Verifikasi Command Transfer Protokol Menggunakan SPIN

Ervin Kusuma Dewi
Prodi S2/S3 Ilmu Komputer.
FMIPA Universitas Gadjah Mada,
Yogyakarta,
dew1_07@yahoo.com

Reza Pulungan
Jurusan Ilmu Komputer dan Elektronika.
FMIPA Universitas Gadjah Mada,
Yogyakarta,
pulungan@ugm.ac.id

Abstrak*— Formal verification adalah metode untuk membuktikan bahwa sebuah implementasi betul-betul mengimplementasikan apa yang dijabarkan dalam bentuk spesifikasinya. Model checking merupakan salah satu metode formal verification. Model checking merupakan salah satu cabang metode formal. Penerapan model checking secara umum memerlukan tiga tahapan, yaitu pemodelan, formalisasi properti dan verifikasi. Pemodelan protokol dilakukan dengan menggunakan bahasa PROMELA, sedangkan formalisasi properti dilakukan dengan menggunakan LTL (Linear Temporal Logic). Model dan properti formal yang dihasilkan menjadi masukan untuk tool verifier SPIN. Tujuan penelitian ini adalah memodelkan sebuah protokol dalam PROMELA dan kemudian memverifikasinya dengan menggunakan SPIN. Penelitian ini menggunakan kasus protokol Command Transfer Protocol (CTP) yang dikembangkan oleh Lev Naumov. Terdapat sembilan properti yang akan dicek. Hasil verifikasi protokol menunjukkan bahwa kesembilan properti tersebut terpenuhi.

Kata kunci* : Verifikasi formal, model checking, Command Transfer Protocol, PROMELA, SPIN

I. PENDAHULUAN

Komunikasi antar komputer dengan menggunakan protokol sering dilakukan dan digunakan secara luas. Oleh karena itu pelu dilakukan pembuktian bahwa protokol yang digunakan sudah tidak memiliki kesalahan *(bugs)*, cacat *(defect)* dan memenuhi spesifikasi yang dibutuhkan untuk komunikasi data sehingga tidak merugikan pengguna. Pada tahun 1996, Roket Ariane 5 meledak setelah 40 menit diluncurkan karena cacat *(defect)* pada *Control Software* yang mengakibatkan kerugian besar [1]. Kejadian-kejadian seperti ini menambah bukti bahwa perlu dilakukan verifikasi protokol yang digunakan untuk menangani kompleksitas, mengurangi jumlah kesalahan *(bugs)* yang hasilnya dapat mengurangi *cost* (biaya dan waktu).

Pemaparan di atas membuka pandangan betapa pentingnya verifikasi. Untuk melakukan verifikasi metode *formal verification* bisa digunakan. *Formal verification* membuktikan bahwa suatu implementasi betul-betul mengimplementasikan apa saja yang dijabarkan dalam spesifikasinya. *Formal verification* dapat dilakukan dengan berbagai metode, termasuk *equivalence checking*, *model checking* dan *theorem proving*. Yang paling umum digunakan adalah metode *model checking*, karena dengan metode ini, proses verifikasi dapat dilaksanakan secara otomatis dan memiliki *tool support* yang matang [2]. SPIN merupakan salah satu *tool* yang digunakan untuk membuat spesifikasi sistem dan melakukan verifikasi terhadap sistem tersebut [3]. SPIN cocok digunakan untuk verifikasi protokol karena SPIN menganalisa konsistensi *logical* pada sistem terdistribusi terutama untuk komunikasi data. Selain itu kelebihan dari SPIN adalah kebutuhan komputasionalnya yang relatif lebih rendah dibanding tool-tool yang lain [4].

PROMELA merupakan bahasa pemodelan untuk verifikasi suatu desain dan digunakan sebagai masukan pada *tool* SPIN. SPIN fokus pada pembuktian kebenaran pada *process interaction* [5]. PROMELA dapat memuat tiga tipe objek yaitu *processes*, *variables*, dan *message channels*.

