FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVOD ZA ELEKTRONIČKE SUSTAVE I OBRADBU INFORMACIJA

SUSTAVI ZA PRIJENOS I TELEMERIJU

SEMINAR

RPC

REMOTE PROCEDURE CALL

DANIJEL BREZAK
0036382933

LIPANJ, 2005.

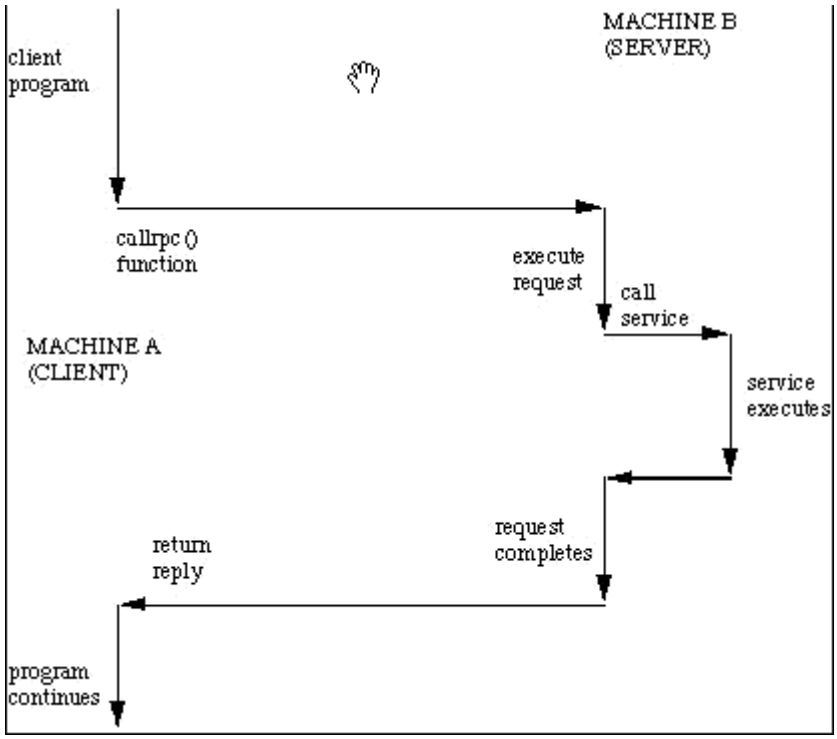
SADRŽAJ

1. RPC MODEL

RPC (remote procedure call) model je vrlo sličan local procedure call modelu.
U local slučaju pozivatelj postavlja argumente proceduri na nekoj vrlo poznatoj lokaciji npr. registar rezultata. Tada kontrolaprebacuje proceduri i eventualno se dobiva nekog vremena natrag. U toj situaciji rezultati procedure se izvlače sa dobro poznate lokacije i pozivatelj nastavlja svoj rad.

Sličnost RPC modela je u tome da kotrolu u nekim trenucima imaju dva procesa – jedan je korisnički, a drugi serverski. Tj. korisnički proces pošalje pozivnu poruku serverskom procesu i čeka (zamrzne se) povratnu poruku tj. odgovor. Pozivna poruka od strane korisnika sadrži parametre procedure i ostale informacije. Povratna poruka sadrži rezultate procedure i ostale informacije. Nakon što se dobije odgovor rezultati procedure se izvlače i korisnik nastavlja svoj rad.

Na strani servera proces čeka dolazak pozivne poruke. Kada ona stigne serverski proces izvlači parametre procedure, izračuna rezultate, šalje povratnu poruku i nastavlja čekati sljedeću pozivnu poruku.



Sl.1. RPC model

Primječuje se da je u ovom modelu , samo jedan od dva procesa aktivan u danom trenutku. Ipak, ovaj model je dan samo kao primjer. RPC protokol nema ograničenja na implementirane modele. Npr. implementacijom se može odabrati da se RPC poziv izvodi asinkrono tako da korisnik može nastaviti svoj rad dalje dok čeka odgovor sa servera. Sljedeća mogućnost je da server stvori neki zadatak da procesira ulazni zahtjev tako da može primati daljnje zahtjeve.

