**CHƯƠNG 2. GIỚI THIỆU PHẦN MỀM MÔ PHỎNG OBS-NS**

## 2.1. Giới thiệu về phần mềm mô phỏng mạng NS2

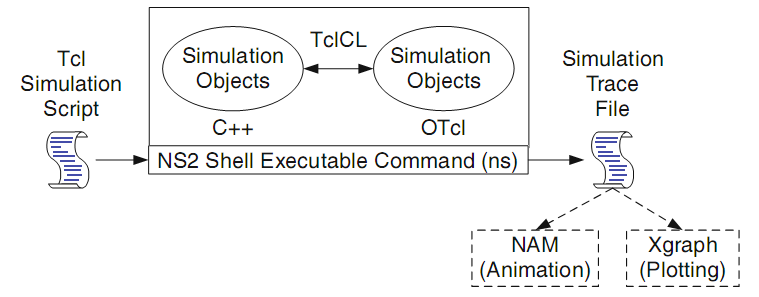
Phần mềm mô phỏng mạng NS2 là một công cụ mô phỏng hướng sự kiện đã được cải tiến để hữu ích trong việc học tập, nghiên cứu hoạt động của những hệ thống mạng. Bốn lợi ích lớn nhất của NS-2 phải kể đến đầu tiên là:

* Khả năng kiểm tra tính ổn định của các giao thức mạng đang tồn tại.
* Khả năng đánh giá các giao thức mạng mới trước khi đưa vào sử dụng.
* Khả năng thực thi những mô hình mạng lớn mà gần như ta không thể thực thi được trong thực tế.
* Khả năng mô phỏng nhiều loại mạng khác nhau.

NS2 có thể mô phỏng các thuật toán và giao thức của mạng hữu tuyến cũng như mạng vô tuyến (như là những thuật toán định tuyến, các giao thức TCP, UDP, …). Do đặc tính linh động và đặc tính mô đun hóa, NS2 được sử dụng phổ biến trong giới nghiên cứu từ năm 1989. Hệ mô phỏng NS được phát triển ở trường đại học Berkeylay cộng tác với một số cơ quan khác, bây giờ là một phần trong dự án VINT (Virtual Internet Testbed) của phòng thí nghiệm quốc gia Lawrence Berkeley. NS2 là phần mềm mã nguồn mở có thể cài được cả nền Windows 32 và Linux.

### 2.1.1 Kiến trúc cơ bản

NS2 bao gồm 2 ngôn ngữ chính: C++ và OTcl. C++ được dùng để định nghĩa những thành phần bên trong của những đối tượng mô phỏng, OTcl dùng để thiết lập mô phỏng bằng cách kết hợp và cấu hình những đối tượng. C++ và OTcl liên kết với nhau nhờ ngôn ngữ Tcl. NS2 là bộ biên dịch Tcl mở rộng hướng đối tượng; bao gồm các đối tượng bộ lập lịch sự kiện, các đối tượng thành phần mạng và các mô đun trợ giúp thiết lập mạng (hay các mô đun Plumbing).



*Hình 2.1 - Kiến trúc NS2*

Để sử dụng NS2, người dùng lập trình bằng ngôn ngữ kịch bản OTcl. Người dùng có thể thêm các mã nguồn OTcl vào NS2 bằng cách viết các lớp đối tượng mới trong OTcl. Những lớp này khi đó sẽ được biên dịch cùng với mã nguồn gốc. Kịch bản OTcl có thể thực hiện những việc sau:

* Khởi tạo Bộ lập lịch Sự kiện
* Thiết lập mô hình mạng dùng các đối tượng thành phần mạng
* Báo cho nguồn traffic khi nào bắt đầu truyền và ngưng truyền gói tin trong bộ lập lịch sự kiện

Sau khi mô phỏng, phụ thuộc vào mục đích của người dùng, đối với kịch bản mô phỏng OTcl mà kết quả mô phỏng có thể được lưu trữ như file trace. Định dạng file trace sẽ được tải vào trong các ứng dụng khác để thực hiện phân tích:

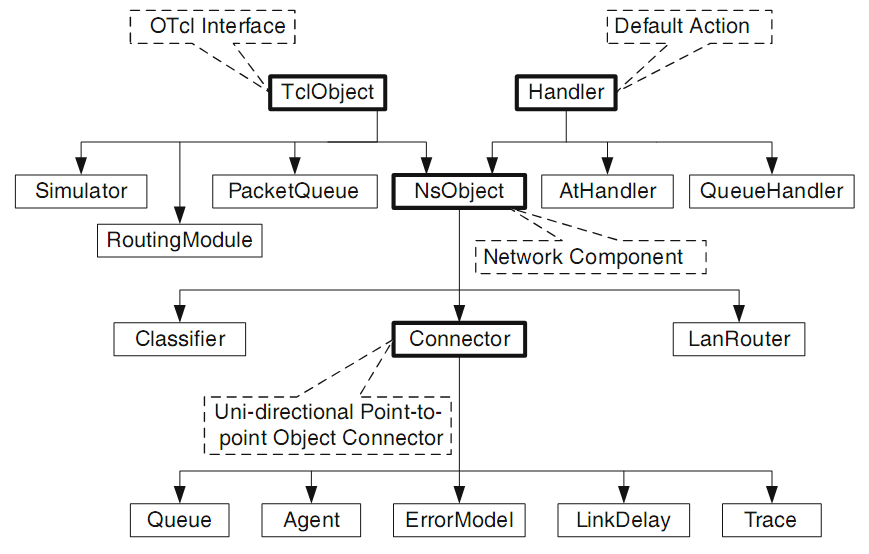
* File nam trace (file .nam) được dùng cho công cụ minh họa mạng NAM
* File Trace (file .tr) được dùng cho công cụ lần vết và giám sát mô phỏng XGRAPH hay TRACEGRAPH



*Hình* *2.2 - Luồng các sự kiện cho file Tcl chạy trong NS*

### 2.1.2 Các thành phần chính của NS2

Hình 2.3 thể hiện một bức tranh tổng quát về kiến trúc phân cấp của các lớp C++ trong NS2. Toàn bộ cấu trúc có trên 100 lớp C++ và các loại dữ liệu cấu trúc.

