEC-104 i VanComm) i pokušaćemo da izvučemo neke zaključke vezane za svaki protokol uz posebni akcenat na VanComm protokol.

Prilikom implementacije bilo kog protokola važna je mogućnost testiranja protokola pre puštanja u rad kako bi se izbegle neželjene situacije sa potencijalno opasnim ishodima koji mogu da donesu velike gubitke bilo u tehničkom ili ljudskom sektoru usled nekih previda i nepredviđenih situacija. RTU-evi su veoma skupi i teški za konfiguraciju i pristup, a RTU-evi ili programska rešenja koja služe za testiranje protokola često nisu opremljena tako da se mogu ispitati svi mogući slučajevi koji se mogu desiti na terenu.

**NAPOMENA:**
**Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Vukmirović, vanr. prof.**

Ovaj problem se može rešiti softverskim putem implementiranjem odgovarajućeg slave-a koji će se ponašati kao RTU sa svim ključnim karakteristikama. Postoje razni protokoli koji na svoj način vrše komunikaciju između komponenti SCADA sistema, te svaki od tih protokola ima svoje prednosti i mane, a samim tim i RTU-evi koji komuniciraju preko tog protokola mogu imati veću ili manju kompleksnost, te implementacija slave-a nosi veću ili manju težinu u smislu korisnosti same aplikacije. U daljim poglavljima opisaćemo implementaciju jednog takvog slave-a koji radi koristeći VanComm protokol i za razliku od testnog RTU-a podržava širi opseg zahteva i komandi, a zbog kompleknosti samog VanComm protokola donosi i veliku prednost i olakšanje za korisnika.

Postoje već implementirani simulatori koji se ponašaju kao slave za VanComm protokole ali su oni većinom limitirani i uglavnom ne predstavljaju dobro rešenje. Primer jednog takvog simulatora je Perl[1] simulator koji podržava samo jednostavne zahteve i komande kao što su čitanje analognih i digitalnih ulaza i izlaza, ali mu fale kompleksniji zahtevi kao što je to npr. komandovanje preko izlaznih tačaka koji će kao posledicu promeniti ulaznu tačku vezanu za taj izlaz.

Ukratko ćemo opisati redosled poglavlja i njihov sadržaj . U poglavlju „Teorijske osnove" opisan je SCADA sistem, kao i vrste njegovih protokola i njihove ključne razlike. Poglavlje „Implementirano programsko rešenje" predstavlja opis implementacije same aplikacije. Poglavlje „Verifikacija programskog rešenja" prikazuje rezultate testiranja implementiranog rešenja uz poređenje sa Perl simulatorom i njegovim performansama i mogućnostima. Unutar poglavlja „Zaključak" opisujemo moguća poboljšanja aplikacije, dok u poglavlju „Literatura" navodimo izvore koji su korišćeni prilikom izrade rada.

## 2. TEORIJSKE OSNOVE

Usled potrebe industrije za kontrolom procesa sa udaljenih lokacija osmišljen je SCADA [2] sistem. Osnovni problem koji SCADA rešava je kontrola više tačaka sa udaljene lokacije koje takođe mogu biti „razbacane" gledajući njihovu geografsku poziciju. Osnovni delovi svakog SCADA sistema su MTU (Master Terminal Unit) koji predstavlja osnovnu jedinicu tj. „mozak" SCADA sistema, komunikaciona oprema u koju spadaju svi različiti načini komunikacije preko različitih prenosnih medija kao što su npr. internet, radio veza, telefonska veza itd, RTU (Remote Terminal Unit) koji predstavlja jedinicu preko koje master strana dobavlja željene podatke za tačke koje se nalaze „ispod" RTU-a ili

komanduje njima i menja im stanja i vrednosti i istorijska baza podataka u kojoj se čuvaju sve promene sa terena.

Komunikacija između MTU-a i RTU-eva je implementirana koristeći razne protokole koji predstavljaju pravila tj. na koji način će se vršiti komunikacija između komponenti. Svi podaci se moraju preneti preko nekog protokola kako bi znali kako da iz pristigle poruke izvučemo željene podatke koji su pritom tačno interpretirani. Postoje dva načina prenosa poruka preko protokola, prva vrsta je inicirana od strane MTU-a i namenjena je RTU-u, a druga vrsta je inicirana od strane RTU-a i namenjena je MTU-u.

