VNTELECOM.ORG
ADNET
NETSIM

MÔ PHỎNG TRONG NS-2

TÀI LIỆU THAM KHẢO
LĨNH VỰC MÔ PHỎNG VÀ ỨNG DỤNG

Dịch thuật bởi: Nhóm TE
Biên tập bởi: Nhóm CM

Hà Nội 12 - 2009

Tài liệu này được tôi tổng hợp và định dạng lại từ các bản dịch. Trong quá trình định dạng, tôi đã cố gắng sửa các lỗi như: tham chiếu các phần, lỗi chính tả, thiếu dấu ... nhưng có thể còn bỏ sót. Trong trường hợp bạn đọc phát hiện được lỗi, các bạn hãy gửi thư về địa chỉ ở cuối tài liệu này. Các lỗi này sẽ được sửa đổi trong phiên bản kế tiếp

Xin được trích dẫn một câu trong cuốn sách *"The Last Lecture"*, tác giả cuốn sách đã làm việc tại trường Đại học Carnegie Mellon (CMU). Lý do trích dẫn xin mời bạn đọc tìm hiểu ...!

"...
Chúng ta không đổi được những quân bài đã chia,
chỉ có thể đổi cách chơi những quân bài đó ..."

The Last Lecture by Randy Pausch
Carnegie Mellon University

Lời nói đầu

Kể từ khi ban quản trị diễn đàn vntelecom.org thành lập nhóm AdNet cho tới nay đã được hơn một năm. Từ khi thành lập nhóm đã phát động dự án NETSIM – Dự án biên tập tài liệu mô phỏng bằng tiếng Việt. Đến tháng 04/2009 nhóm đã hoàn thành tập một của dự án, tài liệu có tên là - *"Lý thuyết chung về mô phỏng"*. Tập một đã được công bố chính thức trên diễn đàn vntelecom.org.

Sau khi hoàn thành tập một, nhóm tiếp tục phát triển dự án với mục tiêu là hoàn thành tập hai: *Mô phỏng trong NS-2*. Sau nhiều lần trì hoãn vì những lý do khác nhau, đến tháng 12-2009 nhóm đã hoàn thành tập hai.

Ở tập một chúng tôi đã biên dịch và giới thiệu với bạn đọc những khái niệm tổng quan trong mô phỏng. Những khái niệm về đánh giá hiệu năng, các lỗi thường gặp khi mô phỏng, các kỹ thuật mô phỏng, nguyên lý về mô phỏng, giới thiệu về mô phỏng rời rạc trong hệ thống mạng máy tính và mạng viễn thông. Ngoài ra còn đề cập đến việc so sánh và đánh giá các chương trình mô phỏng phổ biến hiện nay. Như vậy, nội dung trong tập một là nền tảng để những người quan tâm đến lĩnh vực mô phỏng có thể tìm hiểu. Đồng thời nó là cơ sở để bạn đọc tìm hiểu các vấn đề được đề cập trong tập hai. Những vấn đề này thường hay bị bỏ qua đối với những người mới tìm hiểu.

Trong tập hai, chúng tôi giới thiệu với bạn đọc một bộ công cụ mô phỏng cụ thể - đó là bộ công cụ mô phỏng NS-2-allinone. Trong tài liệu này gọi tắt là NS-2. Lý do chúng tôi chọn NS-2 đơn giản là đây là bộ mô phỏng mã nguồn mở, miễn phí với người dùng. Nó là bộ công cụ mô phỏng mạnh, được cộng đồng mạng sử dụng phổ biến ở khắp nơi trên thế giới, thậm chí có những diễn đàn chuyên về chúng. Bộ mô phỏng này phù hợp với các đối tượng sinh viên, nghiên cứu sinh. Thông tin chi tiết được trình bày ở chương 1 của cuốn này.

Mục đích của việc dịch tài liệu

Để xây dựng nội dung cho tập hai, chúng tôi đã lựa chọn và biên dịch từ các tài liệu [1] [2] [3] [4], đây là những tài liệu không thể thiếu được với những ai đã đang và sẽ tìm hiểu về mô phỏng. Sở dĩ nhóm tiến hành dự án biên dịch này là để phổ biến NS-2 tới các bạn sinh viên, các nghiên cứu sinh.... Ngoài ra để các bạn mới tìm hiểu

và có ý định tìm hiểu rút ngắn được thời gian tiếp cận. Để đảm báo tính chính xác và các vấn đề bản quyền, chúng tôi đã đưa thông tin và nguồn gốc của bản gốc – bản được lựa chọn để dịch ở đầu mỗi chương. Nếu cảm thấy chưa an tâm với bản dịch các bạn có thể tham khảo trực tiếp các bản gốc.

Nội dung các chương trong tập hai

Từ chương 1 tới chương 4 giới thiệu tổng quan về NS2, các thành phần trong NS2, giới thiệu về hai ngôn ngữ chính được sử dụng trong NS-2 là TCL và C++, mô tả về file vết sau khi chạy mô phỏng. . Hiểu và nắm chắc các phần này bạn đọc sẽ sử dụng NS-2 được tốt hơn.

Để các bạn mới tìm hiểu hứng thú hơn với NS-2, chúng tôi trình bày trong chương 5 một "tutorial" để hướng dẫn bạn đọc có thể tạo được một chương trình đơn giản đầu tiên. Đặc biệt, chương 5 còn cung cấp mã nguồn của các chương trình mẫu, bạn đọc có thể tải về, chạy thử và quan sát kết quả.

Chương 6, 7, 8,9 cung cấp cho bạn đọc kiến thức về các kỹ thuật để mô tả về giao thức truyền tải TCP, UDP, mô tả về TCP/IP trong NS, loại bỏ ngẫu nhiên sớm. Đây là các kỹ thuật cơ bản để áp dụng vào một chương trình cụ thể.

Từ chương 10 đến chương 14 giới thiệu với bạn đọc chi tiết hơn về các mô-dun mô phỏng đã phổ biến. Nội dung của từng chương được tập trung để làm sáng tỏ các vấn đề từ lý thuyết cho đến lúc áp dụng vào mô hình mô phỏng. Nội dung các chương này có thể nói ở mức độ nâng cao hơn, phù hợp với những sinh viên năm cuối. Những người làm đề tài tốt nghiệp có thể tham khảo để phục vụ cho báo cáo của mình. Điển hình là:

Chương 10: Mô phỏng mạng LAN.

Chương 11: Mô phỏng MPLS [4]

Chương 12: Mô phỏng hàng đợi.

Chương 13: Mô phỏng trong mạng di động.

Chương 14: Mô phỏng mạng vệ tinh.

Với hơn 200 trang tài liệu, không quá dài mà cũng không quá ngắn. Nhóm biên soạn hi vọng sẽ cung cấp cho bạn đọc những trang tài liệu bổ ích. Dựa vào đây bạn đọc có thể tiếp cận, kế thừa, phát triển và sáng tạo ra các kịch bản mô phỏng. Nếu đủ khả năng có thể viết được các mô-dun dựa trên NS2. Để tạo ra tài liệu này các nhóm làm việc đã sử dụng những tiện ích được cung cấp trên mạng internet, gắn kết và liên lạc với nhau. Các nhóm được tổ chức theo quy trình: lựa chọn tài liệu, dịch, biên tập, định dạng và xuất bản. Thông tin nhóm các bạn có thể tham khảo ở phần cuối cùng của tài liệu.

Với tinh thần tự nguyện, chủ động và không ngừng học hỏi đã giúp chúng tôi hoàn thành tài liệu này. Sản phẩm này là công sức chung của cả nhóm, nhân đây ban biên tập chân thành cám ơn sự cộng tác của các cá nhân, các sinh viên, các thầy cô từ các trường đại học, đồng thời xin được gửi lời cám ơn tới ban quản trị diễn đàn vntelecom.org đã tạo ra sân chơi này, đây là một sân chơi thực sự bổ ích với những người quan tâm đến lĩnh vực mạng viên thông nói riêng và truyền thông nói chung.

Cuốn sách này được thành viên trong nhóm NETSIM định dạng và hiệu chỉnh. Trong quá trình làm việc đã cố gắng hết khả năng để thể hiện tài liệu được khoa học nhất, trong quá trình định dạng đã chỉnh sửa lại các phần tham khảo, các chú thích theo bản gốc, một số chỗ người dịch không đưa vào bản dịch. Do đó khi đọc tài liệu này các bạn có thể tham khảo cùng với bản gốc nếu có khả năng Anh văn tốt. Đặc biệt, việc thống nhất các thuật ngữ rất cần bạn đọc và những người có kinh nghiệm góp ý thêm. Trong phiên bản tới, nhóm sẽ bổ xung các thiếu sót (thuật ngữ, lỗi chính tả, cú pháp câu, việt hóa hình ảnh . . .), bổ xung các mô-dun mô phỏng WiMax, mô phỏng WDM, mô phỏng SDR. . . Việc cài đặt NS-2 đã có rất nhiều tài liệu và hướng dẫn bằng tiếng Việt các bạn có thể tham khảo từ mạng internet, chúng tôi không đưa vào trong tài liệu này.

Mọi ý kiến đóng góp bạn đọc có thể gửi về thư điện tử của nhóm:
adnet@vntelecom.org ban biên tập sẽ tiếp nhận, sửa chữa và bổ xung kịp thời.

Ngày 24 tháng 12 năm 2009
Tác giả

Nhóm biên soạn

Mục lục

Danh sách hình vẽ

Danh sách bảng

Những từ viết tắt

AODV	Ad hoc On Demand Distance Vector
CDMA	Code Division Multiple Access
CIR	Committed Information Rate
CTS	Clear to Send
CR-LDP	Constraint-based Label Distribution Protocol
CR-LSP	Constraint-based routed label switched path
CSMA/CD	Carrier Sense Multiple Access with Collision Detect
Diffserv	Differentiated Services
ED	Early Drops
ERB	Explicit Route information Base
FDDI	Fiber Distributed Data Interface
FDMI	Frequency Division Multiple Access
FTP	File Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
Intserv	Integrated Service
IP	Internet Protocol
LAN	Local Area Network
LDP	Label Distribution Protolcol
LIB	Label Information Table
LSP	Label Switch Path
LSR	Label Switch Router
MAC	Medium Access Control
MPLS	Multiprotocol Label Switching
MRED	Multi RED
NAM	Network Animator
NS	Network Simulaton
NS 2	Network Simulaton version 2
PFT	Partial Forwarding Table
PHB	Per Hop Behavior
QoS	Quality of service
RED	Random Early Detection
RFC	Request for Comments
RR	Round Robin

TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VINT	Virtual InterNetwork Testbed
WIRR	Weighted Interleaved Round Robin

Chương 1

Giới thiệu về NS-2

Mục lục

Người dịch: *Nguyễn Quốc Huy*
Biên tập: *Hà Tất Thành.*
Bản gốc: *The ns Manual, Chapter 3* [1]

NS là một bộ mô phỏng hướng đối tượng, được viết bằng ngôn ngữ C++, với "trình thông dịch" OTcL như là lớp vỏ (frontend). Bộ mô phỏng này hổ trợ một nhánh lớp trong C++ (còn được gọi là nhánh đã được biên dịch "compiled hierarchy" trong tài liệu này), và một nhánh lớp tương tự trong trình thông dịch OTcL (trong tài liệu này gọi là nhánh được thông dịch "interpreted hierarchy"). Hai hệ thống này có mối tương quan khá mật thiết; nhìn từ phía người sử dụng thì đây là mối tương quan một - một giữa một lớp ở nhánh thông dịch và một lớp ở nhánh biên dịch. Phần gốc của nhánh này là lớp TcLObject. Người dùng tạo các đối tượng của bộ mô phỏng mới thông qua bộ thông dịch; các đối tượng này được tạo sẵn bên trong trình thông dịch, và cũng được phản ảnh bởi một đối tượng tương ứng trong nhánh biên dịch. Nhánh lớp đã được thông dịch sẽ được thành lập một cách tự động thông qua các phương pháp được định nghĩa trong lớp TcLClass. Các đối tượng đã được tạo sẵn được phản ánh thông qua các phương pháp được định nghĩa trọng lớp TclObject. Có các nhánh khác trong mã C++ và đoạn mã Otcl; những nhánh khác này không được phản ảnh trong cách của TclObject.

