DATA NETWORK FINAL PROJECT

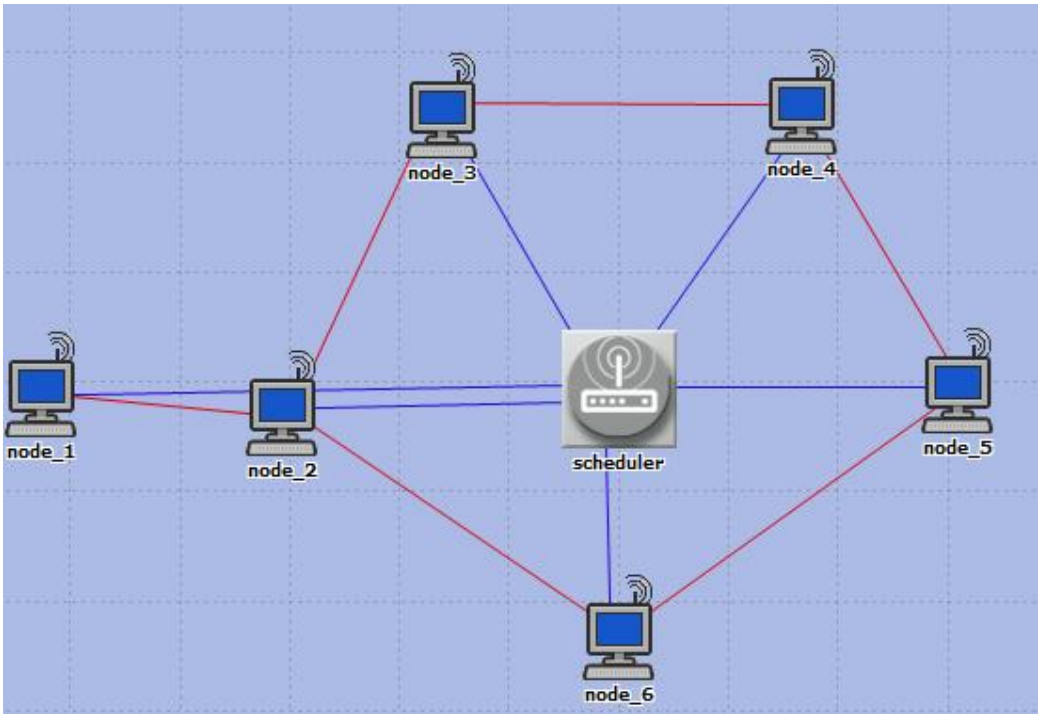
::

Wireless Routing-Scheduling Agents with OPNET Simulator

Ramyad Hadidi
Spring 2013
88109971

۱. معرفی

در این پروژه قصد داریم کلیه ی لایه های دخیل در یک شبکه را طراحی کنیم. شبکه ی ما یک شبکه ی wireless است که با لینک های مجازی به هم متصل شده اند. در این شبکه برای routing پکت ها از الگوریتم MTA[1] استفاده می کنیم.
شبکه ی ما همانطور که در شکل ۱ نشان داده شده است؛ از ۶ کاربر (Client) و یک هماهنگ کننده (Scheduler) تشکیل شده است.



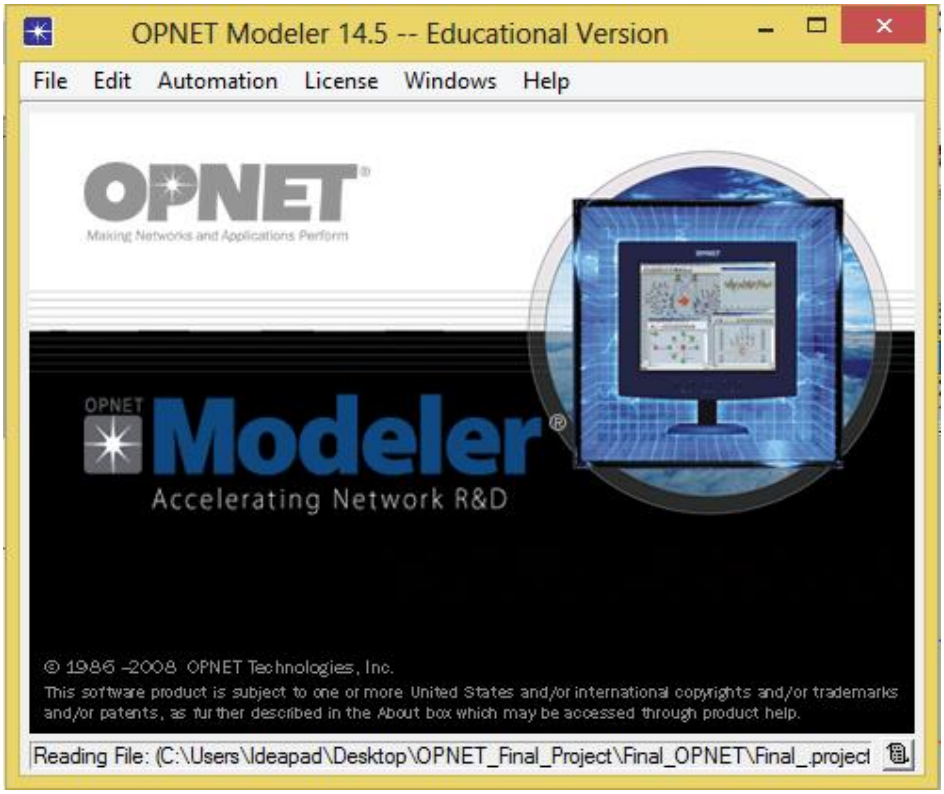
شکل ۱

در این گزارش ابتدا به نحوه ی ساختن تک تک لایه های می پردازیم و سپس به مسائل جزئی تر می پردازیم. در واقع مرحله ی اول در هر پروژه ی دیگری هم می تواند انجام شود.

۲. شرح ساختن عناصر شبکه

ابتدا همانند شکل ۲ OPNET را اجرا می کنیم.

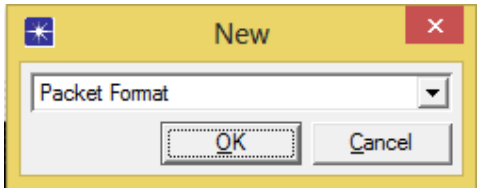
[1] Maximum Throughput Algorithm



شکل ۲

Packet Format .۲,۱

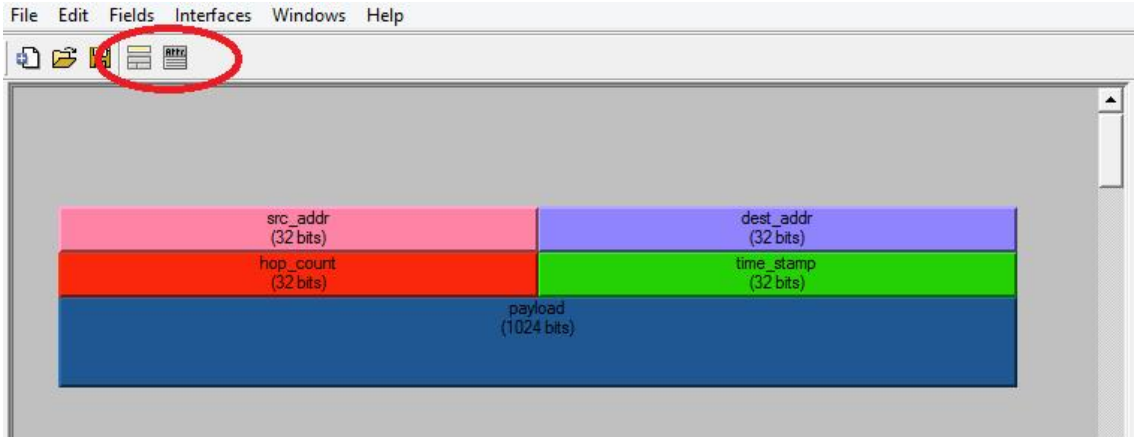
برای ساختن Packet Format از منوی File گزینه ی New را انتخاب کرده. و همانند شکل ۳ Packet Format را انتخاب می کنیم.



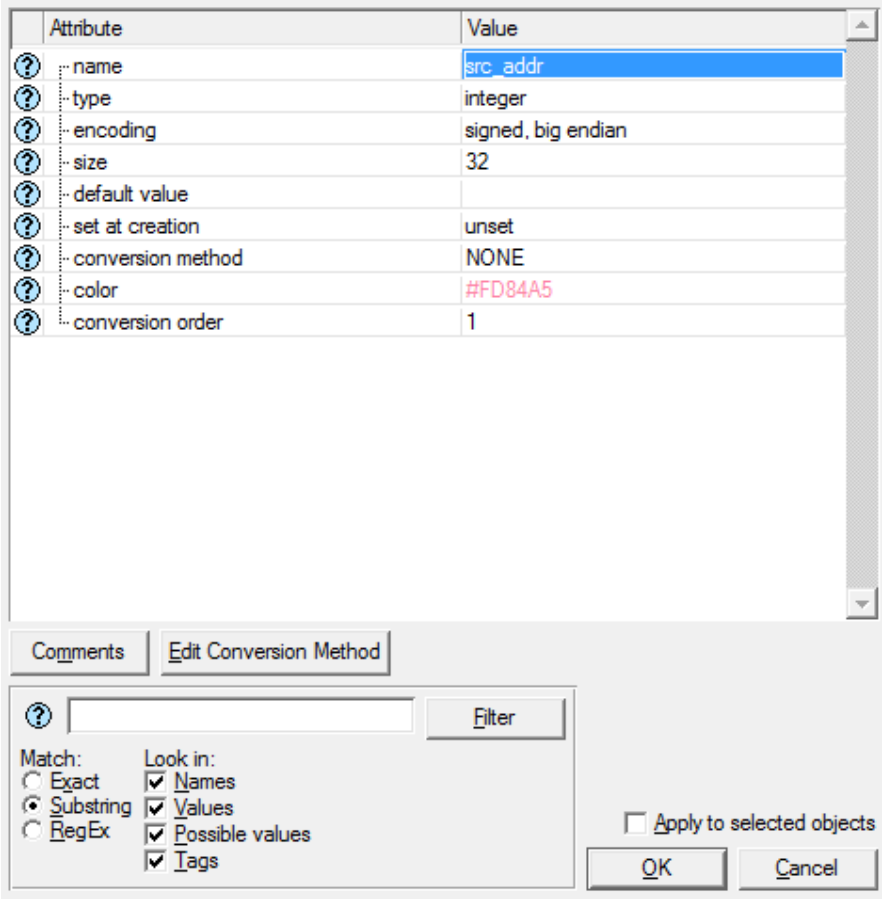
شکل ۳

حال در پنجره ی جدید Packet Field های خود را تعریف کرده و Attribute آنها را مشخص می کنیم. (شکل ۴)

در پنجره ی Attribute می توانیم مشخصات هر بخش را مشخص کنیم. (شکل ۵) به عنوان مثال می توانیم نوع داده و سایز آنها را مشخص کنیم. در این پروژه من برای راحتی کلیه ی Filed ها را ۳۲ بیتی تعریف کردم. مقدار Default هر Filed هم در صورتی که set at creation را فعال کنیم به هر Filed اختصاص داده می شود.



شکل ٤



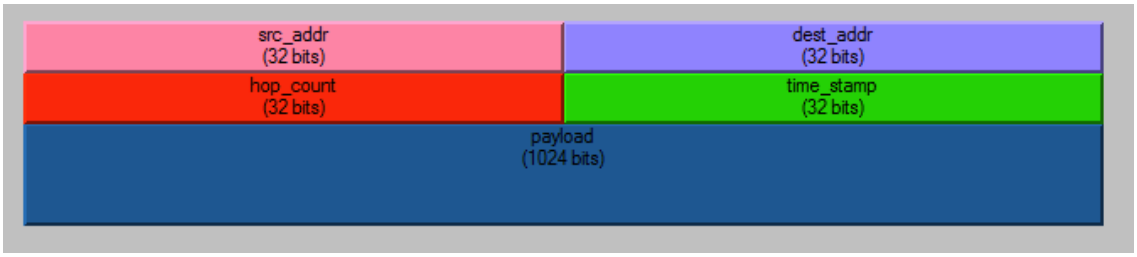
شکل ٥

پس از اتمام تعریف Packet های خود آنها را ذخیره می کنیم. همچنین می توانیم در حین انجام پروژه آنها را تغییر دهیم.

من در این پروژه دو نوع Packet تعریف کردم. اولی Packet های Poll که به ارتباط بین Scheduler و Node ها مربوط می شود. دومی Data که به ارتباط بین Node ها مربوط می شود. شکل ۶ و ۷ به ترتیب ساختار این پکت ها را نشان می دهد.

باید توجه کرد که این ساختار برای این شبکه تعریف شده. (مخصوصا پکت poll). می توان این پکت را به گونه ای تعریف کرد که هر بار اطلاعات یک Queue با آدرس آن ارسال شود. ولی از آنجا که در ابتدا به این نحو تعریف کردم و کد زدم این ساختار را تغییر ندادم. (در صورت پروژه هم گفته نشده بود که به گونه ی Optimized تعریف بکنیم.) پس فقط به گونه ی راحت تر برای کد زنی و استفاده تعریف کردم.

شکل ۶



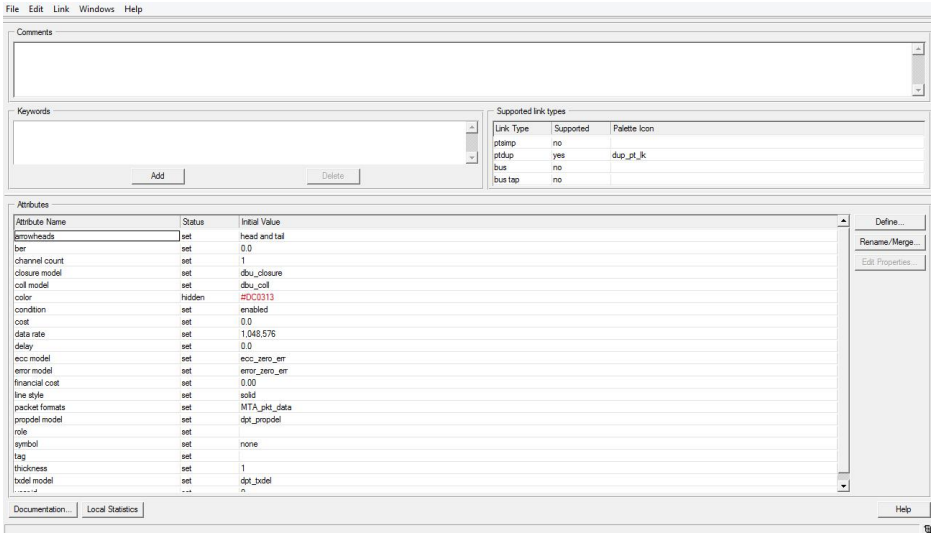
شکل ۷

Link Model .۲,۲

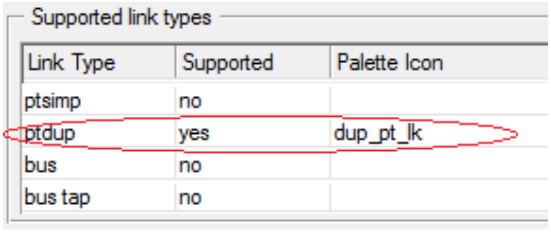
این بخش به تعریف Link های جدید می پردازد. برای تولید یک Link جدید در شکل ۳ Link Model را انتخاب می کنیم. پنجره ی شکل ۸ ظاهر می شود. در این پنجره در بخش supported Link Type انتخاب می کنیم که چه Link ای می خواهیم. از آنجا که در این پروژه لینک های Duplex می خواهیم همانند شکل ۹ تنظیمات را اجرا می کنیم. سپس در بخش Attributes می توانیم ویژگی های دیگری را مشخص کنیم. در این پروژه من از لینک های مختلفی برای انتقال داده استفاده کردم. تفاوت این Link ها در بخش Supported Packet Format است و در سایر موارد مانند یکدیگر هستند. شکل ۱۰ این تنظیمات را نشان می دهد. ما در این لینک ها errorای ندارم و سرعت آنها را به حدی زیاد گرفتم که مشکلی در کندی ارتباطات نداشته باشیم. در صورت نیاز می توان تمامی مقادیر را تغییر داد.

همچین باید در بخش ..File→Declare External Files همانند شکل ۱۱ Link-delay را انتخاب کنیم.

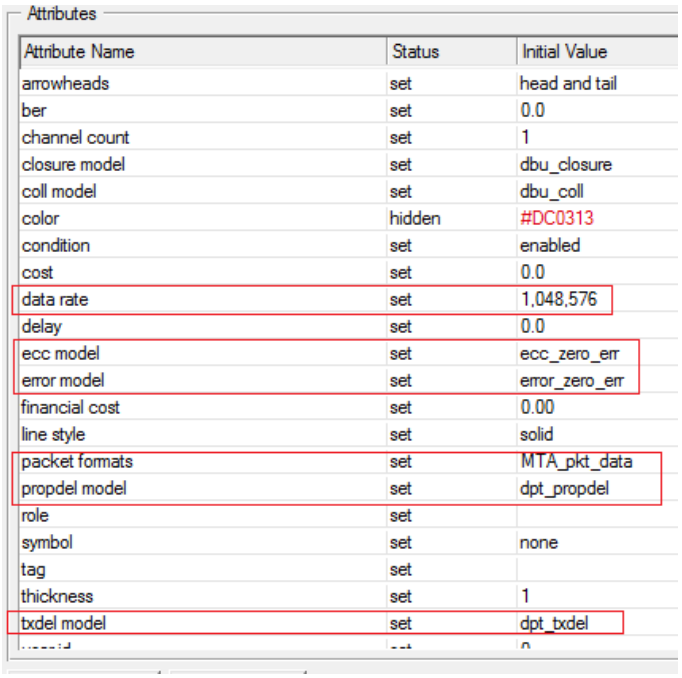
حال ما پکت ها و لینک های ارتباطی را ساخته ایم.