Commands Transfer Protocol (CTP) merupakan protokol baru yang dikembangkan oleh Naumov [6] yang digunakan untuk menyelesaikan permasalahan komunikasi *parallel networking* atau protokol terdistribusi. Protokol ini merupakan protokol yang baru dikembangkan dalam proyek CAMEL *(Cellular Automata Modeling Environment & Library*). Karena protokol CTP merupakan protokol baru, maka belum diketahui apakah protokol ini sesuai dengan spesifikasi yang dibuat. Penelitian kali ini membahas mengenai verifikasi protokol CTP; yaitu apakah protokol tersebut sesuai dengan spesifikasinya.

Terdapat beberapa penelitian tentang verifikasi salah satunya penelitian [7] yaitu menganalisa dan membuktikan *deadlock* pada protokol DHCP, hasil dari penelitian tidak ditemukan *deadlock* pada protokol DHCP. Penelitian dengan membandingkan dua metode yaitu SPIN dan VIS pada kasus protokol *Asyncronous Transfer Mode Ring* (ATMR), hasil dari penelitian penggunaan memori lebih kecil dan menggunakan SPIN lebih cepat serta mampu mendeteksi *deadlock*, *livelock* dan berbasis GUI, namun untuk *syncronous* VIS lebih baik dari pada SPIN[4]. Penelitian pengenai aplikasi paralel yang dilakukan oleh quiroz-fabian menggunakan SPIN dengan mengusulkan lima formalisasi properti yaitu : *no message are lost*, *termintaion consensus*, *no premature termination*, *correct termination*, dan *deadlock*. Hasil dari penelitian menunjukkan implementasi dari model yang diadaptasi untuk memecahkan masalah N-Queues serta menterjemahkan model untuk bahasa C-MPI[8].

II. METODE PENELITIAN

A. *Command Transfer Protocol (CTP)*

Command Transfer Protocol (CTP) merupakan protokol untuk *cluster networking* yang dikembangkan oleh Naumov [6]. *Cluster* merupakan gabungan dari *workstation*. *Computational cluster* adalah sebuah *cluster* yang dibangun dengan perhitungan. Di bawah ini merupakan spesifikasi

sistem yang menyatakan *requirement* untuk fungsi sebuah jaringan *(network)*. Terdapat lima persayaratan utama komunikasi dalam *cluster networking* [6]:

1. Pertukaran data yang cepat *(rapid data interchange)*
2. Pertukaran data yang handal *(reliable data interhcange)*
3. Pada *cluster networking*, setiap node bisa menjadi master dan slave *(peer-to-peer networking)*
4. *Broadcasting support*
5. Pertukaran data yang sangat besar *(huge data blocks interchange support)*

Pertama yang diperlukan adalah mendefinisikan pengirim dan penerima. CTP tetap menggunakan IP Address, yang menjadi alasan IP address digunakan adalah karena IP address sudah digunakan secara luas dan memenuhi persyaratan. Pada protokol CTP, bahasa "*command*" dan "*message* " merupakan sinonim yang berarti "*command*" adalah "*message*" tetapi tidak untuk sebaliknya.

Persyaratan *(requirement)* yang dipenuhi oleh CTP pada *cluster networking* adalah sebagai berikut :

1. Untuk peningkatan kecepatan pertukaran informasi, maka menggunakan UDP sebagai protokol standar. Menggunakan UDP tanpa membuat jaringan *dow*n. Selain itu akan lebih mengamankan *user* jika terjadi masalah.
2. Kehandalan pertukaran data diimplementasi. Setiap paket yang dikirim diurutkan sampai penerima sudah meng-konfirmasi urutan data tersebut. Selain itu untuk menjaga mekanisme, paket-paket disediakan dengan id (*identifier*). Indentifikasi akan dilakukan dengan memberikan *id integer number* pada sisi pengirim.
3. *Broadcasting* merupakan alasan kenapa menggunakan UDP sebagai protokol standar.
4. Protokol CTP diimplementasikan untuk pertukaran data yang besar. Jika pesan lebih besar dari batas (65400 byte default) maka dibagi dalam beberapa bagian kemudian dikirimkan secara terpisah satu persatu. Pada sisi penerima akan menggabungkan untuk mengatur inisial *command*. Catatan penting aplikasi penerima akan memperoleh informasi *command* yang datang hanya setelah semua bagian sudah diterima seperti *command* yang akan dinamai sebagai *large command* tetapi pada prakteknya mayoritas *command* sudah normal.
5. Untuk pertukaran data *peer-to-peer*, implementasi CTP memasukan fungsi *client* dan *server* sebagai satu kesatuan.