2. PRIJENOS I SEMANTIKA

RPC protokol (RFC 1050) je neovisan o prijenosnom protokolu tj. RPC ne zanima kako je poruka prenešena s jednog u drugi proces. Protokol se bavi samo specifikacijom i interpretacijom poruke.

Važno je naglasiti da RPC ne nastoji implementirati nikakvu pouzdanost i da aplikacija mora biti upoznata s tim koji se transportni protokol koristi u RPC-u.. Ako zna da radi na prijenosu največe pouzdanosti poput TPC/IP transporta, tada je večina posla za tu asplikaciju završila. U drugu ruku ako radi sa UDP/IP prijenosom mora implementirati svoju vlastitu retransmisiju i time-out osiguranje jer RPC sloj ne pruže te uslige.

Zbog neovisnosti o transportu, RPC protokol ne pridružuje nikakvu semantiku udaljenim procedurama ili njihovom izvođenju. Semantike se uključuju i moraju biti definirane od strane nižeg transportnog protokola. Npr. Uzmimo u obzir RPC koji radi sa nisko-pouzdanim prijenosom poput UDP/IP. Ako procedura retransmira poruku nakon kratkog time-out-a jedina stvar koju može zaključiti ako ne primi odgovor je da je procedura izvršena nula ili više puta. Ako dobije odgovor može zaključiti da se procedura izvršila barem jedanput.

Server možda želi zapamtiti prethodno dozvoljeni zahtjev od strane korisnika i ne izvršavati ih redom da bi osigurao neki stupanj izvršavanja semantike najviše jedanput. Server to može učiniti iskorištavajući transakciju ID-a koji je pakiran sa svakim RPC zahtjevom. Glavno iskorištenje te transakcije je u korišničkom sloju u spajanju odgovora sa zahtjevom. Kako god, korisnička aplikacija može odabrati da ponovno iskoristi neki prethodni transakcijski ID kada retransmira zahtjev. Serverka aplikacija znajuči tu činjenicu može odabrati da zapamti taj ID nakon što odobri zahtjev i ne dozvoli zahtjev sa istim ID-om kako bi osigurala izvršenje semantike najviše jedanput. Serveru nije dozvoljeno da prouči taj ID u bilo kojem slučaju osim da ispituje jednakost.

U drugu ruku ako se koristi pouzdani prijenos kao TCP/IP, aplikacija može zaključiti iz odgovora da je procedura izvršena točno jedamput, ali ako ne dobije odgovor ne može pretpostaviti da je udaljena procedure nije izvršena. Uočimo da čak i kod conection-oriented protokola kao TCP, aplikacija još uvjek treba time-out i rekonekcije da bi se nosila sa rušenjima servera.

Postoje druge mogućnosti za transport pored datagrama ili conection-oriented protokola. Na primjer request-reply protokol kao VMPT je možda najprirodniji način prijenosa za RPC.

Bilješka: U SAN-u RPC je trenutno implementiran na TCP/IP i UDP/IP protokolima.

3. NEZAVISNOST SPAJANJA I SASTAVLJANJA

Čin spajanja korisnika sa serverom nije dio RPC (remote procedure call) specifikacija. Ta važna i potrebna funkcija je ostavljena nekom višem softvware-u.

Implementatori bi trebali zamisliti RPC protokol kao jump-subroutine instrukciju mreže (JSR). Loader (binder)tj. poveznik čini JSP korisnim i on sam koristi JSR da obavi zadatak. Isto tako i mreža čini RPC korisnim, koristeći RPC da obavi svoj zadatak.

4. AUTENTIFIKACIJA

RPC protokol daje polja potrebna da se korisnik registrira kod servera i vice-versa. Sigurnosni i pristupni kontrolni mehanizam može biti izgrađen na vrhu poruke autentifikacije. Podržano može biti nekoliko različitih autentifikacijskih protokola. Polje u RPC header-u pokazuje koji protokol se koristi. Više o autentifikaciji u poglavlju 7.