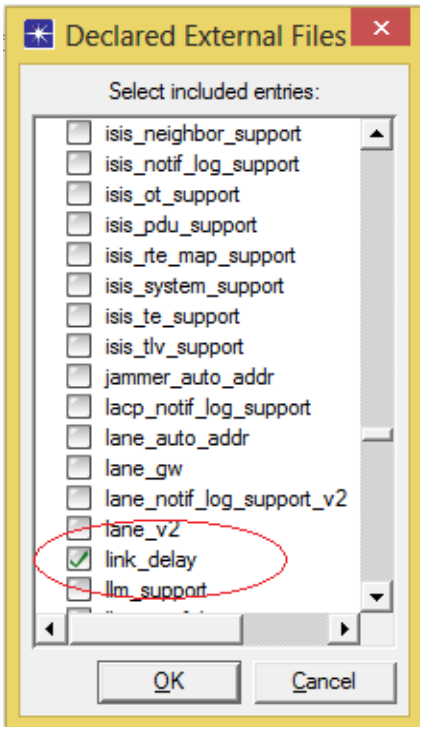
شکل ۸



شکل ۹



شکل ۱۰



شکل ۱۱

Node Model .۲,۳

در این بخش عناصر داخل Nodeها را از فرستنده، گیرنده و پروسسور تعریف می کنیم. ابتدا همانند شکل ۳ Node Model را انتخاب می کنیم. در Toolbar این پنجره که در شکل ۱۲ آمده است می توانیم عناصر مختلف را استفاده کنیم.

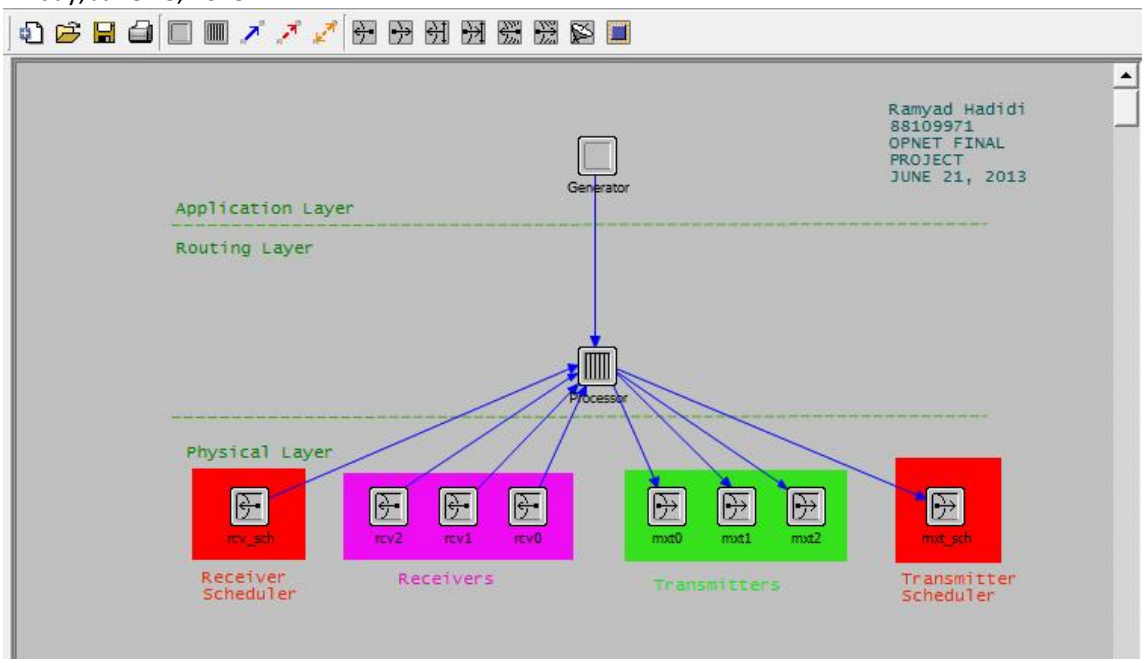
شکل ۱۲

به عنوان مثال فرستنده های Point to Point، Bus و رادیویی تعریف کنیم. همچنین می توان جداگانه پروسسور و Queue تعریف کرد.

برای اتصال بین عناصر لینک های آبی شکل ۱۲ به عنوان Packet stream شناخته می شوند و وظیفه ی انتقال Packetها را به عهده دارند. از لینک های دیگر استفاده ای نکردم. این لینک ها برای انتقال داده های کنترلی استفاده می شود.

حال به بررسی دقیق مدل هایی که برای این پروژه طراحی کردم می پردازیم و نحوه ی ساخت آنها را می بینیم.

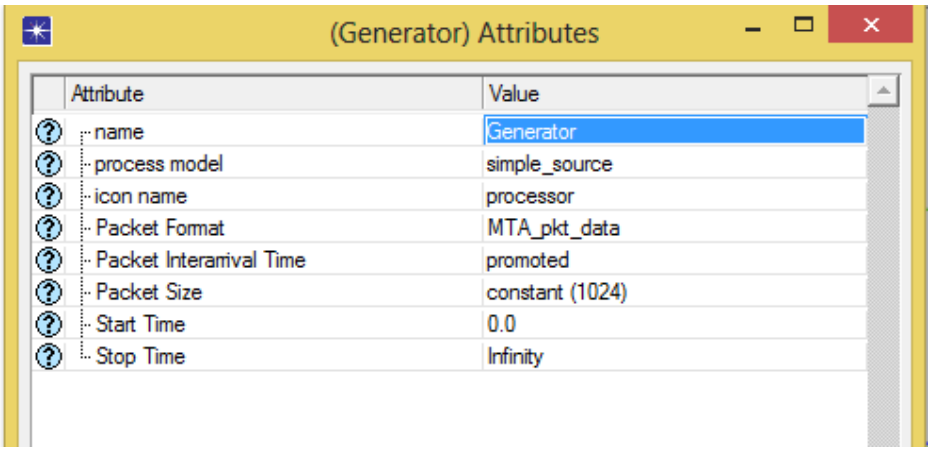
در شکل ۱۳ یکی از Nodeهای Client را می بینیم.



شکل ۱۳

همانطور که می بینیم یک پروسسور برای تولید پکت های قرار دادیم. سپس از یک پروسسور دیگر برای پردازش های اصلی استفاده کرده ایم. فرستنده ها و گیرنده ها را نیز برای هر لینک ورودی و خروجی تعریف کردیم. از آنجا که تعداد ماکزیمم لینک های ورودی داده ۳ و مینیم آن ۱ است؛ تعداد آنها در هر Node ۳ عدد است. همچنین هر Node برای ارتباط با Scheduler یک فرستنده و گیرنده ی خاص دارد.

ابتدا به تنظیمات Generator می پردازیم. ابتدا با قرار دادن یک Processor در شکل و سپس right-click کردن روی آن و انتخاب Edit attribute به شکل ۱۴ می رسیم.

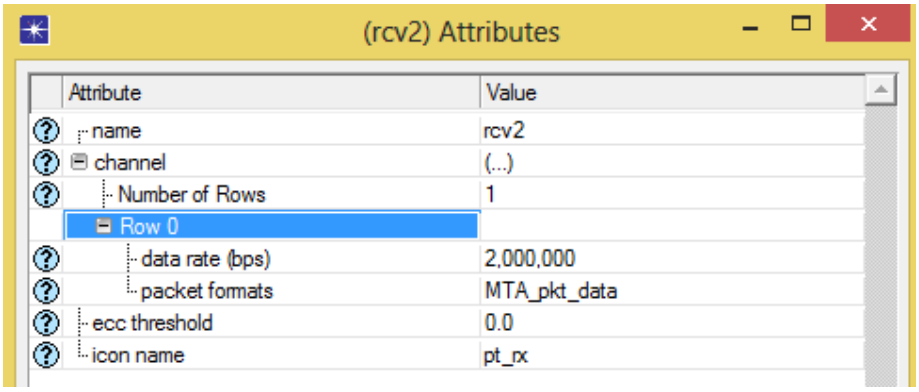


شکل ۱۴

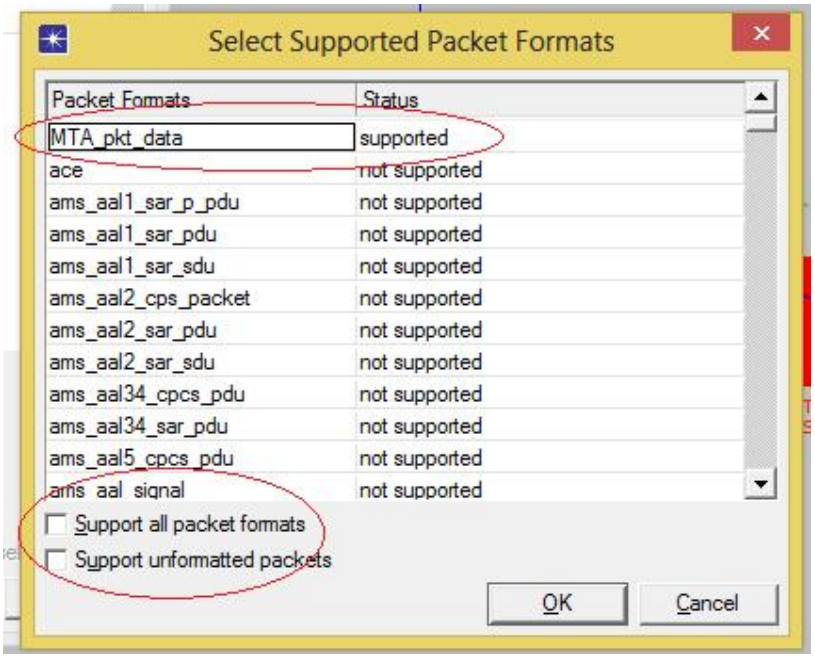
در این شکل Process model را simple source انتخاب می کنیم. نوع پکت های تولیدی را data و Interarrival Time را با right-click، Promoted می کنیم تا بتوانیم در لایه های بالاتر کنترل کنیم. سایر موارد هم مانند شکل تنظیم می کنیم.

پروسسور بعدی را هم قرار می دهیم. (توضیح ادامه ی این پروسسور بعد از کامل شدن بخش بعدی)

برای گیرنده و فرستنده ها هم با توجه به کاربرد آنها نوع پکت پشتیبانی شده را انتخاب می کنیم. شکل ۱۵ و ۱۶ مربوط به تنظیمات یکی از گیرنده های data است.

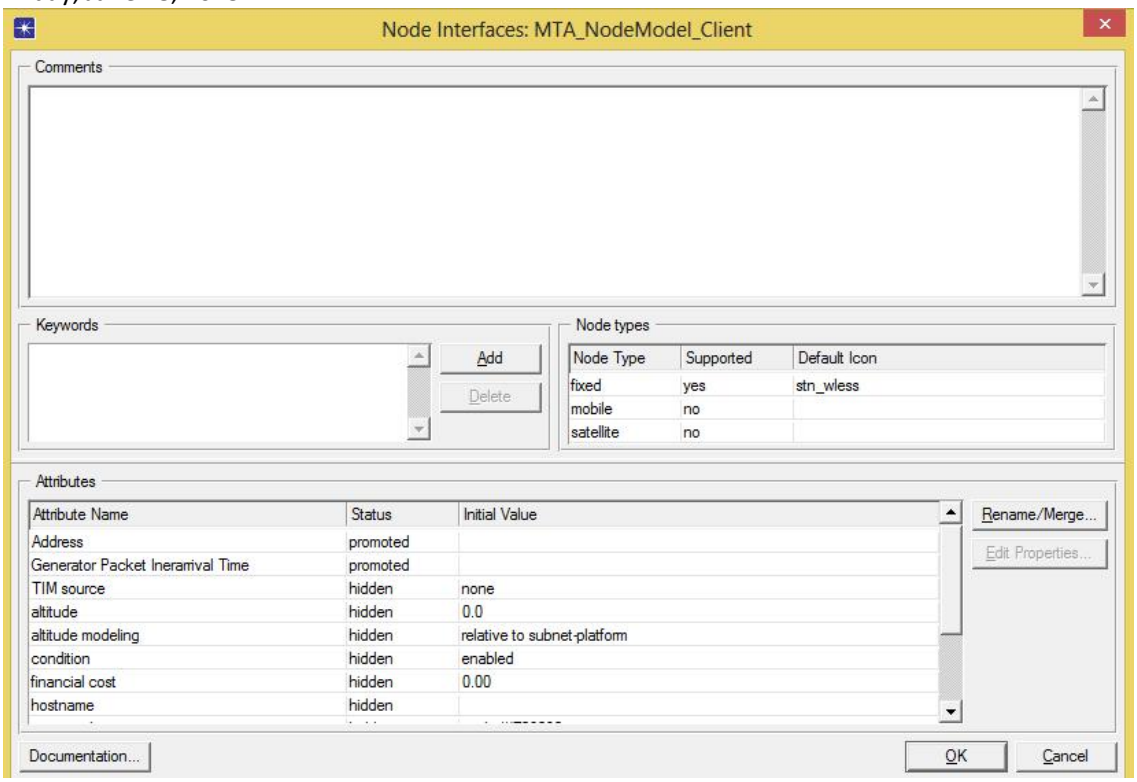


شکل ۱۵

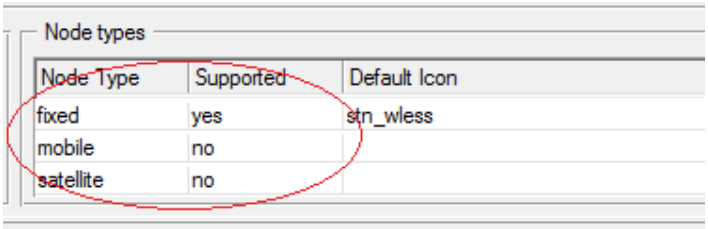


شکل ۱۶

حال روی منوی Interfaces → Node Interfaces کلیک کنید. شکل ۱۷ ظاهر می شود. در این شکل از آنجا که ما این فرض می کنیم Nodeها ثابت اند طبق شکل ۱۸ این بخش را تنظیم کنید. (همچین در این شکل می توان Icon را نیز تغییر داد.)

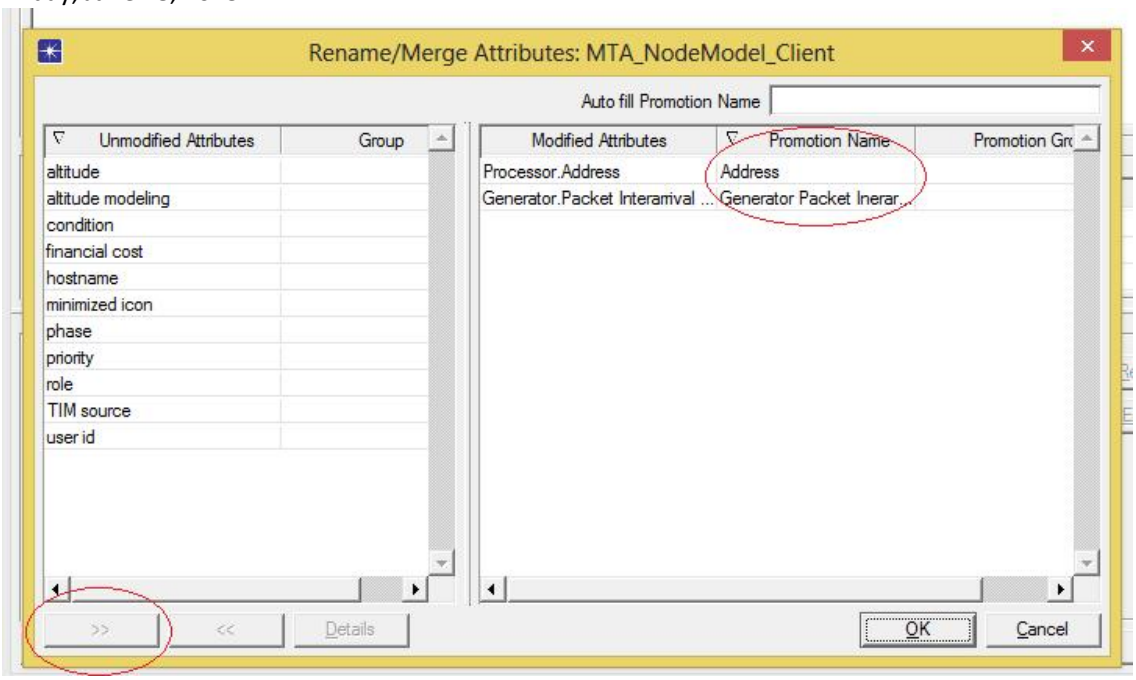


شکل ۱۷

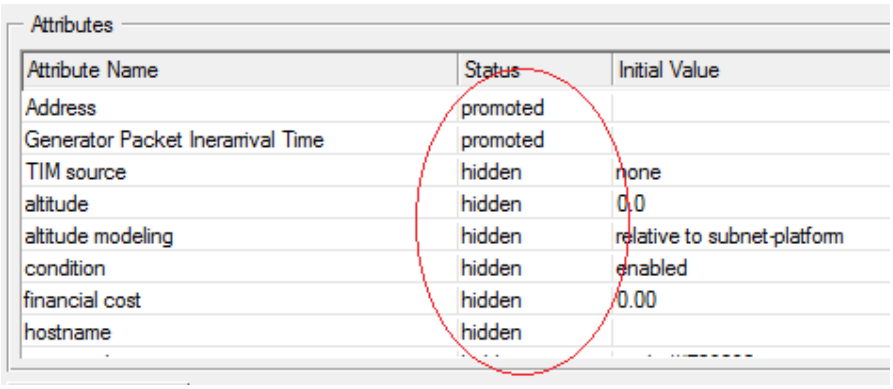


شکل ۱۸

حال کلید Rename/Merge را در سمت راست شکل ۱۷ فشار دهید. در این بخش عناصری که قبلا Promoted کرده بودید همراه با سایر عناصر نشان داده می شوند. حال متغیر های خود را انتخاب کرده و طبق شکل ۱۹ اسامی جدید و بهتری به آنها دهید. این اسامی در لایه ی بالاتر یا همان Project در Attribute این Node دیده می شود. (توجه کنید که Processor.Address در بخش بعدی ساخته می شود.) حال OK را بزنید و مانند شکل ۲۰ سایر پارامترهای غیر ضروری را Hidden کنید. و مقادیر مورد نیاز در لایه ی بالاتر را Promoted.