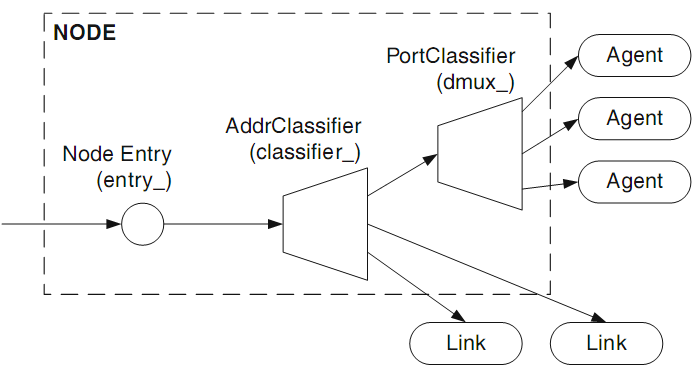


*Hình 2.3 - Một phần kiến trúc phân lớp C++ trong NS*

Tất cả các lớp đều bắt nguồn từ lớp TclObject, đây là gốc của tất cả các đối tượng trong thư viện Otcl (như lịch trình của sự kiện, thành phần mạng, thời gian, xử lý gói tin, liên quan đến mô phỏng, định tuyến và những đối tượng khác). NSObject là lớp cơ sở của tất cả các thành phần trong NS2, bao gồm nhiều thành phần khác nhau nhằm để xử lý gói tin, điều khiển mô phỏng. Lớp này được chia thành hai lớp con là Connector và Classifier. Trong lớp Connector bao gồm các đối tượng liên quan đến độ trễ (delay), hàng đợi (queue), lưu dấu (trace) …

#### 2.1.2.1 Node và định tuyến

Node là thành phần cơ bản của mạng, trong NS2 node có thể hành động như là một máy tính (ví dụ là nguồn hoặc đích), hoặc có thể là một bộ định tuyến (nút trung gian). Nó nhận những gói tin từ một ứng dụng được gắn vào hoặc từ một luồng nào đó và hướng nó tới những liên kết gắn với nó đã được chỉ định trong bảng định tuyến (chức năng như một bộ định tuyến) hoặc chuyển chúng tới những cổng đã được chỉ định trong header của gói tin (chức năng như một host).



*Hình 2.4 - Kiến trúc 1 node trong NS2*

Kiến trúc cơ bản của một nút được miêu tả trong hình 2.4, gói tin sẽ đi vào qua đối tượng Node entry (entry\_: là một đối tượng Connector), sau đó gói tin sẽ đi vào bộ phân loại đia chỉ (address classifier). Nếu node này không phải đích của gói tin thì nó sẽ được hướng tới liên kết thích hợp trong bảng định tuyến. Ngược lại gói tin sẽ được chuyển tới bộ tách kênh (demux\_) hoặc phân loại cổng (Port Classifier). Trong NS có hai loại node cơ bản đó là node Unicast và nút Multicast. Tại các node chúng ta có thể gắn thêm các agent (UDP agent, TCPagent) để truyền các luồng dữ liệu được tạo từ các ứng dụng, dịch vụ (FTP, CBR,…)

|  |  |
| --- | --- |
|  |  |

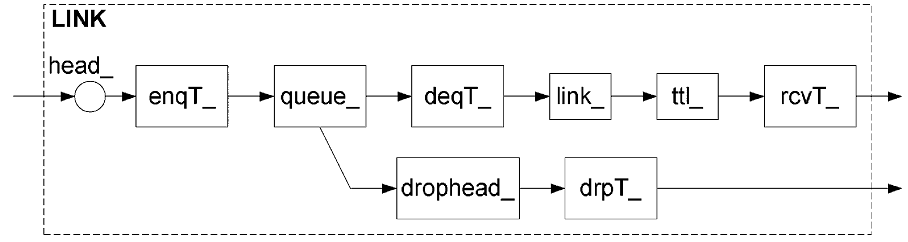
*Hình 2.5 - Hai loại node cơ bản trong NS2*

Bộ định tuyến trong NS2 bao gồm 4 thành phần chính:

* Routing agent: thu thập thông tin (ví dụ như cấu trúc mạng) cần thiết để tính toán bảng định tuyến.
* Route logic: sử dụng thông tin thu thập được từ routing agent để tính toán bảng định tuyến.
* Classiﬁer: sử dụng bảng định tuyến đã được tính toán cho việc chuyển gói tin.
* Routing module: làm nhiệm vụ như một điểm quản lý nhóm các classsifier trong một node. Nó sẽ lấy những câu lệnh cấu hình từ routing agent, route logic, và từ một node và truyền chúng tới những bộ classifier liên quan.

#### 2.1.2.2 Liên kết (Link)

Một liên kết là một đối tượng OTcl kết nối 2 node và mang những gói tin từ node đầu đến node cuối. Một đối tượng liên kết sử dụng rộng rãi nhất là **SimpleLink**, bao gồm những đối tượng cơ bản và những đối tượng lưu vết.



*Hình 2.6 Kiến trúc một đối tượng liên kết SimpleLink*

Trên mỗi liên kết giữa các node chúng ta có thể sử dụng các hàng đợi, nó được thi hành như một phần của liên kết. Khi trong hệ thống xảy ra sự cố như độ trì hoãn cao hay nghẽn mạch (congestion) thì hàng đợi sẽ lưu các gói tin này. Gói tin sẽ ra khỏi hàng đợi và truyền đi đến nơi nhận khi trên hệ thống không còn bị tắc nghẽn, trường TTL (Time-To-Live) sẽ tính toán và cập nhật lại thời gian cho mỗi gói tin truyền đi thành công.

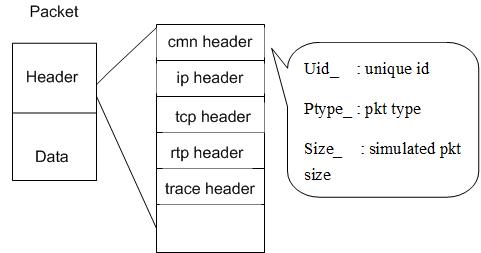
Lưu vết sẽ ghi lại chi tiết gói tin khi chúng chuyển qua mạng, nơi có đối tượng **Trace** được cài đặt. Thông tin ghi lại từ các đối tượng trace sẽ dùng để phân tích, thống kê, đánh giá, hoặc xem lại hoạt động qua các phần mềm hỗ trợ.