B. SPIN/ PROMELA

SPIN merupakan toolss yang cocok untuk digunakan sebagai *verifier*, karena SPIN memiliki kelebihan. Kelebihan pada SPIN yaitu :

1. SPIN lebih cocok digunakan untuk memverifikasi protokol karena SPIN menganalisa konsistensi logical pada sistem terdistribusi terutama untuk komunikasi data [9].
2. Target SPIN pada Software karena *basic* model SPIN adalah *Interleaving Model* [4].
3. Penggunaan CPU lebih cepat [4].
4. SPIN merupakan toolss yang berbasis *Graphic User Interface* [4].
5. SPIN dapat mendeteksi *livelock*, *deadlock* dan properti lainya. Karena SPIN berbasis opensource sehingga dapat dikembangkan [10].

PROMELA *(Protocol Meta Language)* adalah bahasa pemodelan untuk verifikasi suatu desain yang digunakan sebagai input dalam SPIN, memverifikasi dan membuktikan *correctness* dalam bentuk sintaks *Linear Temporal Logic* (LTL). Model PROMELA dapat memuat tiga tipe objek yang berbeda yaitu *processes*, *variables*, dan *message channels*. PROMELA tidak ditunjukkan untuk bahasa implementasi melainkan sebagai bahasa untuk mendeskripsikan sistem dan yang paling menonjol dari bahasa PROMELA adalah pemodelan, sinkronisasi, dan koordinasi proses.

C. Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) adalah pernyataan tingkah laku sistem yang diharapakan dalam bahasa logika dengan melibatkan operator-operator temporal. Sifat dari protokol dirumuskan ke dalam LTL dan digunakan sebagai inputan ke dalam SPIN. Basic formula LTL adalah *atomic propotition*[1].

Protokol dikatakan *correctness* jika dapat memenuhi *requirement*. Untuk mendapatkan *correctness*, perlu menyusun *requirement* protokol CTP. *Requirement* perlu dinyatakan ke dalam bahasa yang dipahami oleh *verifier*, dengan menggunakan bahasa LTL (*Linear Temporal Logic*) .

□artinya adalah *"always"*

◊ artinya adalah *"eventualy"*

Safety property merupakan sebuah properti yang digunakan untuk menunjukan bahwa suatu sistem *"nothing bad will happen"*. Sedangkan *liveness property* merupakan sebuah properti *liveness* yang menyatakan bahwa sesuatu yang baik akhirnya terjadi *"something good eventually happens"*

III. PEMODELAN DAN FORMALISASI PROTOKOL CTP

Pada sub bab ini akan dibahas mengenai pemodelan terhadap *behaviour* dari protokol CTP. Selian pemodelan, juga dibahas mengenai formalisasi properti protokol CTP.

A. Pemodelan protokol CTP

Pemodelan protokol berdasarkan *behaviour* dari protokol CTP. `Proctype` merupakan deklarasi dari semua *behaviour*. `Proctype` dibangun untuk digunakan sebagai pendeklarasian sebuah proses, di dalam sebuah `proctype` terdapat *variabel local* dan *message channel*.

Pemodelan memerlukan notasi simbolik untuk nilai konstan untuk verifikasi, pada PROMELA disebut dengan `mtype`. Selain itu `mtype` juga bisa digunakan sebagai tipe data yang dapat digunakan dalam deklarasi `chan`. Gambar 1 merupakan deklarasi `mtype`.

```
mtype = {command, conf1, conf2, conf3
```