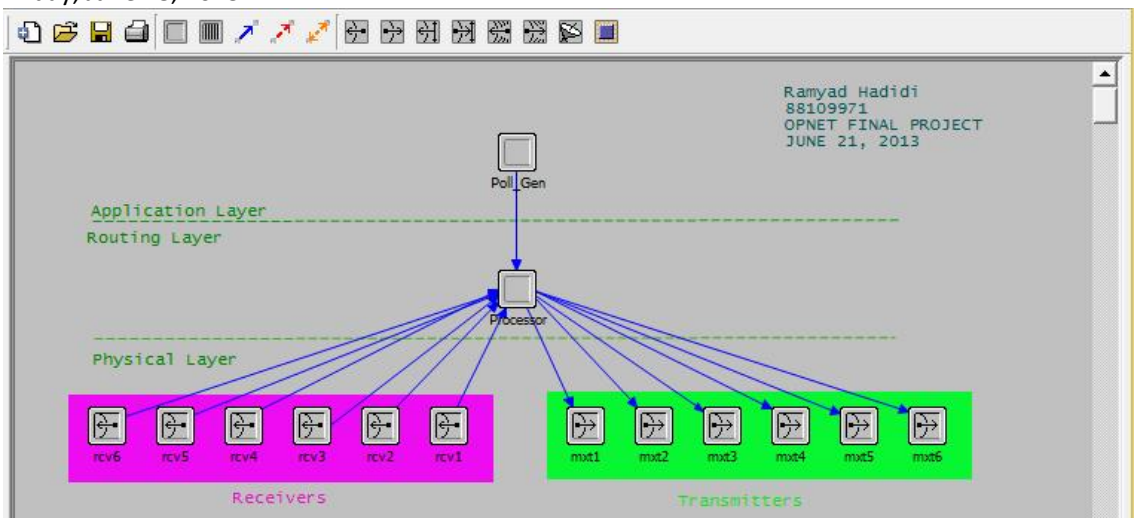


شکل ۱۹



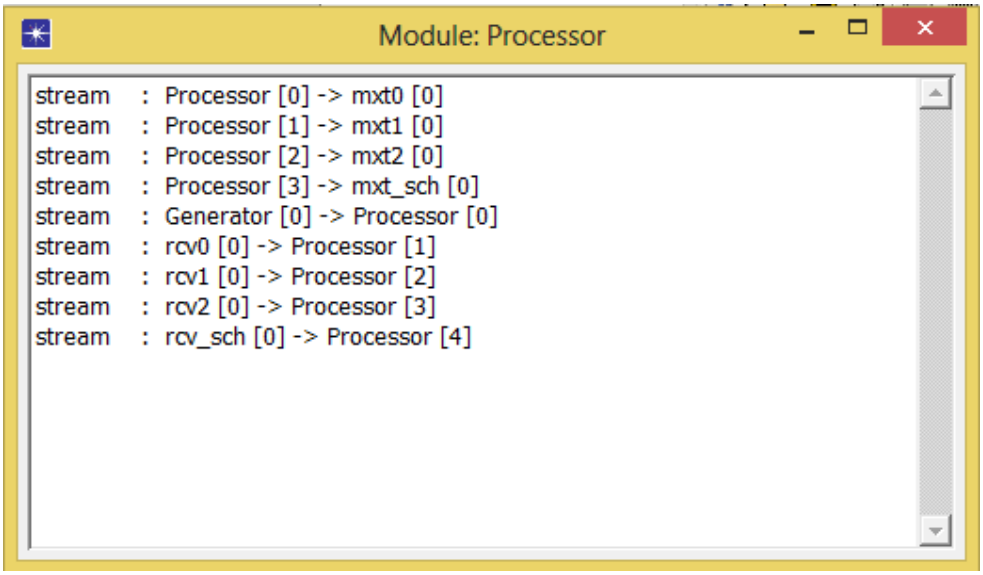
شکل ۲۰

مراحل بالا را برای ساختن Scheduler هم تکرار کنید. با این تفاوت که در اینجا نوع پکت ها از نوع Poll هستند. شکل ۲۱ این Node را نشان می دهد.

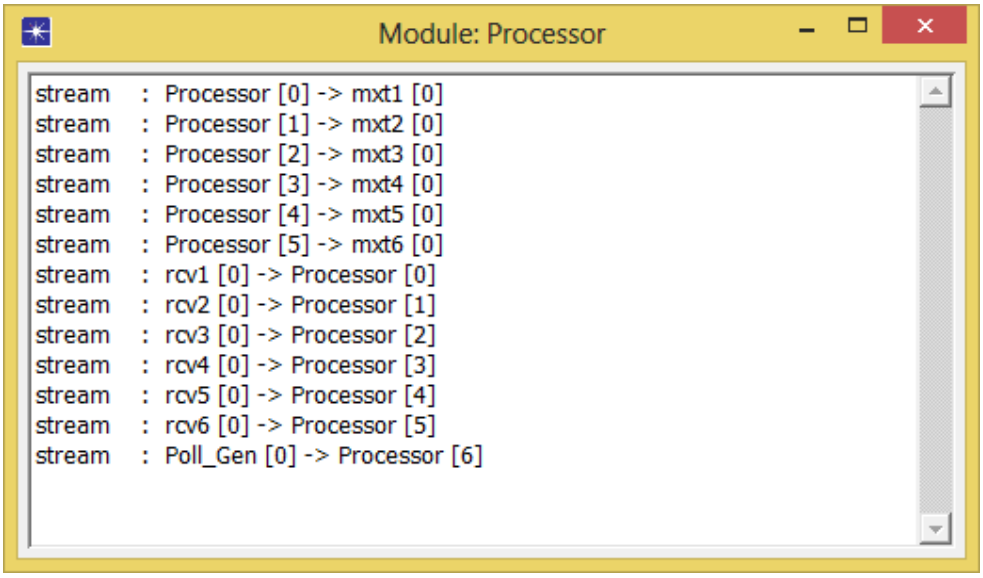


شکل ۲۱

همچنین برای مطمئن شدن از اتصال Processor به سایر ماژول ها با right-click کردن روی آن و انتخاب Show Connectivity می تواند کلیه ی اتصالات را ببینید. شکل ۲۲ برای Client و شکل ۲۳ برای scheduler است.



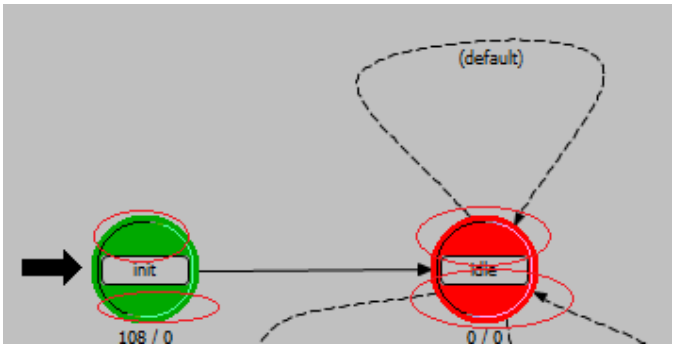
شکل ۲۲



شکل ۲۳

Process Model ۲,۴

در شکل ۳ Process Model را انتخاب کنید. در این صفحه می خواهیم نوع رفتار Processor را تعریف کنیم. نحوه ی تعریف با استفاده از FSM[2] است. هر FSM شامل چند حالت است، که انتقال بین این حالت ها با Transitionهای معمولی یا با شرط اتفاق می افتد. در OPNET هر حالت شامل یک بخش ورودی و یک بخش خروجی است. بخش ورودی هنگام وارد شدن به حالت اجرا می شود و بخش خروجی هنگام خروج از آن. شکل ۲۴ این بخش ها مشخص شده اند.



شکل ۲۴

در این شکل دو نوع حالت می بینم. سبز و قرمز. سبز حالت Forced است و به این معناست که OPNET بدون توقف در این حالت از بعد از اجرای کد درون آن از آن خارج می شود. حالت قرمز حالت Not Forced است. که OPNET با ورود به آن بخش ورودی را اجرا می کند و منتظر بروز اتفاقی (interrupt) می ماند. می توان نوع این اتفاق را تعریف کرد. مثلا در شکل ۲۴ این اتفاق default است که شامل کلیه ی اتفاقات می شود.

[2] Finite State Machine

همانطور که می بینیم Transition ها هم دو نوعند، ساده یا همواره درست که خطوط Solid هستند و شرطی که خطوط Dashed هستند (شرط default هم نوع شرط است.)

فلش سیاه شکل ۲۴ نیز جهت ورود به Process را در هنگام شروع برنامه نشان می دهد.

هر Process Model علاوه بر کد های موجود در حالت ها کد های دیگری هم دارد که توسط Toolbar قابل دسترسی است. شکل ۲۴ این Toolbar را نشان می دهد.

شکل ۲۵

- SV: State Variables مقادیری که در این بخش تعریف می شوند همواره در همه جای Process قابل دسترسی هستند و از بین نمی روند.
- TV: Temporary Variables مقادیر temp در این بخش تعریف می شوند. کاربرد اصلی این مقادیر در header block یا کد های خارج از function در stateها است که نمی توانیم گاهی متغیری تعریف کنیم.
- HB: Header Block که همانند Header کلیه ی فایل های C است.
- FB: Function Block در این بخش می توانیم Functionها را بنویسیم. این توابع در کلیه ی حالت ها قابل استفاده اند.

در بخش Header با توجه به شکل ۲۳ و ۲۲ کلیه ی stream های ورودی و خروجی را define می کنیم.
شکل ۲۶ این بخش را برای Client و شکل ۲۷ برای Scheduler نشان می دهد.

```
/* packet stream definitions */
#define MXT0_OUT_STRM           0
#define MXT1_OUT_STRM           1
#define MXT2_OUT_STRM           2
#define MXT_SCH_OUT_STRM        3

#define GENERATOR_IN_STRM       0
#define RCV0_IN_STRM            1
#define RCV1_IN_STRM            2
#define RCV2_IN_STRM            3
#define RCV_SCH_IN_STRM         4
```

شکل ۲۶

```
/* packet stream definitions */
#define MXT1_OUT_STRM            0
#define MXT2_OUT_STRM            1
#define MXT3_OUT_STRM            2
#define MXT4_OUT_STRM            3
#define MXT5_OUT_STRM            4
#define MXT6_OUT_STRM            5


#define RCV1_IN_STRM             0
#define RCV2_IN_STRM             1
#define RCV3_IN_STRM             2
#define RCV4_IN_STRM             3
#define RCV5_IN_STRM             4
#define RCV6_IN_STRM             5
#define POLL_GEN_IN_STRM         6
#define QUEUE_CHECK_GEN_IN_STRM  7
```

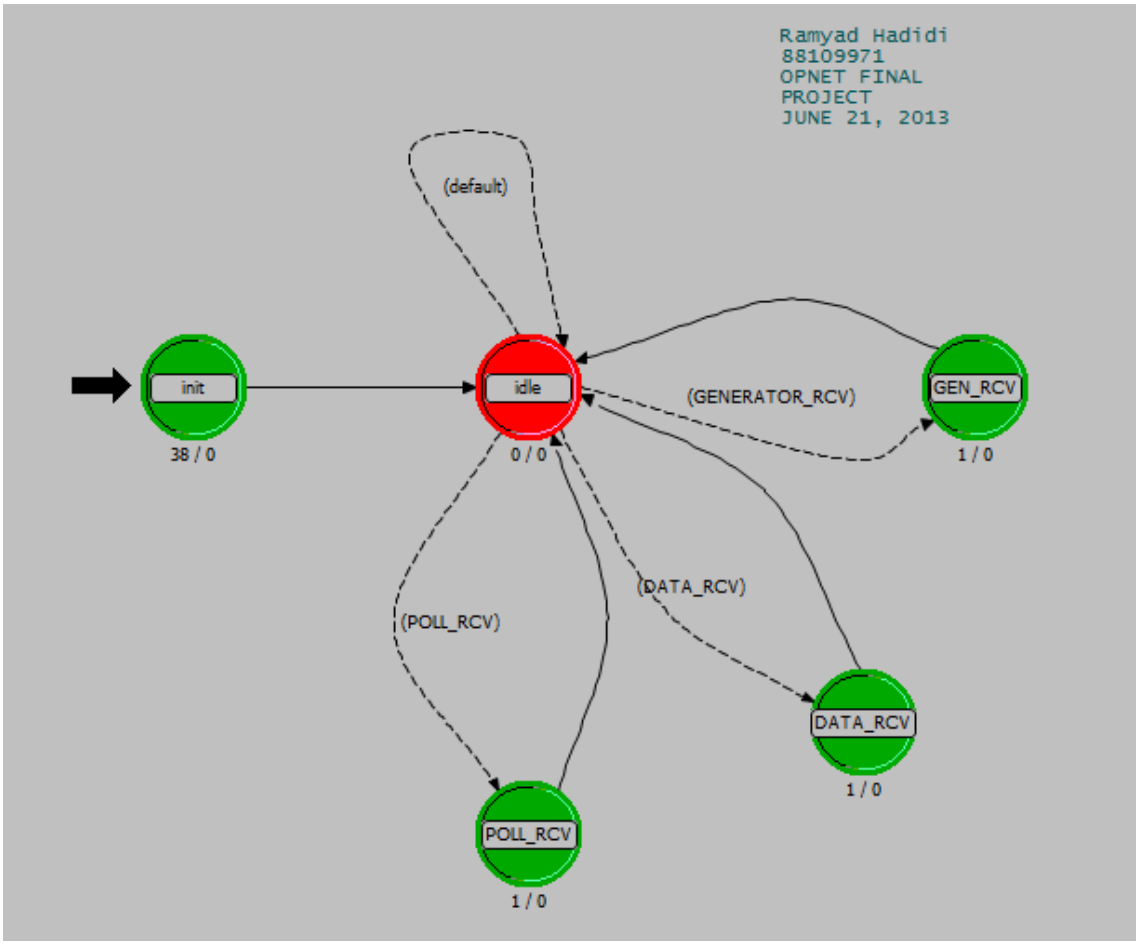
شکل ۲۷

حال کلیه ی لوازم لازم برای طراحی شبکه را در اختیار داریم. در بخش بعدی کد ها و مراحل دیگر را با جزئیات بیشتر توضیح می دهیم.

۳. توضیحات مربوط به پروژه

۳,۱.ساخت Clients

طراحی Process در شکل ۲۸ آمده است.



شکل ۲۸

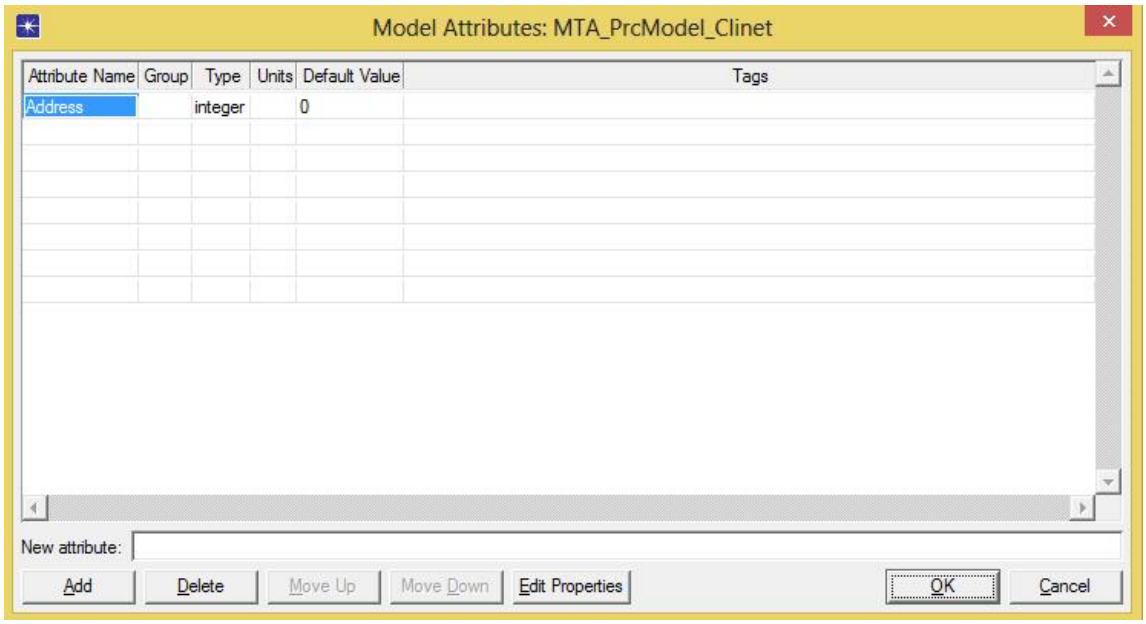
همانطور که می بینیم برای هر حالت یک شرط گذاشته ام. این شرط ها هم در Header تعریف می شوند. شکل ۲۹ این بخش را نشان می دهد.

```
/* transition macros */
#define POLL_RCV (op_intrpt_type() == \
    OPC_INTRPT_STRM && op_intrpt_strm() == RCV_SCH_IN_STRM)
#define DATA_RCV (op_intrpt_type() == \
    OPC_INTRPT_STRM && (op_intrpt_strm() == RCV0_IN_STRM || \
                        op_intrpt_strm() == RCV1_IN_STRM || \
                        op_intrpt_strm() == RCV2_IN_STRM))
#define GENERATOR_RCV (op_intrpt_type() == \
    OPC_INTRPT_STRM && op_intrpt_strm() == GENERATOR_IN_STRM)
```

شکل ۲۹

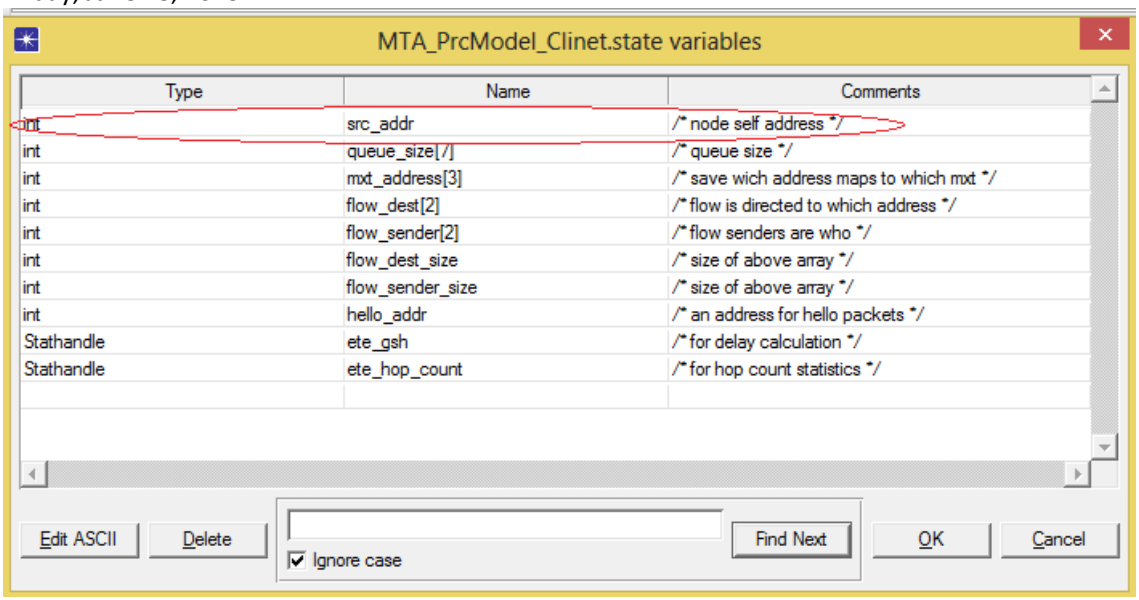
به عنوان مثال حالت POLL_RCV وقتی اتفاق می افتد که از Scheduler یک پکت به Processor برسد. برای سایر شرط ها هم همین طور.