#### 2.1.2.3 Gói tin (Packet) và luồng dữ liệu

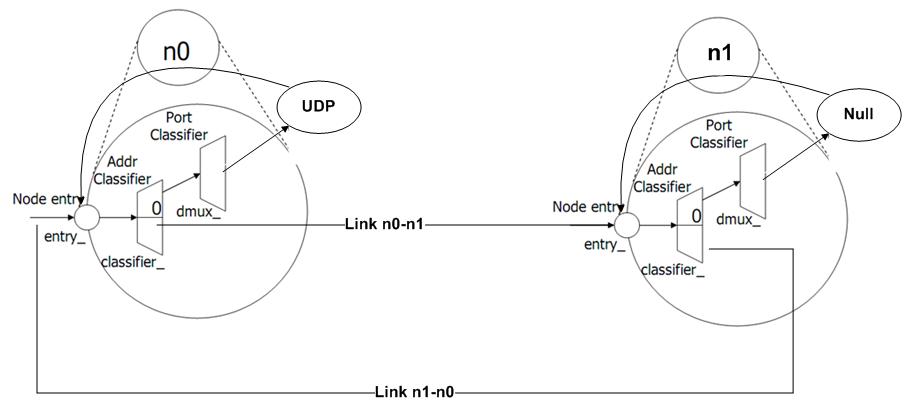
Một gói tin bao gồm phần đầu (header) và dữ liệu, phần header của gói tin lưu trữ các thuộc tính cần thiết của gói tin (như địa chỉ ip nguồn, địa chỉ ip đích, …), chi tiết được mô tả trong hình 2.7.

Để mô tả hoạt động của luồng trong NS2, ta xét ví dụ: để truyền một gói tin “\*p” từ agent UDP tới null ta sẽ qua các bước sau :

* Agent UDP sẽ gởi gói tin \*p tới đối tượng entry\_ của node 1.
* Gói tin sẽ được gởi tới đối tượng classifier\_ của nút 1.
* Đối tượng này sẽ kiểm tra header của gói tin \*p, trong trường hợp gói tin có đích là node 2, nó sẽ hướng gói tin tới liên kết với node 2.
* Gói tin sẽ được đưa vào hàng đợi, tại cuối liên kết gói tin sẽ được kiểm tra thời gian sống (TTL), nếu TTL < 0 sẽ đánh rớt gói tin. Ngược lại gói tin sẽ được gởi tới classifier\_ của node 2.
* Node 2 kiểm tra xem gói tin có đúng gởi cho mình không, sau đó gói tin sẽ được gởi tới đối tượng tách kênh (dmux\_), đối tượng này sẽ hướng gói tin tới agent null.



*Hình 2.7 - Cấu trúc gói tin trong NS2*



*Hình 2.8 Mô tả liên kết và gửi gói tin trên hai nút*

## 2.2. Giới thiệu gói mô phỏng mạng OBS-ns

OBS-ns được phát triển bởi phòng nghiên cứu mạng DAWN thuộc trường đại học Maryland, nó là một phần mở rộng của phần mềm mô phỏng NS2 dùng để mô phỏng cho mạng OBS. Phiên bản mới nhất là 0.9a, nhưng chưa hoàn chỉnh và có nhiều hạn chế. Trung tâm nghiên cứu Internet quang và viện công nghệ cao SamSung (SAIT[[1]](#footnote-1)), Hàn Quốc đã có một số cải tiến để đưa ra gói OBS-ns với những tham số mới, mục đích chính là giải quyết những vấn đề của phiên bản 0.9a .

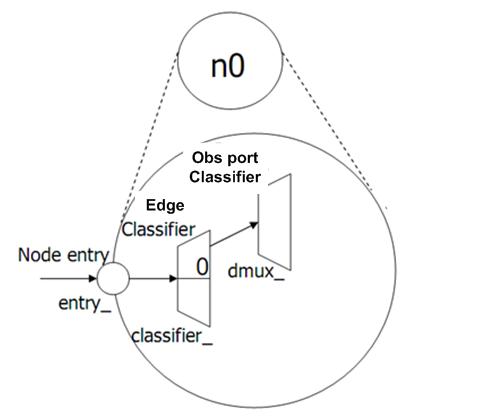
Cũng như NS2, OBS-ns là phần mềm mô phỏng hướng sự kiện. Nó kế thừa những mô đun, thư viện, đặc tính của NS2. Mô hình đối tượng cũng được phát triển dựa trên C++ và OTcl. Trong gói OBS-ns có định nghĩa thêm những đối tượng mới giúp cho việc mô phỏng mạng OBS. Một loại gói tin mới (IPKT) được định nghĩa riêng cho mạng OBS, trong đó có thông số để chỉ rõ khi nào gói tin ipkt là gói tin điều khiển, khi nào là burst dữ liệu. Hầu hết các đối tượng dùng trong OBS-ns đều dựa trên các đối tượng đã được định nghĩa trong NS2. Ngoài ra có một số đối tượng mới được định nghĩa để phù hợp với mạng OBS.

### 2.2.1 Node

Mạng OBS được kết nối với nhau bằng 2 loại nút cơ bản: node biên và node lõi, trong đó node biên có 2 loại là node biên vào và node biên ra có cấu trúc tương tự nhau, chỉ khác nhau đối tượng tập hợp burst ở node đầu vào và bộ phân tách burst ở node đầu ra. Tại các node có thể được thiết lập các bộ làm trễ quang FDL.

#### 2.2.1.1 Node biên

Chức năng chính của node biên là tập hợp các gói tin thành burst (node biên vào) và phân tách burst thành các gói tin riêng lẻ (node biên ra). Tại node biên được xây dựng một bộ quản lý burst (Burst manager) với một số chức năng chính như: phân chia các hàng đợi ưu tiên cho dịch vụ QoS, cài đặt các thuật toán tập hợp burst phổ biến (dựa trên ngưỡng thời gian, kích thước và lai). Ngoài ra còn có bộ lập lịch với thuật toán lập lịch LAUC. Node biên có các đối tượng classifier là Edge Classifier làm nhiệm vụ phân loại và định tuyến gói tin. Các loại gói tin sẽ được đưa tới cổng thích hợp nhờ đối tượng OBS Port Classifier.