Svi protokoli ne podržavaju mogućnost da se RTU sam „javlja" MTU-u i zbog toga protokole takođe možemo podeliti na dve vrste: balansirane (RTU se može „javiti" sam) i nebalansirane (RTU se ne može „javiti" sam) protokole. Posmatrani su sledeći protokoli: DNP3 [3], IEC-60870-5-104 [4] i VanComm [5], te su opisane ključne razlike između gore navedenih protokola:

| **Protokol** | **DNP3** | **IEC-104** | **VanComm** |
|---|---|---|---|
| **Standard i organizacija** | DNP3 Users group | IEC standard | Ferranti |
| **Adresiranje uređaja** | 16-bitno adresiranje uređaja | Adresa Master-a je 8 bita, adresa RTU-a i objekta koji se posmatra 16 bita | Adresa RTU-a i Master-a su dužine 8 bita |
| **Mehanizmi za pouzdanost** | 16-bitni CRC, Start/Stop bit | Kontrolna polja | 8-bitna BCH kontrolna suma |
| **Fizički sloj i procedure sloja veze** | Nebalansirani mod, Balansirani mod, TCP/IP | Balansirani mod, TCP/IP | Nebalansiran mod, Sinhrona serijska komunikacija |
| **Aplikacioni sloj** | Kontrolisanje i obrada više tačaka po poruci;Više tipova podataka po poruci;Ima konfirmaciju događaja; | Jedan tip podataka po poruci; Kontrolisanje i obrada jedne tačke po poruci; | Kontrolisanje i obrada više tačaka po poruci;Jedan tip podataka po poruci; Potvrda da je master dobio odgovor (Station Check, Status Acknowledge,) |
| **Dominantno tržište** | Severna Amerika, Australija, Kina | Evropa | Amerika |

## 3. IMPLEMENTIRANO PROGRAMSKO REŠENJE

U ovom poglavlju prikazana je implementacija programskog rešenja tj. slave-a koji koristi Vancomm protokol, uz opis arhitekture samog rešenja. Slave prima zahteve sa master strane, obrađuje ih , izvršava zahteve i zatim vraća odgovarajuće odgovore nazad ka master-u.

### 3.1. Arhitektura realizovanog rešenja

Osnovne komponente implementiranog rešenja su:

- VanCommSlaveLibrary
- VanCommSlaveConsole
- CommonVanComm
- SlaveCommonVanComm
- VanCommDataLoader

VanCommSlaveLibrary komponenta predstavlja glavnu komponentu implementiranog rešenja. Ona je zadužena za komunikaciju sa master-om, učitavanje konfiguracije iz NMS-a, obrađivanje i procesiranje zahteva koji su stigli sa master-a, kao i slanje odgovora nakon što je zahtev uspešno ili neuspešno obrađen.

VanCommSlaveConsole komponenta predstavlja konzolnu aplikaciju pomoću koje startujemo slave. Možemo izabrati vrstu konfiguracije koju želimo da učitamo (iz XML fajla ili sa NMS servisa), zatim pomoću VanCommSlaveConsole možemo menjati vrednosti za željene konfigurisane analogne ili diskretne tačke, takođe možemo menjati vrednost UnitStatus-a za odgovarajući RTU i kad završimo sa korišćenjem istog prekinuti rad slave-a.

CommonVanComm komponenta je biblioteka koja sadrži sve zajedničke podatke i metode važne za rad celog sistema kao što su na primer: Enums, ByteParityCalculator, SecurityCoding itd.

SlaveCommonVanComm je biblioteka koja formira odgovore za odgovarajuće zahteve. to znači da unutar SlaveCommonVanComm komponente formiramo frejmove, zaglavlja i poruke koje se kasnije koriste kao odgovor na zahtev.

VanCommDataLoader je komponenta zadužena za popunjavanje memorije odgovarajućim podacima iz XML fajlova.

### 3.2. Implementacija realizovanog rešenja

Opisaćemo na koji način je implementirano programsko rešenje. Podelićemo poglavlje na više celina kako bi lakše ispratili sve funkcionalnosti slave-a kao i na koji način je izvršena njihova implementacija. Programsko rešenje implementirano je u C# programskom jeziku.