Gambar 1 Penggunaan `mtype`

Pemodelan juga menggunakan beberapa `channel`. `Channel` merupakan sebuah obyek yang dapat menyimpan beberapa nilai yang dijadikan satu[3]. Selain `channel` pemodelan CTP dibagi menjadi empat `proctype` yaitu : `proctype timer`, `proctype masternode`, `proctype node1`, `proctype node2`, `proctype node3` dan `proctype udp`. Gambar 2 merupakan potongan dari pemodelan protokol CTP.

```
mtype = { command, conf1, conf2, conf3 };
chan to_msnode                  = [0] of { mtype, bit
, byte};
chan to_udp             = [0] of {mtype, bit, byte};
chan to_n1              = [0] of { mtype, bit , byte};
chan to_n2              = [0] of {mtype, bit, byte};
chan to_n3              = [0] of {mtype, bit, byte};
chan time_out           = [0] of {bit};

active proctype timer()
{
        int i;
        do
        :: time_out!0;
        od
        }
active proctype Masternode ()
{       bit cbit_out, rbit_in;
        mtype msg;bit sent=0;
        bit t;bit confirms[3];
        confirms[0];
        confirms[1];
        confirms[2];
        byte i=0;byte r_i;
        int time=100;
do
::  to_udp!command, cbit_out, i ->sent=1;
        do
        :: to_msnode?msg, rbit_in, r_i ->
                if
                :: (cbit_out==rbit_in) ->
                if
                :: (msg==conf1) ->
confirms[0]=1;
                :: (msg==conf2) ->
confirms[1]=1;
                :: (msg==conf3) ->
confirms[2]=1;
                fi;
        :: else -> skip;
                fi;
        od
:: time_out?t, r_i ->sent;
        if
        :: sent -> to_udp!command, cbit_out;
        :: else -> skip
        fi
od
}
active proctype udp()
{
        bit  rbit_in, cbit_out;
        mtype msg;
do
        ::to_udp?command, rbit_in, i->
                if
                :: to_n1!command, cbit_out,i;
                :: skip;
        do
        :: to_n1?command, rbit_in,r_i ->
                to_udp!conf1,cbit_out,i
        :: to_n1?command,lost ,r_i->
        if
                :: (cbit_out!=rbit_in)->
                cbit_out==rbit_in;
                :: else -> skip;
                fi
        :: to_udp!conf1,cbit_out,i
        od
        }
active proctype Node2 ()
{       bit rbit_in, lost, cbit_out;
        byte r_i;
        byte i;
        do
        :: to_n2?command,rbit_in, r_i ->
           to_udp!conf2,cbit_out,i
        :: to_n2?command,lost,r_i ->
             if
             :: (cbit_out!=rbit_in)-> cbit_out==rbit_in;
             :: else -> skip;
             fi
        :: to_udp!conf2, cbit_out,i
        od
        }
```

Gambar 2 Pemodelan protokol CTP

B. Formalisasi properti

Properti yang akan diverfikasi berdasarkan standar SPIN dan berdasarkan beberapa properti yang mengacu properti penelitian[11]. Protokol dikatakan *correctness* jika dapat memenuhi *requirement*. Untuk mendapatkan *correctness*, perlu menyusun *requirement* protokol CTP. *Requirement* perlu dinyatakan ke dalam bahasa yang dipahami oleh *verifier*, bahasa yang akan digunakan adalah LTL (*Linear Temporal Logic*) .

1) Idle

```
[](¬(sendcom) -> ( ¬(konfirmasi))
```

Tidak ada pengiriman `command sendcom` maka tidak ada konfirmasi `konfirmasi`. Keterangan dari formula properti ke-1 adalah sebagai berikut ini :

`Sendcom` : proses pengiriman `command sendcom`.
`konfirmasi` : proses menerima konfirmasi `konfirmasi`.

2) Hak tertinggi (precedence)

```
[](message->(message U konfirmasi))
```

Setiap *state* akan memenuhi jika mengirimkan `command message` ke `node` dan `node` menerima `message` sampai `node` mengirimkan konfirmasi `konfirmasi` ke `masternode`. Keterangan dari LTL tersebut adalah :

`Message` : Proses `masternode` mengirimkan `command message` ke `node`.
`Konfirmasi` : Sampai `masternode` menerima konfirmasi `konfirmasi` dari `node`.

3) Confirm

```
[]((conf_node1)&&(conf_node2)&&
(conf_node3)-> <>(konfirmasi))
```

Konfirmasi yang dikirimkan oleh `node 1 conf_node1`, `node 2 conf_node2` , dan `node 3 conf_node3` akan `konfirmasi` jika semua konfirmasi `konfirmasi` tersebut diterima oleh `masternode`. Keterangan dari LTL tersebut adalah :

`conf_node1` : Proses pengiriman konfirmasi `node 1 conf_node1` ke `masternode`.