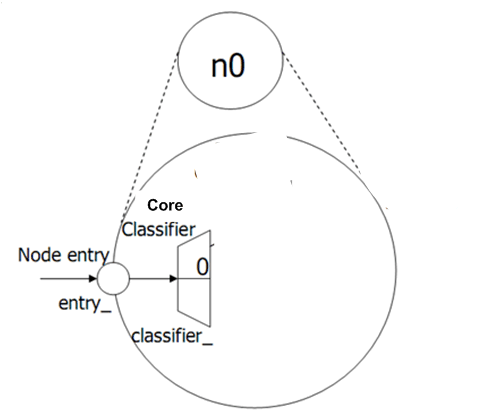


*Hình 2.9 - Cấu trúc node biên*

Tại node biên vào các gói tin được đưa tới những hàng đợi thích hợp (dựa trên đích đến hoặc dựa trên sự ưu tiên), tập tập hợp thành các burst dữ liệu dựa trên các thuật toán đã được cài đặt. Gói tin điều khiển được tạo và gởi trên những kênh riêng, burst dữ liệu cũng sẽ được gởi trên những kênh tách biệt. Khi tới đích, node biên đầu ra làm nhiều vụ phân tách burst thành những gói tin riêng lẻ.

#### 2.2.1.2 Nút lõi

Chức năng chính của node lõi là định tuyến và giải quyết tắc nghẽn. Node lõi được thiết kết khá đơn giản, chỉ có bộ phận phân loại địa chỉ để phân loại gói tin và định tuyến cho gói tin đi.



*Hình 2.10 - Cấu trúc node lõi*

#### 2.2.2 Liên kết

Các liên kết quang được thiết lập ở 2 chế độ đơn công và song công. Liên kết có thêm các tham số về độ trễ, tổng số bước sóng, số bước sóng cho dữ liệu, số bước sóng cho điều khiển.

1. *Lưu lượng (traffic)*

Ngoài các traffic (TCP, FTP, …) truyền thống trong ns2, OBS-ns định nghĩa 2 loại traffic mới :

* Traffic mang đặc tính Self-similar, đây là loại traffic phổ biến của internet hiện nay.  
  Application/Traffic/SelfSimilar
* Theo phân phối gamma   
  RandomVariable/Gamma

1. *Luồng gói tin*

Các gói tin từ các nguồn khác nhau tới nút biên sẽ đi vào cổng entry\_, tại đây các gói tin sẽ chuyển tới đối tượng phân giải địa chỉ, gói tin sẽ được so sánh xem có phải gói tin của mạng OBS (ipkt) hay không, gói tin ipkt tiếp theo sẽ được phân loại gói tin là điều khiển hay dữ liệu để có hành động thích hợp. Gói tin điều khiển sẽ được xử lý và chuyển đi ngay, trong khi đó gói tin dữ liệu sẽ được thêm độ trễ để có thể đi sau gói tin điều khiển khoảng thời gian offset và có đủ thời gian để lập lịch gói tin đi. Gói tin tới nút lõi sẽ được định tuyến tới node thích hợp.

**2.3 Các thành phần của OBS-NS**

**2.3.1 Thành phần CPP**

*a. Classifier*

Base Classifier cung cấp cho ta nhiều hàm chức năng cơ bản trong suốt quá trình truyền và nhận burst dữ liệu. Trong Base Classifier có các tham số sau:

bind("address\_", (int\*)(&address\_) ); //địa chỉ của node

bind("type\_", &type\_);//Kiểu của node(0 là node biên, 1 là node //lõi)

bind("proc\_time", &proc\_time\_); //thời gian xử lí gói điều khiển //trên mỗi node

bind( "nfdl", &FS\_.nfdl\_); //số kênh fdl trên mỗi node

bind( "fdldelay", &FS\_.fdl\_delay\_); //thời gian trễ tại mỗi node

bind ("maxfdls", &FS\_.max\_fdls\_); //số kênh FDL sử dụng tối đa tại mỗi node

Trong một mạng chuyển mạch chùm quang, một burst dữ liệu bao gồm nhiều gói tin IP được truyền qua mạng toàn quang. Một gói tin điều khiển được truyền đi trước một khoảng thời gian offset để cấu hình các chuyển mạch dọc theo đường đi. Base Classifier cung cấp cho ta 2 hàm điều khiển gói tin điều khiển và burst dữ liệu.

void BaseClassifier::handleControlPacket( Packet \*p ) {

int fdl\_count;

LaucScheduler \*ls = sg.search( getNextHop( iph->daddr() ) );

if( ls == NULL ) {

char debugStr[100];

sprintf(debugStr, "Laucscheduler not found for destination %s at node %d",

iph->daddr(), address\_ );

Debug::debug( \_\_FILE\_\_, \_\_LINE\_\_, debugStr );

// if this error occur check the TCL initialization of the laucschedulers

// also check if the nextHop method returns the current next-hop.

exit (-1);

}

double bhpDur = ls->duration( ch->size() );

double burstDur = ls->duration( hdr->C\_burst\_size() );

double curTime = Scheduler::instance().clock();

Khi một burst tới một node, nó cần được cấp cho một kênh bước sóng ở ngõ ra. Ngoài ra, nhằm làm giảm thiểu khoảng thời gian trống giữa 2 burst truyền đi trên cùng một kênh bước sóng, người ta dùng thêm bộ sắp xếp các burst tại tất cả các node tham gia trong mạng, bộ đó được gọi là bộ xếp lịch (channel scheduling). Đoạn lệnh trên dùng để lập lịch cho gói điều khiển.

Trong quá trình truyền gói điều khiển thỉnh thoảng vẫn xảy ra trường hợp gói điều khiển bị drop.Tại nút lõi có các trường hợp sau:

if (type\_ == 1) //core node; note that for egress edge node,

//the current function is not invoked

hdr->offset\_time\_ += (hdr->FDL\_delay\_ - proc\_time\_ -

hdr->tx\_delay\_);

if( hdr->offset\_time\_ <= 0 )

{

StatCollector &sc = StatCollector::instance();

sc.updateEntry("BHPDROP", sc.getValue( "BHPDROP" ) + 1.0 );

drop(p);

return;

}

Đây là trường hợp burst được gửi đi khi chưa hoàn thành dự trữ tài nguyên ở các node trung gian, tức là thời gian offset time quá ngắn, gói điều khiển chưa được xử lí xong thì burst dữ liệu đã được gửi đi.