1.1. Khái niệm tổng quan

Tại sao lại là hai ngôn ngữ? NS sử dụng hai ngôn ngữ bởi vì bộ mô phỏng có hai kiểu công việc khác nhau cần phải làm. Một mặt, mô phỏng chi tiết các giao thức yêu cầu một ngôn ngữ lập trình hệ thống có thể thao tác một cách hiệu quả đến các byte, các tiêu đề gói, và các thuật toán triển khai mà có thể chạy trên một tập dữ liệu lớn. Với nhiệm vụ này thì tốc độ chạy chương trình và thời gian xoay vòng (gồm chạy mô phỏng, tìm lỗi, sữa lỗi, biên dịch lại, chạy lại) là ít quan trọng hơn.

Mặt khác, một phần lớn các nghiên cứu về mạng bao gồm các tham số hoặc các cấu hình ít thay đổi, hay nghiên cứu nhanh chóng một số hoạt cảnh. Trong các trường hợp đó, thời gian lặp lại (thay đổi mẫu và chạy lại) quan trọng hơn. Khi cấu hình chạy một lần (tại thời điểm bắt đầu mô phỏng), thời gian chạy của phần nhiệm vụ này thì ít quan trọng hơn.

NS đáp ứng cả hai yêu cầu này với 2 ngôn ngữ, C++ và OTcl. Với C++ chạy chương trình nhanh nhưng thay đổi lại chậm hơn, điều đó khiến nó phù hợp cho việc

triển khai giao thức chi tiết. OTcl chạy chậm hơn nhiều nhưng có thể được thay đổi rất nhanh (và có thể tương tác), và nó là ý tưởng cho việc cấu hình mô phỏng. NS (thông qua tclcl) cung cấp sự kết dính để tạo nên các đối tượng và các biến trong cả hai ngôn ngữ.

Để biết thêm thông tin về ý tưởng về các ngôn ngữ tạo bản và chương trình phân chia ngôn ngữ, tham khảo tiêu đề của Ousterhout trong IEEE Computer. Thông tin về chương trình phân cấp phân chia cho mô phỏng mạng, xem trang NS[2].

Ngôn ngữ nào để làm gì? Có hai ngôn ngữ cót thể lựa chọn dẫn đến làm tăng số câu hỏi rằng ngôn ngữ nào nên được dùng cho mục đích gì. Lời khuyên cơ bản của chúng tôi là dùng Otcl để :

- Cấu hình, cài đặt, và các công việc thực hiện một lần.
- Nếu bạn có thể làm điều gì bạn muốn bằng việc thao tác với các đối tượng có sẵn trong C++ và dùng C++.
- Nếu bạn sẽ làm bất cứ thứ gì mà yêu cầu xử lý mỗi gói của một luồng.
- Nếu bạn phải thay đổi hoạt động của một lớp có sẵn trong C++ bằng những cách mà chưa từng được biết đến.

Ví dụ, các kênh truyền là các đối tượng OTcl mà tập hợp độ trễ, việc xếp hàng, và các môđun có thể bị mất. Nếu bạn có thể sử dụng thành thạo các công cụ này thì tốt. Nếu không, thay vào đó bạn muốn làm điều gì đó riêng biệt (một quy tắc xếp hàng đặc biệt hoặc các kiểu mất mát), thì bạn cần một đối tượng C++ mới.

Có những phần không rõ ràng: hầu hết việc định tuyến được thực hiện trong OTcl (mặc dù thuật toán lõi Dijkstra ở trong C++). Chúng ta đã có những bộ mô phỏng HTTP mà trong đó mỗi luồng được bắt đầu trong OTcl và quá trình xử lý trên một gói đã được định nghĩa trong C++. Điều này ổn khi chúng ta đã có 100các luồng bắt đầu trên một giây trong thời gian mô phỏng. Nói chung, nếu chúng ta cần đến Tcl nhiều lần trên một giây, thì nên chuyển sang C++.

1.2. Tổng quan về mã nguồn

Trong tài liệu này, chúng ta sử dụng từ "interpreter" để đồng nghĩa với trình thông dịch OTcl. Mã để giao tiếp với trình thông dịch này nằm trong một thư mục riêng, tclcl. Phần còn lại của mã bộ mô phỏng nằm trong thư mục ns-2. Chúng ta sẽ sử dụng ký hiệu tclcl/hfilei để đề cập đến một phần hfilei riêng biệt trong Tcl. Tương tự, chúng ta dùng ký hiệu, ns/hfilei để gán cho phần hfilei trong thư mục ns-2 .

Có một số lượng các lớp được định nghĩa trong tclcl/. Chúng ta chỉ tập trung vào lớp thứ 6 được dùng trong ns. Lớp Tcl (phần 1.3) bao gồm các phương pháp mà

mã C++ sẽ dùng để truy cập trình thông dịch. Lớp TclObject (phần 1.4) là lớp cơ bản cho tất cả các đối tượng của bộ mô phỏng mà cũng được phản ánh vào trong hệ thống cấp bậc được biên dịch. Lớp TclClass (phần 1.5) xác định các hệ thống cấp bậc lớp được biên dịch, và các phương pháp để cho phép người dùng diễn giải TclObjects. Lớp TclCommand (phần 1.6) được sử dụng để định nghĩa các lệnh biên dịch toàn cục cơ bản. Lớp EmbeddedTcl (phần 1.7) bao gồm các phương thức để tải các lệnh có sẵn ở cấp độ cao hơn, điều này làm cho việc mô phỏng cấu hình trở nên dễ dàng hơn. Cuối cùng, lớp InstVar (phần 1.8) là các phương pháp để truy cập các biến thành viên của C++ như là cac biến trong OTcl.

Các thủ tục và các hàm được mô tả trong chương này có thể được tìm thấy trong tclcl/Tcl.cc, h, tclcl/Tcl2.cc, tclcl/tcl-object.tcl, và tclcl/tracedvar.cc, h. File tclcl/tcl2c++.c được dùng để xây dựng ns, và được đề cập một cách ngắn gọn trong chương này.

1.3. Lớp Tcl

Lớp class Tcl tóm gọn trường hợp thực của bộ thông dịch OTcl, và cung cấp các phương pháp để truy cập và giao tiếp với bộ thông dịch này. Các phương thức được mô tả trong phần này liên quan đến người lập trình ns, tức người sẽ viết đoạn mã C++. Nó còn cho ta các phương pháp để cho các hoạt động sau đây:

- Đạt được tham chiếu tới Tcl instance.
- Gọi ra các thủ tục OTcl thông qua bộ thông dịch.
- Khôi phục hoặc gửi lại các kết quả của trình thông dịch.
- Báo cáo các tình trạng lỗi và thoát ra như bình thường.
- Lưu trữ và tra cứu các "TclObjects".
- Giành quyền truy cập trực tiếp vào trình thông dịch.

1.3.1. Đạt được một tham chiếu với ví dụ về lớp Tcl

Một instance đơn lẻ của lớp được khai báo trong file tclcl/Tcl.cc như một biến thành viên tĩnh; người lập trình phải nhận được một tham chiếu đến instance này để có thể truy cập tới các phương pháp khác được mô tả trong phần này. Câu lệnh để truy cập instance này là.

```
Tcl& tcl = Tcl::instance();
```

1.3.2. Dẫn chứng về các thủ tục của OTcl

Có 4 phương pháp khác nhau để gọi ra một lệnh OTcl thông qua trường tcl. Chúng khác nhau một cách cơ bản về các đối số gọi. Mỗi hàm chuyển một chuỗi vào trình thông dịch, mà sau đó đánh giá chuỗi này trong một ngữ cảnh chung (global context). Các phương pháp này sẽ trả về cho hàm gọi nếu trình thông dịch trả về bản tin TCL_OK. Hay nói cách khác, nếu trình thông dịch trả về TCL_ERROR, các phương thức này sẽ gọi thủ tục *tkerror{}*. Người dùng có thể làm quá tải thủ tục này để có bỏ qua một số loại lỗi nào đó. Những phức tạp của việc lập trình OTcl nằm ngoài phạm vi của tài liệu này. Phần tiếp theo (phần 1.3.3) sẽ mô tả các phương thức để truy cập kết quả được trả về bởi bộ thông dịch.

- tcl.eval(char* s) invokes Tcl_GlobalEval() to execute s through the interpreter.
- tcl.evalc(const char* s) preserves the argument string s. It copies the string s into its internal buffer; it then invokes the previous eval(char* s) on the internal buffer.
- tcl.eval() assumes that the command is already stored in the class' internal bp_; it directly invokes tcl.eval(char*bp_). A handle to the buffer itself is available through the method tcl.buffer(void).
- tcl.evalf(const char* s, . . .) is a Printf(3) like equivalent. It uses vsprintf(3) internally to create the input string.
- tcl.eval(char* s) viện dẫn Tcl_GlobalEval() để thực thi s thông qua trình thông dịch.
- tcl.evalc(const char* s) lưu giữ chuỗi đối số s. Nó copy chuỗi s vào bộ đệm bên trong; sau đó nó dẫn nhập eval(char* s) trước đó lên bộ đệm bên trong.
- tcl.eval() cho rằng lệnh đã được lưu trữ sẵn sàng trong bp_ nội của lớp; nó dẫn nhập tcl.eval(char*bp_) một cách trực tiếp. Mặt khác bộ đệm của nó luôn sẵn sàng thông qua thủ tục tcl.buffer(void).
- tcl.evalf(const char* s, . . .) tương đương với Printf(3). Nó sử dụng vsprintf(3) để tạo nên một chuỗi nội tại bên trong nó.

Ví dụ, dưới đây là một số cách để sử dụng các phương pháp nói trên:

```
Tcl& tcl = Tcl::instance();
char wrk[128];
strcpy(wrk, "Simulator set NumberInterfaces_ 1");
tcl.eval(wrk);
sprintf(tcl.buffer(), "Agent/SRM set requestFunction_ %s", "Fixed");
```

```
tcl.eval();
tcl.evalc("puts stdout hello world");
tcl.evalf("%s request %d %d", name_, sender, msgid);
```

1.3.3. Truy nhập các kết quả đến/từ trình thông dịch

Khi trình thông dịch viện dẫn một phương thức từ C++, nó mong muốn có được một kết quả phản hồi trong một biến thành viên riêng biệt, tcl_->result. Có hai cách để tạo nên biến này:

- tcl.result(const char* s)
  Dẫn nhập chuỗi kết quả s trở về lại trình thông dịch.
- tcl.resultf(const char* fmt, . . .)
  varargs(3) sự thay đổi như trên là để định dạng kết quả sử dụng vsprintf(3), dẫn nhập chuỗi kết quả trở về lại trình thông dịch.

```
if (strcmp(argv[1], "now") == 0) {
    tcl.resultf("%.17g", clock());
    return TCL_OK;
}
tcl.result("Invalid operation specified");
return TCL_ERROR;
```

Tương tự, khi một phương thức của C++ viện dẫn một lệnh OTcl, trình thông dịch sẽ trả về kết quả trong tcl_->result.

- tcl.result(void) phải được sử dụng để tìm lại kết quả. Lưu ý rằng, kết quả là một chuỗi, phải được chuyển sang một định dạng bên trong phù hợp với kiểu của kết quả.

```
tcl.evalc("Simulator set NumberInterfaces_");
char* ni = tcl.result();
if (atoi(ni) != 1)
    tcl.evalc("Simulator set NumberInterfaces_ 1");
```

1.3.4. Thoát và báo cáo lỗi

Phương pháp này cung cấp một cách thức quy chuẩn để báo cáo các lỗi trong mã được biên dịch.

- tcl.error(const char* s) thực hiện các chức năng sau đây: ghi s vào stdout; ghi tcl_->result vào stdout; thoát ra với mã lỗi 1.

```
tcl.resultf("cmd = %s", cmd);
tcl.error("invalid command specified");
            /*NOTREACHED*/
```

Lưu ý, không có sự khác nhau nhiều giữa việc trả về bản tin TCL_ERROR như chúng ta đã làm trong phần trước (phần 3.3.3), và việc gọi Tcl::error(). Bộ định dạng sẽ tạo một ngoại lệ bên trong trình thông dịch; người dùng có thể giữ ngoại lệ này và có thể khôi phục lại từ lỗi. Nếu người dùng đã không định nghĩa bất cứ ngoại lệ nào, thì trình thông dịch sẽ in một vết cụm và thoát. Tuy nhiên, nếu mã gọi ra error(), thì người dùng mô phỏng không thể giữ lỗi; thêm vào đó, ns sẽ không in bất kỳ một vết cụm nào.

1.3.5. Các hàm hash trong trình thông dịch

ns lưu trữ một tham chiếu với mọi TclObject trong nhánh được biên dịch trong một một bảng hash; điều này cho phép truy cập nhanh đến các đối tượng. Bảng hash này nằm ở bên trong bộ thông dịch. ns dùng tên của TclObject như là chìa khoá để truy nhập, tra cứu, hoặc xoá TclObject đó trong bảng hash.