Client ها باید آدرس داخلی خود را بدانند. برای این کار در منوی Interfaces→Model Attributes که در شکل ۳۰ آمده است متغیر آدرس را تعریف می کنیم.



شکل ۳۰

حال باید این متغیر را در کد به متغیر خود اختصاص دهیم. این کار را در حالت init که اولین حالتی است که اجرا می شود می کنیم. برای این کار ابتدا در بخش SV (شکل ۳۱) متغیری با نام دلخواه تعریف می کنیم. و سپس کد شکل ۳۲ را برای Assign کردن به آن استفاده می کنیم.



شکل ۳۱

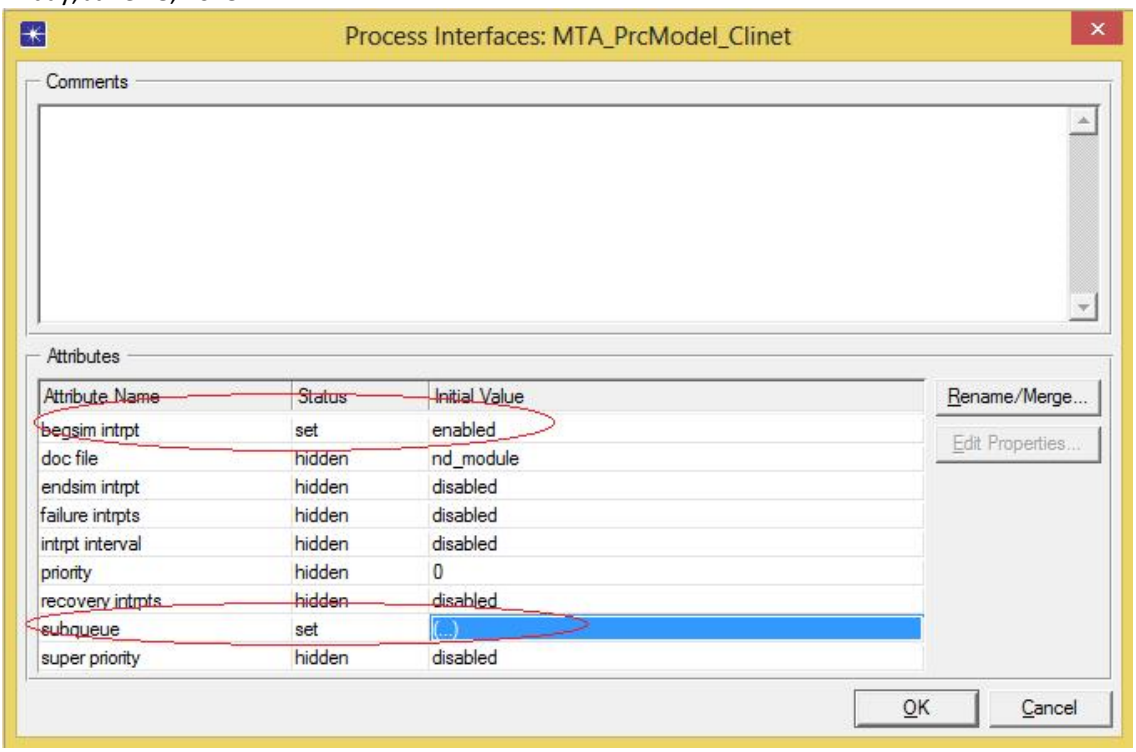
```
//get the self address
op_ima_obj_attr_get(op_id_self(), "Address", &src_addr);
```

شکل ۳۲

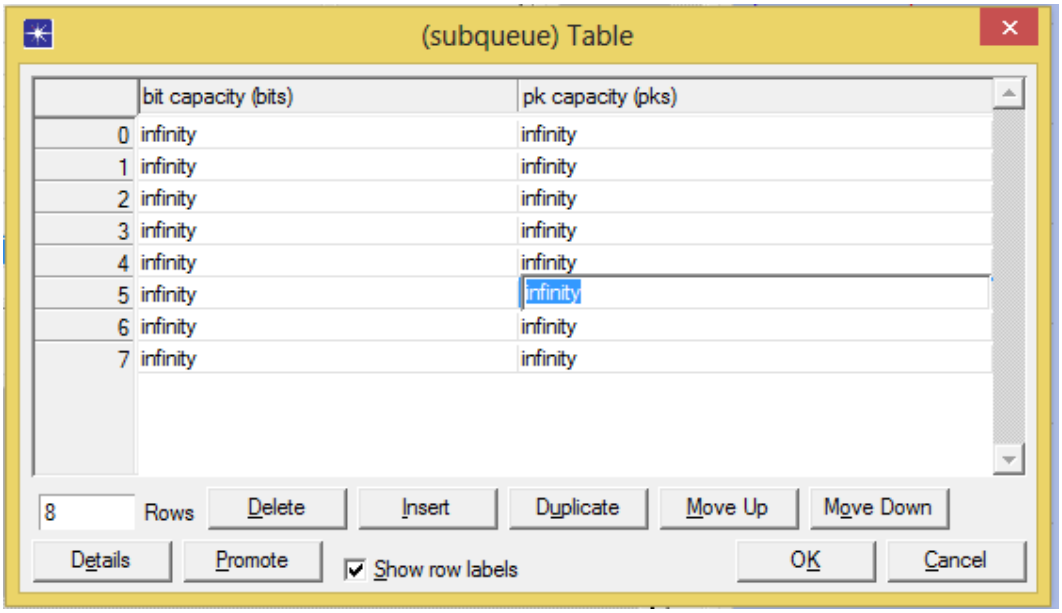
(این آدرس را در بخش های قبل Promoted کردیم و به لایه های بالاتر رساندیم.)

حال به بخش Interfaces→Process Interfaces می رویم. (شکل ۳۳) در این بخش ابتدا begsim intrpt را فعال می کنیم تا در هنگام شروع به این Process یک interrupt بیاید. همچنین برای تعریف Queueهای مورد نیاز روی sub queue کلیک کرده و Queue های خود را تعریف می کنیم. (شکل ۳۴)

برای دسترسی به این Queue ها می توانیم از کد های شکل ۳۵ استفاده کنیم. استفاده از این Queue ها بسیار آسان است؛ چون دو مقدار Head و Tail را به عنوان یک متغیر در اختیار ما می گذارد.



شکل ۳۳



شکل ۳٤

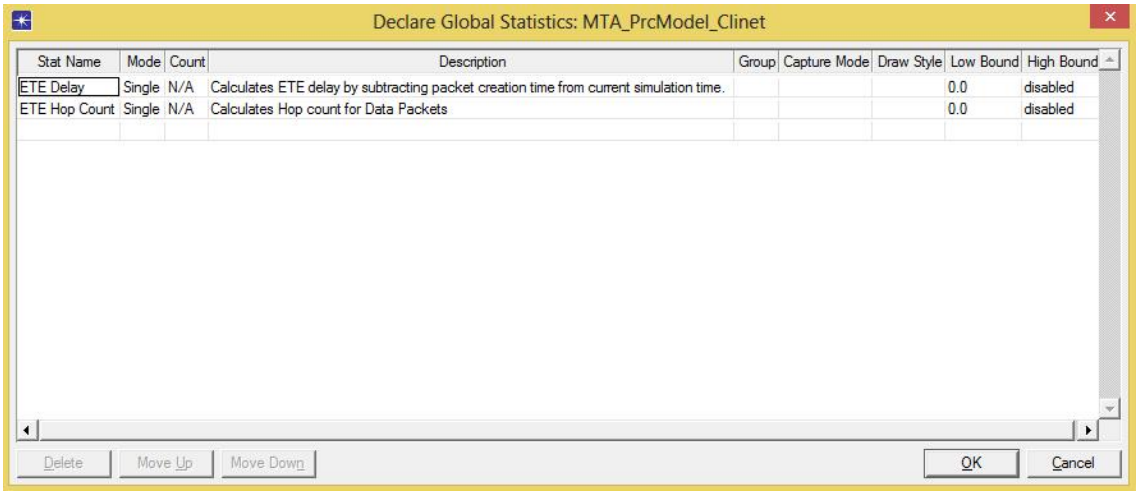
Insert or remove a packet from a specified subqueue.

- *op_subq_pk_insert (subq_index, pkptr, pos_index)* –> completion code
- *op_subq_pk_remove (subq_index, pos_index)* –> pointer to packet removed from the specified subqueue

شکل ۳۵

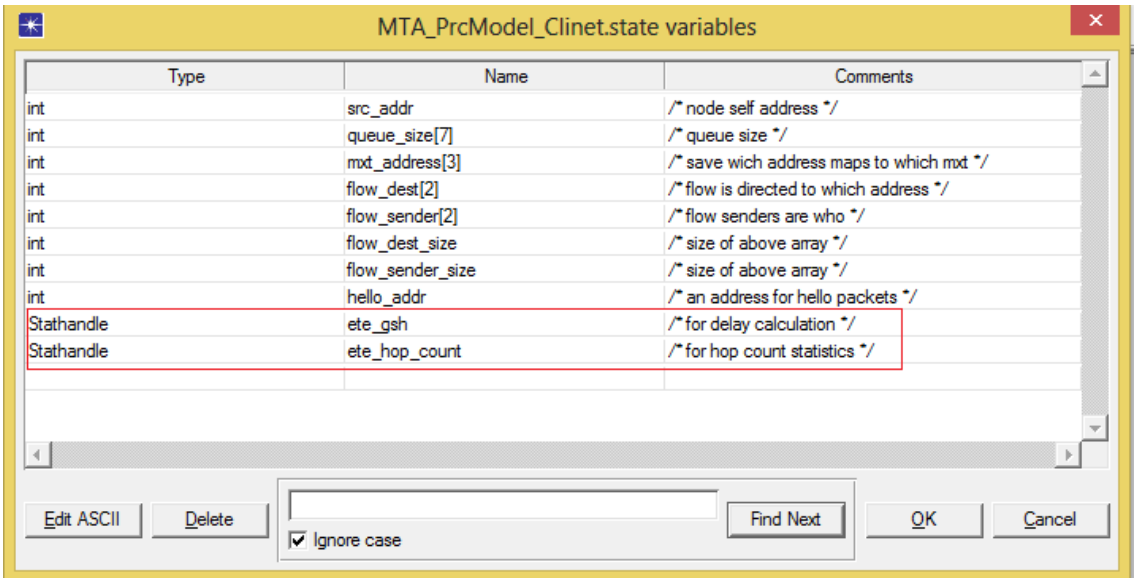
حال برای جمع آوری Statistics باید مراحل زیر را طی کنیم.

ابتدا در منوی Interfaces→Global Statistics را انتخاب کرده و همانند شکل ۳۶ آن را پر کنیم. از آنجا که در این پروژه به Delay و Hopcount نیاز است، من این مقادیر را تعریف کردم.



شکل ۳۶

حال در بخش SV یک متغیر Global داخلی را طبق شکل ۳۷ به آنها اختصاص می دهیم. توجه کنید که نوع آنها Standhandle است. حال دوباره در بخش init این مقادیر را به آنها Assign می کنیم. (شکل ۳۸)



شکل ۳۷

```
//*************************************************
//*******STATISTICS
ete_gsh = op_stat_reg ("ETE Delay",OPC_STAT_INDEX_NONE,OPC_STAT_GLOBAL);
ete_hop_count = op_stat_reg ("ETE Hop Count",OPC_STAT_INDEX_NONE,OPC_STAT_GLOBAL);
```

شکل ۳۸

در پایان نیز، در هنگامی که پکت به مقصد رسید مقادیر را به این متغیر ها می دهیم. شکل ۳۹ کد این بخش را نشان می دهد.

```
//other packets so check them if they reach to the destination
if (pkt_dest_addr == src_addr)
    {

    ete_delay = op_sim_time () - op_pk_creation_time_get (pkptr);
    op_stat_write (ete_gsh, ete_delay);

    op_pk_nfd_get_int32 (pkptr, "hop_count", &hop_count_temp);
    hop_count_temp++;
    op_stat_write (ete_hop_count, hop_count_temp);

    op_pk_destroy (pkptr);
    FOUT;
    }
```

شکل ۳۹

۳,۲. الگوریتم کاری Clientها

ابتدا هر Client برای شناختن همسایه های خود و آدرس های آنها یک پکت با آدرس ۹۹۹ که به عنوان آدرس Hello در همه ی آنها مشخص شده به کلیه ی خط های متصل به خود می فرستد. در نتیجه هر Client با دریافت این Packet می تواند بفهمد در هر گیرنده و فرستنده اش چه کسی و با چه آدرسی در نشسته. این پروسس در ابتدا اجرا می شود پس شبکه ی ما در هنگام کار نمی تواند تغییرات را احساس کند. از طرفی هر Client پس از update کردن جدول اتصالات خود تغییرات را به Scheduler نیز می فرستد. (این خاصیت برای dynamic routing نوشته شد که در نهایت به علت سختی ادامه داده نشد!)

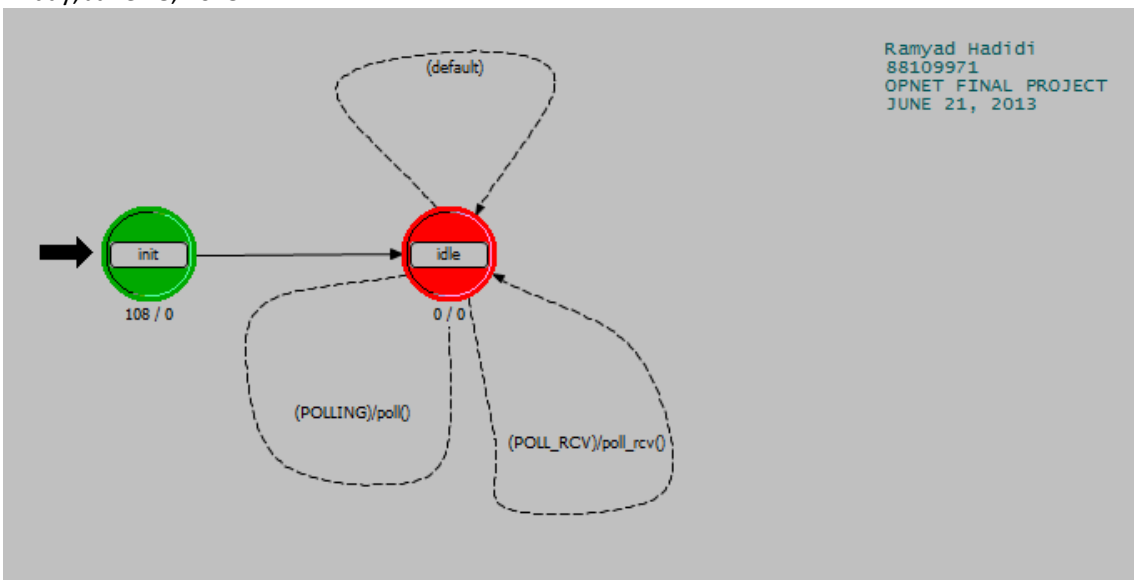
Client ها با استفاده از جدول های فرستنده و گیرنده ای که در خود آنها تعریف شده می فهمند که آیا پکت رسیده از Generator را آیا باید در queue مربوطه ذخیره کنند یا خیر. مثلا در این پروژه ارسال کنند ها گره ۲ و ۶ هستند. پس گره ۳ در صورت گرفتن پکت داده ای از Generator خود آن را Ignore می کند و گره ۲ آن را در queue مربوط به گیرنده اش ذخیره می کند.

Client ها متغیر هایی به عنوان سایز Queue ها در خود دارند که آن را در هر poll از طرف scheduler به او می فرستند. مقادیر این متغیر با گرفتن و فرستادن داده تغییر می کند و ربطی به درست رسیدن یا نرسیدن داده ندارد.

سایر موارد همانند تعریف پروژه انجام می شود و می توان برای فهم بهتر به پروژه مراجعه کرد.

۳,۳. ساخت Scheduler

شکل ۴۰ ساختار Process، Scheduler را نشان می دهد. همانطور که می بینیم در این ساختار از نوع دیگری از Transition ها استفاده شده. در این نوع اگر شرط تحقق پیدا کند؛ تابع بعد از / اجرا می شود. این نمایش تفاوتی در عملکرد با مدل قبلی ندارد.



شکل ۴۰

سایر موارد همانند Client درست می شود. حال به بررسی الگوریتم می پردازیم.

۳,۴. الگوریتم کاری Scheduler

نحوه ی کار Scheduler به این نحو است که در بازه های مشخص با ارسال پکت Client Poll ها را poll می کند. توجه کنید که طبق شکل ۶ و شکل ۴۱ این پکت برای poll ،command کردن و فرستادن اطلاعات استفاده می شود و command_type Field برای این منظور استفاده می شود.

```
/*packet formats*/
#define pkt_poll_poll_request   0
#define pkt_poll_poll_answer    1
#define pkt_poll_command        2
#define pkt_poll_hello_info     3
```

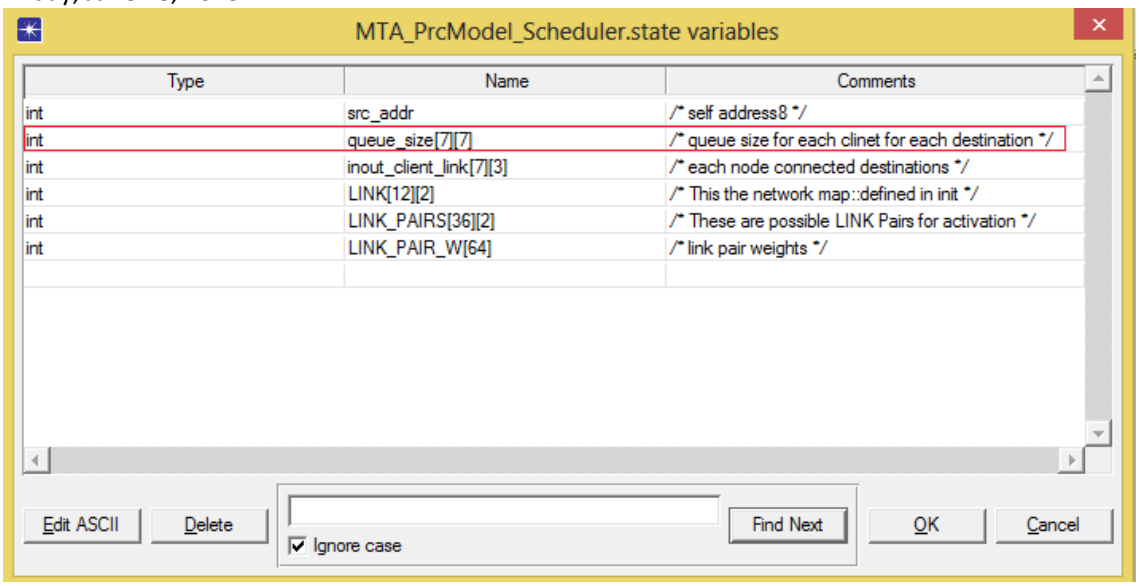
شکل ۴۱

در صورتی هم که پکت اطلاعاتی مربوط به Queue های هر Clinet پیدا کند، جدول های داخلی خود را update میکند. شکل ۴۲ این متغیر را نشان می دهد. سایز این متغیر ۷ در ۷ تعریف کردم که مانند آدرس هر نود باشد.