5. ZAHTJEVI RPC PROTOKOLA

RPC protokol se mora pobrinuti za sljedeće:

1. Jedinstvenu specifikaciju procedure koju treba pozvati
2. Provizije za sparivanje odgovorne i zahtjevne poruke
3. Provizije za autentifikaciju korisnika servisu i vice-versa

Pored tih zahtjeva, opcije koje detektiraju sljedeće je vrijedno podržati zbog roll-over-errors protokola, implementacijskih grešaka (bug-ova), korisničkih grešaka i mrežne administracije:

1. Greške sparivanja RPC protokol a
2. Greške sparivanja verzija udaljenih programa protokola
3. Greške u protokolu (kao pogrešno specificiranje parametara procedure)
4. Uzroka zbog kojih autentifikacija nije uspjela
5. Bilo koji drugih uzroka zašto željena procedura nije bila pozvana.

5.1. RPC PROGRAMI I PROCEDURE

RPC pozivna poruka ima tri nedodjeljena polja: broj udaljenog programa, broj verzije udaljenog programa i broj udaljene procedure. Ta tri polja jedinstveno identificiraju proceduru koja će biti pozvana. Brojevi programa su dodjeljeni od strane nekog centralnog autoriteta (kao SUN). Jednom kad implementator dobije broj programa, može implementirati svoj udaljeni program. Prva implementacija će najvjerojatnije imati broj verzije 1. Zbog toga što se većina novih protokola razvije u bolje, stabilnije i zrelije protokole, polje verzije u pozivnoj poruci identificira koju verzija protokola korisnik koristi. Brojevi verzije omogučavaju starim i novim protokolima da komuniciraju što je moguće više kroz isti serverski proces.

Broj procedure identificira proceduru koja se poziva. Ovi brojevi su dokumentirani u specifičnim specifikacijama programskog protokola. Npr. datotekovni servisni protokol može specificirati da je broj procedure 5, čitanje („read“), a broj procedure 12 pisanje („write“). Isto tako kao što udaljeni programski protokol može promjeniti nekoliko verzija, tako se i sam protokol RPC poruke može promjeniti. Zbog toga pozivna poruka također sadrži i broj verzije RPC protokola, koji je uvijek jednak 2 za verziju ovdje opisanu.

Povratna poruka na pozivnu poruku ima dovoljno informacija da ralikuje stanja sljedećih grešaka:

1. Udaljena implementacija RPC-a ne razumije protokol verzije 2. Vraćaju se najniži i najviši brojevi podržane PC verzije.
2. Udaljeni program nije dostupan na udaljenom sistemu.
3. Udaljeni program ne podržava zahtjevani broj verzije. Vraćaju se najniži i najviši brojevi verzije podržanog udaljenog programa
4. Zahtjevani broj procedure ne postoji.
5. Parametri pozivnoj proceduri liče na smeće gledano sa stajališta servera.

5.2. AUTENTIFIKACIJA

Provizija o autentičnosti pozivatelja serveru i vice-versa je dana kao dio RPC protokola. Pozivna poruka ima dva polja za dokazivanje autentičnosti: kredibilitet i verifikacija. Povratna poruka ima jedno polje za dokazivanje autentičnosti: povratna verifikacija.

Interpretacija i semantika podataka sadržanih unutar polja za dokazivaje autentičnosti je specificirana od strane individualnih, nezavisnih specifikacija protokola za dokazivanje autentičnosti.

Ako su parametri autentičnosti odbijeni, povratna poruka sadrži informaciju koja sadrži odgovor zbog čga su odbijeni.

5.3. DODJELJIVANJE BROJEVA PROGRAMA

Brojevi programa su dani u grupama od 20000000 hexadecimalno (536870912 decimalno) prema sljedećoj tablici:

0 - 1fffffff	definirani od SUN-a
20000000 - 3fffffff	definirani od korisnika
40000000 - 5fffffff	prijelazno
60000000 - 7fffffff	rezervirano
80000000 - 9fffffff	rezervirano
a0000000 - bfffffff	rezervirano
c0000000 - dfffffff	rezervirano
e0000000 - ffffffff	rezervirano

Prva grupa je niz brojeva administriranih od strane SUN Mycrosystems i trebali bi biti identični za sve stranice. Drugi niz je za aplikacije tipične za određene stranice. Taj niz je primarno namjenjen za debuggiranje novih programa. Kada stranica razvije aplikaciju koja može biti od općeg interesa, toj aplikaciji trebao se dodjeliti broj iz prve grupe. Treča grupa je za aplikacije koje brojeve programa generiraju dinamički. Posljednje grupe su rezervirane za daljnju upotrebu i nebi smjele biti upotrebljenje.