- tcl.enter(TclObject* o) sẽ chèn một con trỏ vào TclObject o bên trong bảng hash.
  Nó được dùng bởi TclClass::create_shadow() để chèn một đối tượng vào bên trong bảng, khi đối tượng đó được tạo ra.

- tcl.lookup(char* s) sẽ tìm lại được TclObject với tên s.
  Nó được dùng bởi TclObject::lookup().

- _ tcl.remove(TclObject* o) sẽ xoá các quan hệ với TclObject o từ bảng hash.
  Nó được dùng bởi TclClass::delete_shadow() để xoá một entry đang tồn tại từ bảng hash, khi đối tượng đó bị xoá.

Các hàm này được dùng bên trong bởi lớp TclObject và lớp TclClass.

1.3.6. Các hoạt động khác trên trình thông dịch

Nếu các phương pháp trên không đủ, thì chúng ta phải nắm được trình thông dịch và viết ra các hàm riêng cho mình.

- tcl.interp(void) trả về các hàm bên trong trình thông dịch mà được lưu trữ trong lớp Tcl.

1.4. Lớp TclObject

class TclObject là lớp cơ bản cho hầu hết các lớp khác trong các nhánh thông dịch và biên dịch. Mọi đối tượng trong lớp TclObject được tạo ra bởi người dùng từ bên trong bộ thông dịch. Một đối tượng phần bóng tương đương được tạo ra ở nhánh biên dịch. Hai đối tượng này có mối tương quan chặt chẽ với nhau. Lớp TclClass, mà sẽ được mô tả trong phần tiếp theo, chứa các cơ cấu để thực hiện quá trình bóng hoá (quá trình ánh xạ từ đối tượng ở nhánh thông dịch sang đối tượng ở nhwránh biên dịch).

Trong phần còn lại của tài liệu này, chúng ta sẽ thường xuyên gọi một đối tượng là một TclObject1. Bằng cách đó, ta xem xét một đối tượng cụ thể mà ở cả trọng lớp TclObject hoặc trong một lớp mà được tạo ra từ lớp TclObject. Nếu cần thiết, ta xác định xem đối tượng đó là một đối tượng trong bộ thông dịch hay là trong mã biên dịch. Trong những trường hợp như vậy, chúng ta sẽ sử dụng các thuật ngữ tương ứng là "đối tượng thông dịch", và "đối tượng biên dịch" để phân biệt hai loại đối tượng trên.

Những sự khác biệt từ ns v1 Không như ns v1, lớp TclObject[1] gộp các hàm trước đây của lớp NsObject. Vì thế, nó lưu trữ những liên kết biến giao diện (phần 1.4.2) mà dùng để nối các biến instance của Otcl trong đối tượng thông dịch với các biến thành viên C++ tương ứng trong đối tượng biên dịch. Sự ràng buộc này mạnh hơn trong ns v1 vì trong đó bất kỳ sự thay đổi các biến OTcl đều bị khoá, và các giá trị C++ và OTcl hiện tại đều được làm cho phù hợp sau mỗi lần truy cập thông qua bộ thông dịch. Sự đồng nhất đó được thực hiện thông qua lớp InstVar (phần 3.8). Khác nls v1, các đối tượng trong lớp TclObject cũng không còn được lưu như một danh sách liên kết toàn cục. Thay vào đó, chúng được lưu trữ trong một bảng hash thuộc lớp Tcl (phần 1.3.5).

Cấu hình tiêu biểu của một TclObject Ví dụ sau đây mô tả cấu hình của một tác tử SRM (class Agent/SRM/Adaptive).

```
set srm [new Agent/SRM/Adaptive]
\$srm set packetSize\_{} 1024
\$srm traffic-source \$s0
```

Theo quy ước trong ns, lớp Agent/SRM/Adaptive là một lớp con của Agent/SRM, là một lớp con của Agent, là một lớp con của TclObject. Nhánh lớp biên dịch tương ứng là ASRMAgent, được lấy từ SRMAgent, SRMAgent lại được lấy từ Agent, và cuối cùng là từ TclObject một cách tuần tự. Dòng đầu tiên của ví dụ trên cho thấy một TclObject được tạo ra (hoặc bị huỷ) như thế nào (phần 3.4.1); dòng tiếp theo

[1]Trong phiên bản gần đây nhất của ns và ns/tclcl, đối tượng này đã được đổi tên thành SplitObjefct, phản ánh một cách chính xác hơn sự bản chất của sự tồn tại của nó. Tuy nhiên, lúc này, chúng ta sẽ tiếp tục sử dụng thuật ngữ TclObject để chỉ những đối tượng này và lớp này

cấu hình một biến giới hạn (phần 3.4.2); và cuối cùng, dòng cuối mô tả đối tượng được thông dịch bằng cách gọi ra một phương thức C++ ngay khi nó đã là một thủ tục ví dụ.

1.4.1. Tạo và huỷ các TclObject

Khi người dùng tạo/huỷ một TclObject mới, sử dụng các thủ tục *new{}* và *delete{}*; các thủ tục này được định nghĩa trong tclcl/tcl-object.tcl. Chúng có thể được dùng để tạo hay huỷ các đối tượng trong tất cả các lớp, bao gồm các TclObjects[2] Trong phần này,chúng ta sẽ mô tả các hoạt động bên trong được thực thi khi một TclObject được tạo ra.

Tạo các TclObject bằng cách dùng *new{}*, người dùng tạo một TclObject đã được thông dịch. bộ thông dịch xây dựng cấu trúc cho đối tượng đó, *init{}*, các đối số của phần này được cung cấp bởi người dùng. ns chịu trách nhiệm tạo ra đối tượng đã biên dịch một cách tự động. Đối tượng bóng được tạo ra bởi lớp cơ bản của hàm tạo TclObject. Vì thế, hàm tạo cho một TclObject mới phải gọi hàm tạo lớp nguồn của nó trước. *new{}* trả về một chức danh cho đối tượng, mà có thể được dùng cho các hoạt động sau này trên đối tượng đó.

Ví dụ sau đây minh họa hàm tạo Agent/SRM/Adaptive:

```
Agent/SRM/Adaptive instproc init args \{
    eval \$self next \$args
   \$self array set closest\_{} "requestor 0 repairor 0"
   \$self set eps\_{} [\$class set eps\_{}]
\}
```

Chuỗi các hoạt động sau đây được thực hiện bởi trình thông dịch như một phần của việc thể hiện một TclObject mới . Để cho dễ theo dõi, chúng ta sẽ mô tả từng bước để tạo nên một đối tượng Agent/SRM/Adaptive. Các bước đó là:

1. Tạo được một tên duy nhất cho một đối tượng mới từ không gian tên TclObject. Tên này được trả về cho người dùng. Hầu hết tên trong *ns* đều có dạng_<NNN>, với <NNN> là một số nguyên. Tên này được tạo bởi *getid{}*. Nó có thể được tìm thấy từ C++ bằng phương thức *name(){}*.

2. Tạo hàm tạo cho một đối tượng mới. Bất kỳ các đối số được quy định bởi người dùng nào cũng được truyền tới hàm tạo này. Hàm tạo này phải gọi ra hàm tạo có liên kết với lớp nguồn của nó.

[2]Như ví dụ, Các lớp, Simulator, Node, Link, hay rtObject, là các lớp mà không xuất phát từ lớp TclObject. Các đối tượng trong những lớp này vì thế không phải là TclObjects. Tuy nhiên, một đối tượng Simulator, Node, Link, hoặc route cũng được tạo ra bằng cách sử dụng thủ tục new trong ns.

Trong ví dụ trên, Agent/SRM/Adaptive gọi lớp nguồn của nó ở dòng chính đầu tiên.

Lưu ý rằng mỗi hàm tạo đó lần lượt gọi ra hàm tạo của lớp nguồn của nó ad *nauseum*. Hàm tạo cuối cùng trong ns là hàm tạo TclObject. Hàm tạo này chịu trách nhiệm thiết lập đối tượng bóng, và thực hiện những việc khởi tạo thao và các ràng buộc khác, như chúng tôi giải thích bên dưới. Tốt hơn là nên gọi các cấu trúc nguồn trước khi thực hiện việc nạp các thông số được yêu cầu trong lớp này. Điều này cho phép đối tượng bóng được thiết lập, và các ràng buộc biến được thiết lập.

3. Hàm tạo TclObject gọi ra thủ tục ví dụ *create-shadow{}* cho lớp Agent/SRM/Adaptive.

4. Khi đối tượng bóng được tạo ra, ns gọi tất cả các bộ câu trúc cho đối tượng đã biên dịch, mỗi bộ như vậy có thể thiết lập các ràng buộc biến cho các đối tượng trong lớp đó, và thực hiện các khởi tạo cần thiết khác. Do đó lệnh huấn thị trước đây của chúng ta phù hợp hơn để gọi ra các hàm tạo nguồn trước khi thực hiện việc khởi tạo lớp.

5. Sau khi đối tượng bóng được tạo ra thành công bằng lệnh *create_ shadow(void)*

   (a) Đối tượng mới vào bảng hash của các TclObject đã được mô tả trước đây (phần 1.3.5).

   (b) Thực hiện *cmd{}*, một thủ tục của đối tượng thông dịch mới được tạo. Thủ tục ví dụ này gọi ra phương thức *command()* của đối tượng biên dịch. Trong một phần nhỏ sau (Section 1.4.4), chúng tôi sẽ mô tả cách thức command được định nghĩa , và được gọi ra như thế nào.

Lưu ý rằng tất cả các cơ chế bóng hoá trên chỉ hoạt động khi người dùng tạo một TclObject mới thông qua trình thông dịch. Nó sẽ không làm việc nếu người lập trình tạo một TclObject được biên dịch một cách đơn phương. Vì thế, người lập trình bị cấm không dùng phương thức mới C++ để tạo các đối tượng biên dịch một cách trực tiếp.

Xoá các TclObject: lệnh *delete* huỷ đối tượng thông dịch, và đối tượng bóng tương ứng.Ví dụ, use-scheduler{hscheduleri} dùng thủ tục *delete* để xoá bỏ bộ lập lịch trình bảng kê mặc định, và tạo ra một bộ lịch trình thay thế trong không gian của nó.

```
Simulator instproc use-scheduler type {
        \$self instvar scheduler\_{}
        delete scheduler\_{} ;       # xoá bộ lịch trình bảng
                                      kê đang tồn tại trước
        set scheduler\_{} [new Scheduler/\$type]
}
```

1.4.2. Sự ràng buộc biến

Trong hầu hết các trường hợp, việc truy cập đến các biến thành viên biên dịch bị giới hạn tới mã biên dịch, và truy cập đến các biến thành phần đã được thông dịch cũng bị hạn chế, phải truy nhập thông qua mã thông dịch; tuy nhiên, có thể thiết lập các ràng buộc hai chiều như cả hai biến thành phần đã thông dịch và biến thành phần đã biên dịch, truy cập cùng dữ liệu, và việc thay đổi giá trị của cả hai biến có thể thay đổi giá trị của biến được cặp đôi tương ứng sang cùng một giá trị.

Sự ràng buộc được tạo ra bởi hàm tạo được biên dịch khi đối tượng đó được tạo; có thể truy cập một cách tự động bởi đối tượng được thông dịch như một biến instanc. ns hổ trợ năm kiểu dữ liệu khác nhau: các giá trị thực, các biến định trị băng thông, các biến định trị thời gian, các số nguyên, và các số nhị phân. Cú pháp để mô tả các giá trị đó được đặc tả trong OTcl là khác nhau ứng với mỗi kiểu.