Sau đây là trường hợp thứ 2 drop gói điều khiển:

double burstStartTime = curTime + hdr->offset\_time\_;

fdl\_count = hdr->fdl\_count\_; //GMG -- added passing of fdl\_count to scheduler

Schedule data = ls->schedData(burstStartTime, burstDur, fdl\_count );

//GMG -- Adjust hdr->offset\_time\_ for any FDLs that are scheduled

if (FS\_.option\_ == 1) // max # FDLs used per node

hdr->offset\_time\_ += (double)(fdl\_count) \* (FS\_.fdl\_delay\_);

else if (FS\_.option\_ == 2) // max # FDLs used per path

hdr->offset\_time\_ += (double)(fdl\_count - hdr->fdl\_count\_) \*

(FS\_.fdl\_delay\_);

hdr->fdl\_count\_ = fdl\_count;

if( data.channel() < 0 ) {

StatCollector &sc = StatCollector::instance();

//GMG - fixed following line; "BHPDROP" was written "BHPROP"

sc.updateEntry("BHPDROP", sc.getValue( "BHPDROP" ) + 1.0 );

drop(p);

return;

}

Dựa vào bộ lập lịch trên kênh dữ liệu, khi mà tất cả các kênh dữ liệu đều bận tức là theo lịch tất cả các kênh dữ liệu đều đã dung để truyền các burst dữ liệu nên không còn kênh nào để truyền burst dữ liệu tương ứng với gói điều khiển này nên gói điều khiển bị drop.Đây là trường hợp rất phổ biến do trong mạng có rất nhiều burst được truyền đi nên rất dễ xảy ra xung đột.Số lượng Burst càng nhiều thì càng dễ dẫn đến xung đột

Tại nút biên thì chỉ có 1 trường hợp gói điều khiển bị drop xảy ra:

if (type\_ == 0) //edge node; update offset to account for

//any electronic buffering of DB

hdr->offset\_time\_ += (data.startTime() - burstStartTime);

Schedule control = ls->schedControl( bhpStartTime, bhpDur );

if( control.channel() < 0 ) {

char str[200];

StatCollector &sc = StatCollector::instance();

sc.updateEntry("BHPDROP", sc.getValue("BHPDROP") + 1.0 );

FS\_.FdlSchedRestore();

drop(p);

return;

}

Trường hợp này xảy ra tại nút biên. Khi các gói tin được tổ chức thành các burst thì gói điều khiển được tạo ra và truyền đi trước trên các kênh điều khiển. Trường hợp các gói điều khiển được phát sinh quá nhiều, các kênh điều khiển không đủ để sử dụng thì các gói điều khiển sẽ bị drop.

Sau mỗi lần drop gói điều khiển thì ta đều dùng câu lệnh:

StatCollector &sc = StatCollector::instance();

sc.updateEntry( "BHPDROP", sc.getValue( "BHPDROP" ) + 1.0 );

// tăng giá trị trường BHPDROP của StatCollector lên 1, dùng để thống kê kết quả mô phỏng.

Ngoài hàm điều khiển gói điều khiển trên, Base Classifier cũng cung cấp hàm điều khiển burst dữ liệu:

void BaseClassifier::handleDataBurst( Packet \*p ) {

if( p == NULL )

return;

Debug::markTr( address\_, p );

hdr\_cmn \*ch = hdr\_cmn::access( p );

hdr\_ip \*iph = hdr\_ip::access( p );

hdr\_IPKT \*hdr = hdr\_IPKT::access( p );

if( ( ch->ptype() != PT\_IPKT ) || ( iph->prio\_ != 2 ) ) {

Debug::debug( \_\_FILE\_\_, \_\_LINE\_\_,"Critical error: DataburstHandler reeived a non-burst ipkt" );

exit( -1 );

}

HashEntry \*he = lswitch.erase( (unsigned long) hdr->C\_burst\_id() );

if( he == NULL ) {

/\* critical error bhp is ahead of the burst or no scheduler found\*/

/\* char str[100];

sprintf( str, "Dropping burst: %d", hdr->C\_burst\_id() );

Debug::debug( str ); \*/

StatCollector &sc = StatCollector::instance();

sc.updateEntry("BURSTDROP", sc.getValue( "BURSTDROP" ) + 1.0 );

Ta thấy khi gói điều khiển tương ứng với burst dữ liệu này bị drop thì burst dữ liệu bị drop theo. Khi drop burst dữ liệu thì trường BURSTDROP của StatCollector cũng được tăng lên 1. Sau đó, burst dữ liệu sẽ được phân loại và tăng giá trị trường drop tương ứng trong StatCollector:

if ( tcpch->ptype() == PT\_TCP)

{

sc = StatCollector::instance();

sc.updateEntry( "TCPDROP", sc.getValue( "TCPDROP" ) + 1.0 );

}

else if ( tcpch->ptype() == PT\_ACK)

{

sc = StatCollector::instance();

sc.updateEntry("ACKDROP", sc.getValue( "ACKDROP" ) + 1.0 );

}

else if ( tcpch->ptype() == PT\_UDP)

{

sc = StatCollector::instance();

sc.updateEntry( "UDPDROP", sc.getValue( "UDPDROP" ) + 1.0 );

}

else if ( tcpch->ptype() == PT\_CBR)

{

sc = StatCollector::instance();

sc.updateEntry( "CBRDROP", sc.getValue( "CBRDROP" ) + 1.0 );

}

else if ( tcpch->ptype() == PT\_EXP)

{

sc = StatCollector::instance();

sc.updateEntry("EXPDROP", sc.getValue("EXPDROP") + 1.0 );

}

else if ( tcpch->ptype() == PT\_PARETO)

{

sc = StatCollector::instance();

sc.updateEntry( "PARDROP", sc.getValue("PARDROP" ) + 1.0 );

}

//free( \*tcp\_pkt ); //GMG -- fixed this; originally was tcp\_pkt

// rather than \*tcp\_pkt

//GMG -- further change -- changed from free to drop to allow for trace objects

drop( \*tcp\_pkt );

tcp\_pkt++;

}

drop(p);

return;