#### 3.2.1. Komunikacija sa Master-om

Komunikacija između master-a i slave-a se uspostavlja preko TCP konekcije koristeći socket-e. Glavna klasa koja je zadužena za sve aspekte komunikacije se naziva CommunicationEngine. Pre nego što master pokuša da uspostavi konekciju sa slave-om potrebno je kreirati i pokrenuti TCPServer-e koje iščitavamo iz konfiguracije.

Klasa TCPServer je klasa koja je zadužena za kreiranje soketa, njihovo održavanje, pravovremeno gašenje ili stopiranje ako je to potrebno. TCPServer takođe prihvata i uspostavlja konekciju sa master-om, ili gasi konekciju ukoliko dođe do neke greške prilikom komunikacije. Svaki TCPServer je pokrenut na jedinstvenoj kombinaciji IP adrese i porta.

ConnectionState je pomoćna klasa služi za kontrolu stanja trenutne konekcije između master-a i pokrenutih TCPServer-a na slave-u.

#### 3.2.2. Konfiguracija podataka

Svi podaci koje slave koristi za svoj rad se iščitavaju iz konfiguracije, kako bi te podatke mogli predstaviti što jednostavnije i bolje. Bilo je potrebno modelovati podatke i kreirati odgovarajuće klase koje sadrže odgovarajuće delove podataka koji zajedno čine jednu veliku celinu.

### 3.2.3. Parsiranje i procesiranje pristiglih zahteva

Klasa Manager je klasa koja kontroliše ceo rad slave-a. Poziva metode za čitanje podataka u konfiguraciju, startuje CommunicationEngine i startuje RequestReceiver koji je zadužen za parsiranje i obradu pristiglih zahteva. Manager takođe stopira servise, ali glavna uloga Manager-a je da uz pomoć event-ova pokreće parsiranje pristiglog zahteva i šalje nazad odgovore koji su formirani od strane slave-a.

Klasa RequestReceiver je glavna klasa za parsiranje pristiglih zahteva od strane master-a. Svaki tip zahteva ima odgovarajući procesor koji kasnije ukoliko do takvog zahteva dođe parsira zahtev i nakon toga formira odgovor. Procesori se mapiraju pomoću dictionary-ja requestProcessorsMap, dok se linkovi između master-a i slave-a mapiraju pomoću dictionary-ja ipPortToLinkMap. Popunjavanje samih dictionary-ja se vrši u metodama ConfigureLinksMap i PopulateRequestProcessorMap.

Svaki zahtev koji pristigne sa master-a na slave se procesuira u metodi ProcessRequest. Metoda prvo parsira sirove podatke tako da popuni sva potrebna bolja i konvertuje sirove podatke u čitljiv zahtev. Zatim iz zahteva izvlačimo informacije koje su nam potrebne kako bi formirali odgovor. Pomoću polja StationAddress proveravamo da li imamo konfigurisan RTU pod tom adresom, proveravamo OperationCode i iz dictionary-ja procesora prozivamo odgovarajući procesor koji je zadužen za tu vrstu zahteva. Procesor dalje obrađuje zahtev i formira odgovor, vraća odgovor nazad, te se odgovor šalje master strani koja je poslala zahtev.

### 3.2.4. Formiranje odgovora za odgovarajući zahtev

Osnovnu metodu predstavlja metoda Process koja kao parametre prima formiran zahtev od strane master-a i RtuSlaveModel koji predstavlja RTU nad kojim se izvršava neka akcija ili čitanje podataka. Svaki procesor zadužen je za tačno jedan tip zahteva, tj. za formiranje odgovarajućeg odgovora.

### 3.2.5. Konzolna aplikacija

Konzolna aplikacija nam služi za pokretanje samog slave-a, kao i za još neke dodatne funkcionalnosti koje ćemo objasniti u ovom podpoglavlju. Prilikom pokretanja prvo je potrebno da izaberemo odakle želimo da učitamo konfiguraciju koja će napuniti našu memoriju podacima.

Pored učitavanja konfiguracije preko konzole imamo mogućnosti da setujemo analogni ulaz, analogni izlaz, UnitStatus bajt, kao i da ugasimo slave. Ove opcije su omogućene kako bi preko konzole mogli da simuliramo spontane promene koje će kasnije master strana prepoznati i obraditi.