- Các biến giá trị thực và số nguyên được đặc tả theo dạng "bình thường" . Ví dụ,
  ```
  $object set realvar 1.2e3
  $object set intvar 12
  ```

- Băng thông được đặc tả như một giá trị thực, được điền thêm tiền tố là một 'k' hay 'K' để chỉ giá trị hàng ngàn, hoặc 'm' hay 'M' để chỉ giá trị hàng triệu. Một ký tự 'B' sau cùng thể hiện giá trị Byte trên một giây. Theo mặc định, băng thông được thể hiện ở dạng bps. Ví dụ, tất các các dòng lệnh sau là tương đương:
  ```
  $object set bwvar 1.5m
  $object set bwvar 1.5mb
  $object set bwvar 1500k
  $object set bwvar 1500kb
  $object set bwvar .1875MB
  $object set bwvar 187.5kB
  $object set bwvar 1.5e6
  $object set bwvar 1500kb
  $object set bwvar .1875MB
  $object set bwvar 187.5kB
  $object set bwvar 1.5e6
  ```

- Thời gian được quy định như là một giá trị thực,có thể đi kèm với hậu tố m để thể hiện đơn vị mili giây, và 'n' nếu là nano giây, hoặc là pico giây với 'p'. Giá trị mặc định là giây (s). Ví dụ, tất cả các giá trị sau là tương đương nhau:
  ```
  $object set timevar 1500m
  $object set timevar 1.5
  $object set timevar 1.5e9n
  ```

```
$object set timevar 1500e9p
```
Lưu ý rằng ta có cũng thể thêm 1 chữ s để biểu thị đơn vị là giây. ns w sẽ bỏ qua tất cả các giá trị khác ngoài các con số, hoặc kết thúc bằng chữ 'm', 'n', hay 'p'.

- Các giá trị nhị phân có thể được biểu diễn ở cả hai dạng như một số nguyên, hay như 'T' hoặc 't' với giá trị đúng. Các ký tự theo sau chữ cái đầu tiên được bỏ qua. Nếu giá trị không phải là một số nguyên hay một giá trị đúng thì đương nhiên được xem là sai. Ví dụ,

```
$object set boolvar t ;            # đặt giá trị là đúng
$object set boolvar true
$object set boolvar 1 ;           # hay bất kỳ giá trị không bằng không
nào $object set boolvar false ;      # đặt giá trị là sai
$object set boolvar junk
$object set boolvar 0
```

Các ví dụ sau đây cho thấy hàm tạo của ASRMAgent[3]

```
ASRMAgent::ASRMAgent() {
bind("pdistance_", &pdistance_); /* biến giá trị thực */
bind("requestor_", &requestor_); /* ibiến giá trị nguyên */
bind_time("lastSent_", &lastSessSent_); /* biến giá trị thời gian
*/
bind_bw("ctrlLimit_", &ctrlBWLimit_); /* biến giá trị băng thông
*/
bind_bool("running_", &running_); /* biến giá trị nhị phân */
```

Chú ý rằng tất cả các hàm trên đều có hai đối số, tên của một biến OTcl, và địa chỉ của biến thành viên được biên dịch tương ứng được liên kết lại. Trong khi thường có trường hợp là các ràng buộc được thiết lập bởi hàm tạo của đối tượng, không cần phải luôn được thực hiện theo kiểu này. Ta sẽ thảo luận các phương pháp thay thế khi đến mục mô tả lớp InstVar (phần 3.8) một cách chi tiết ở phần sau.

Mỗi biến có ràng buộc sẽ tự động được khởi tạo với giá trị mặc định khi đối tượng được tạo ra. Các giá trị mặc định này được đặc tả như các biến lớp được thông dịch. Quá trình khởi tạo này được thực hiện bởi tuyến *init-instvar{}*, được gọi ra bởi các phương pháp trong lớp Insvar, sẽ được mô tả sau (phần 1.8). hàm *init-instvar{}* kiểm tra lớp đối tượng thông dịch, và tất cả các lớp nguồn của đối tượng đó, để tìm kiếm lớp đầu tiên mà các biến được định nghĩa. Nó dùng giá trị của biến trong lớp đó để khởi tạo đối tượng đó. Hầu hết các giá trị khởi tạo liên quan được định nghĩa trong ns/tcl/lib/ns-default.tcl. Ví dụ, nếu các biến lớp sau đây được định nghĩa cho ASRMAgent:

[3]Lưu ý rằng hàm tạo này được tô điểm thêm để mô tả các đặc điểm của cơ chế ràng buộc biến

```
Agent/SRM/Adaptive set pdistance_ 15.0
Agent/SRM set pdistance_ 10.0
Agent/SRM set lastSent_ 8.345m
Agent set ctrlLimit_ 1.44M
Agent/SRM/Adaptive set running_ f
Agent/SRM/Adaptive set pdistance_ 15.0
Agent/SRM set pdistance_ 10.0
Agent/SRM set lastSent_ 8.345m
Agent set ctrlLimit_ 1.44M
Agent/SRM/Adaptive set running_ f
```

Do đó, mọi đối tượng Agent/SRM/Adaptive mới sẽ có tham số `pdistance_` với giá trị đặt là 15.0; `lastSent_` là 8.345m từ việc cài đặt biến lớp của lớp nguồn; `ctrlLimit_` có giá trị là 1.44M sử dụng biến lớp của lớp nguồn hai lần đã bị xoá; running được đặt là sai; biến trường hợp pdistance_ không được gán tham số đầu vào, vì không có biến lớp tồn tại trong bất kỳ hệ thống phân cấp lớp của đối tượng thông dịch. Trong những trường hợp như vậy, `init-instvar{}` sẽ gọi ra `warn-instvar{}`, để in ra một cảnh báo về một biến chẳng hạn. Người dùng có thể bỏ qua một cách có chọn lọc thủ tục này trong các bản mô tả mô phỏng, để lướt qua cảnh báo này.

Chú ý rằng sự ràng buộc thực tế được thực hiện băng việc tạo các đối tương trong lớp InstVar. Mỗi đối tượng trong lớp InstVar ràng buộc một biến thành viên biên dịch với một biến thành viên thông dịch. Một TclObject lưu trữ một danh sách các đối tượng InstVar tương ứng với mỗi biến thành viên của nó mà bị ràng buộc trong kiểu này. Phần đầu của danh sách được lưu trong biến thành viên của nó `instvar_` của TclObject.

Điểm cuối cùng cần xem xét là `NS` sẽ bảo đảm các giá trị thực của biến, cả trong đối tượng thông dịch và đối tượng biên dịch, sẽ giống nhau mọi lúc. Tuy nhiên, nếu các phương thức và các biến khác của compiled object mà bám theo giá trị của biến đó, chúng phải được gọi ra rõ ràng hoặc được thay đổi bất kỳ khi nào giá trị của biến đó bị thay đổi. Điều này thường đòi hỏi các từ gốc mà người dùng cần biết. Có một cách để cung cấp các từ gốc trong `NS` là thông qua phương thức `command()` sẽ được mô tả trong phần tiếp theo.

1.4.3. Bám vết biến

Thêm vào các ràng buộc biến, TclObject còn hổ trợ bám vết của các biến trường hợp trong C++ và Tcl. Một biến bám vết có thể được tạo và cấu hình trong C++ hay Tcl. Để tạo ra quá trình bám vết biến ứng với mức Tcl, biến phải được nhìn thấy trong Tcl, điều đó có nghĩa rằng nó phải là một biến trường hợp Tcl thuần túy hoặc biến C++/Tcl bị giới hạn . Thêm vào đó, đối tượng sở hữu biến bám vết cũng phải được yêu cầu để thiết lập quá trình bám vết sử dụng phương thức Tcl `trace` của

TclObject. Đối số đầu tiên trong phương thức `trace` phải là tên biến. Đối số thứ 2 (không bắt buộc) đặc tả đối tượng bám vết, chịu trách nhiệm bám vết của biến đó. Nếu đối tượng vết không được đặc tả, thì đối tượng sở hữu biến đó chịu trách nhiệm bám vết nó.

Với một TclObject, để bám vết các biến, nó phải mở rộng phương thức bán mà nó được định nghĩa trong TclObject. Lớp bám vết thực hiện một phương thức `trace` đơn giản, vì thế, nó có thể hoạt động như một người bám vết chung cho tất cả các biến.

```
class Trace  public Connector {:
      ...
      virtual void trace(TracedVar*);
};
```

Dưới đây là một ví dụ đơn giản cho việc cài đặt quá trình bám vết biến trong Tcl:

```
\# \$tcp tracing its own variable cwnd\_
\$tcp trace cwnd_


\# the variable ssthresh_ of \$tcp is traced by a generic \$tracer
set tracer [new Trace/Var]
\$tcp trace ssthresh_ \$tracer
```

Đối với một biến C++ để có thể bám vết được, nó phải thuộc về một lớp mà bắt nguồn từ TracedVar. Lớp cơ sở ảo TracedVar lưu trữ tên biến, người sở hữu, và các bộ bám vết. Các lớp mà bắt nguồn từ TracedVar sẽ phải cung cấp phương thức ảo value, phương thức mà lấy bộ đệm ký tự như là một đối số và ghi giá trị của biến vào đó.

```
class TracedVar {
      ...
      virtual char* value(char* buf) = 0;
protected:
      TracedVar(const char* name);
      const char* name_; // name of the variable
      TclObject* owner_; // the object that owns this variable
      TclObject* tracer_; // callback when the variable is changed
      ...
};
```

Thư viện TclCL xuất ra 2 lớp của TracedVar là: `TracedInt` và `TracedDouble`. Các lớp này có thể được dùng với các kiểu dữ liệu cơ bản của int và double tương ứng. Cả TracedInt và TracedDouble làm quá tải tất cả các hoạt động mà có thể thay

đổi giá trị của biến như phép gán, tăng và giảm. Các hoạt động bị quá tải này dùng phương thức `assign` gán giá trị mới cho biến và gọi bộ bám vết nếu giá trị mới khác với giá trị cũ trước đó. TracedInt và TracedDouble cũng bổ sung các phương thức về `value` của chúng mà đưa các giá trị biến vào trong chuỗi. Độ rộng và tính chính xác của đầu ra có thể được đặc tả trước.

1.4.4. Các phương thức command: Định nghĩa và gọi ra

Đối với mỗi TclObject được tạo ra, ns thiết lập thủ tục trường hợp, `cmd{}`, như là một bản lề để thực thi các phương thức thông qua đối tượng bản sao đã được biên dịch. Thủ tục `cmd{}` gọi ra phương thức `command()` của đối tượng bóng một cách tự động, nhập các đối số cho `cmd{}` như là một vector đối số với phương thức `command()`.

Người dùng có thể gọi phương thức `cmd{}` bằng một trong 2 cách sau: gọi ra một cách rõ ràng thủ tục, đặc tả hoạt động mong muốn như là đối số đầu tiên, hoặc một cách rõ ràng , nếu đã có một thủ tục trường hợp của cùng tên đó như hoạt động mong muốn. Hầu hết các bản mô phỏng sẽ dùng kiểu sau này, vì thế, ta sẽ mô tả kiểu dẫn chứng kia trước.

Cho rằng tính toán khoảng cách trong SRM đã được thực hiện bởi đối tượng biên dịch; tuy nhiên, nó lại thường được sử dụng bởi đối tượng thông dịch và thường được gọi ra như sau:

```
$srmObject distance? <agentAddress>
```

Nếu không có thủ tục nào gọi là distance, trình thông dịch sẽ gọi ra thủ tục unknown, được định nghĩa trong TclObject lớp cơ bản. Thủ tục unknown sau đó sẽ gọi ra

```
$srmObject cmd distance? <agentAddress>
```

để thực hiện hoạt động thông qua thủ tục command() của đối tượng biên dịch.

Dĩ nhiên, người dùng có thể gọi hoạt động này một cách trực tiếp. Một lý do cho điều này là để có thể làm quá tải hoạt động đó bằng cách sử dụng một thủ tục trùng tên. Ví dụ,

```
Agent/SRM/Adaptive instproc distance? addr {
   $self instvar distanceCache_
   if ![info exists distanceCache_($addr)] {
       set distanceCache_($addr) [$self cmd distance? $addr]
   }
   set distanceCache_($addr)
}
```

Bây giờ ta sẽ mô tả cách mà phương thức `command()` sử dụng `ASRMAgent`:command(): như ví dụ sau:

```
int ASRMAgent:command(int argc, const char*const*argv) :
    Tcl& tcl = Tcl:instance();:
    if (argc == 3) {
        if (strcmp(argv[1], "distance?") == 0) {
            int sender = atoi(argv[2]);
            SRMinfo* sp = get_state(sender);
            tcl.tesultf("%f", sp->distance_);
            return TCL_OK;
         }
    }
    return (SRMAgent:command(argc, argv));:
}
```

Chúng ta có thể phân tích đoạn mã này như sau:

- Hàm được gọi với 2 đối số: Đối số đầu tiên (argc) biểu thị số lượng của các đối số được đặc tả trong dòng lệnh cho trình thông dịch. Vector các đối số dòng lệnh (argv) bao gồm
  - argv[0] chứa tên của phương thức, "cmd".
  - argv[1] đặc tả hoạt động mong muốn.
  - Nếu người dùng đã đặc tả bất kỳ đối số nào, thì chúng phải được đặt trong argv[2...(argc - 1)]. Các đối số được truyền như các chuỗi; chúng phải được chuyển đổi sang kiểu dữ liệu phù hợp.
- Nếu phép toán là đúng, thì nó sẽ trả về kết quả phép toán sử dụng các phương thức đã được mô tả trước đó (phần 1.3.3).
- Bản thân hàm command() phải trả về hoặc là TCL_OK hay TCL_ERROR để biểu thị thành công hay thất bại.
- Nếu phép toán trong phương thức này không đúng, nó phải gọi ra phương thức lệnh gốc, và trả về kết quả tương ứng. Điều này cho phép người dùng hình dung được các phép toán khi có được các thuộc tính kế thừa như là các thủ tục trường hợp, hoặc các phương pháp biên dịch. Trường hợp phương thức command này được định nghĩa cho một lớp với sự đa kế thừa, người lập trình có thể tự do chọn một trong hai thành phần bổ sung sau:
  1. Chúng có thể gọi ra một trong các phương thức command gốc, và trả về kết quả của sự gọi ra đó, hoặc là.
  2. Chúng có thể là một trong các phương thức command gốc ở một số chuỗi, và trả về kết quả của sự gọi ra đầu tiên thành công. Nếu không thành công, thì chúng sẽ trả về một lỗi.