Ngược lại, nếu gói điều khiển của burst dữ liệu không bị drop thì burst dữ liệu được lập lịch và gửi đi:

double curTime = Scheduler::instance().clock();

double sendTime = 0.0;

if( he->arrTime >= curTime ) {

if( type\_ == 0 ) {

char str[100];

Scheduler::instance().schedule( slot\_[iph->daddr()], p, sendTime );

if (type\_ == 0)

{

hdr->ebuf\_ind = 1;

hdr->bc\_ingress = this;

}

}

}

Ngoài ra, Base Classifier còn cung cấp cho chúng ta các hàm khác như :

Hàm dùng để drop gói tin

void BaseClassifier::drop(Packet\* p)

{

if (drop\_ != 0)

drop\_->recv(p);

else

Packet::free(p);

}

Hàm gửi gói tin:

void BaseClassifier::recv( Packet \*pkt, Handler \*h ) {

Classifier::recv( pkt, h );

}

Hàm lấy địa chỉ của hop tiếp theo:

int BaseClassifier::getNextHop( nsaddr\_t addr ) {

Tcl& tcl = Tcl::instance();

sprintf( tcl.buffer(), "$ns getnexthop %d %d", address\_, addr );

tcl.eval();

return atoi( tcl.result() );

}

*b. StatCollector*

StatCollector là một đối tượng dùng để thống kê kết quả của mô phỏng, qua đó đánh giá hiệu quả của mô hình mạng. Trong StatCollector có một số trường sau cần chú ý:

TCPSND, // TCP Send

TCPRCV, // TCP recv

BURSTSND, // Burst snd

BURSTDROP, // Burst dropped

BURSTRCV, // Burst received

BHPDROP, // BHP Dropped

BHPSND, // BHP Sent

BHPRCV, // BHP Received

TCPDROP, // TCP segments dropped

ACKDROP, // ACK segments dropped

UDPSND, // UDP Send -- added by GMG

UDPRCV, // UDP Receive -- added by GMG

ACKSND, // ACK Send -- added by GMG

ACKRCV, // ACK Receive -- added by GMG

CBRSND, // CBR Send -- added by GMG

CBRRCV, // CBR Receive -- added by GMG

…

Trong StatCollector cũng có một số hàm hỗ trợ việc thống kê kết quả mô phỏng như :

* Hàm lấy giá trị một trường của StatCollector

StatEntry& StatEntry::getStatEntry( string keyStr ) {

map<string, StatEntry>::iterator result = mapList\_\_.find( keyStr );

return (\*result).second;

}

* Hàm hiển thị giá trị của tất cả các trường của StatCollector ra màn hình:

void StatEntry::displayEntry( StatEntry se ) {

char s[100];

double pktrcv1;

switch (se.type() )

{

case TCPDLY:

pktrcv1 = StatEntry::getStatEntry("TCPRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case UDPDLY:

pktrcv1 = StatEntry::getStatEntry("UDPRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case CBRDLY:

pktrcv1 = StatEntry::getStatEntry("CBRRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case EXPDLY:

pktrcv1 = StatEntry::getStatEntry("EXPRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case PARDLY:

pktrcv1 = StatEntry::getStatEntry("PARRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case SSIMDLY:

pktrcv1 = StatEntry::getStatEntry("SSIMRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case BURSTDLY:

pktrcv1 = StatEntry::getStatEntry("BURSTRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

case BHPDLY:

pktrcv1 = StatEntry::getStatEntry("BHPRCV").value();

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(),

se.infoStr().c\_str(),

(pktrcv1 != 0.0 ? se.value()/pktrcv1 : 0.0) );

break;

default:

sprintf( s, "Type: %d Str: %s Value: %lf ", se.type(), se.infoStr().c\_str(), se.value() );

break;

}

Debug::debug( s );

}

Giá trị của tất cả các trường của StatCollector được khởi tạo ban đầu bằng 0 bởi hàm:

StatCollector::StatCollector() {

for( int i = 0; i < 43; i++ ) {

StatEntry::addStatEntry((statType)i, StatEntry::baseInfoList[i], 0.0 );

}

}

Trong quá trình truyền dữ liệu trong mạng, giá trị của các trường trong StatCollector thay đổi liên tục, hàm sau cho phép chúng ta cập nhật lại gía trị của trường tương ứng:

void StatCollector::updateEntry( string entry, double value ) {

StatEntry::getStatEntry( entry ).value() = value;

}

*c. Lauc – Scheduler*

Khi một burst tới một node, nó cần được cấp cho một kênh bước sóng ở ngõ ra, vì vậy tất cả các node trong hệ thống mạng đều phải có bộ wavelength converter. Ngoài ra, nhằm làm giảm thiểu khoảng thời gian trống giữa 2 burst truyền đi trên cùng một kênh bước sóng, người ta dùng thêm bộ sắp xếp các burst tại tất cả các node tham gia trong mạng, bộ đó được gọi là bộ xếp lịch (channel scheduling).

Khi header của burst dữ liệu tới được nút lõi, các thông số về burst dữ liệu sẽ được nhận biết ở nút lõi như chiều dài burst (burst duration), thời gian burst đó tới nút (arrival time)… Dựa vào những thông số này, nút lõi sẽ xác định được kênh bước sóng thích hợp nhất dành cho burst dữ liệu nhờ thuật toán sắp xếp của bộ channel scheduling.

Có nhiều giải thuật lập lịch nhưng trong gói OBS-NS sử dụng giải thuật LAUC (Latest Available Unschedule Channel).