چون Routing ما static است؛ در scheduler متغیر های مربوط به نحوه ی وصل شدن link ها را در متغیر LINK همانند شکل ۴۳ ذخیره کرده ایم.

از طرفی کلیه ی لینک های دوتایی ممکن را نیز که می توانند با هم فعال شوند را در متغیر LINK_PAIR ذخیره کرده ام و آنها را طبق شکل ۴۴ در ابتدا مقدار دهی کرده ام.

در طول اجرا هر بار متغیر LINK_PAIR_W که وزن این زوج لینک ها است محاسبه کرده و دستور ارسال را با آن نود ها می دهیم.



شکل ۴۲

	Link ID	Begin node	End node	Capacity
0	1 → 2	1	2	∞
1	2 → 1	2	1	∞
2	2 → 3			
3	3 → 2			
4	3 → 4			
5	4 → 3			
6	4 → 5			
7	5 → 4			
8	5 → 6			
9	6 → 5			
10	6 → 2			
11	2 → 6			

شکل ۴۳

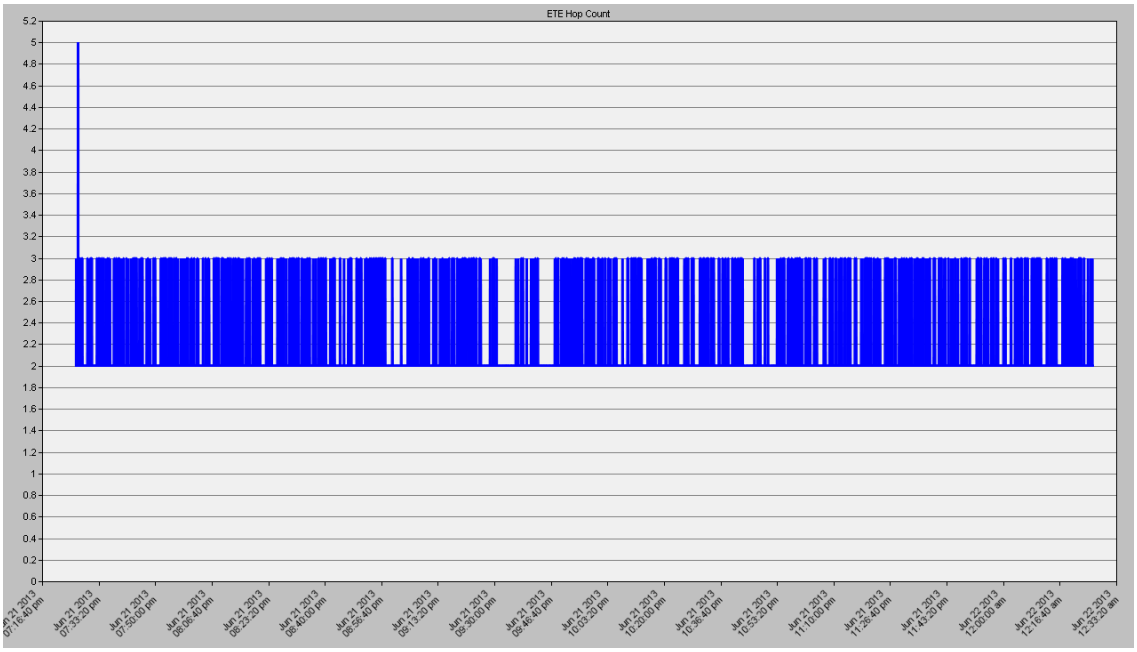
```
//********POSSIBLE LINK PAIRS
LINK_PAIRS[0][0] = 0 ; LINK_PAIRS[0][1] = 4;
LINK_PAIRS[1][0] = 0 ; LINK_PAIRS[1][1] = 5;
LINK_PAIRS[2][0] = 0 ; LINK_PAIRS[2][1] = 6;
LINK_PAIRS[3][0] = 0 ; LINK_PAIRS[3][1] = 7;
LINK_PAIRS[4][0] = 0 ; LINK_PAIRS[4][1] = 8;
LINK_PAIRS[5][0] = 0 ; LINK_PAIRS[5][1] = 9;

LINK_PAIRS[6][0] = 1 ; LINK_PAIRS[6][1] = 4;
LINK_PAIRS[7][0] = 1 ; LINK_PAIRS[7][1] = 5;
LINK_PAIRS[8][0] = 1 ; LINK_PAIRS[8][1] = 6;
LINK_PAIRS[9][0] = 1 ; LINK_PAIRS[9][1] = 7;
LINK_PAIRS[10][0] = 1 ; LINK_PAIRS[10][1] = 8;
LINK_PAIRS[11][0] = 1 ; LINK_PAIRS[11][1] = 9;

LINK_PAIRS[12][0] = 2 ; LINK_PAIRS[12][1] = 6;
LINK_PAIRS[13][0] = 2 ; LINK_PAIRS[13][1] = 7;
LINK_PAIRS[14][0] = 2 ; LINK_PAIRS[14][1] = 8;
LINK_PAIRS[15][0] = 2 ; LINK_PAIRS[15][1] = 9;

LINK_PAIRS[16][0] = 3 ; LINK_PAIRS[16][1] = 6;
LINK_PAIRS[17][0] = 3 ; LINK_PAIRS[17][1] = 7;
LINK_PAIRS[18][0] = 3 ; LINK_PAIRS[18][1] = 8;
LINK_PAIRS[19][0] = 3 ; LINK_PAIRS[19][1] = 9;

LINK_PAIRS[20][0] = 10 ; LINK_PAIRS[20][1] = 4;
LINK_PAIRS[21][0] = 10 ; LINK_PAIRS[21][1] = 5;
LINK_PAIRS[22][0] = 10 ; LINK_PAIRS[22][1] = 6;
LINK_PAIRS[23][0] = 10 ; LINK_PAIRS[23][1] = 7;

LINK_PAIRS[24][0] = 11 ; LINK_PAIRS[24][1] = 4;
LINK_PAIRS[25][0] = 11 ; LINK_PAIRS[25][1] = 5;
LINK_PAIRS[26][0] = 11 ; LINK_PAIRS[26][1] = 6;
LINK_PAIRS[27][0] = 11 ; LINK_PAIRS[27][1] = 7;

LINK_PAIRS[28][0] = 4 ; LINK_PAIRS[28][1] = 8;
LINK_PAIRS[29][0] = 4 ; LINK_PAIRS[29][1] = 9;
LINK_PAIRS[30][0] = 4 ; LINK_PAIRS[30][1] = 10;
LINK_PAIRS[31][0] = 4 ; LINK_PAIRS[31][1] = 11;

LINK_PAIRS[32][0] = 5 ; LINK_PAIRS[32][1] = 8;
LINK_PAIRS[33][0] = 5 ; LINK_PAIRS[33][1] = 9;
LINK_PAIRS[34][0] = 5 ; LINK_PAIRS[34][1] = 10;
LINK_PAIRS[35][0] = 5 ; LINK_PAIRS[35][1] = 11;
```

شکل ٤٤

۵. گزارش های شبیه سازی

زمان تولید پکت های داده را برای نود های ۲ و ۶، هر دو را با توزیع نمایی و میانگین ۱ تعریف کردم. زمان poll کردن هم هر ۰٫۸ ثانیه یک بار قرار دادم.

در این صورت نمودار Hop count به صورت شکل ۴۵ و میانگین آن به صورت شکل ۴۶ در آمد.

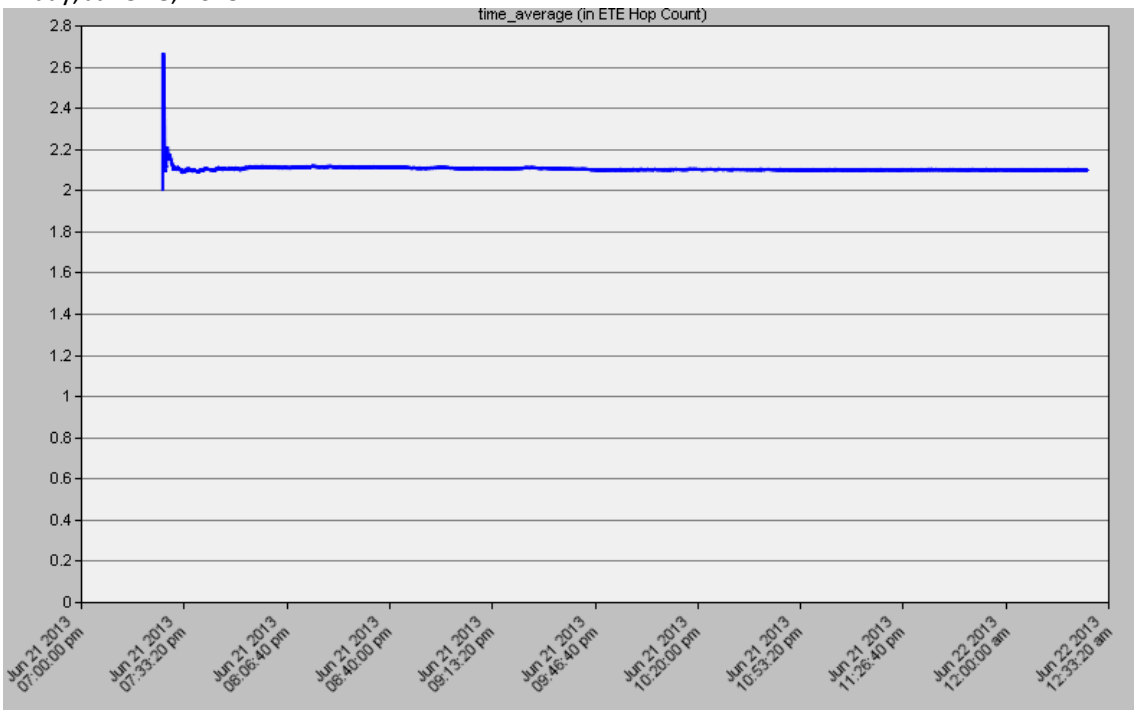


شکل ۴۵- poll هر ۰٫۸ ثانیه و تولید با exp(1)

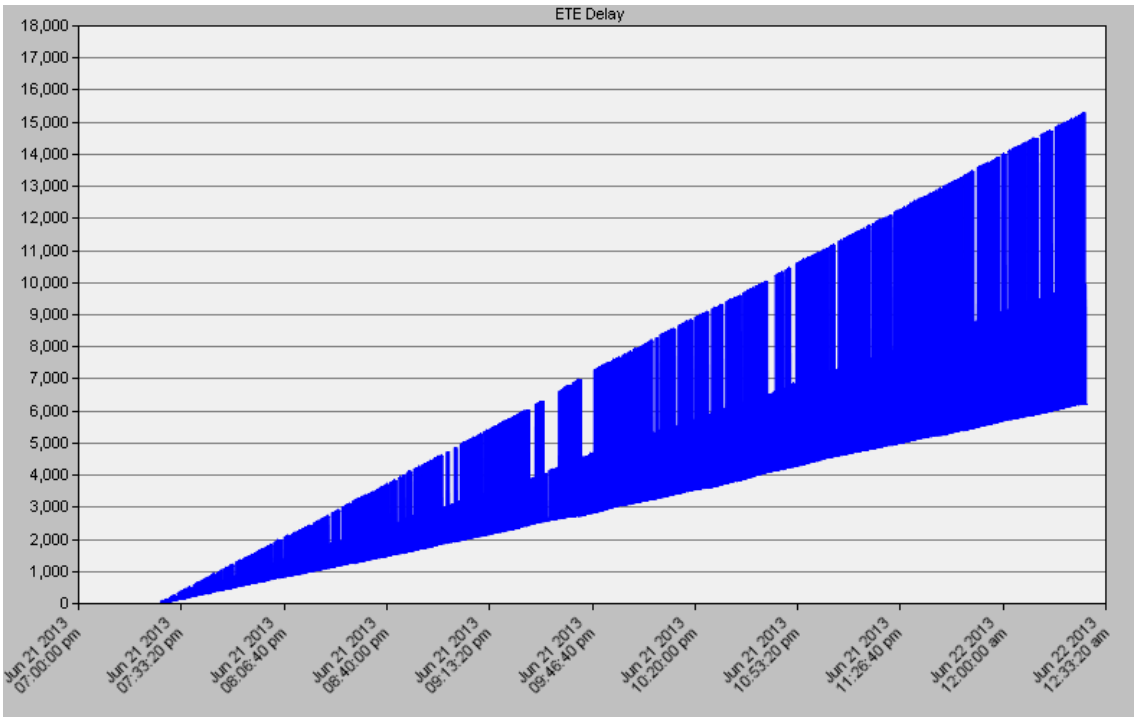
همانطور که می بینیم اکثر پکت ها مسیر ۲ تایی را طی می کنند. یعنی مسیر کوتاه تر ولی گاهی اوقات هم مسیر ۳ تایی را که به علت فعال بودن برخی از نود ها در آن زمان است. پس میانگین ما به ۲ بیشتر خواهد بود.

شکل ۴۷ و ۴۸ هم delay را نشان می دهد. همانطور که می بینیم delay به مرور زیاد می شود. و این به دلیل پر شدن بافر های میانی است. زیرا زمان poll و command ما حداقل ۰٫۸ ثانیه است، در حالی که حجم تولید پکت ها بیشتر است.

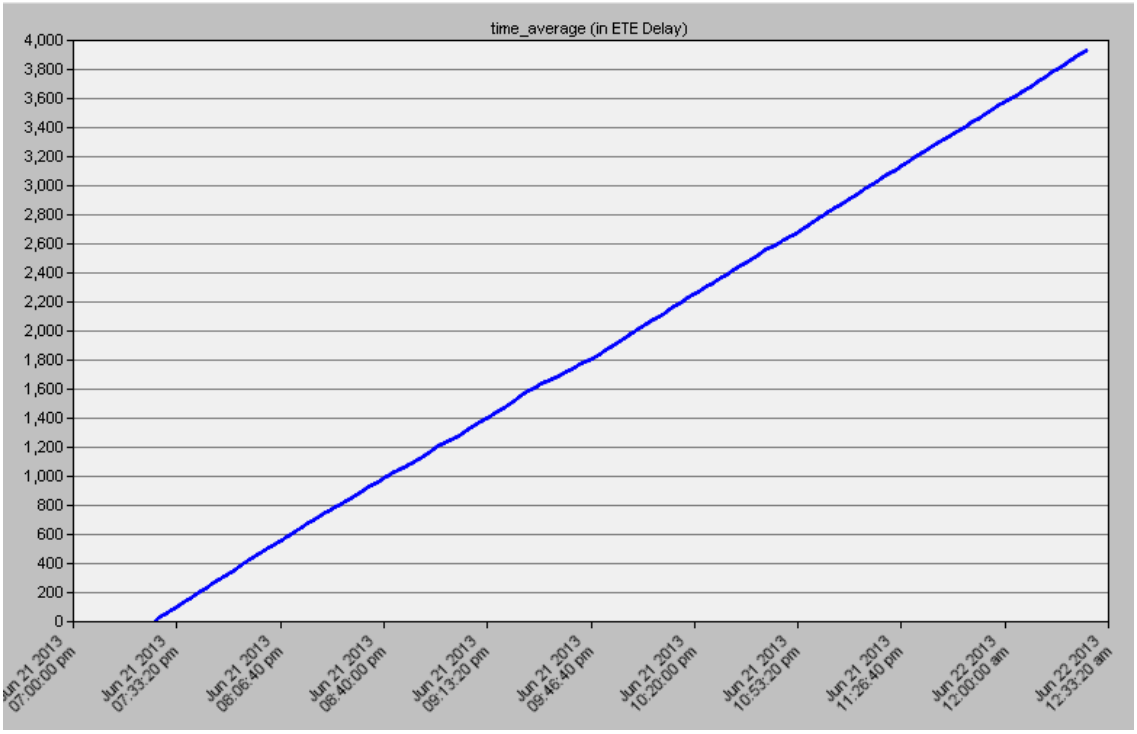
ولی از طرفی می بینیم که مقادیر hop count مقادیر کاملا optimize ای هستند و علت آن این است که چون بافر ها همگی پر هستند و شبکه شلوغ تر است scheduler بهتر می تواند مسیر های مناسب را پیدا کند.



شکل ٤٦ poll - هر ۰٫۸ ثانیه و تولید با exp(1)



شکل ٤٧ - poll هر ۰٫۸ ثانیه و تولید با exp(1)



شکل ۴۸ - poll هر ۰٫۸ ثانیه و تولید با exp(1)

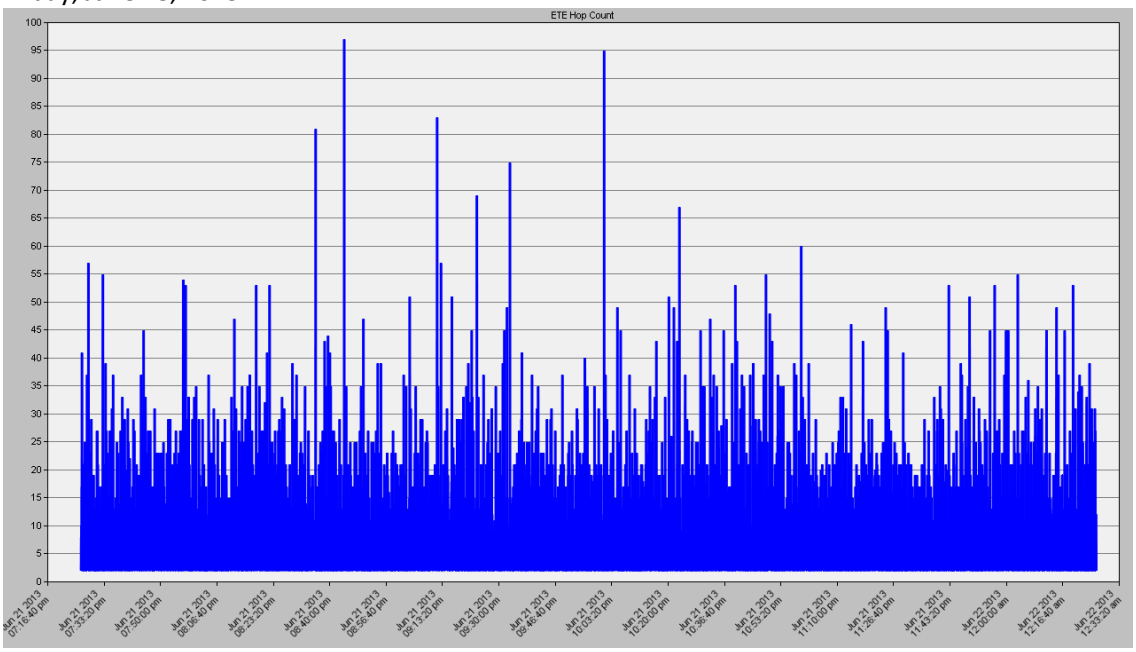
حال طبق توضیحی که در بخش قبل دادیم بیایم و زمان های poll را برای کاهش delay افزایش دهیم. مقدار تولید داده ها همان مقدار قبلی می ماند ولی polling با exp(0.2). در این صورت همان پیش بینی که کرده بودیم صورت می گیرد. از آنجا که در اکثر مواقع به علت خلوت بودن شبکه و خالی بودن بافر ها scheduler نمی تواند بهترین مسیر را پیدا کند hop count ما مانند حالت قبل بهترین نیست.