Trong tài liệu này, chúng ta gọi các phép toán được thực thi thông qua `command()` là `instproc-likes`. Nó phản ảnh công dụng của những hoạt động này như thể chúng là các thủ tục trường hợp OTcl của một đối tượng, nhưng có thể sẽ rất khác trong sự nhận thức và sử dụng.

1.5. Lớp TclClass

Lớp biên dịch này `(class TclClass)` là một lớp hoàn toàn ảo. Các lớp đã thu được từ lớp cơ sở này cung cấp 2 chức năng: xây dựng nên nhánh lớp thông dịch để phản ánh nhánh lớp biên dịch; và cung cấp các phương thức để tạo ra các TclObject mới. Mỗi lớp thu được như vậy kết hợp với một lớp biên dịch riêng biệt trong nhánh lớp biên dịch, và có thể tạo các đối tượng mới trong lớp kết hợp.

Ví dụ, xét một lớp như `RenoTcpClass`. Nó được lấy ra từ lớp `TclClass`, và được kết hợp với lớp RenoTcpAgent. Nó sẽ dẫn ra các đối tượng mới trong lớp `RenoTcpAgent`. Nhánh lớp biên dịch cho `RenoTcpAgent` là lớp lần lượt được tạo ra từ `TcpAgent`, TcpAgent: lại được tạo ra từ `Agent`, Agent lại được tạo ra từ TclObject. `RenoTcpClass` được định nghĩa như sau

```
static class RenoTcpClass: public TclClass {
public:
RenoTcpClass() : TclClass("Agent/TCP/Reno") {}
TclObject* create(int argc, const char*const* argv) {
return (new RenoTcpAgent());
}
} class_reno;
```

Chúng ta có thể đưa ra một số phân tích từ định nghĩa trên:

1. Lớp này chỉ định nghĩa hàm tạo, và một phương thức thêm vào, để `create` các đối tượng của `TclObject` kếp hợp.

2. NS sẽ thực hiện hàm tạo hàm tạo `RenoTcpClass` cho biến tĩnh `class_reno}`, khi nó được bắt đầu lần đầu tiên. Việc này cài đặt các phương thức thích hợp và nhánh lớp phiên dịch.

3. Hàm tạo đặc tả lớp thông dịch một cách rõ ràng như `Agent/TCP/Reno`. Điều này cũng đặc tả nhánh lớp thông dịch một cách hoàn toàn.

   Gọi lại thoả thuận kia trong ns is là để dùng ký tự gạch chéo `('/')` như là cách phân chia. Với bất kỳ lớp được cho là `A/B/C/D`, thì lớp `A/B/C/D` là 1 lớp con

của A/B/C, A/B/C là lớp con của A/B, lần lượt A/B là lớp con của A. A chính nó lại là lớp con của TclObject.

Trong trường hợp trên, hàm tạo TclClass tạo ra 3 lớp, Agent/TCP/Reno là lớp con của Agent/TCP là lớp con của Agent là lớp con của TclObject.

4. Lớp này kếp hợp với lớp RenoTcpAgent; nó tạo ra các đối tượng mới trong lớp được kết hợp này.

5. Phương thức RenoTcpClass::create trả các TclObject về trong lớp RenoTcpAgent.

6. Khi người dùng đặc tả new Agent/TCP/Reno, thủ tục tiếp theo RenoTcpClass::create được gọi ra

7. Vector các đối số (argv) bao gồm
– argv[0] chứa tên của đối tượng.
– argv[1...3] chứa $self, $class, và $proc. Kể từ khi create is được gọi thông qua thủ tục trường hợp create-shadow, argv[3] bao gồm create-shadow. – argv[4] chứa bất kỳ đối số bổ sung nào (được xem như là một chuỗi) được cung cấp bởi người dùng.

class Trace minh hoạ việc trình bày đối số bằng các phương thức TclClass.

```
class TraceClass : public TclClass \{
public:\tab
     {}TraceClass() : TclClass("Trace") \{\}
     {}TclObject* create(int args, const char*const* argv) \{
     {}if (args >= 5)\untab
              {}return (new Trace(*argv[4]));
     {}else
         {}return NULL;
      {}\}
{}\}  trace\_{}class;
```

Một đối tượng Trace được tạo ra như

```
new Trace "X"
```

Cuối cùng, về cơ bản, cách mà nhánh lớp thông dịch được tạo nên như sau:

1. Hàm tạo đối tượng được thực hiện khi ns khởi động lần đầu.

2. Hàm tạo này gọi hàm tạo TclClass với cái tên của lớp được thông dịch cũng như đối số của nó.

3. Hàm tạo TclClass lưu trữ tên của lớp, và chèn đối tượng này vào một bảng kê liên kết của các đối tượng TclClass.

4. Trong suốt quá trình khởi tạo của bộ mô phỏng, hàm `Tcl_AppInit(void)` gọi ra `TclClass::bind(void)`

5. Với mỗi đối tượng trong bảng kê các đối tượng `TclClass`, hàm bind() gọi ra `register{}`, bằng cách đặc tả tên của lớp thông dịch như đối số của nó.

6. Hàm `register{}` thiết lập lớp nhánh, tạo các lớp được yêu cầu, và chưa được tạo ra.

7. Cuối cùng, hàm `bind()` định nghĩa các thủ tục đối tượng `creat-shadow` và `delete-shadow` cho chính lớp mới này.

1.5.1. Làm thế nào để ràng buộc các biến thành viên lớp C++ tĩnh

Trong phần 1.4, ta đã thấy cách phơi bày các biến thành viên của một đối tượng C++ sang không gian OTcl. Tuy nhiên, điều này không áp dụng cho các biến thành viên tĩnh của một lớp C++. Dĩ nhiên, có thể tạo ra một biến OTcl cho biến thành viên tĩnh của mọi đối tượng C++; một cách hiển nhiên điều này làm mất hoàn toàn ý nghĩa của các biến thành viên tĩnh.

Chúng ta không thể giải quyết vấn đề ràng buộc bằng cách sử dụng giải pháp tương tự như việc ràng buộc trong TclObject dựa trên lớp InstVar, bởi vì lớp InstVar trong TclCL yêu cầu sự hiện hữu của một TclObject. Tuy nhiên, ta có thể tạo một phương thức trong lớp TclClass tương ứng và truy cập các thành viên tĩnh của một lớp C++ thông qua các phương thức TclObject tương ứng của nó. Thủ tục ấy như sau:

1. Tạo TclClass của chính bạn như đã mô tả ở trên;

2. Khai báo các phương thức `bind()` và `method()` trong lớp do bạn tạo ra;

3. Tạo các phương thức ràng buộc của bạn trong hàm `bind()` của bạn với hàm `add_method("your_method")`, và sau đó bổ sung biến điều khiển trong `method()` bằng cách tương tự như bạn sẽ làm trong `TclObject::command()`. Lưu ý, số lượng biến hợp quy cách với `TclClass::method()` là khác so với `TclObject::command()`. Người tạo có nhiều hơn 2 đối số trước.

Như ví dụ sau, chúng ta biểu diễn một phiên bản đơn giản `PacketHeaderClass` trong `~ns/packet.cc`. Giả sử rằng chúng ta có lớp `Packet` sau đây có chứa một biến tĩnh `hdrlen_` mà ta muốn truy cập từ OTcl:

```
class Packet {
......
```

```
    static int hdrlen_;
    };
```

Sau đó ta làm tiếp theo như bên dưới để xây dựng một bộ truy cập cho biến này:

```
class PacketHeaderClass:public TclClass {
protected:
        PacketHeaderClass(const char* classname, int hdrsize);
        TclObject* create(int argc, const char*const* argv);
              /* Hai thành phần của các phương thức truy nhập
              trong lớp Tcl */
        virtual void bind();
        virtual int method(int argc, const char*const* argv);
};
void PacketHeaderClass::bind()
{
               /*Gọi đến lớp cơ bản bind() phải làm trước
               add_method()*/
        TclClass::bind();
        add_method("hdrlen");
{
int PacketHeaderClass::method(int ac, const char*const* av)
{
        Tcl& tcl = Tcl::instance();
              /*Lưu ý việc chuyển đổi đối số này; ta có thể sau đó
              đối xử với chúng như nếu là trong TclObject::command()*/
        int argc = ac - 2;
        const char*const* argv = av  2;+
        if (argc == 2) {
           if (strcmp(argv[1], "hdrlen") == 0) {
              tcl.resultf("%d", Packet::hdrlen_);
              return (TCL_OK);
           }
        } else if (argc == 3) {
           if (strcmp(argv[1], "hdrlen") == 0) {
              Packet::hdrlen_ = atoi(argv[2]);
              return (TCL_OK);
           }
        }
        return TclClass::method(ac, av);
{
```

Sau bước này, ta có thể dùng lệnh OTcl sau để truy cập và thay đổi các giá trị của biến `Packet::hdrlen_`:

```
PacketHeader hdrlen 120
set i [PacketHeader hdrlen]
```

1.6. Lớp Tcl Command

Lớp này (`class TclCommand`) chỉ cung cấp cơ chế cho `NS` to để xuất ra các lệnh đơn giản cho trình thông dịch, mà sau đó có thể được thực thi trong một bối cảnh toàn cục bởi trình thông dịch. Có hai hàm được định nghĩa trong file `~ns/misc.cc` là: `ns-random` và `ns-version`. Hai hàm này được khởi tạo bởi hàm `init_misc(void)`, được định nghĩa trong `~ns/misc.cc; init_misc` và được gọi ra bởi `Tcl_AppInit(void)`
trong suốt quá trình khởi động.

- `class VersionCommand` định nghĩa lệnh `ns-version`. Nó không cần đối số, và trả về phiên bản `NS` hiện hành.

  ```
  % ns-version ;      # lấy phiên bản hiện hành
  2.0a12
  ```

- `class RandomCommand` định nghĩa lệnh `ns-random`. Không cần đối số, `ns-random returns` trả về một số nguyên, và được phân bố đều trong khoảng $[0, 2^{31} - 1]$.