`conf_node2` : Proses pengiriman konfirmasi node 2 `conf_node2` ke `masternode`.

`conf_node3` : Proses pengiriman konfirmasi `node 3 conf_node3` ke `masternode`.

`konfirmasi` : Proses `masternode` menerima semua konfirmasi `konfirmasi` dari semua `node`.

4) Deadlock

End State merupakan nama label yang dimulai dengan akhiran urutan tiga karakter, *end state* dapat digunakan dalam `proctype`, *trace* dan deklarasi notrace. *End state* digunkan untuk mengetahui apakah dalam komunikasi protokol terdapat *deadlock*.

Deadlock adalah suatu kondisi dimana dua proses atau lebih saling menunggu proses yang lain untuk melepaskan *resource* yang sedang dipakai. Karena proses saling menunggu, maka tidak terjadi kerja pada proses tersebut. Oleh karena itu perlu dilakukan verifikasi dengan menggunakan *end-state* yang merupakan standar dari SPIN, untuk mengetahui apakah pada komunikasi tersebut terdapat *deadlock* atau tidak.

5) Mengosongkan daftar pengiriman

```
[](sending)->([]DafSending1 ==0&&
[]DafSending2==0&[]DafSending3==0)
```

Setiap proses mengirimkan `sending` semua `command`, maka selalu daftar pengiriman `DafSending` kosong.

`Sending` : Proses mengirimkan `command`.

`DafSending` : Jumlah `command` yang masih tersisa diproses `DafSending` adalah 0.

6) `Command` yang dikirim diterima

```
[]((sendcom)-> (<>(rcvcom)))
```

Selalu semua `command sendcom` yang dikirimkan diterima oleh `node rcvcom`. Keterangan dari LTL tersebut adalah :

`sendcom` : Proses `masternode` mengirimkan `command sendcom` ke `node`.

`rcvcom` : Proses `node` menerima `command rcvcom` dari `masternode`.

7) Transmisi tidak berhasil, namun `node` menerima `command`

```
[](sendcom)-><>((!konfirmasi) &&
(menerima))
```

Selalu `masternode` men-*transmisi* `command sendcom`, *transmisi* tidak berhasil `!konfirmasi`, namun `node` menerima `command`. Keterangan dari LTL tersebut adalah:

`sendcom` : Proses melakukan pengiriman `command`.

`Konfirmasi` : Proses mengirimkan konfirmasi `konfirmasi`.

`menerima` : Proses `node` menerima `command`.

8) Transmisi sukses, namun `node` tidak menerima semua `command`

```
[](sendcom)-> <>((konfirmasi) &&
(!menerima))
```

Selalu `masternode` melakukan transmisi `sendcom`, transmisi dilaporkan sukses `konfirmasi`, namun `node` tidak menerima semua `command !receive`. Keterangan dari LTL tersebut adalah :

`Sendcom` : Proses melakukan pengiriman `command`.

`konfirmasi` : Proses mengirimkan konfirmasi `konfirmasi`.

`menerima` : Proses `node` menerima `command`.

9) Paralelisme

```
[]((sendcom) -> <> (konfirmasi) &&
(!konfirmasi U resend)
```

Selalu mengirimkan `command sendcom`, *eventually* menerima `konfirmasi`, jika tidak menerima konfirmasi `!konfirmasi` maka mengirimkan ulang `command resend`. Keterangan dari LTL tersebut adalah :

`Sendcom` : Proses mengirimkan `command`.

`Konfirmasi` : Proses menerima konfirmasi dari `node`

`Resend` : Mengirimkan ulang `command` yang belum mendapatkan konfirmasi dari `node`.

IV. PEMBAHASAN

Verifikasi dilakukan dengan menggunakan *tools* SPIN yaitu iSPIN. CTP yang sudah dimodelkan dengan menggunakan bahasa PROMELA menjadi masukan pada SPIN. SPIN akan memeriksa apakah model yang telah dibangun dalam bahasa PROMELA telah memenuhi *requirement* atau tidak.