شکل ۴۹ و ۵۰ نمودار های hop count هستند. همانطور که می بینیم مقادیر بسیار متفاوت هستند و مقدار میانگین هم حدودا ۵ است که برخلاف حالت قبل optimize نیست.

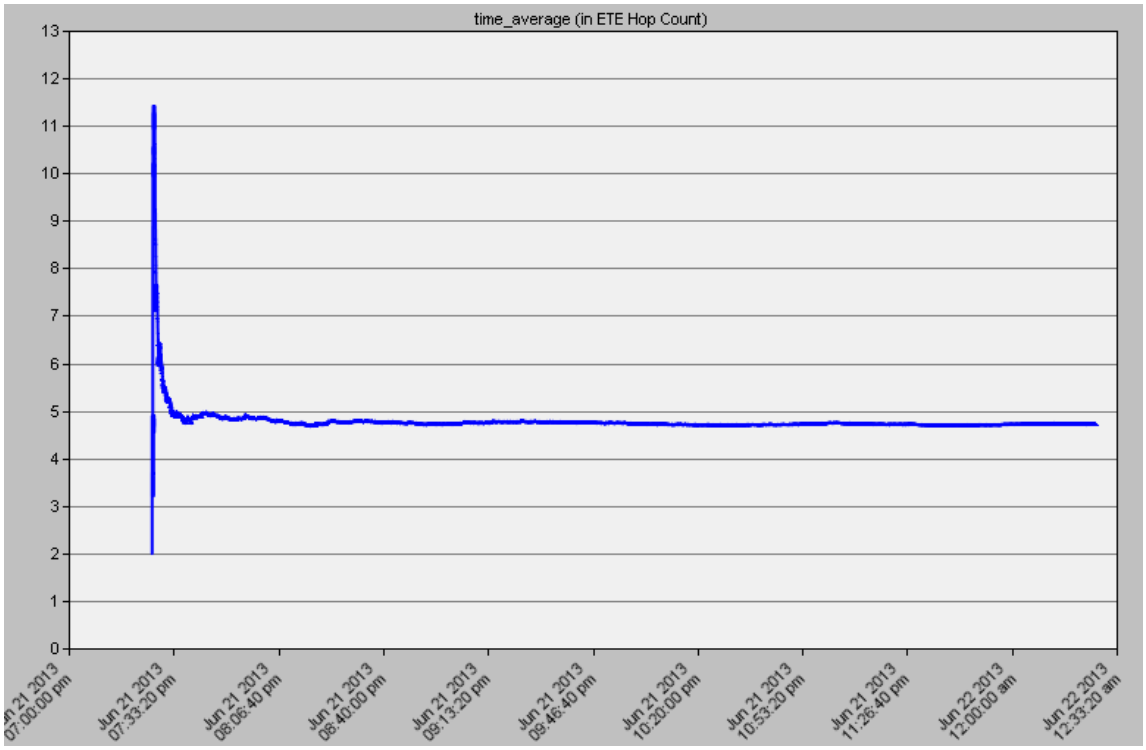
شکل ۵۱ و ۵۲ هم نمودار های delay هستند. همانطور که گفتیم delay را به یک مقدار ثابت رساندیم ولی از طرفی گره ها بار بیشتری برای انتقال داده ها تحمل می کنند (به علت hop count).

شکل ۵۳ هم queue size نود ۲ در این حالت نشان می دهد. که می بینیم مقدار آن ثابت است. ولی همانطور که در شکل ۵۴ می بینیم queue size در simulation اول به مرور زیاد می شود.

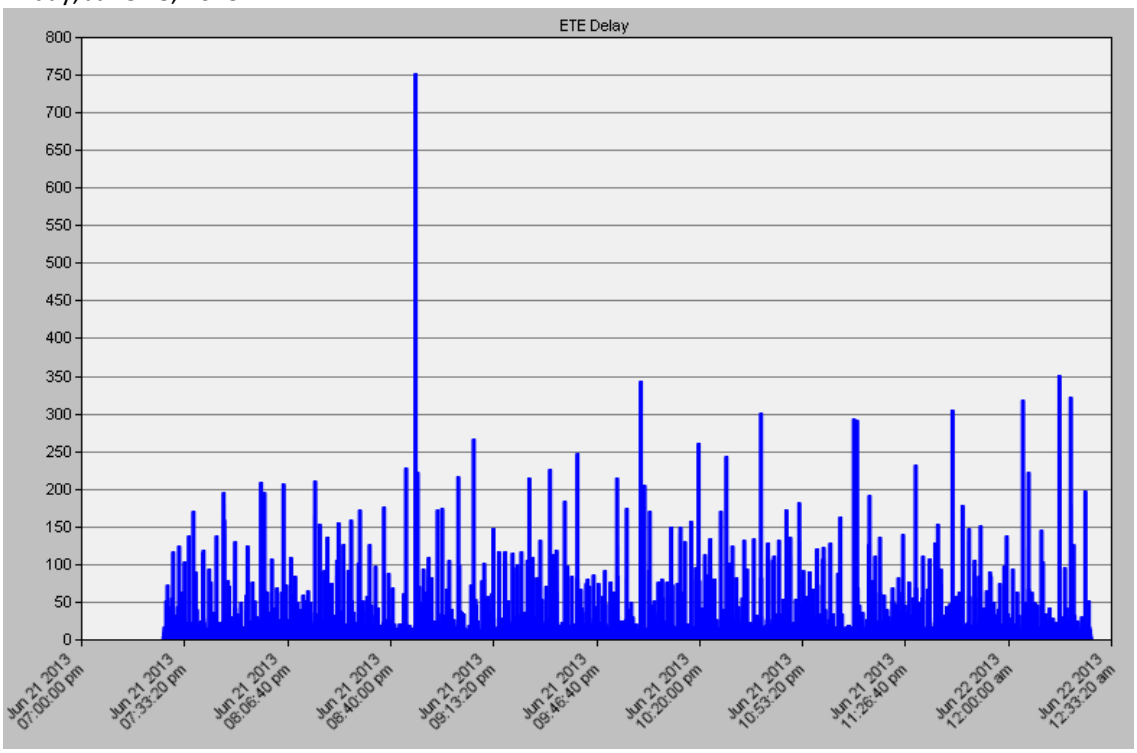
پس همانطور که دیدم، الگوریتم درست عمل می کند و حالت واقع گرانه ی عملی برای ما حالت دوم است که در آن delay ثابت می شود و شبکه به پایداری می رسد ولی hop countها مقادیر optimize نیستند.



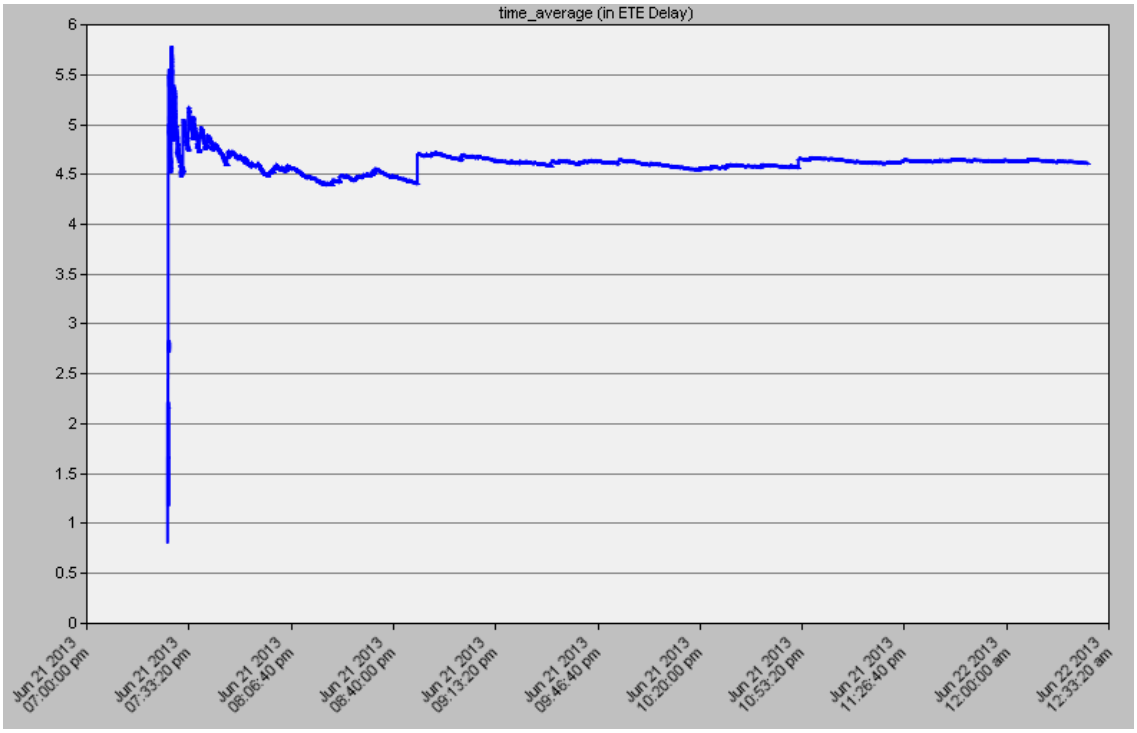
شکل ۴۹- poll exp(0.2) و داده exp(1)



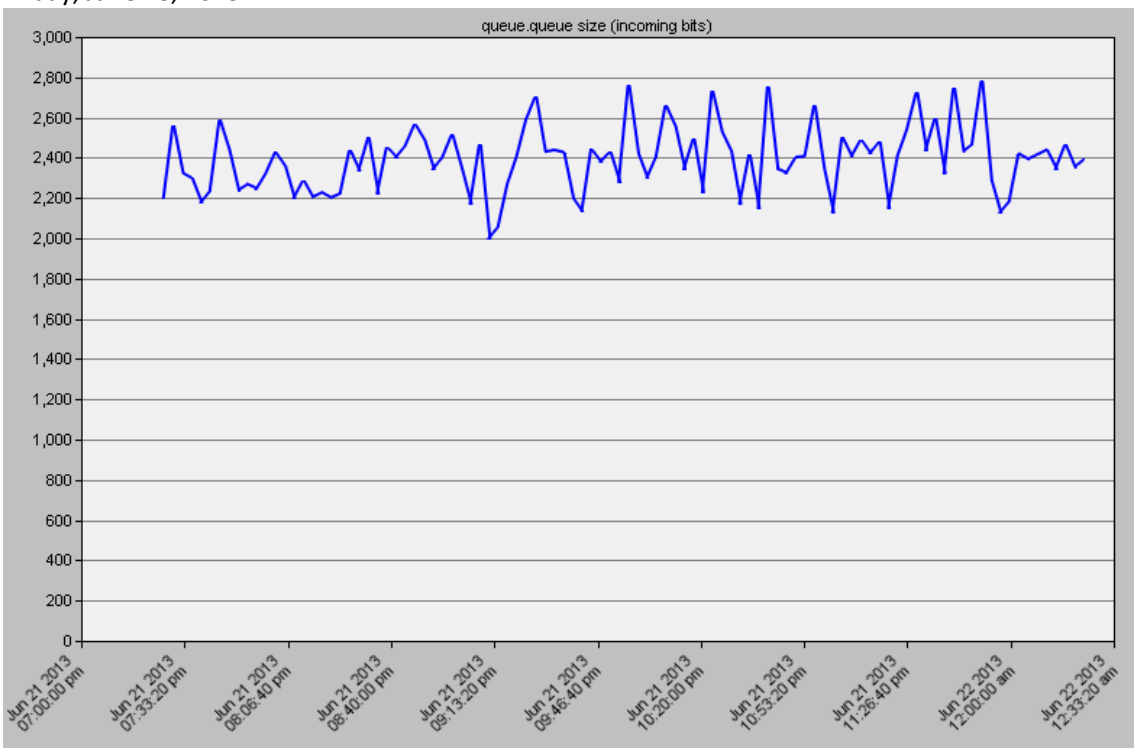
شکل ۵۰ - poll exp(0.2) و داده exp(1)



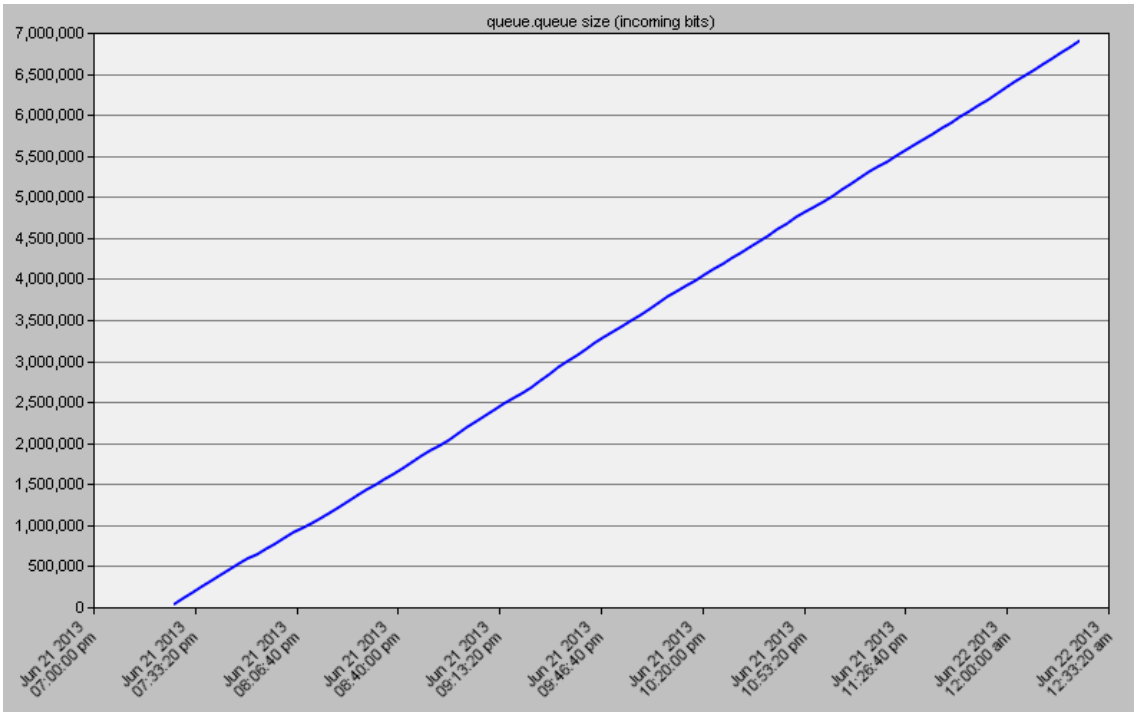
شکل ۵۱ - exp(0.2) و داده exp(1)



شکل ۵۲ - exp(0.2) و داده exp(1)



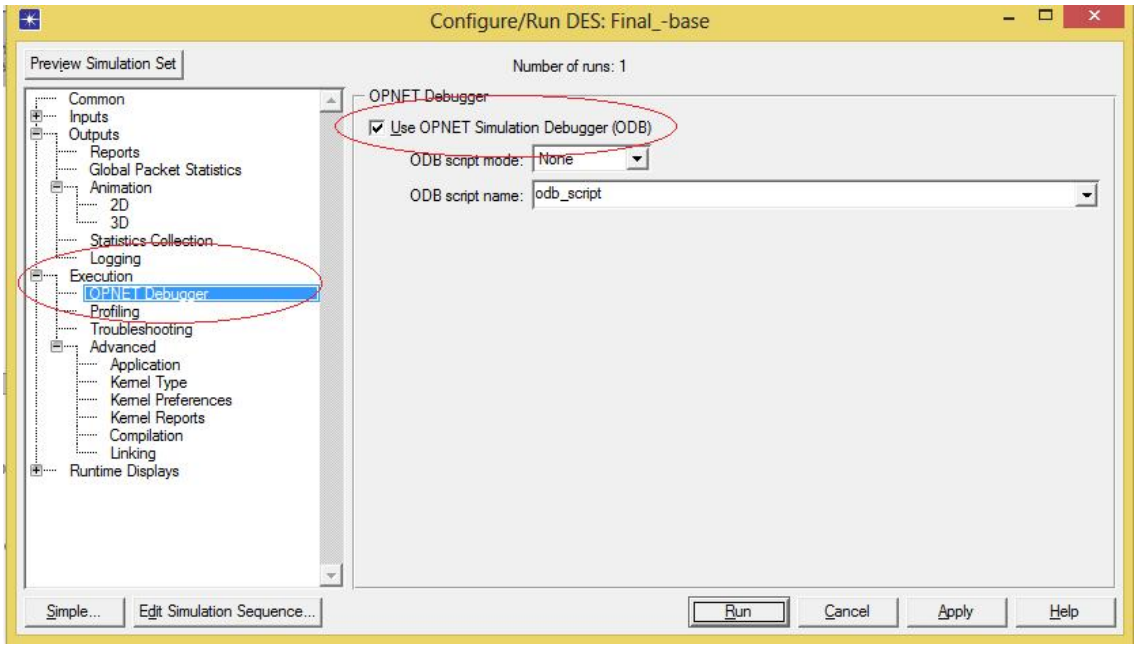
شکل ۵۳ – queue size نود دوم



شکل ۵۴ – queue size نود دوم در simulation اول

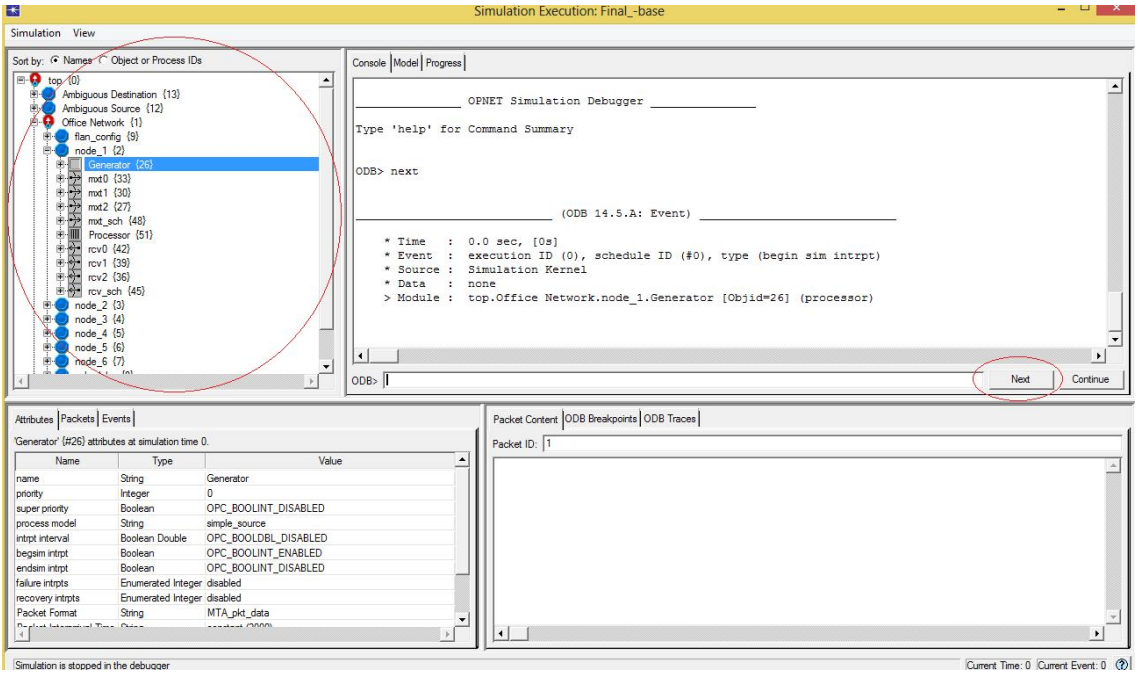
۶. ضعف های الگوریتم MTA

ابتدا برای فعال سازی ODB باید مراحل زیر را طی کنیم. پس از رفتن به منوی DES و زدن configure/run discrete event simulation.... در شکل ۵۵ گزینه ی ODB را فعال کنیم.