5.4. OSTALE UPOTREBE RPC PROTOKOLA

Predviđena upotreba ovog protokola je za pozivaje udaljenih procedura. Drugim riječima, svaka pozivna poruka je sparena sa povratnom porukom. Ipak, protokol sam po sebi je protokol za prolazak poruka sa kojim ostali (ne RPC) protokoli mogu biti implementirani. SUN trenutno koristi, ili bolje reći iskorištava RPC protokol za poruke za sljedeća dva (ne RPC) protokola: batching (pipelining) i broadcast RPC.

5.4.1. Batching

Batching dozvoljava korisniku da šalje arbitražno velike nizove pozivnih poruka serveru. Batching tipično koristi pouzdani byte stream protokole (TCP/IP) za transport. U slučaju baching-a, korisnik nikada ne čeka na odgovor od servera, a server ne šalje odgovore na batch zahtjeve. Sekvenca batch poziva obično je prekinuta od strane legitimirano RPC-a s ciljem da se pročisti pipeline (sa pozitivnom potvrdom).

5.4.2. Broadcast RPC

U protokolima baziraima na broadcast RPC-u, korisnik šalje broadcast paket mreži i ćeka na brojne odgovore. Broadcast RPC koristi nepouzdan, protokol baziran na paketima (UDP/IP) za transport. Server koji podržava broadcastz protokol odgovara jedino kad je zahtjev uspješno procesiran i kad nema grešaka u prijenosu. Broadcast koristi Port Maper RPC srvis kako bi postigao svoju semantiku.