A. Hasil verifikasi

Pertama yang dilakukan setelah pemodelan adalah mengecek sintaks. *Syntax Check* pada SPIN berfungsi untuk mengecek apakah pemodelan terdapat *error syntax* pada penulisan PROMELA atau tidak. Jika terdapat kesalahan pada *error syntax* maka SPIN akan memberikan keterangan *error syntax* namun sebaliknya, jika tidak terdapat *error syntax* maka pada SPIN akan muncul keterangan *nothing to report*.

1) Idle

Hasil *running* verifikasi properti. Total memori yang digunakan adalah 64.539 MB. Sedangkan jumlah transisi state sebanya 46 transisi.

Hasil *running* verifikasi menunjukan tidak terdapat *error*, artinya properti ini tidak dilanggar atau properti

ini memenuhi *requirement*-nya yaitu pada saat tidak ada pengiriman `command` maka tidak ada konfirmasi yang diterima oleh `masternode`.

2) *Hak tertinggi (precedence)*
   Lama waktu yang dibutuhkan untuk *running* verifikasi adalah 0.001 *second* dan memori yang digunakan adalah 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 46 transisi. Tidak terdapat *error* artinya properti ke-2 telah memenuhi *requirement*-nya, bahwa `masternode` akan mengirimkan pesan sampai mendapat konfirmasi. Hasil dari verifikasi properti ke-2 yang menunjukkan *satisfy*, yaitu ditandai dengan *no error found* pada *verifier*.
3) *Confirm*
   Lama waktu yang dibutuhkan untuk *running* verifikasi adalah 0.001 *second* serta total memori yang digunakan 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 46 transisi. Tidak terdapat *error* pada verifikasi properti ke-3. Karena tidak terdapat *error*, artinya properti ke-3 telah memenuhi *requirement*-nya, yaitu ketika semua konfirmasi yang diterima oleh `masternode` menandai bahwa masing-masing `node` menerima `command` yang dikirimkan oleh `masternode` artinya bahwa semua telah ter-*confirm*. Hasil verifikasi properti ke-3 yang menunjukkan *satisfy*, yaitu ditandai dengan *no error found* pada *verifier*.
4) *Deadlock*
   Pengecekan *deadlock* ini merupakan stadar dari pengecekan SPIN dengan memilih opsi verifikasi yaitu *safety property* serta mengaktifkan opsi *end state* pada *verifier*. Opsi tersebut digunakan untuk membuktikan *end state* atau yang biasa dikenal dengan *deadlock*.
   Lama waktu yang dibutuhkan saat *running* verifikasi properti ke-4 adalah 0.001 *second*. Total memori yang digunakan 64.539 MB. Jumlah transisi state sebanyak 46 state serta *no error found* yang artinya properti ke-4 memenuhi *requirement*. Hasil verifikasi properti yang menunjukkan bahwa properti ke-4 *satisfy*.
5) *Mengosongkan daftar pengiriman*
   Lama waktu yang dibutuhkan untuk *running* verifikasi adalah `0.001` *second*. Memori yang dibutuhkan adalah 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 46 transisi. Tidak terdapat *error* artinya properti ke-5 ini tidak dilanggar.
   Ketika properti ke-5 tidak dilanggar artinya properti ke-5 memenuhi *requirement*-nya, yaitu ketika keseluruhan `command` sudah terkirim maka daftar pengiriman akan dikosongkan, mekanisme pengosongan penyimpanan pengiriman jika `command` sudah terkirim semua. Hasil verifikasi properti ke-5 yang menunjukkan *satisfy*, yaitu ditandai dengan *no error found* pada *verifier*.