Trong Lauc Scheduler có các tham số sau:

void LaucScheduler::alloc( u\_int ncc, u\_int ndc, u\_int maxChannels,

BaseClassifier \*parent ) {

assert( ( ncc > 0 ) && ( ndc > 0 ) && ( maxChannels > 0 ) );

assert( (ncc + ndc) == maxChannels );

ncc\_ = ncc;

ndc\_ = ndc;

maxChannels\_ = maxChannels;

unschTime\_ = new double[maxChannels];

startTime\_ = new double[maxChannels];

endTime\_ = new double[maxChannels];

memset( unschTime\_, 0, maxChannels \* sizeof( double ) );

memset( startTime\_, 0, maxChannels \* sizeof( double ) );

memset( endTime\_, 0, maxChannels \* sizeof( double ) );

}

Trong đó

* ncc : số kênh điều khiển
* ndc : số kênh dữ liệu
* maxChannels : tổng số kênh , maxChannels=ncc+ndc
* unschTime : thời gian mà kênh đó rãnh
* startTime : thời gian mà kênh đó bắt đầu truyền dữ liệu
* endTime : thời gian mà kênh đó chấm dứt việc truyền dữ liệu

Các kênh điều khiển tiến hành lập lịch như sau :

Schedule LaucScheduler::schedControl(double schedTime, double schedDur ) {

Schedule result;

double diffTime = HUGE\_VAL;

assert( ( schedTime >= 0. ) && ( schedDur >= 0. ) );

assert( ( ncc\_ > 0 ) && ( ndc\_ > 0 ) && ( ( ncc\_ + ndc\_ ) == maxChannels\_ ) );

for( u\_int i = 0; i < ncc\_; i++ ){

if( schedTime >= unschTime\_[i] )

if( ( schedTime - unschTime\_[i] ) < diffTime ) {

diffTime = schedTime - unschTime\_[i];

result.channel() = i;

result.startTime() = schedTime;

}

}

int ch = result.channel();

if( ch >= 0 )

unschTime\_[ch] = schedTime + schedDur;

return result;

}

Trong hàm này có 2 tham số là schedTime và schedDur. Trong đó, schedTime là thời gian mà gói điều khiển dự định được truyền, shedDur là thời gian truyền hết gói tin điều khiển. diffTime chính là thời gian trống lãng phí tức là chênh lệch giữa thời gian dự định bắt đầu truyền gói tin và thời gian mà kênh rảnh. Duyệt tất cả các kênh điều khiển, kênh nào có thời gian rảnh nhỏ hơn hoặc bằng thời gian mà gói tin điều khiển đó bắt đầu truyền, đồng thời thời gian trống lãng phí phải nhỏ thì kênh đó sẽ được dùng để truyền gói tin điều khiển. diffTime sẽ được gắn bằng hiệu giữa thời gian bắt đầu truyền gói tin điều khiển và thời gian rãnh của kênh. Các tham số của kênh thay đổi như: tham số startTime chính bằng thời gian mà gói điều khiển bắt đầu truyền và unschTime chính bằng tổng thời gian bắt đầu truyền gói tin điều khiển và thời gian dùng để truyền hết gói tin đó.

Các kênh dữ liệu được lập lịch bằng hàm sau :

Schedule LaucScheduler::schedData( double schedTime, double schedDur, int &fdl\_count ){

int count = fdl\_count;

Schedule result = search( schedTime, schedDur, count );

fdl\_count = count;

int ch = result.channel();

if( ch >= 0 )

update( ch, result.startTime(), schedDur );

return result;

}

Trong hàm trên có 2 tham trị là schedTime, schedDur và 1 tham biến là fdl\_count. Trong đó, schedTime là thời gian mà burst dữ liệu dự định bắt đầu truyền, schedDur là thời gian cần thiết để truyền hết burst dữ liệu đó và fdl\_count là số FDL đã được sử dụng trong quá trình truyền dữ liệu. Hàm trên đã sử dụng hàm “search” để tìm kiếm 1 kênh dữ liệu đang rãnh và hàm “update” để cập nhật lại giá trị các tham số của kênh đó.

Schedule LaucScheduler::search(double schedTime, double schedDur, int &count ) {

Schedule result;

int max, fdl;

double diffTime = HUGE\_VAL;

while(1) {

for( u\_int i = ncc\_; i < maxChannels\_; i++ )

{

// try to schedule in a void

if( schedTime >= startTime\_[i] )

if( ( endTime\_[i] - schedTime ) >= schedDur )

if( ( schedTime - startTime\_[i] ) < diffTime )

{

diffTime = schedTime - startTime\_[i];

result.channel() = i;

result.startTime() = schedTime;

}

// try to schedule after the void

if( schedTime >= unschTime\_[i] )

if( ( schedTime - unschTime\_[i] ) < diffTime )

{

diffTime = schedTime - unschTime\_[i];

result.channel() = i;

result.startTime() = schedTime;

}

}

break;

}

return (result);

}

Để tìm kiếm kênh dữ liệu có thể truyền burst dữ liệu, ta tiến hành duyệt tất cả kênh dữ liệu mà mình có. Kênh dữ liệu sẽ được chọn nếu thỏa một trong 2 trường hợp sau:

Thứ nhất, thời gian bắt đầu truyền burst dữ liệu lớn hơn hoặc bằng thời gian mà kênh đó bắt đầu truyền dữ liệu theo lịch sắp xếp, hiệu giữa thời gian kết thúc truyền dữ liệu của kênh theo lịch sắp xếp và thời gian bắt đầu truyền burst dữ liệu phải lớn hơn hoặc bằng thời gian cần thiết để truyền hết burst dữ liệu đó đồng thời chênh lệch giữa thời gian bắt đầu truyền burst dữ liệu và thời gian kênh bắt đầu truyền dữ liệu theo lịch không quá lớn.

Thứ hai, thời gian bắt đầu truyền burst dữ liệu phải lớn hơn hoặc bằng thời gian mà kênh đó rãnh đồng thời chênh lệch giữa thời gian bắt đầu truyền burst dữ liệu và thời gian rãnh của kênh đó phải không quá lớn.

Việc cập nhật lại giá trị các tham số của kênh được thực hiện bởi hàm sau:

void LaucScheduler::update(u\_int channel, double schedTime, double schedDur )

{

if(schedTime == unschTime\_[channel])

unschTime\_[channel] = schedTime + schedDur;

else if( schedTime > unschTime\_[channel] ) {

startTime\_[channel] = unschTime\_[channel];

unschTime\_[channel] = schedTime + schedDur;

} else {

// scheduled in the void

// i.e sched\_time < unsch\_time[channel]

startTime\_[channel] = schedTime + schedDur;

}

}

Trong hàm này có 3 tham số là channel, schedTime, schedDur. Trong đó, channel là số thứ tự kênh mà mình cần thay đổi, schedTime là thời gian bắt đầu truyền dữ liệu và schedDur là thời gian truyền hết dữ liệu. Việc thay đổi thông tin được tiến hành như sau:

Nếu thời gian bắt đầu truyền dữ liệu bằng thời rảnh của kênh thì thời gian rảnh của kênh sẽ bằng tổng thời gian bắt đầu truyền dữ liệu và thời gian truyền hết dữ liệu đó .