## 4. VERIFIKACIJA PROGRAMSKOG REŠENJA

VanComm slave služi za simulaciju velikih industrijskih SCADA sistema sa više RTU-eva i više tačaka nakačenih na te RTU-eve. U ovom poglavlju ćemo testirati skalabilnost implementirane aplikacije tj. maksimalni broj RTU-eva i tačaka gde će naše programsko rešenje raditi bez problema i zadovoljiti standarde, takođe izvršićemo poređenje sa već postojećim VanComm slave simulatorom ,,Strawberry Perl" i pokazati zašto je naše implementirano rešenje bolje, tj. koja funkcionalna proširenja smo dobili našim rešenjem koja ranije nisu bila podržana.

Kapacitet implementiranog rešenja u mnogome zavisi od kapaciteta računara na kojem smo pokrenuli VanComm slave, kao i od sposobnosti mreže preko koje se izvršava komunikacija master-a i slave-a, zbog toga bi bilo bespotrebno testirati performanse u obliku merenja brzine formiranja odgovora, ili brzine slanja svih odgovora na sve zahteve pristiglih u nekom vremenskom periodu. Jedina ,,jedinica" kojom možemo dati težinu našem radu je kao što smo već rekli broj RTU-eva i tačaka gde će VanComm slave i dalje raditi bez problema. U tabeli prikazaćemo scenarija za koje je slave testiran uz napomenu da se za svaki RTU vršilo periodično skeniranje svih tačaka na 30 sekundi i za svaki zahtev je stigao odgovarajući odgovor, a slave je radio u periodu od 10 minuta. Memorija koju zauzima slave prilikom inicajalizovanja podataka za virtuelne RTU-eve je zanemarljiva i nećemo je posmatrati kao faktor rada slave-a. Testiranje smo vršili na računaru sa 24.3 GB RAM memorije i Inter Core i3-3210 procesoru 3.20 Ghz.

| **Broj RTU-eva** | **Ukupan broj analognih i digitalnih tačaka** | **Prosek obrađenih zahteva u periodu od 30 sekundi** | **Slave pravilno radi** |
|---|---|---|---|
| 10 | 2000 | 133,44 | Da |
| 20 | 3000 | 241,61 | Da |
| 30 | 4000 | 273,42 | Da |
| 40 | 5000 | **386,17** | Da |
| 50 | 6000 | 317,28 | **NE** |

Prilikom testiranja sa 50 RTU-eva i 6000 tačaka došlo je do grešaka prilikom odgovora na zahteve i za taj slučaj možemo reći da VanComm slave nije kompetentan. VanComm slave je radio najbrže sa konfiguracijom od 40 RTU-eva i 5000 analognih i digitalnih tačaka gde je u proseku na svakih 30 sekundi obrađivao 386,17 različitih zahteva.

Strawberry Perl je aplikacija koja predstavlja slave stranu za više različitih protokola kao što su DNP3, Series5, Modbus i VanComm, dok ćemo se mi fokusirati samo na deo vezan za VanComm protokol. Aplikacija sama po sebi ima dosta nedostataka koji su rešeni implemenitranjem našeg VanComm slave-a.

Strawberry Perl podržava terminal server-e ali samo sa jednim RTU-em nakačenim ,,ispod" tog terminal server-a, ili drugačije rečeno samo jedan RTU ukoliko ne postoje nikakvi terminal server-i, dok naša aplikacija nema to ograničenje i moguće je simuliranje više RTU-eva odjednom i konektovanje master strane na njih uz omogućeno paralelno izvršavanje zahteva i vraćanje odgovora na zahteve. Perl podržava promenu i iščitavanje analognih tačaka u potpunosti, ali prilikom promene ili čitanja digitalnih tačaka moguće je promeniti ili iščitati samo nultu grupu, tj. ne postoji ni jedna više grupa sem nulte grupe što predstavlja veliki nedostatak koji je rešen implementacijom našeg slave-a. Perl simulator pored čitanja analognih i statusnih tačaka podržava report by

exception, read internal status, raise lower immediate execute, control select, control execute i control immediate execute zahteve, ali ne podržava još dosta važnih zahteva kao što su prvenstveno setpoint select, setpoint execute i setpoint immediate execute koji programski izvršavaju promene na analognim izlazima i predstavljaju pandan control zahtevima koji se kao što je već rečeno koriste prilikom promene digitalnih izlaza i tačaka vezanih za te izlaze.