  Khi đã đặc tả một đối số, nó lấy đối số đó như hạt giống. Nếu hạt giống này có giá trị là 0, lệnh dùng một giá trị hạt giống heuristic; hay nói cách khác, nó đặt hạt giống cho hàm tạo số ngẫu nhiên một giá trị cụ thể.

  ```
  % ns-random;         # trả về một số ngẫu nhiên
  2078917053
  % ns-random 0;       #set the seed  heuristically
  858190129
  % ns-random 23786;   #đặt hạt giống về giá trị được đặc tả
  23786
  ```

Lưu ý rằng, nói chung không nên xây dựng các lệnh ở mức cao cho người dùng. Bây giờ ta mô tả cách để định nghĩa một lệnh mới bằng cách sử dụng ví dụ `class say_hello`. Ví dụ định nghĩa lệnh hi, để in chuỗi "hello world", được theo sau bởi bất kỳ các đối số dòng lệnh nào được đặc tả bởi người dùng. Ví dụ

```
% hi this is ns [ns-version]
hello world, this is ns 2.0a12
```

1. Lệnh phải được định nghĩa bên trong một lớp được xuất phát từ class TclCommand. Việc định nghĩa lớp này như sau:

```
class say_hello : public TclCommand {
public:
say_hello();
int command(int argc, const char*const* argv);
};
```

2. Hàm tạo cho lớp phải gọi ra hàm tạo TclCommand với lệnh và đối số; v.v.,

   `say_hello() : TclCommand("hi") {}`

   Hàm tạo TclCommand đặt "hi" như là một thủ tục toàn cục mà gọi ra `TclCommand::dispatch_cmd()`

3. Phương thức `command()` phải thực hiện được việc mong muốn. Phương thức có 2 đối số, một là argc, chứa số lượng đối số thực tế được thông qua bởi người dùng. Các đối số thực tế được truyền vào bởi người sử dụng đều được xem như là một vector đối số (argv) và bao gồm:

   — `argv[0]` chứa tên của lệnh (hi).
   — `argv[1...(argc - 1)]` chứa các đối số khác được đặc tả trên dòng lệnh bởi người dùng. command() được gọi ra bởi `dispatch_cmd()`.

```
#include <streams.h>   /* bởi vì ta dùng dòng I/O */
int say_hello::command(int argc, const char*const* argv) {
cout << "hello world:";
for (int i = 1; i < argc; i++)
cout << ' ' << argv[i];
cout << 'n n';
return TCL_OK;
}
```

4. Cuối cùng, ta yêu cầu một đối tượng cho lớp này. Các đối tượng `TclCommand`được tạo ra trong thủ tục `init_misc(void)`. `new say_hello;`

Lưu ý rằng, ở đây thường dùng nhiều hàm hơn như ns-at và ns-now mà có thể truy cập được. Hầu hết các hàm này sẽ được gom vào trong một lớp hiện hành. Cụ thể là, ns-at và ns-now có thể truy cập được thông qua bộ lập lich thời gian của TclObject. Các hàm được định nghĩa trong ns/tcl/lib/ns-lib.tcl

```
% set ns [new Simulator] ;    # Lấy trường hợp của bộ mô phỏng
_o1
% $ns now ;      # truy vấn bộ mô phỏng về thời gian hiện tại
0
% $ns at ... ;   #  chỉ rõ các hoạt động cho bộ mô phỏng
```

1.7. Lớp EmbeddedTcl

NS cho phép sự phát triển của các tính năng trong cả mã biên dịch, hoặc thông qua mã thông dịch, mà được đánh giá trong quá trình khởi tạo. Ví dụ, các bản trong *~tclcl/tcl-object.tcl* hay trong *~ns/tcl/lib*. Việc tải và đánh giá các đoạn mã như vậy được thực hiện thông qua các đối tượng trong lớp `class EmbeddedTcl`.

Cách dễ nhất để mở rộng ns is là thêm mã OTcl vào hoặc *~tclcl/tcl-object.tcl* hoặc qua các đoạn mã ở thư mục ns/tcl/lib. Lưu ý rằng, trong những trường hợp sau này, ns tự động tạo *~ns/tcl/lib/ns-lib.tcl*, và vì thế người lập trình phải thêm hai dòng lệnh vào file này để bản dịch của chúng sẽ tự động lấy nguồn từ ns khi khởi động. Ví dụ, file *~ns/tcl/mcast/srm.tcl* xác định một số thủ tục trường hợp để chạy SRM. Trong *~ns/tcl/lib/ns-lib.tcl*, chúng ta có các dòng lệnh:

```
source tcl/mcast/srm.tcl
```

để tự động lấy srm.tcl được tạo bởi *NS* khi khởi động.

Có ba điểm cần lưu ý với mã EmbeddedTcl đó là: đầu tiên nếu mã có một lỗi được tạo ra trong quá trình đánh giá thì ns sẽ không chạy. Thứ hai, người dùng có thể không cần để ý đến bất kỳ mã nào trong các bản dịch. Đặc biệt, chúng có thể tái tạo toàn bộ đoạn mã sau khi sau khi đã có sự thay đổi nội tại. Cuối cùng, sau khi thêm các bản dịch cho `ns/tcl/lib/ns-lib.tcl`, và từ đó về sau bất cứ khi nào đoạn mã thay đổi, người dùng phải biên dịch lại ns để làm sự thay đổi đó có hiệu lực. Dĩ nhiên, trong hầu hết trường hợp [4], người dùng chó thể tạo đoạn mã của mình để bỏ qua mã được nhúng.

Phần còn lại này sẽ mô tả cách để tích hợp các bản độc lập trực tiếp vào trong ns. Bước đầu tiên là chuyển đổi bản dịch vào trong một đối tượng `EmbeddedTcl`. Các dòng bên dưới mở rộng `ns-lib.tcl` và tạo ra đối tượng `EmbeddedTcl` được gọi là `et_ns_lib`:

```
tclsh bin/tcl-expand.tcl tcl/lib/ns-lib.tcl | n
                        ../Tcl/tcl2c++ et_ns_lib > gen/ns_tcl.cc
```

Đoạn mã, *~ns/bin/tcl-expand.tcl* mở rộng *ns-lib.tcl* bằng cách thay thể tất cả các dòng source lines bởi các file nguồn tương ứng. Chương trình, *~tclcl/tcl2cc.c*, chuyển đổi mã OTcl thành một đối tượng EmbeddedTcl tương đương, `et_ns_lib`.

Trong quá khởi tạo, việc gọi ra phương thức `EmbeddedTcl::load` sẽ đánh giá mảng đó một cách rõ ràng.

- *~tclcl/tcl-object.tcl* được định trị bởi phương thức `Tcl::init(void); Tcl_AppInit()` gọi ra `Tcl::Init()`. Cú pháp lệnh chíng xác cho tải là:

[4] Một ít trường hợp mà điều này có thể sẽ không xảy ra là khi các biến nào đó có thể phải được định nghĩa hay không được định nghĩa, hoặc nói cách khác bản dịch chứa mã khác hơn là thủ tục, và việc định nghĩa biến và thực thi các hoạt động một cách trực tiếp là không thể đảo ngược được

```
et_tclobject.load();
```

- Một cách tương tự, *~ns/tcl/lib/ns-lib.tcl* được định trị trực tiếp bởi `Tcl_AppInit` trong *~ns/ns_tclsh.cc*.

```
et_ns_lib.load();
```

1.8. Lớp InstVar

Phần này mô tả các bản chất của lớp InstVar. Lớp này xác định các phương thức và cơ cấu để ràng buộc một biến thành viên C++ trong đối tượng bóng biên dịch với một biến trường hợp OTcl được đặc tả trong đối tượng thông dịch tương đương. Việc ràng buộc được thiết lập để giá trị của biến có thể được đặt hoặc được truy cập từ cả trình thông dịch hoặc từ mã biên dịch ở mọi lúc.

Có năm lớp biến đối tượng: `class InstVarReal, class InstVarTime, class InstVarBandwidth, class InstVarInt, và class InstVarBool,` tương ứng với các ràng buộc cho các biến giá trị thực, thời gian, băng thông, số nguyên, và nhị phân một cách lần lượt.

Bây giờ, ta mô tả cơ cấu cài đặt các biến trường hợp. Ta dùng class InstVarReal to để diễn giãi khái niệm. Tuy nhiên, cơ cấu này có thể ứng dụng được với cả 5 kiểu biến trường hợp. Khi việc cài đặt một biến thông dịch để truy cập một biến thành viên, các hàm thành viên của lớp InstVar giả sử rằng chúng đang thực hiện các phương thức thích hợp, do đó chúng sẽ không truy vấn trình thông dịch để xác định bối cảnh mà biến đó phải tồn tại.

Để bảo đảm hoàn cảnh thực thi phương thức chính xác, một biến phải được ràng buộc nếu lớp của nó đã được thiết lập trong bộ trình thông dịch, và trình thông dịch đang làm việc trên một đối tượng của lớp đó. Để ý rằng bộ định dạng yêu cầu khi một phương thức trong một lớp cho sẵn sẽ làm các biến của nó có thể truy cập thông qua trình thông dịch, đó phải là một lớp TclClass (phần 1.5) liên kết đã được định nghĩa để nhận dạng nhánh lớp phù hợp cho trình thông dịch. Vì thế, bối cảnh thực thi phương thức phù hợp, có thể được tạo ra bằng một trong 2 cách.

Một giải pháp ẩn xuất hiện bất cứ khi nào một TclObject được tạo ra bên trong trình thông dich. Điều này thiết lập bối cảnh thực thi phương thức trong trình thông dịch. Khi đối tượng bóng biên dịch của TclObject thông dịch được tạo ra, hàm tạo cho đối tượng biên dịch đó có thể ràng buộc các biến thành viên của đối tượng đó với các biến đối tượng thông dịch trong bối cảnh của đối tượng thông dịch mới được tạo .

Một giải pháp nổi để định nghĩa một phép toán `bind-variables` trong một hàm `command`, mà có thể được gọi ra thông qua phương thức cmd. Bối cảnh thực thi phương thức đúng được thiết lập để thực hiện phương thức cmd. Tương tự, mã biên

dịch bây giờ sẽ hoạt động trên đối tượng bóng thích hợp, và vì thế có thể ràng buộc chắc chắn cấc biến thành viên được yêu cầu.

Một biến đối tượng được tạo bởi việc đặc tả tên của biến thông dịch, và địa chỉ của biến thành viên trong đối tượng biên dịch. Hàm tạo cho lớp cơ sở InstVar tạo ra một trường hợp của biến đó trong trình thông dịch, và sau đó thiết lập một thủ tục bẫy để bắt tất cả các truy cập đến biến thông qua trình thông dịch.

Mỗi khi biến được đọc thông qua trình thôn dịch, thủ tục bẫy được gọi ra trước ngay khi có thao tác đọc. Thủ tục gọi ra hàm get thích hợp tương ứng sẽ trả về giá trị hiện tại của biến. Giá trị này sau đó được dùng để đặt giá trị của biến thông dịch và sau đó được đọc bởi trình thông dịch.

Tương tự, mỗi khi biến được lập thông qua trình thông dịch, thủ tục bẫy được gọi ra chỉ sau khi việc ghi hoàn tất. Thủ tục lấy giá trị hiện tại được lập bởi trình thông dịch, và gọi ra hàm set tương ứng chịu trách nhiệm lập giá trị của thành viên biên dịch đến giá trị hiện tại được lập bên trong trình thông dịch.

Chương 2

Cơ bản về TCL và OTCL

Mục lục

Người dịch: *Hà Tất Thành*
Biên tập: *Lê Tiến Anh.*
Bản gốc: *NS Simulator for beginners, Chapter 1* [2]

2.1. Tổng quan về NS

Bộ mô phỏng NS dựa trên hai ngôn ngữ: một bộ mô phỏng hướng đối tượng, được viết bằng C++, và một bộ thông dịch OTcl (phần mở rộng hướng đối tượng của Tcl), được dùng để thực hiện các đoạn mã kịch bản (script) của người dùng.

NS có một thư viện đồ sộ các đối tượng giao thức và mạng. Có 2 nhánh các lớp trong NS: nhánh gồm mã ngôn ngữ C++ đã được biên dịch và nhánh gồm mã lệnh OtcL đã đươc thông dịch, giữa chúng có sự tương ứng 1-1.

Nhánh gồm mã ngôn ngữ C++ đã được biên dịch cho phép ta đạt hiệu quả mô phỏng với thời gian thực hiện nhanh hơn. Điều này đặc biệt hữu ích cho việc định nghĩa và mô phỏng hoạt động một cách thật chi tiết của các giao thức cụ thể, cho phép ta giảm thời gian xử lí sự kiện và gói tin.

Sau đó, ở đoạn mã Otcl được nhập bởi người dùng, ta có thể định nghĩa một cấu trúc (topo) mạng cụ thể, các giao thức và ứng dụng riêng biệt mà ta muốn mô phỏng (tập tính của chúng được định nghĩa ở bên nhánh mã ngôn ngữ C++ đã được biên

dịch) và dạng của dữ liêu đầu ra muốn nhận được từ bộ mô phỏng. Otcl có thể tận dụng các đối tượng đã được biên dịch trong C++ thông qua một liên kết Otcl (được thực hiện bằng việc sử dụng TclCL[1])[5] , liên kết Otcl này tạo ra một ánh xạ giữa đối tượng Otcl với mỗi đối tượng trong C++.

NS là một bộ mô phỏng sự kiện rời rạc, trong đó, độ tăng của thời gian phụ thuộc vào việc định thời của các sự kiện, độ tăng này được duy trì với một bộ định trình. Một sự kiện là một đối tượng trong nhánh C++ với một ID duy nhất, một thời gian định trước và con trỏ trỏ tới đối tượng sử dụng sự kiện đó. Bộ định trình nắm quyền quản lý một cấu trúc dữ liệu có thứ tự (có 4 cấu trúc, nhưng theo mặc định, NS sử dụng một danh sách liên kết đơn giản) cùng với các sự kiện chuẩn bị được thực hiện, và thực thi từng sự kiện một bằng cách kích hoạt thẻ (handle) của sự kiện đó.