شکل ۵۵

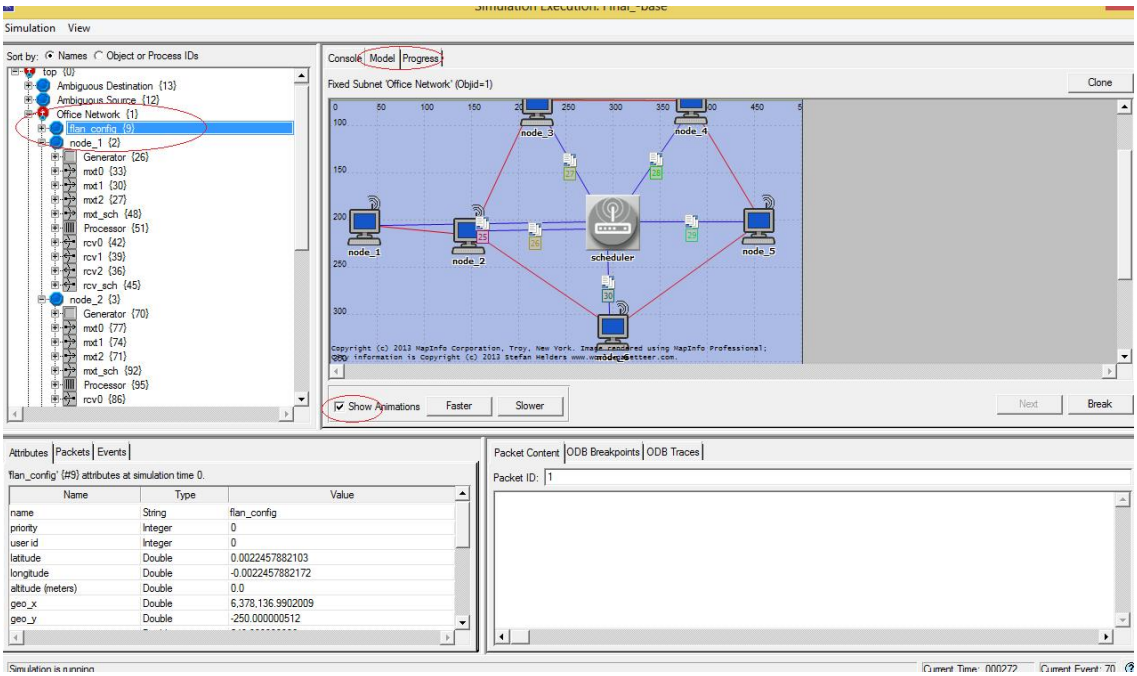
حال با run کردن project وارد debugger می شویم.



شکل ۵۶

حال با زدن یک بار next مانند شکل ۵۶، منوی سمت راست update می شود.

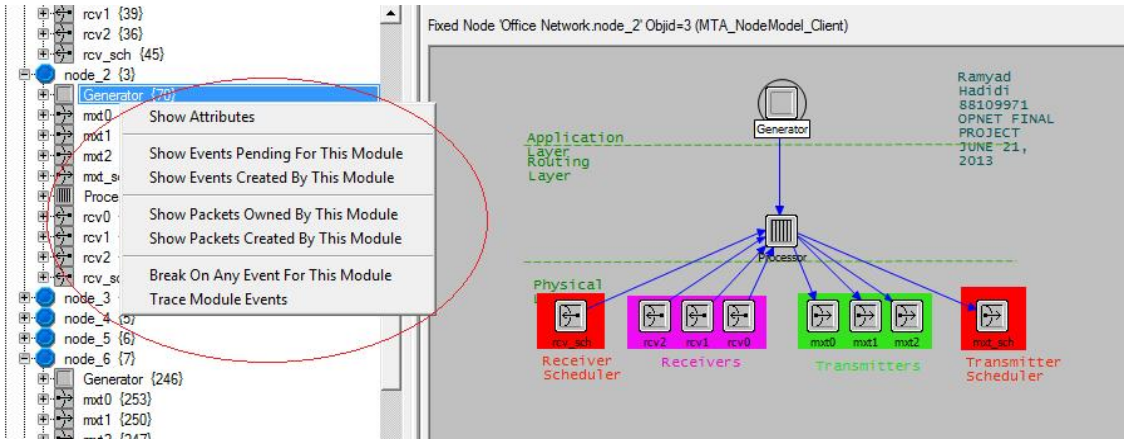
یکی از ویژگی های قابل تحسین این debugger، نشان دادن انتقال پکت ها به صورت شهودی است.



شکل ۵۷

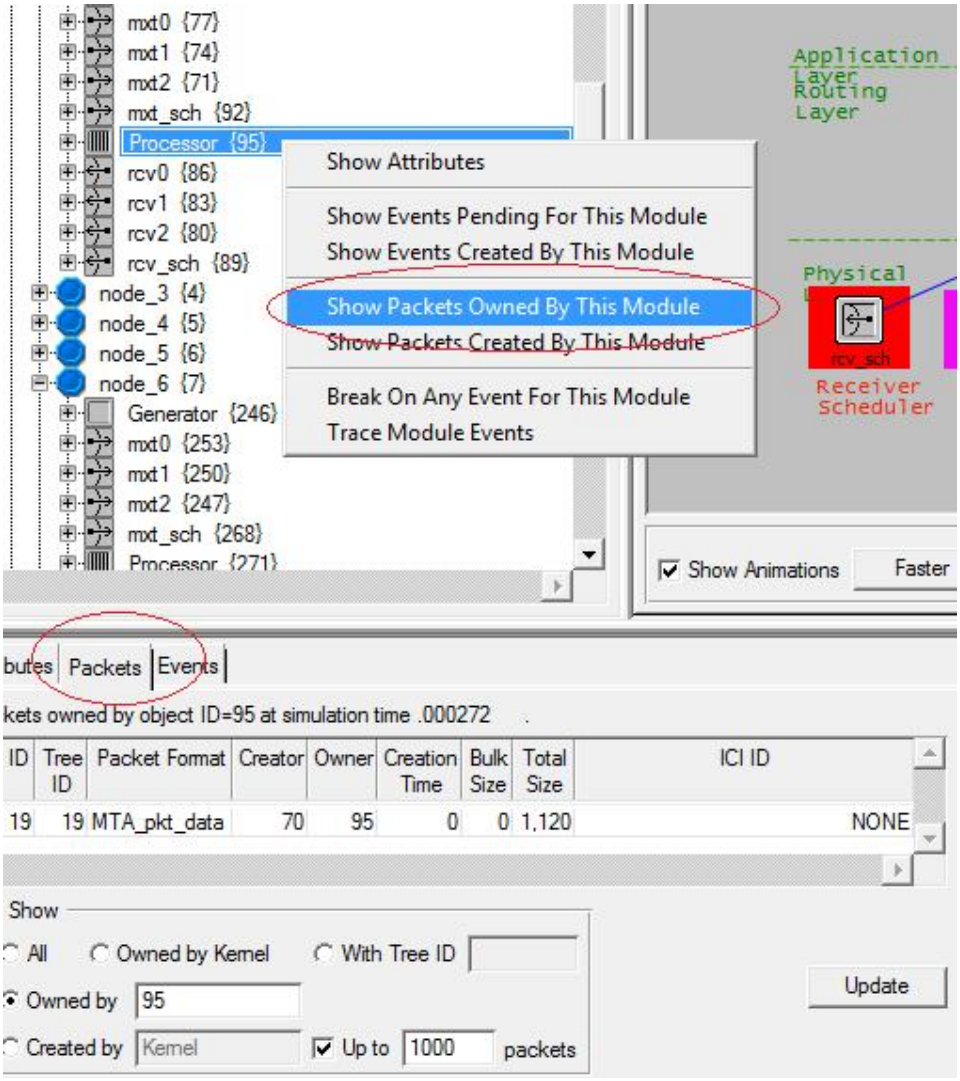
مانند شکل ۵۷ به tab model بروید و show animation را tick کنید. حال با انتخاب شبکه از سمت راست و زدن دگمه ی continue می توان حرکت پکت ها را مشاهده کرد.

ولی با این کار نمی توان پکت ها را دنبال کرد و یا منتظر یک اتفاق خاص بود. به همین علت این debugger قابلیت trace و افزودن break point دارد. با right-click کردن روی هر ماژول می توان این موارد را به debugger اضافه کرد. (شکل ۵۸)



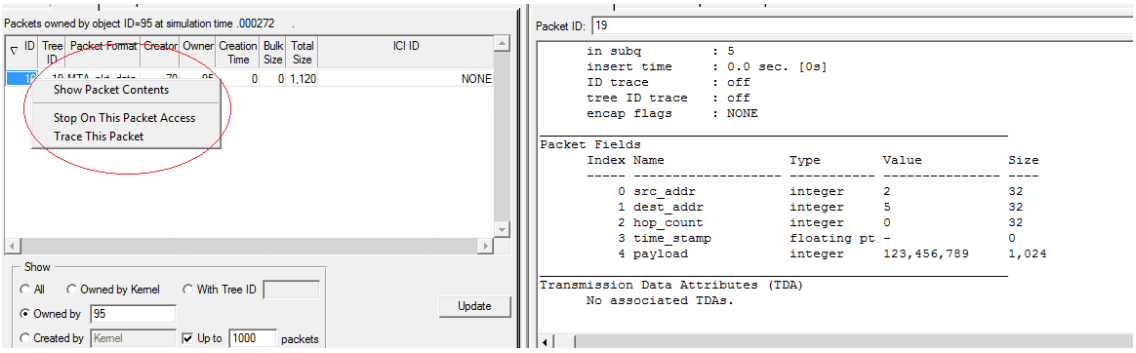
شکل ۵۸

حال صرف نظر از سایر موارد به trace کردن پکت ها می پردازیم. ابتدا همانند شکل ۵۹ روی گزینه ی نشان داده شده کلیک کنید و tab packets را بزنید. حال می توانید کلیه ی packetهای داده را ببینند.



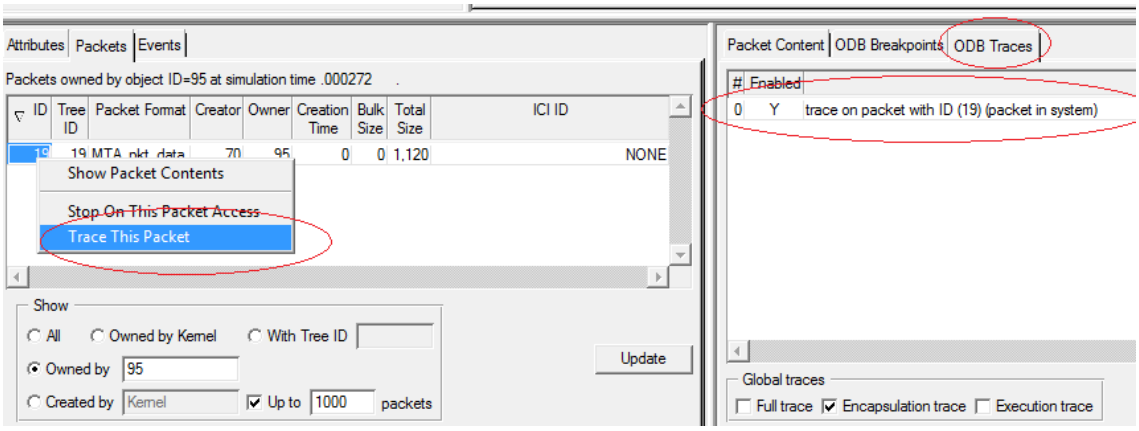
شکل ۵۹

حال با right-click کردن روی packet ID می توان همانند شکل ۶۰ به مقادیر جالبی دست پیدا کرد. مثلا در اینجا می توانیم محتوی پکت را ببینیم.



شکل ۶۰

برای trace کردن روی trace this packet کلیک کنید. حال در tab trace این پکت به لیست اضافه شده. (شکل ۶۱)



شکل ۶۱

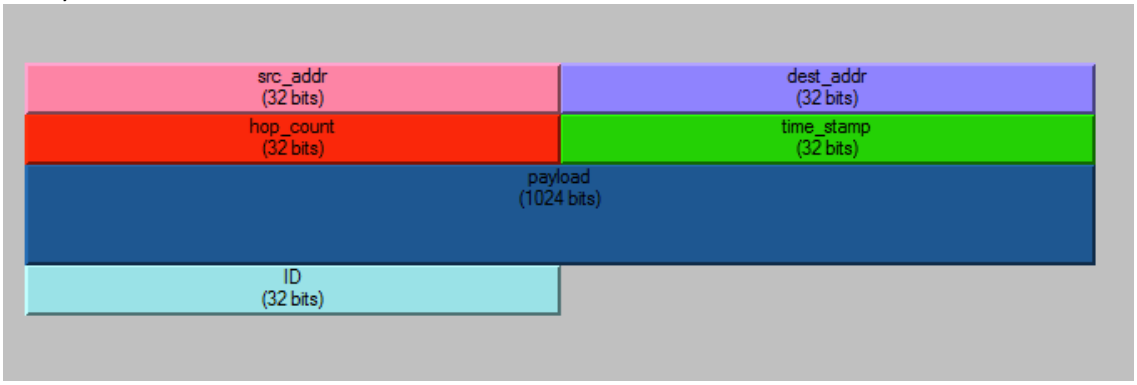
حال opnet در console کلیه ی اتفاقات افتاده روی این packet را همانند شکل ۶۲ نشان می دهد.

```
_____________________________ (ODB 14.5.A: Event) _____________________________

* Time   :  3.291162910892 sec, [3s . 291ms 162us 910ns 892ps]
* Event  :  execution ID (1838), schedule ID (#1845), type (stream intrpt)
* Source :  execution ID (1836), top.Office Network.node_5.rcv1 [Objid=215] (pt-pt receiver)
* Data   :  instrm (2), packet ID (19)
> Module :  top.Office Network.node_5.Processor [Objid=227] (queue)

    +- op_pk_get (instrm_index)
    |       strm. index      (2)
    |       packet ID        (19)
    +-------

    +- op_pk_nfd_get_int32 (pkptr, fd_name, value_ptr)
    |       packet ID        (19)
    |       field name       (dest_addr)
    |       completion code  (success)
    |       value            (5)
    +-------

    +- op_pk_nfd_get_int32 (pkptr, fd_name, value_ptr)
```

شکل ۶۲

با این روش می توان تمام اتفاقات افتاده روی packet را دید. ولی از آنجا که مسیر کلی هر packet برای ما اهمیت دارد من از روش دیگری استفاده می کنم. در هر نود وقتی که scheduler به آن دستور می دهد که داده ای را بفرستد این نود اطلاعات مفید را برای ما printf می کند. برای این کار نیاز به یک packet ID در پکت خود داریم که آن را مانند شکل ۶۳ اضافه می کنیم. حال کد های مربوطه را با اضافه کردن یک ID که به مرور زیاد می شود در SV کامل می کنیم. (شکل ۶۴). حال طبق شکل ۶۵ تمام اطلاعات مربوطه به راحتی چاپ می شود.

خروجی نمونه هم در شکل ۶۶ آمده است.