6) *Command yang dikirim diterima*
   Lama waktu yang dibutuhkan untuk *running* verifikasi adalah 0.001 *second* serta memori yang dibutuhkan adalah 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 20 transisi. Tidak terdapat *error* artinya properti ke-6 ini tidak dilanggar. Tidak dilanggarnya properti ke-6 maka properti ini telah memenuhi *requirement*-nya, yaitu ketika keseluruhan `command` sudah terkirim maka `masternode` akan mengosongkan daftar pengiriman. Hasil verifikasi properti ke-6 menunjukkan *satisfy*, yaitu ditandai dengan *no error found* pada *verifier*.
7) *Transmisi tidak berhasil, namun receiver menerima* `command`
   Lama waktu yang dibutuhkan untuk *running* verifikasi adalah 0.007 *second* serta memori yang dibutuhkan adalah 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 3 transisi. Terdapat *erorr* artinya properti ke-6 ini dilanggar. Pelanggaran properti tersebut yang ingin dicapai, karena CTP diklaim mampu melakukan *retransmisi*. Sedangkan properti yang dicek adalah masternode tidak berhasil melakukan transmisi, namun `node` menerima `command`. Karena CTP mampu melakukan *retransmisi*, maka properti tersebut harus dilanggar. Setiap `command` yang dikirimkan oleh masternode jika tidak mendapatkan konfirmasi dari `node` maka `command` tersebut akan dikirimkan ulang.
   Counterexample properti ke-6 pada line 2 SPIN memverifikasi model dengan properti yang sudah di-*generate* kedalam bahasa PROMELA, pada proses tiga `node` 3 mengirimkan konfirmasi ke `masternode` malui UDP, namun pada *line* 5 proses UDP menghilangkan konfirmasi yang dikirimkan oleh `node` 3. *Line* 6 *verifier* melakukan verifikasi kembali model dengan properti, namun kejadian yang sama terulang sehingga properti tersebut dilanggar. Dari *counterexample* tersebut, ingin menunjukan bahwa properti mengehendaki bahwa proses transmisi tidak berhasil sedangkan CTP memiliki mekanisme *retransmisi*. Hasil tersebut berbanding balik, oleh karena itu pengecekan tidak memenuhi properti yang ditunjukkan dengan adanya 1 *error*. Namun dengan tidak dipenuhi properti tersebut, membuktikan bahwa protokol CTP konsisten dengan mekanisme *retransmisi*. Hasil dari verifikasi memang diharapkan properti tidak memenuhi properti artinya dengan tidak memenuhi properti, CTP tetap konsisten terharap mekanisme *retransmisi*. Namun sebaliknya jika properti tersebut dipenuhi, maka protokol CTP tidak konsisten terhadap *requirement*-nya.
8) *Transmisi sukses, namun receiver tidak menerima semua* `command`
   Lama waktu yang dibutuhkan untuk *running* verifikasi adalah 0.007 *second* serta memori yang dibutuhkan adalah 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 3 transisi. Terdapat *error* artinya properti ke-8 ini dilanggar. Pelanggaran properti tersebut yang ingin dicapai, karena CTP diklaim mampu melakukan *retransmisi*. Sedangkan properti yang dicek adalah `masternode` berhasil melakukan transmisi, namun `node` tidak menerima `command`. `Masternode` akan menandai trasmisi sukses dengan menerima konfirmasi dari `node`. Namun jika tidak mendapatkan