6. PORUKE RPC PROTOKOLA

Ovdje ćemo definirati poruke RPC protokola pisane XDR data opisnim jezikom. Ova poruka je definirana kao top-down stil.

```
enum msg_type {
   CALL  = 0,
   REPLY = 1
};

/*
 * Odgovor na pozivnu poruku može poprimiti dvije forme:
 * Poruka je ili prihvaćena ili odbijena.
 */
enum reply_stat {
   MSG_ACCEPTED = 0,
   MSG_DENIED   = 1
};

/*
 * S obzirom da je pozivna poruka prihvačena, slijedi
 * status pokušaja da se pozove udaljena procedura.
 */
enum accept_stat {
   SUCCESS       = 0, /* RPC proveden uspješno*/
   PROG_UNAVAIL  = 1, /* ne postoji traženi program  */
   PROG_MISMATCH = 2, /* nepodržana verzija#  */
   PROC_UNAVAIL  = 3, /* program ne podržava proceduru */
   GARBAGE_ARGS  = 4  /* procedura ne može dekodirati parametre */
};

/*
 * Razlozi zašto je poruka odbijena:
 */
enum reject_stat {
   RPC_MISMATCH = 0, /* Broj RPC verzije != 2          */
   AUTH_ERROR = 1    /* Neuspjela autentifikacija pozivatelja */
};

/*
 * Zašto nije uspjela autentifikacija:
 */
enum auth_stat {
   AUTH_BADCRED      = 1,  /* loš kredibilitet */
   AUTH_REJECTEDCRED = 2,  /* korisnik mora započeti iznova*/
   AUTH_BADVERF      = 3,  /* loš verifikator    */
   AUTH_REJECTEDVERF = 4,  /* verifikator istekao  */
   AUTH_TOOWEAK      = 5   /* odbijeno zbog sigurnosnih razloga */
};

/*
 * RPC poruka:
 * Sve poruke počinju sa identifikacijom transakcije,xid,
 * slijedi dvoručna diskriminirajuča unija. Diskriminanta unije je
 * msg_type koji prebacuje na jedan ili drugi tip poruke
 * Xid povratne poruke se uvijek poklapa sa onim u inicijalnoj
 * pozivnoj poruci
 */
```

```
struct rpc_msg {
   unsigned int xid;
   union switch (msg_type mtype) {
   case CALL:
      call_body cbody;
   case REPLY:
      reply_body rbody;
   } body;
};

/*
 * Tijelo pozivne RPC poruke:
 * U verziji 2 specifikacije RPC protokola rpcvers mora biti
 * jednak 2.  Polja prog, vers, i proc definiraju udaljeni
 * program, broj njegove verzije i proceduru unutar koje je udaljeni
 * program pozvan. Nakon ovih polja dolaze dva parametra
 * autentifikacije: cred (kredibilitet autentifikacije)
 * i verf (verifikacija autentifikacije).Iza njih slijede parametri
 * udaljene procedure koji su specificirani pomoću specifičnog
 * programa protokola.
 */
struct call_body {
   unsigned int rpcvers;          /* mora biti 2*/
   unsigned int prog;
   unsigned int vers;
   unsigned int proc;
   opaque_auth cred;
   opaque_auth verf;
   /* specifični parametri procedure počinju ovdje*/
};

/*
 * Tijelo odgovora na RPC zahtjev:
 * Pozivna poruka je ili prihvačena ili odbijena.
 */
union reply_body switch (reply_stat stat) {
case MSG_ACCEPTED:
   accepted_reply areply;
case MSG_DENIED:
   rejected_reply rreply;
} reply;

/*

 * Odgovor na RPC zahtjev koji je prihvačen od servera:
 * moguća je pogreška iako je zahtjev prihvaćen.
 * Prvo polje je verifikacija autentičnosti koju server generira
 * kako bi validirao sebe pozivatelju. Nju slijedi unija čija
 * diskriminanta je accept_stat.
 * SUCCESS dio unije jedefiniran protokolom.
 * PROG_UNAVAIL, PROC_UNAVAIL, i GARBAGE_ARGS
 * dijelovi unije su void.PROG_MISMATCH dio specificira
 * najniži i najviši broj verzije udaljenog programa
 * podržanog od servera.
 */
struct accepted_reply {
   opaque_auth verf;
   union switch (accept_stat stat) {
   case SUCCESS:
      opaque results[0];
      /*
```

```
         * rezultati specifične procedure su ovdje
         */
      case PROG_MISMATCH:
         struct {
            unsigned int low;
            unsigned int high;
         } mismatch_info;
      default:
         /*
          * Void.  Cases include PROG_UNAVAIL, PROC_UNAVAIL,
          * and GARBAGE_ARGS.
          */
         void;
      } reply_data;
};

/*
 * Odgovor na RPC zahtjev koji je odbijen od servera:
 * Odbijen može biti zbog dva razloga:  ili na serveru
 * ne radi kompatibilna verzija RPC protokola (RPC_MISMATCH),
 * ili server odbija autentifikaciju pozivatelja (AUTH_ERROR)
 * U slučaju nepodudaranja RPC verzija, server vraća najniži i
 * najviši broj podržane RPC verzije.
 * U slučaju odbijene autentifikacije vračen je status o odbijanju.
 */
union rejected_reply switch (reject_stat stat) {
case RPC_MISMATCH:
   struct {
      unsigned int low;
      unsigned int high;

   } mismatch_info;
case AUTH_ERROR:
   auth_stat stat;
};
```

7. AUTENTIFIKACIJSKI PROTOKOLI

Kao što je i prije navedeno, autentifikacijski parametri su ograničeni, ali krajem otvoreni prema ostalim RPC protokolima. Ova sekcija definira neke vrste autentifikacije implementirane (i podržane) od strane SUN-a. Ostale stranice slobodne su da izmisle nove načine autentifikacije, sa istim pravilima dodjeljivanja brojeva vrsti autentifikacije kakva vrijedi i za dodjeljivanje brojeva programima.

7.1. NULL AUTENTIFIKACIJA

Često pozivi moraju biti provedeni gdje pozivatelj nezna tko je ili server ne iteresira tko je pozivatelj. U tom slučuju, vrijednosti kredibiliteta, vrifikacije i povratne verifikacije u RPC porruci su postavljene na „AUTH_NULL“. Bytovi tijela ograničen autentifikacije su nedefinirani, a preporučena dužina im je nula.