Starberry Perl simulator takođe ne podržava zahtev za restartovanje podataka unutar RTU-a (RTU restart zahtev), zahtev za brisanje internih bajtova (Clear internal statuses) i zahtev za proveru stanja samog RTU-a, tj. komunikacije između komponenti (Station Check). Jasno je posle upoređivanja našeg rešenja i Perl simulatora da je naše rešenje mnogo kompleksnije i upotrebljivije za korisnika, te da predstavlja unapređenje u odnosu na prethodna rešenja koja se mogu naći.

## 5. ZAKLJUČAK

Protokoli predstavljaju ključni deo sistema komunikacije unutar SCADA sistema te je njihova implementacija i unapređivanje uvek aktuelna tema. Uporedili smo tri veoma zastupljena protokola i videli na koji način su implementirani, te koje su njihove prednosti i mane, kao i gde se najviše koriste posmatrajući geografsko područije i infrastrukturu koju ti protokoli koriste.

Cilj ovog rada je bio razvoj aplikacije koja će uspešno simulirati rad više RTU-eva koji primaju zahteve od master strane, obrađuju te zahteve i izvršavaju ih uz vidnu promenu „na terenu" koja se može ispratiti iščitavanjem podataka „sa terena". Važna stavka same aplikacije je takođe i obrada i odgovor na zahteve u neko prihvatljivo vreme koje bi nalikovalo na realni sistem, kao i sama skalabilnost aplikacije kako bi taj „realni sistem" mogao biti što veći kako bi mogli testirati scenarije koji bi se mogli desiti u realnom životu. Aplikacija predstavlja unapređenje u odnosu na prethodno implementirana rešenja koja imaju velike nedostatke i kod kojih VanComm protokol nije u potpunosti iskorišćen, tj. nisu iskorišćene sve njegove funkcionalnosti i karakteristike koje ga čine pogodnim za upotrebu.

Moguća unapređenja našeg programskog rešenja su vezana za zahteve koji nisu obrađeni u trenutnoj verziji VanComm slave-a, a to su: 16-bit Accumulator Request, 32-bit Accumulator Request, Clear Accumulators, Defrost Accumulators, Freeze Accumulators i Freeze and Clear Accumulators, gde su akumulatori memorije koje možemo oblikovati po sopstvenim potrebama, npr. ako nam treba podatak koji zauzima 64 bita možemo spojiti dva akumulatora i tako iščitati podatak.

Zahtevi koji takođe nisu podržani od strane našeg slave-a su i SOE Dump, SOE Capture Line, SOE Line Delay, SOE Acknowledge, SOE Purge Events i RTU Clock Set, gde SOE predstavlja Sequence of Events, tj. skup nekih događaja i ovim zahtevima manipulišemo tim događajima, a RTU Clock Set postavlja tačno vreme na strani RTU-a ukoliko je potrebno sinhronizovati vremena na master-u i slave-u. Još jedno od mogućih unapređenja je i menjanje konzolne aplikacije sa nekom modernijom vrstom korisničkog interfejsa koji se može implementirati koristeći npr. WPF tehnologiju.

## 6. LITERATURA


[1] Perl simulator. https://www.perl.org/ (Accessed 23.09.2019.)

[2] What is SCADA? https://inductiveautomation.com/resources/article/what-is-scada . (Accessed 24.09.2019.)

[3] DNP3 SPECIFICATION, Version 2.01, DNP Users Group, 2007.

[4] P.Matoušek, *Description and analysis of IEC 104 Protocol*, Brno University of Technology, 2017.

[5] *Appendix B, Ferranti International Controls, Message Standard: Van-Comm,* Ferranti International Controls, 1986.


### Kratka biografija:


**Marko Bubnjević** rođen je 31.05.1994. godine u Novom Sadu. Završio je osnovnu školu „Dušan Radović" 2009. godine, a Elektrotehničku školu „Mihajlo Pupin" u Novom Sadu 2013. godine. Osnovne akademske studije na Fakultetu Tehničkih Nauka u Novom Sadu završio je 2018 godine, a iste godine upisao master akademske studije na Fakultetu Tehničkih Nauka, smer Primenjeno softversko inženjerstvo. Ispunio je sve obaveze i položio je sve ispite predviđene studijskim programom sa prosečnom ocenom od 8.83.

Kontakt: marko.bubnjevic94@hotmail.com