konfirmasi maka `masternode` akan terus mengirimkan `command`. Maka properti tersebut harus dilanggar.

Counterexample properti ke-8 sama dengan properti ke-7 yaitu proses pertama SPIN mencocokan model dengan properti. Proses *line* 2 `node` mengirimkan konfirmasi melaui UDP, namun UDP menghilangkan konfirmasi tersebut yang dapat dilihat pada *line* 5. Dari *counterexample* menunjukkan bahwa CTP tetap konsisten dengan *requirement* retrasmisi.

9) *Paralelisme*

Lama waktu yang dibutuhkan untuk *running* verifikasi adalah 0.002 *second* serta memori yang dibutuhkan adalah 64.539 MB. Jumlah transisi *state* secara keseluruhan adalah 46 transisi. Tidak dilanggarnya properti ke-9 maka properti ini telah memenuhi *requirement*-nya, yaitu ketika keseluruhan `command` sudah terkirim maka `masternode` akan mengosongkan daftar pengiriman. Hasil verifikasi properti ke-9 yang menunjukan *satisfy*, yaitu di tandai dengan *no error found* pada *verifier*.

B. *Hasil pengujian*

Hasil dari verifikasi dengan menggunakan tiga `node` pada sembilan properti adalah *satisfy*. Pengujian pada Tabel 1 bahwa dengan menggunakan tiga sampai dua puluh `node` menunjukkan hasil verifikasi adalah *satisfy*. Namun *depth reached* dan transisi jumlahnya bertambah dikarenakan bertambahnya perpindahan *state* pada komunikasi protokol CTP. Memori yang digunakan untuk *running* verifikasi hingga `node` dua puluh adalah 64.539 MB.

TABLE I. HASIL PENGUJIAN

Jumlah Node	Depth reached	Transisi	Memori (MB)	Lama waktu verifikasi	error	Ketera-ngan
3	12	46	64.539	0.001	0	*Satisfy*
4	15	73	64.539	0.001	0	*Satisfy*
7	24	190	64.539	0.002	0	*Satisfy*
10	33	361	64.539	0.003	0	*Satisfy*
20	63	1321	64.539	0.007	0	*Satisfy*

KESIMPULAN

Berdasarkan penelitian dan pembahasan yang telah dilakukan maka diperoleh kesimpulan bahwa verifikasi protokol CTP dilakukan dengan memodelkan dalam bahasa PROMELA dan diverifikasi secara otomatis menggunakan *tools model checker* SPIN. Berdasarkan hasil verifikasi, protokol CTP memenuhi sembilan properti yang artinya bahwa sembilan properti adalah *satisfy*, sehingga protokol CTP memenuhi *requirement* dari protokol CTP.

Hasil pengujian dengan menggunakan `node` berbeda didapat hasil *depth reached* dan transisi *state* bertambah, namun memori yang digunakan untuk *running* verifikasi sama yaitu 64.539. Dari semua pengujian `node` tidak ditemukan error artinya dengan jumlah `node` berbeda properti tetap dipenuhi.

DAFTAR PUSTAKA

[1] Baier, C., & Katoen, J.-P., Pinciples of Model Checking, The MIT Press, Cambridge, 2008.

[2] Clarke, E. M., Wing, J. M., "Formal Methods : State of the Art and Future Direction", ACM Computing Surveys, 1996, Vol. 28, No. 4.

[3] Raharjo,B., "EL 688-Metode Formla", 2002, *http://www.budi.insan.co.id/courses/ec7030/promela-spin.pdf*, diakses tanggal 8 agustus 2012

[4] Peng, H., Tahar, S., & Khendek, F., "SPIN vs. VIS : A Case Study on the Formal Verification of the ATMR Protocol", York, IEEE International Conference on Formal Engineering Methods ICFEM2000, 4-6 September, York, pp.79-87, 2000.

[5] Holzmann, G. J., "The Model Checker SPIN", IEEE Transaction on Software Enggineering Vol 23, No 5, 1997.

[6] Naumov, L., "Command Transfer Protocol (CTP) - A New Networking Protocol for Distributed or Parallel", 2005, *Computations,http://www.codeproject.com/Articles/6742/Commands-Transfer-Protocol-CTP-A-New_networking-Pr*, diakses tanggal 7 juli 2012

[7] Islam, S. M., Sqalli, M. H., dan Khan, S., Modeling and Formal Verification of DHCP Using SPIN, *International Journal of Computer Science & Applications, 3(6)* pp 145-159, 2006.

[8] Quiroz-Fabian, J.L, Aguilar-Cornejo, M., Roman-Alonso, G., dan Castro-Garcia, M.A., Model Checking for Integrating Dynamic Load Distribution into Parallel Application, *Proceeding Computer Science, 2008.ENC'08.Mexican International Conference on*, 16-10 Oktober, Baja California, pp 221-231, 2008.

[9] Gert, R., "Concise Promela Reference", *http://spinroot.com/spin/Man/Quick.html*, diakses tanggal 29 januari 2013.

[10] Ruys,T. C., "SPIN Beginner's Tutorial", 2002, *http://www.cs.utwente.nl/~ruys*, diakses tanggal 22 januari 2013

[11] Dewi, E.K, "Verifikasi Protokol CTP (Command Transfer Protokol) menggunakan SPIN/PROMELA", *Tesis*, Pascasarjana Ilmu Komputer, Universitas Gadjah Mada, Yogyakarta, 2014.