7.2. UNIX AUTENTIFIKACIJA

Pozivatelj udaljene procedure može zatražiti da se identificira kao što je identificiran na UNIX sistemu. Vrijednost kreditibirane diskriminante RPC pozivne poruke je „AUTH_UNIX“ Vrijednost diskriminate povratne verifikacije primljene u povratnoj poruci od strane servera mogu biti „AUTH_NULL“ ili „AUTH_SHORT“.
U slučaju „AUTH_SHORT“ , bytovi stringa povratne verifikacije okodiraju ograničenu strukturu. Ta nova ograničena struktura može biti proslijeđena umjesto orginala „AUTH_UNIX“ vrste kredibiliteta. Server održava cache koji mapira kratke ograničene strukture prema originalnim kredibilitetima pozivatelja. Pozivatelj može sačuvati mrežni bandwidth i cikluse serverskog CPU-a koristeći nove kreditacije.Server može isprazniti kratke ograničene strukture u bilo kojem trenutku. Ako se to dogodi RPC poruke će biti odbijene zbog greške kod autentifikacije. Razlog će biti „AUTH_REJECTED“. U tom slučaju pozivatelj može pukušati sa originalnim „AUTH_UNIX“ načinom kredibilizacije.

7.3. DES AUTENTIFIKACIJA

UNIX autentifikacija opterećena je sa sljedeća va proble:

1. Imenovanje je UNIX orjentirano
2. Nema verifikacije tako da kredibiliteti mogu biti bez problema lažirani

DES autentifikacija pokušava riješiti ta dva problema

7.3.1. Imenovanje

Prvi problem se rješava tako da se pozivatelj adresira jednostavnim nizom charactera umjesto operativom sistemu tipični integera. Taj niz charactera poznat je kao „netname“ ili kao mrežno ime pozivatelja. Serveru nije dozvoljeno da interpretira sadržaj pozivateljevog imena na bilo koji drugi način osim da se identificira pozivatelj. Iz navedenog razloga netnames

trebaju biti jedinstveni za svakog pozivatelja na Internetu. Generiranje netnamesa ovisi o pojedinom operativnom sistemu koji ima implementiranu DES autentifikaciju. Bitno je da osiguraju jedinstvenost kod poziva udaljeng servera.

7.3.2. Verifikatori DES autentifikacije

Za razliku od UNIX autentifikacije, DES autentifikacija ima verifikatore kako bi server moga vrednovati korisnikov kredibilitet. Sadržaj verifikatora je primarno enkriptirani timestamp. Server može dekriptirati taj timestamp, i ako je blizu stvarnom vremenu, to bi značilo da ju je korisnik enkriptirao ispravno. Jedini način da ju enkriptira ispravno je da zna „conversation key“ (konverzacijski ključ) RPC –a i samo takav korisnik je pravi orisnik. Conversation key je DES key koji korisnik generira i obavještava servr o njemu pri svojem prvom RPC pozivu. Taj ključ je enkriptiran tako da se koristi javna key shema pri prvoj transakciji. Određena public key shema određena pri DES autentifikaciji je Diffie-Hellman shema sa 128-bitnim ključem. Da bi sve to radilo server i koriasnik trebaju imati istu predodžbu o trenutnom vremenu. Ako se ne može garantirati sinkronizacija mrežnog vremena, tada se korisnik može sinkronizirati sa serverom prije početka razgovora konzultirajući Internet Time Seerver.

Način na koji server određuje dali je korisnisnikov timestamp validan je donekle komplicirano. Za bilo koji osim prvog prijenosa server jednostavno provjerava sljedeće dvije stvari:

1. timestamp je veći od onog prethodno viđenog od istog korisnika.
2. timestamp nije istekao

Prilikom prvog prijenosa server samo provjerava dali je istekao timestamp. Kada bi to bilo sve što bi se provjeravalo tada bi korisnik mogao poslati slučajne podatke i imao šanse da uspije. Kao dodatna provjera, korisnik šalje enkrioptirani podatak poznat kao „window verifier“ koji mora biti jednak window minus 1 ili će server odbaciti kredibilitet.

Korisnik pored toga mora provjeriti dali je verifikator vraćen od server legitiman. Server korisniku šalje enkriptiran timestamp koju je sam zaprimio od korisnika, minus jedna sekunda. Ako se klijentu vrati išta drugo znaći da je odbijen.