Nếu thời gian bắt đầu truyền dữ liệu lớn hơn thời gian rảnh của kênh thì thời gian bắt đầu làm việc của kênh bằng thời gian rảnh của kênh và thời gian rảnh của kênh bằng tổng thời gian bắt đầu truyền dữ liệu và thời gian cần thiết để truyền hết dữ liệu

Nếu không rơi vào 2 trường hợp trên, thời gian bắt đầu làm việc của kênh bằng tổng thời gian bắt đầu truyền dữ liệu và thời gian cần thiết để truyền hết dữ liệu.

**2.3.2 Phần TCL**

OBS 0.9a cung cấp cho ta 3 file thư viện là :

***a. /obs-0.9a/tcl/lib/ns-obs-defaults.tcl***

Thư viện này thiết lập các tham số Otcl mà tương ứng với phản ánh thành viên của lớp C++ với các giá trị mặc định. Tập tin này phải luôn luôn được gọi đến, nếu không việc khởi tạo các biến OTcl / C++ sẽ bị cảnh báo lỗi.

***b. /obs-0.9a/tcl/lib/ns-obs-lib.tcl***

Thư viện này có chứa các phương thức của lớp mô phỏng OTcl:

* Tạo các bước truyền tiếp theo trong một hướng:

Simulator instproc getnexthop { src dest } {

set r [ $self get-routelogic ]

set ret [ $r lookup $src $dest ]

return $ret

}

* Nhận được số lượng các bước truyền trong một lộ trình:

Simulator instproc nhops { src dest } {

set r [ $self get-routelogic ]

set ret [ $r nhops $src $dest ]

return $ret

}

* Xây dựng một node lõi mới:

Simulator instproc create-core-node { totcore } {

set n [$self node]

set nodeid [$n id]

set cl [new Classifier/BaseClassifier/CoreClassifier]

$n set classifier\_ $cl

[$n set classifier\_] set address\_ $nodeid

[$n set classifier\_] set type\_ 1

# puts " create-core-node: node $nodeid created"

return $n

}

* Xây dựng mới node biên:

Simulator instproc create-edge-node { totedges } {

set n [$self node]

set nodeid [$n id]

set cl [new Classifier/BaseClassifier/EdgeClassifier]

$n set classifier\_ $cl

[$n set classifier\_] set address\_ $nodeid

[$n set classifier\_] set type\_ 0

set p0 [new Classifier/OBSPort]

$p0 set address\_ $nodeid

set iagent0 [new Agent/IPKT]

$iagent0 initiagent $totedges

$p0 install-iagent $iagent0

# dump the default burst-params

#$iagent0 dumpburstdefaults

$n set dmux\_ $p0

[$n set classifier\_] install $nodeid $p0

$n attach $iagent0

#puts " create-edge-node: node $nodeid created"

return $n

}

* Tạo ra một kết nối FTP:

Simulator instproc create-ftp-connection { src des } {

set source [new Agent/TCP]

set sink [new Agent/TCPSink]

$self attach-agent $src $source

$self attach-agent $des $sink

$self connect $source $sink

set ftp [new Application/FTP]

$ftp attach-agent $source

return $ftp

}

* Xây dựng bảng định tuyến (định tuyến đường đi ngắn nhất):

Simulator instproc build-routing-table { } {

$self instvar Node\_ link\_

set r [$self get-routelogic]

$self compute-routes

$self dump-routelogic-nh

set n [Node set nn\_]

puts "The total number of nodes in this topology is $n"

set i 0

while { $i < $n } {

if ![info exists Node\_($i)] {

puts "Error node $i does not exist"

incr i

continue

}

set n1 $Node\_($i)

set j 0

while { $j < $n } {

if { $i != $j } {

set nh [$r lookup $i $j]

if { $nh >=0 } {

puts "setting up route between $i $j with $nh"

[$n1 set classifier\_] install $j [$link\_($i:$nh) head]

}

}

incr j

}

incr i

}

}

* Tạo ra một liên kết đôi:

Simulator instproc createDuplexFiberLink { node1 node2 bwpc delay ncc ndc maxch } {

$self instvar link\_

$self obs\_duplex-FiberLink $node1 $node2 $bwpc $delay Null $maxch

set id1 [$node1 id]

set id2 [$node2 id]

if [info exists link\_($id1:$id2)] {

[$node1 set classifier\_] install-scheduler $id2 $ncc $ndc $maxch $bwpc

puts "Adding schedulers between $node1 and $node2"

}

if [info exists link\_($id2:$id1)] {

[$node2 set classifier\_] install-scheduler $id1 $ncc $ndc $maxch $bwpc

puts "Adding schedulers between $node2 and $node1"

}

}

* Bật NodeTrace:

Simulator instproc nodetrace-all {file} {

set ndtr [ new NodeTrace ]

$ndtr set trace\_on 1

$ndtr attach $file

}

* Xuất ra những file NodeTrace:

Simulator instproc flush-nodetrace {} {

set ndtr [ new NodeTrace ]

$ndtr flush

}

***c. /obs-0.9a/tcl/lib/ns-obs-link.tcl***

Bao gồm các phương thức Otcl của lớp mô phỏng được thêm vào bởi phương thức createDuplexFiberLink trong ns-obs-lib, thuận lợi cho việc tạo các đối tượng DuplexFiberLink.

**Kết luận chương**

Trong chương này giúp chúng tôi đã hiểu được cấu trúc và hoạt động của phần mềm mô phỏng mạng NS2, và gói OBS-NS được phát triển từ NS2 hỗ trợ cho việc mô phỏng mạng OBS. Qua đó giúp cho ta thấy được cấu trúc cũng như hoạt động cơ bản của gói mô phỏng OBS-NS. Vấn đề chính mà nội dung chương này đặt ra là tìm hiểu được phần code C++ bên trong OBS, giúp cho các nghiên cứu tiếp theo có thể nắm bắt được vấn đề sử dụng và xây dựng các cải tiến đối với một số thuật toán quan trọng như: các thuật toán định tuyến, định tuyến lệch hướng , lập lịch, …

1. [↑](#footnote-ref-1)