2.2. Lập trình Tcl và Otcl

- Trong Tcl, các biến không được định kiểu, vì vậy, một biến có thể là kiểu chuỗi hoặc kiểu số nguyên tùy thuộc vào giá trị mà bạn gán cho nó. Ví dụ, khi ta muốn in kết quả của phép chia 1/60, nếu ta viết: `puts "[expr 1/60]"`, thì kết quả sẽ là `0`! Để có được kết quả chính xác, ta cần chỉ ra rằng ta không làm việc với số nguyên, và cần nhập như sau: `put"[expr 1.0/60.0]"`

- Dấu # bắt đầu một dòng chú thích không nằm trong chương trình, trình thông dịch tcl sẽ bỏ qua dòng này.

- Để tạo một file, ta phải cho nó một cái tên, giả sử là "filename", và gán một con trỏ tới nó, con trỏ này sẽ được dùng trong chương trình tcl để liên kết với file đó, giả sử là `"file1"`. Việc này được thực hiện bởi câu lệnh:

  `set file1 [open filename w]`.

- Lệnh puts được dùng để in kết quả đầu ra. Chú ý: mỗi lần lệnh puts được dùng, một dòng mới được bắt đầu. Để tránh mở dòng mới, ta phải thêm hậu tố -nonewline sau lênh `"puts''`. Nếu muốn in kết quả ra file (giả sử là file ta đã định nghĩa ở trên), gõ: puts `$file1` "đoạn cần nhập". Khoảng nhẩy trống được tạo ra bằng cách thêm ⌢Ví dụ, nếu một biến, giả sử là x, có giá trị là 2 và ta nhập lệnh `puts $file1 "x $x"` sau đó nó sẽ in một lênh vào file có tên là filename, với hai thành phần "x" và "2" được ngăn cách bởi khoảng nhẩy trống.

- Thực thi một câu lệnh unix: gõ "exec" và câu lệnh. Ví dụ, ta muốn ns hiển thị một đường cong từ dữ liệu được ghi trong một file có 2 cột, với tên là "data"

[1]TCL web page[5]

trong phạm vi chương trình mô phỏng. Điều này có thể thực hiện được bằng cách dùng lệnh xgraph như sau:

```
Exec xgraph data &
```
(chú ý: dấu "&" được dùng để điều khiển cho lệnh thực hiện ngầm).

- Cấu trúc của một câu lệnh if như sau:

```
If  { biểu thức}{
   <khối lệnh>
   } else {
   {<khối lệnh>
}
```

Câu lệnh if có thể được bao trong các câu lệnh "if" khác và với các câu lệnh "else". Chúng có thể xuất hiện trong phần "<khối lệnh>''. Chú ý, khi kiểm tra điều kiện logic tương đương, ta nên dùng dấu "==" chứ không nên dùng dấu "=". Điều kiện logic bất tương đương có cú pháp là "!=".

- Các vòng lặp có dạng như sau:

```
for {set i 0}{$ i < 5}{incr i}{
   <khối câu lệnh>
}
```

Trong ví dụ này, khối câu lệnh trong vòng lặp sẽ được thực hiện 5 lần. Sau toán tử for, cụm "{set i 0} khai báo biến i là bộ đếm của vòng lặp đồng thời khởi tạo giá trị 0 cho nó. Phần thứ hai trong {} là điều kiện tiếp tục của vòng lặp, nó có nghĩa là "tiếp tục thực hiện vòng lặp nếu i<5. Phần cuối cùng của câu lệnh thể hiện sự thay đổi trong biến đếm, ở đây, ta liên tục tăng i thêm 1, nhưng ta cũng có thể giảm hoặc dùng bất kì một biểu thức toán học nào khác để thay đổi giá trị bộ đếm.

```
#Create a procedure+\\
proc test {} {+\\
set a 43
set b 27
set c [exrp $a + $b]
set d [expr [expr $a - $b] * $c]
puts "c = $c d = $d":\\
for{set k 0} {$k < 10} {incr k}{
    if {$k < 5} {
    puts "k>= 5, pod = [expr $d % $k]"
    } else {:\\
```

```
        puts "k >= 5, mod = [expr $d % $k]"
    }
    }
}

calling the procedure
test
```

Bảng 2.1: Chương trình Tcl thực hiện các phép toán

- tcl cho phép tạo các thủ tục. Chúng có thể trả về một vài giá trị trong trường hợp chúng có chứa lệnh "return". Cú pháp tổng quát của một thủ tục có tên ví dụ là ‘‘blue’’ như sau:

```
proc blue {par1 par2 ...} {
global var1 var 2
   <khối câu lệnh>
   Return $<một giá trị nào đó>
}
```

  Thủ tục nhận một vài tham số đầu vào, có thể là các đối tượng, file, hoặc các biến. Trong trường hợp của chúng ta, các biến được đặt tên là par1, par2, v.v.. Thủ tục được gọi ra bằng cách gõ: ‘‘blue x y ...’’ ở đây, giá trị của x và y sẽ được gán cho par1 và par2. Nếu par1 và par 2 bị thay đổi trong thủ tục, điều đó không ảnh hưởng gì tới x và y. Mặt khác, nếu ta muốn thủ tục có khả năng tác động trực tiếp tới các biến ở bên ngoài phạm vi của nó, ta phải định nghĩa các biến đó là biến toàn cục "global’’, ở ví dụ dưới đây là var 1 và var 2.

Ở bảng 2.1, ta thấy một ví dụ thể hiện rất nhiều toán tử số học trong tcl. Chỉ thị pow cho kết quả là hàm mũ bậc k của biến d (d^k).

Ở bảng 2.2, ta lấy ví dụ một chương trình tcl để tính tất cả số nguyên tố nhỏ hơn một giới hạn j cho trước. Ví dụ, để nhận được các số nguyên tố nhỏ hơn 11, gõ: ‘‘ns prime.tcl 11’’. Ví dụ về các số nguyên tố cho thấy cách sử dụng lệnh if, các vòng lặp và thủ tục. Biến argc có chứa các tham số được đưa vào chương trình. Biến argv là một vectơ chứa các tham số được truyền tới chương trình (vậy nên biến argc chính là độ dài của biến argv), và lệnh lindex cho phép chúng ta lấy trường hợp của vectơ được trỏ bởi tham số thứ hai. Do vậy, dòng lệnh set j [lindex $argv 0] gán cho biến j giá trị của tham số đầu tiên được truyền vào chương trình (giá trị đã được lưu vào biến argv).

```
# Cách sử dụng: ns prime.tcl NUMBER
# NUMBER là số các số nguyên tố mà chúng ta muốn nhận

  If {$argc !=1}{
#  Phải nhận lấy một đối số đơn, hoặc chương trình sẽ báo
#lỗi Puts stderr ‘‘ERROR! Ns called with wrong number of argument!($argc)’’
exit 1
  } else{
  Set j [lindex $argv 0]
     proc prime {j}{
#Tính toán tất cả các số nguyên tố cho tới j
       for {set a 2}{$a<=$j}{incr a}{
          set b 0
          for {set i 2}{$i<$a}{incr i}{
              set d [expr fmod($a,$i)]
              if {$d==0}{
                 set b 1}
           }
           if{$b==1}{
            puts ‘‘$a không phải là số nguyên tố’’
           }else{
            puts ‘‘$a là số nguyên tố’’
           }
         }
       }
 prime $j
```

Bảng 2.2: Chương trình Tcl tính toán các số nguyên tố

Chúng tôi chỉ giải thích ngắn gọn thông qua một ví dụ về mẫu lập trình trong Otcl. Nếu bạn chưa biết một ngôn ngữ lập trình hướng đối tượng nào (C++, Java), bạn nên học cách lập trình các ngôn ngữ hướng đối tượng.

Từ khóa dành riêng `Class` đặt sau tên của một lớp được dùng để khai báo một lớp mới trong OTcl. Các phương thức (hàm) của một lớp được khai báo bằng cách dùng từ khóa `instproc` đặt đằng sau tên lớp, tiếp đó là tên của phương thức và các tham số của nó. Phương thức init là hàm khởi tạo của lớp đó. Biến `self` là một con trỏ tới bản thân đối tượng, giống như con trỏ `"this"` trong C++ hoặc Java. Để khai báo một đối tượng (instance), OTcl sử dụng từ khóa `instvar`. Từ khóa `-superclass` được dùng để khai báo rằng một lớp là lớp con (lớp thừa kế) của một lớp khác, trong ví dụ này, lớp `Integer` (số nguyên)là lớp con của lớp `Real` (số thực). Chúng

tôi khuyến cáo các bạn tìm hiểu thêm về OTcl bằng cách đọc thêm các tài liệu khác.

```
# Cách dùng: ns fact.tcl Number
# NUMBER là số mà ta muốn lấy giai thừa
#
if {$argc !=1}{
# Phải lấy đối sô đơn hoặc chương trình báo lỗi
# Puts stderr ‘‘ERROR! Ns called with wrong number
# of argument!($argc)’’
exit 1
} else{
  Set f [lindex $argv 0]
}
   proc Factorial{x}{
   for {ser result 1}{$x>1}{set x [expr $x-1]}{
   set result [expr $result*$x]
}
 return $result
}
Set res [Factorial $f]
Puts ‘‘Factorial of $f is $res’’
```

Bảng 2.3: Chương trình Tcl tính giai thừa của một số

```
Class Real
Real instproc init {a}{
   $self instvar value_
   set value_ $a
}
Real instproc sum {x}{
   $self instvar value_
   set op ‘‘$value_+[$x set value_] = \t’’
   set value_[expr $value_+ [$x set value_]]
   put ‘‘$op $value_’’
}

Real instproc multiply {x}{
```

```
   $self instvar value_
   set op ‘‘$value_*[$x set value_] = \t’’
   set value_[expr $value_* [$x set value_]]
   put ‘‘$op $value_’’
}

Real instproc divide {x}{
   $self instvar value_
   set op ‘‘$value_/[$x set value_] = \t’’
   set value_[expr $value_/ [$x set value_]]
   put ‘‘$op $value_’’
}

Class integer -superclass Real
Integer instproc divide {x}{+
   $self instvar value_
   set op ‘‘$value_/[$x set value_] = \t’’
   set d[expr $value_/ [$x set value_]]
   set value_ [expr round($d)]
   put ‘‘$op $value_’’
   }
   set realA [new Real 12.3]
   set realB [new Real 0.5]

   $realA sum $realB
   $realA multiply $realB
   $realA divide $realB

   set integerA [new Integer 12]
   set integerB [new Integer 5]
   set integerC [new Integer 7]

    $integerA multiply $integerB
   $integerB divide $integerC
```

Bảng 2.4: Chương trình Tcl đơn giản sử dụng đối tượng real và integer

Chương 3

Các thành phần cơ bản trong bộ mô phỏng NS

Mục lục

Người dịch: *Nguyễn Thanh Hải*
Biên tập: *Hà Tất Thành*

Bản gốc: *NS Simulator for beginners, Chapter 2* [2]

Trong chương này chúng ta sẽ trình bày các bước đầu tiên gồm có:

- Việc khởi tạo và kết thúc của bộ mô phỏng ns.
- Định nghĩa các node mạng, link, hàng đợi và topology.
- Định nghĩa các tác nhân (agent) và các ứng dụng.
- Công cụ hiển thị "nam" .
- Bám vết.
- Các biến ngẫu nhiên.

Một vài ví dụ đơn giản được đưa ra sẽ cho phép chúng ta thực hiện các bước đầu tiên với bộ mô phỏng ns

3.1. Khởi tạo và kết thúc

Một mô phỏng ns bắt đầu với câu lệnh

```
set ns [new Simulator]
```

Đây là dòng đầu tiên trong tập lệnh tcl. Dòng này khai báo 1 biến ns mới dùng lệnh `set`, bạn có thể gọi biến ngẫu nhiên này như bạn muốn, tuy nhiên thông thường người ta khai báo nó là ns vì nó là một thể hiện của lớp `Simulator`, do vậy nó là một đối tượng. Thật vậy đoạn mã `[new simulator]` là một trường hợp thể hiện của lớp `Simulator` sử dụng từ khóa `new`. Vậy nên việc sử dụng biến mới ns này chúng ta có thể sử dụng tất cả phương thức của lớp Simulator, chúng ta sẽ thấy điều này ở phần sau.