8. STANDARD ZA BILJEŽENJE ZAPISA

Kada su RPC poruke prosljeđene na vrh byte stream protokola (kao TCP/IP) poželjno je ograničiti jednu poruku od druge da bi se otkrile i po mogućnosti popravile greške korisničkog protokola. Ovo se naziva bilježenje zapisa (record marking RM). SUN koristi RM/TCP/IP transport za prosljeđivanje RPC poruka na TCP streamove. Jedna RPC poruka stane u jedan RM zapis.

Zapis je sastavljen od jednog ili više fragmenata zapisa. Fragment zapisa je 4-bitni header sljeđen sa 0 do 2^{31}-1 byte fragmenata podataka. Byteovi kodiraju binarni broj bez predznaka, kao sa XDR integerima, slijed im je od najvišeg prema najnižem. Broj kodira dvije vrijednosti: booleov koji indicira dali je fragment posljednji fragment zapisa (vrijednost bita 1 indicira da je fragment zadnji) i 31-bitnu binarnu vrijednost bez predznaka koja je duljina u byteovima fragmenata podataka. The boolean vrijednost je najvišeg reda bita headera, a dužina je 31 bit niskog reda. Primjećujemo da specifikacija zapisa nije u formi XDR standarda.

9. RPC JEZIK

Kao što je postojala potreba da se opišu XDR data-typovi u formalnom jeziku, tako postoji i potreba da se tim jezikom opiše i procedura koja upravlja navedenim XDR data tipovima. U tu svrhu koristimo RPC jezik koji je produžetak XDR jezika. Sljedeći primjer je iskorišten kako bi se opisala bit jezika.

9.1 PRIMJER OPISAN U RPC JEZIKU

Ovdje se nalazi primjer specifikacije jednostavnog ping programa:

```
/*
 * Jednostavan ping program
 */
program PING_PROG {
   /*
    * posljednja i najbolja verzija
    */
   version PING_VERS_PINGBACK {
      void
      PINGPROC_NULL(void) = 0;

      /*
       * Ping the caller, return the round-trip time
       * (in microseconds).  Returns -1 if the operation
       * timed out.

       */
      int
      PINGPROC_PINGBACK(void) = 1;
   } = 2;

   /*
```

```
         * Originalna verzija
         */
        version PING_VERS_ORIG {
           void
           PINGPROC_NULL(void) = 0;
        } = 1;
    } = 1;

    const PING_VERS = 2;      /* posljednja verzija */
```

Prva verzija opisana je PING_VERS_PINGBACK sa dvije procedure PINGPROC_NULL i PINGPROC_PINGBACK. PINGPROC_NULL ne uzima argumente i ne vraća rezultate, ali je korisna za izračunavanje round-trip vremenaod korisnika do servera i natrag. Po konvenciji, procedura 0 bilo kojeg RPC protokola trebala bi imati istu semantiku, i nikad ne zahtjeva bilo kakvu autentifikaciju.
Drugu proceduru koriste korisnici kako bi server radio obrnutu ping operaciju natrag prema korisniku i vrača količinu vremena u mikrosekundama koja treba za izvršavanje operacije. Sljedeća verzija, PING_VERS_ORIG je originalna verzija protokola i ne sadrži PINGPROC_PINGBACK proceduru. Ona je korisna za kompatibilnost sa starim korisničkim programima i kako i taj program stari postoji mogučnost da u potpunosti bude odbačen od protokola.

9.2. SPECIFIKACIJA RPC JEZIKA

RPC jezik identičan je XDR jeziku, izuzev dodane definicije "program-def" opisane dolje.

```
program-def:
  "program" identifier "{"
     version-def
     version-def *
  "}" "=" constant ";"

version-def:
  "version" identifier "{"
     procedure-def
     procedure-def *
  "}" "=" constant ";"

procedure-def:

  type-specifier identifier "(" type-specifier ")"
  "=" constant ";"
```

9.3. SINTAKSIČKE BILJEŠKE

(1)Sljedeće ključne riječi su dodane i nemogu se koristiti kao identifikatori:
"program" and "version";

(2) Ime i broj verzije se ne može pojaviti više od jedanput unutar spektra programske definicije.

(3) Ime i broj procedure se ne može pojaviti više od jedanput unutar spektra definicije verzije.

(4) Identifikatori program su u iistom imenskom prostoru kao konstanta i tip identifikatora.

(5) Jedino se nedodjeljenje konstante mogu pripisivati programima, verzijama i procedurama .