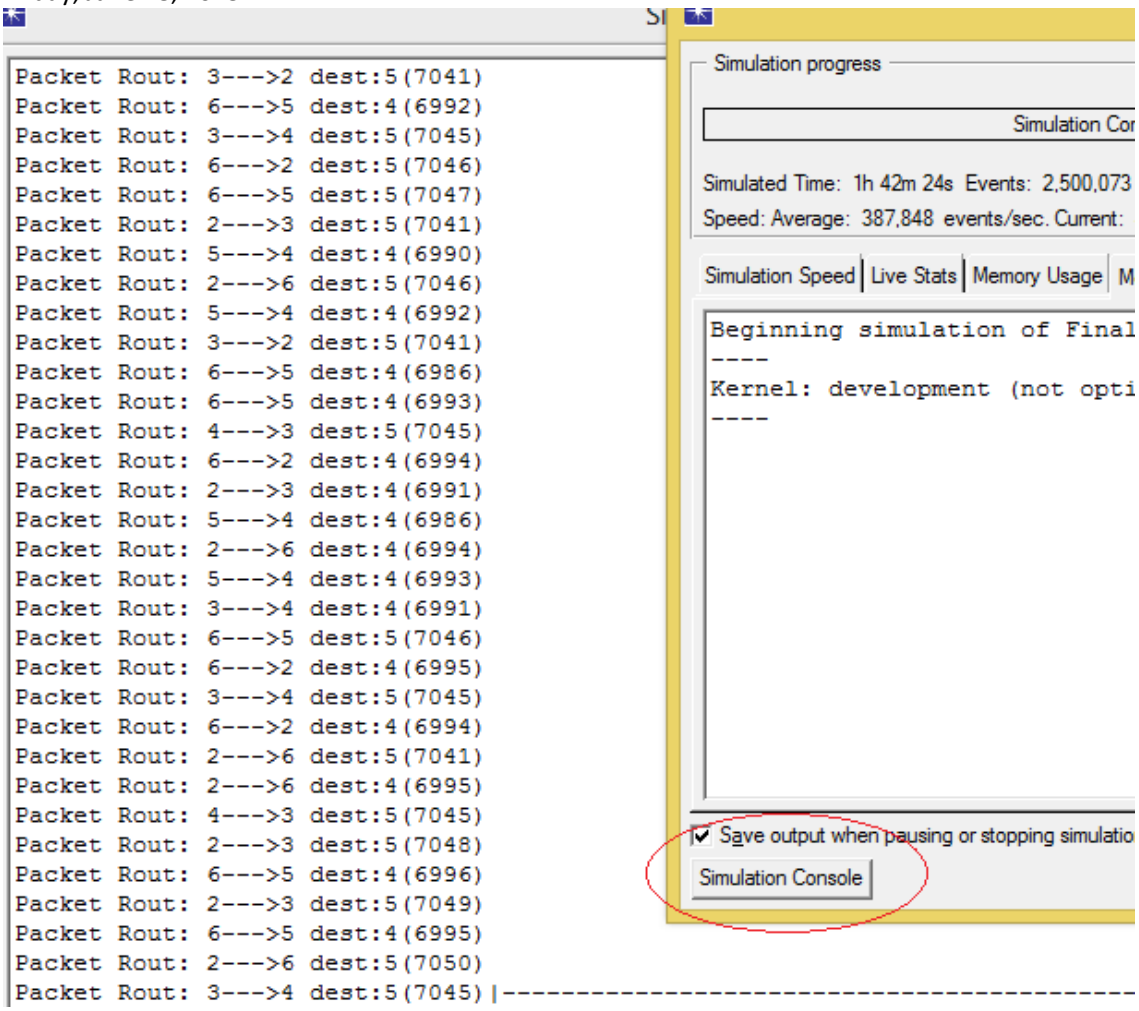
شکل ٦٣

Type	Name	Comments
int	src_addr	/* node self address */
int	queue_size[7]	/* queue size */
int	mxt_address[3]	/* save wich address maps to which mxt */
int	flow_dest[2]	/* flow is directed to which address */
int	flow_sender[2]	/* flow senders are who */
int	flow_dest_size	/* size of above array */
int	flow_sender_size	/* size of above array */
int	hello_addr	/* an address for hello packets */
Stathandle	ete_gsh	/* for delay calculation */
Stathandle	ete_hop_count	/* for hop count statistics */
int	ID	/*a static varible to distinguish between pkt*/

شکل ٦٤

```
//a command, send a packet, update queue sizes
if (command_type == pkt_poll_command)
    {

    op_pk_nfd_get_int32 (pkptr, "queue_dest_addr", &pkt_queue_dest_addr);
    op_pk_nfd_get_int32 (pkptr, "queue_sending_addr", &pkt_queue_sending_addr);
    op_pk_destroy(pkptr);

    if (queue_size[pkt_queue_dest_addr] == 0) { FOUT; } //I have no data for it to send!
    flag=-1;
    for (i=0; i<3; i++) {
        if(mxt_address[i] == pkt_queue_sending_addr) {
        flag = i;
        }
    }
    if(flag==-1) { FOUT; } //I have no connection to it!
    //send data from queue
    //take out from queue
    pkptr = op_subq_pk_remove(pkt_queue_dest_addr, OPC_QPOS_TAIL);
    queue_size[pkt_queue_dest_addr]--;
    op_pk_nfd_get_int32 (pkptr, "pkt_ID", &pkt_ID);

    switch (flag)
        {
        case 0: op_pk_send (pkptr,MXT0_OUT_STRM);
                break;
        case 1: op_pk_send (pkptr,MXT1_OUT_STRM);
                break;
        case 2: op_pk_send (pkptr,MXT2_OUT_STRM);
                break;
        }

    printf("\nPacket Rout: %d--->%d dest:%d(%d)",src_addr,pkt_queue_sending_addr,pkt_queue_dest_addr,pkt_ID);
    }

FOUT;
}
```

شکل ٦٥



شکل ۲۲

حال با ذخیره ی این فایل در یک فایل txt. با find می توانیم با ID و dest یک پکت به راحتی آن را دنبال کنیم. به عنوان مثال:

2->3->4->3->2->6->2->6->5

2->3->4->5

2->3->2->6->5

6->5->4

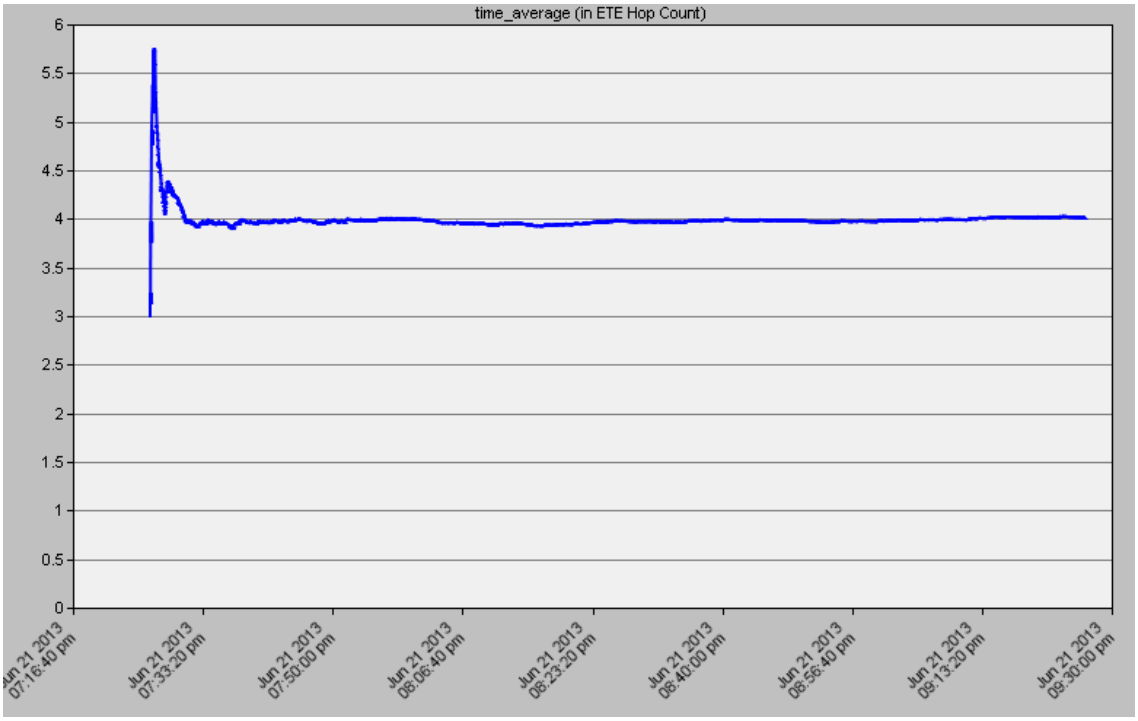
6->2->6->2->3->4

همانطور که می بینیم، در MTA مسیر دقیق معلوم نیست و یک پکت می تواند با توجه به شرایط شبکه از مسیر های متفاوتی عبور کند و یا حتی بار ها بین نود ها دست به دست شود. پس در نتیجه نمی توان یک delay ماکزیمم برای کل شبکه تعریف کرد. از طرفی دیگر این رفتار موجب می شود که سایز queue ها بسیار وابسته به نوع اتصالات و شلوغی شبکه باشد. (با این حال به نظر من با گرفتن یک سایز queue ماکزیمم و استفاده از روش های جلوگیری از سرریز queue|| می توان این مشکل را حل کرد)

۷. از بین رفتن ارتباط بین نود های ۵ و ۶

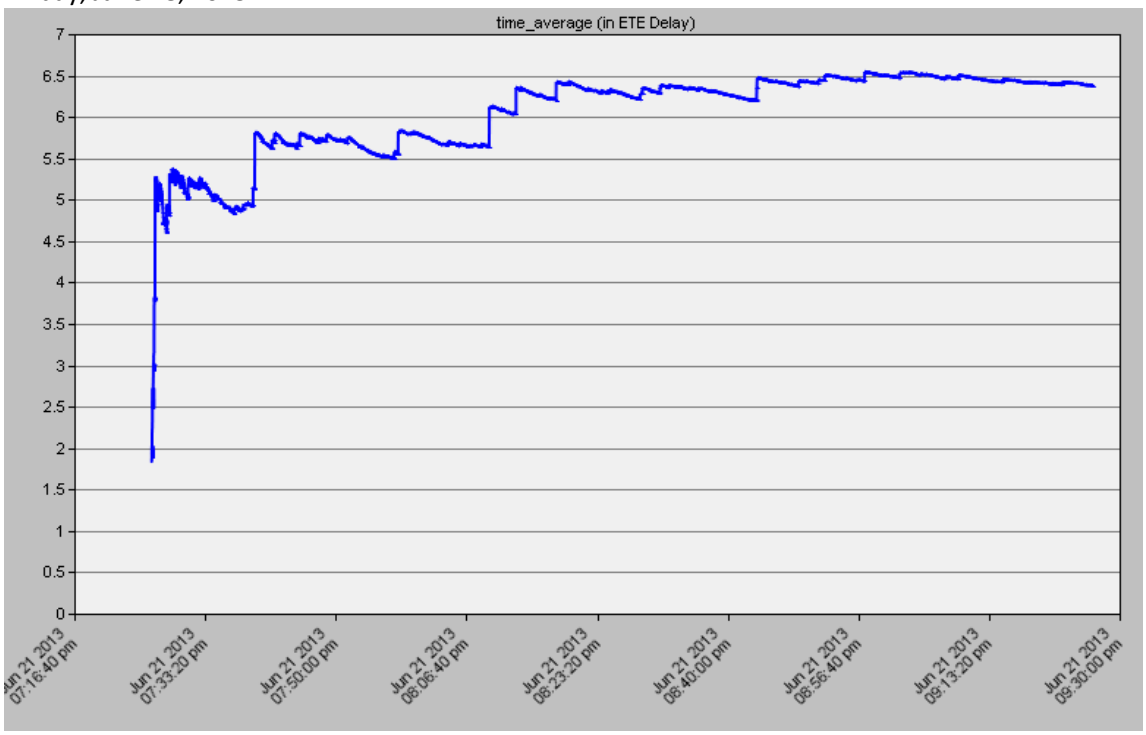
از آنجا که این در این پروژه ما نقشه ی شبکه را به صورت static به scheduler دادیم، اضافه کردن این قابلیت که بتوان برخی از Link ها را از لیست در هنگام simulation حذف کرد مستلزم تغییرات زیادی است. پس بنابرین ما در ابتدا این دو لینک را از شبکه خارج می کنیم و تحلیل را انجام می دهیم. مطمئنا نتایج یکی خواهند بود.

طبق شکل ۶۷ می بینیم که Hop count کم شده. و به این دلیل است که دیگر دو مسیر برای رسیدن به داده وجود ندارد، و احتمال دست به دست شدن داده کمتر می شود.



شکل ۶۷

ولی همانطور که در شکل ۶۸ می بینیم، delay بیشتر شده. و این به دلیل افزایش زمان انتظار در queue ها است. چون یکی از مسیر ها فعال است و برای دوری از تداخل scheduler مجبور است فقط در هر لحظه یکی از داده ها را عبور دهد. و همانطور که می بینیم delay روند افزایش دارد و ممکن از با افزایش زمان شبیه سازی بیشتر هم بشود که نشان دهنده ی درستی صحبت قبلی است.



شکل ۶۸

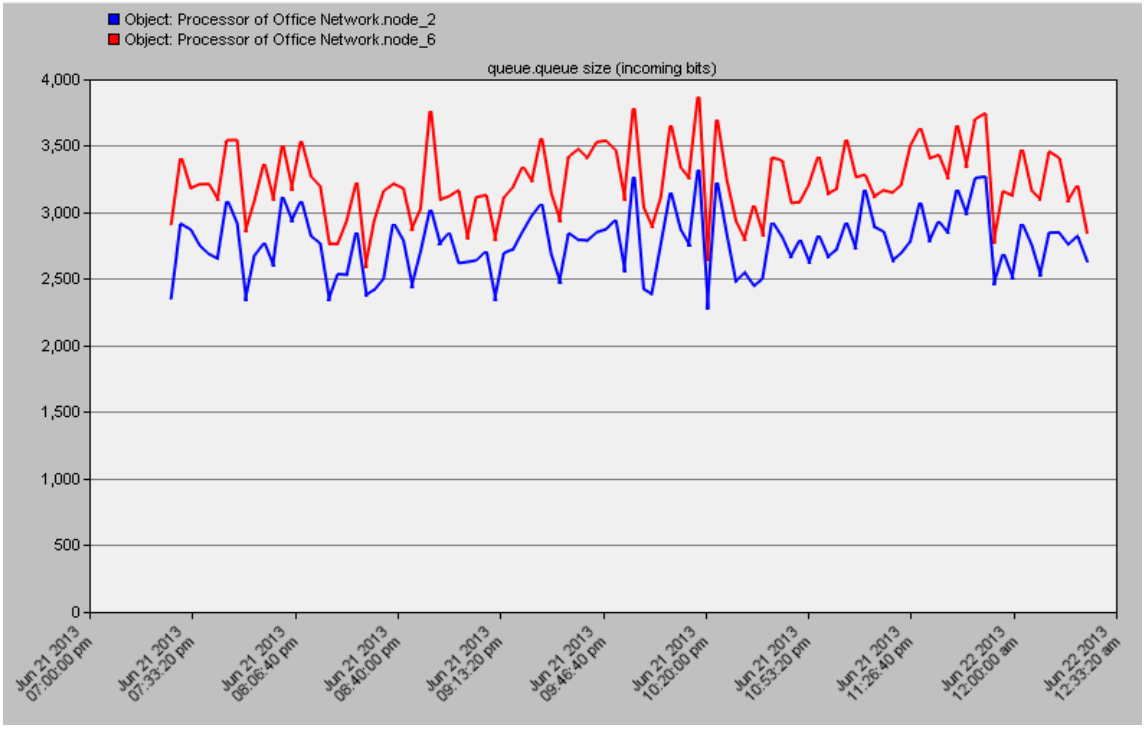
۸. دخالت دادن QoS برای Flow1 و بخش امتیازی آن

برای این کار فقط کافی است تا queueهایی که به مقصد ۵ هستند با ضریب بیشتری محاسبه کنیم. این کار را باید در scheduler انجام دهیم. شکل ۶۹ کد تغییر داده شده را نشان می دهد.

```
//we need to command clinets to send datas
if (command_needed == 1)
    {
    //calculates weigths
    for (i=0;i<36;i++)
        {
        a=LINK[LINK_PAIRS[i][0]][0];     //i pairs: first link start node
        b=LINK[LINK_PAIRS[i][0]][1];     //end node

        c=LINK[LINK_PAIRS[i][1]][0];     //i pairs: second link start node
        d=LINK[LINK_PAIRS[i][1]][1];     //end node

        max1=0;
        max2=0;
        for (j=1;j<7;j++)
            {
                Qos_w=1;
                if (j==5) {Qos_w=2;}     //uncomment this part for QoS
                                         //you can add more for more Qos s

                w1=queue_size[a][j];
                w2=queue_size[b][j];
                if ((w1-w2)>max1) { max1 = (w1-w2) * Qos_w; }

                w1=queue_size[c][j];
                w2=queue_size[d][j];
                if ((w1-w2)>max2) { max2 = (w1-w2) * Qos_w; }
            }
        LINK_PAIR_W[i] = max1+max2;
        }
```

شکل ۶۹

برای اطمینان می توانیم queue دو نود ۶ و ۲ را چک کنیم و همانطور که می بینیم این مقدار برای ۶ بیشتر است. (شکل ۷۰)



شکل ۷۰

۹. دلیل نامگذاری Back Pressure Algorithm

Back pressure به معنای فشار در مایعات است؛ این الگوریتم هم پکت ها را مانند جریان آب انتقال می دهد. یعنی در پکت ها به جایی که اختلاف فشار آنها بیشتر است جاری می شوند. از مزیت های این روش می توان به شکل ۷۱ مراجعه کرد.

The backpressure algorithm operates in slotted time. Every time slot it seeks to route data in directions that maximize the differential backlog between neighbouring nodes. This is similar to how water flows through a network of pipes via pressure gradients. However, the backpressure algorithm can be applied to multi-commodity networks (where different packets may have different destinations), and to networks where transmission rates can be selected from a set of (possibly time-varying) options. Attractive features of the backpressure algorithm are: (i) it leads to maximum network throughput, (ii) it is provably robust to time-varying network conditions, (iii) it can be implemented without knowing traffic arrival rates or channel state probabilities. However, the algorithm may introduce large delays, and may be difficult to implement exactly in networks with interference. Modifications of backpressure that reduce delay and simplify implementation are described below under Improving Delay and Distributed Backpressure.

شکل ۷۱

۱۰. (بخش امتیازی) Dynamic Topology

همانطور که در بخش های قبلی گفتم، من در این پروژه برای شناخت اولیه ی همسایه های هر نود از پکت های hello استفاده کردم و این اطلاعات را هم به scheduler فرستادم. یعنی scheduler می داند که تمام نود ها چه همسایه هایی

دارند. حال تغییرات جدیدی که باید بدهیم این است که پکت های hello را به صورت پریودیک ارسال کنیم، پس در نتیجه اگر تغییری در شبکه رخ دهد، خواهیم فهمید.

در Scheduler نیز باید این قابلیت را اضافه کنیم که بتواند Linkها و جفت linkهای قابل فعال سازی با هم را شناسایی کند و آنها را هر بار با تغییر شبکه update کند.

*کد زنی این بخش پیچیدگی زیاد داشت، و من بعد از چند بار سعی منصرف شدم! برای بخش بعدی هم به همین گونه!