Để có các file đầu ra với dữ liệu mô phỏng (file bám vết) hay các file sử dụng cho hiển thị (các file nam) thì chúng ta cần tạo các file bằng cách sử dụng lệnh `"open"`:

```
#Mở file bám vết
Set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
#Mở file bám vết NAM
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

Đoạn mã trên tạo một file bám vết tên là "out.tr" và một file bám vết hiển thị nam (cho công cụ NAM) tên là "out.nam". Trong tập lênh tcl, các file này không được gọi một cách rõ ràng bằng tên của chúng nhưng thay vào đó bằng các con trỏ được khai báo ở trên và được gọi lần lượt là "tracefile1" và "namefile". Dòng đầu tiên và dòng thứ 4 trong ví dụ này chỉ là các chú thích, chúng không phải là các lệnh mô

phỏng. Chú ý rằng chúng bắt đầu với một ký tự "#". Dòng thứ 2 mở file "out.tr" để sử dụng cho ghi, được khai báo với chữ "w". Dòng thứ 3 sử dụng một phương thức `Simulator` tên là `trace-all` có tham số là tên của file mà các trace sẽ chạy. Với câu lệnh simulator này chúng ta sẽ bám vết tất cả các sự kiện theo một khuôn dạng cụ thể mà chúng ta sẽ giải thích sau chương này

Dòng cuối cùng báo cho bộ mô phỏng ghi lại tất cả sự bám vết thành dạng đầu vào NAM. Nó cũng đưa ra tên file mà sự bám vết sẽ được ghi lại sau đó bằng lệnh `$ns flush-trace` (xem thủ tục ‘‘`finish`’’ ở dưới). Trong trường hợp của chúng ta, file được chỉ tới bằng con trở ‘‘`$ namfile`’’ là file ‘‘`out.tr`’’

Chú ý: các câu lệnh `trace-all` và `namtrace-all` có thể dẫn đến việc tạo ra các file kích thước lớn. Nếu chúng ta muốn tiết kiệm không gian, các lệnh trace khác sẽ được sử dụng bởi vậy khi tạo trace chỉ một tập con các sự kiện mô phỏng là thực sự cần đến. Như các câu lệnh được mô tả trong phần 3.6

Kết thúc chương trình được thực hiện bằng một thủ tục ‘‘`finish`’’

```
#Định nghĩa một thử tục ‘finish’
Proc finish {} {
     glocal ns tracefile1 namfile
     $ns flush-trace
     close $tracefile1
     close $namfile
     exe nam out.nam &
     exit 0
}
```

Từ khóa `proc` khai báo một thủ tục - trong trường hợp này là finish và không có tham số nào. Từ khóa global được dùng để cho biết rằng chúng ta đang dùng các biến được khai báo ngoài thủ tục. Phương thức simulator ‘‘`flush-trace`’’ sẽ đổ (dump) các trace vào các file tương ứng. Lệnh tcl “close” đóng các file bám vết định nghĩa trước đó và ‘‘`exec`” thực thi chương trình nam cho hiển thị. Lưu ý rằng chúng ta đã chuyển tên thật của file bám vết thành nam và không thực hiện điều này với con trỏ namfile vì nó không phải câu lệnh ngoài. Câu lệnh exit sẽ kết thúc ứng dụng và trả về số 0 như trạng thái của hệ thống. Giá trị không là mặc định cho exit không lỗi (clean exit). Các giá trị khác có thể được dùng để cho biết đó là một exit có một vài lỗi gì đó. Ở cuối chương trình ns chúng ta sẽ gọi thủ tục “finish” và chỉ ra thời điểm kết thúc. Ví dụ,

```
$ns at 125.0 ‘‘finish’’
```

sẽ được dùng để gọi “finish” vào thời điểm 125 giây. Thật vậy, phương thức của simulator cho phép chúng ta lập lịch các sự kiện một cách chính xác. Sau đó trình mô phỏng có thể bắt đầu bằng lệnh:

```
$ns run
```

3.2. Định nghĩa một mạng các liên kết và các nút

Cách định nghĩa một nút.

```
set n0 [$ns node]
```

Chúng ta đã tạo một nút được trỏ tới bởi biến n0. Khi muốn tham chiếu nút đó trong tập lệnh thì chúng ta sẽ viết `$n0`.

Một khi định nghĩa vài nút, chúng ta có thể định nghĩa các liên kết mà kết nối tới chúng. Ví dụ định nghĩa một liên kết là:

```
$ns duplex-link $n0 $n3 10Mb 10ms DropTail
```

Ở đây các nút `$n0` và `$n2` được kết nối bằng một liên kết 2 chiều có độ trễ đường truyền 10ms và dung lượng mỗi chiều 10Mbps.

Để định nghĩa một liên kết có chiều thay cho một liên kết 2 chiều, chúng ta thay ‘‘`duplex-link`’’ bằng ‘‘`simplex-link`’’

Trong NS, một hàng đời đầu ra của một nút được thực thi như một phần của mỗi liên kết mà đầu vào của nó là nút kia. Định nghĩa liên kết chứa cách điều khiển tràn bộ nhớ ở hàng đợi. Trong trường hợp của chúng ta, nếu dung lượng bộ đệm của hàng đợi đầu ra bị vượt quá thì gói tin cuối cùng đến sẽ bị loại bỏ (tùy chọn DropTail). Có nhiều tùy chọn nữa khác như là kỹ thuật RED (Loại bỏ sớm ngẫu nhiên), FQ (Sắp hàng đợi cân bằng), DRR (Deficit Round Robin), sắp hàng đợi cân bằng ngẫu nhiên (SFQ) và CBQ (gồm một bộ lập lịch ưu tiên và bộ lập lịch round-robin); chúng ta sẽ trở lại kỹ thuật RED sau một cách chi tiết hơn).

Tất nhiên, chúng ra cũng định nghĩa dung lượng bộ đệm hàng đợi liên quan tới mỗi liên kết. Chẳng hạn:

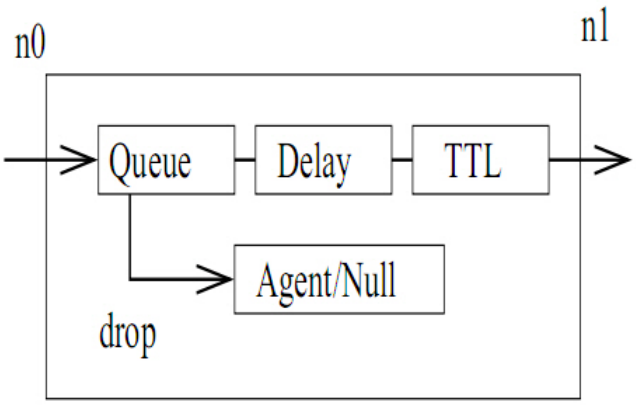
```
#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20
```

Một liên kết đơn công có mẫu được trình bày như trong hình 3.1. Tràn hàng đợi được thực hiện bằng việc gửi các gói tin bị hủy tới một tác nhân Null. Đối tượng TTL sẽ tính tham số Time to Live [1] cho mỗi gói nhận được. Một liên kết song công được xây dựng từ hai liên kết đơn công song song.

Xét một ví dụ của một mạng đơn giản được mô tả trong hình 3.2. Mạng này được định nghĩa thông qua đoạn mã được đưa ra trong bảng 3.1

Chú ý rằng chúng ta đã định nghĩa dung lương bộ đệm tương ứng với một liên kết duy nhất (giữa n2 và n3) Các hàng đợi tương ứng với tất cả liên kết khác đều có

[1] các gói có một vài thẻ kết hợp được cập nhật trong mạng và chỉ ra chúng sẽ ở trong mạng trong thời gian bao lâu trước khi tới được đích. Khi thời gian này quá hạn các gói tin sẽ bị hủy.



Hình 3.1: Kênh truyền đơn công

giá trị mặc định là 50. Giá trị mặc định này nằm trong câu lệnh của ns-default.tcl[2]

```
Queue set limit_ 50
```

```
#Tạo 6 nút
Set n0 [$ns node]
Set n1 [$ns node]
Set n2 [$ns node]
Set n3 [$ns node]
Set n4 [$ns node]
Set n5 [$ns node]

$Tạo các liên kết giữa các nút
$ns duplex-link $n0 $n2 2Mb 10ms DropTial
$ns duplex-link $n1 $n2 2Mb 10ms DropTial
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail
#Đặt kích thước hàng đợi cho liên kết (n2-n3) bằng 20
$ns queue-limit $n2 $n3 20
```

Bảng 3.1: Định nghĩa các nút, kênh truyền và gán kích cỡ hàng đợi

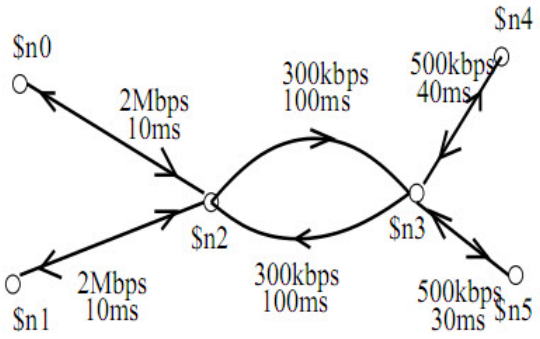
Chúng ta có thể tìm giá trị mặc định này bằng cách nào ? Đầu tiên, kiểm tra file `ns-lib.tcl`, ở đây ta sẽ thấy thủ tục `queue-limit`

```
Simulator instproc queue-limit {n1 n2 limit}{
```

[2]trong is-allinone-2.XXX/ns-2.XXX/tcl/lib, ở đây XXX là chỉ số phiên bản, ví dụ 1b9a.

```
$self instvar link_
[$link_([$n1 id]:[$n2 id]) queue] set limit_ $limit
}
```

Trong đó, chúng ta thấy rằng giới hạn hàng đợi là một phương thức của bộ mô phỏng mà cần 3 tham số: 2 nút định nghĩa kênh truyền và giới hạn hàng đợi. Ở đây ta thấy số giới hạn hàng đợi được đưa ra bởi biến `limit_`.



Hình 3.2: Ví dụ về một mạng đơn giản

```
#Thiết đặt một kết nối TCP
Set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
Set sink [new Agent/TCP]
$ns attach-agent $n4 $tcp
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packetSize_ 552

#Thiết đặt một FTP qua kết nối TCP
Set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

Bảng 3.2: Định nghĩa một ứng dụng FTP sử dụng tác nhân TCP

3.3. Tác nhân và ứng dụng

Vừa rồi chúng ta đã định nghĩa cấu trúc mạng (các nút và kênh truyền), bây giờ ta sẽ tạo dòng lưu lượng qua chúng. Cuối cùng chúng ta cần định nghĩa việc định

tuyến (ở các nguồn và đích cụ thể) cho các trạm (các giao thức định tuyến) và ứng dụng sử dụng chúng.

Trong ví dụ trước, chúng ta muốn chạy một ứng dụng FTP (giao thức truyền file) giữa nút `$n0` và `$n4` và ứng dụng CBR (tốc độ bit không đổi) giữa nút `$n1` và `$n5`. Giao thức internet mà FTP sử dụng là TCP/IP (Giao thức điều khiển vận chuyển/Giao thức Internet) còn CBR sử dụng giao thức UDP (giao thức gói dữ liệu người dùng). Trước tiên chúng ta sẽ định nghĩa (như trong bảng 3.2 một tác nhân TCP giữa nút nguồn `$n0` và nút đích `$n4` và ứng dụng FTP sử dụng nó. Sau đó chúng ta định nghĩa như trong bảng 3.3 tác nhân UDP giữa nút nguồn `$n1` và nút đích `$n5` và ứng dụng CBR sử dụng nó.

3.3.1. FTP trên nền TCP

TCP là một giao thức điều khiển nghẽn tin cậy động sẽ được giải thích chi tiết trong chương 4 (Cuốn "ns for beginer"). Nó sử dụng các báo nhận được tạo từ phía đích để biết gói tin nhận được đầy đủ hay chưa. Các gói bị mất được hiểu là các dấu hiệu tắc nghẽn, như vậy nguyên nhân TCP yêu cầu kênh truyền 2 hướng là để các báo nhận quay trở về nguồn.

Có một vài biến thể của giao thức TCP, như là `Tahoe, Reno, Newreno, Vegas`. Kiểu tác nhân (agent-có thể hiểu là một trạm) xuất hiện trong dòng đầu tiên:

`Set tcp [new Agent/TCP]` Câu lệnh này cũng đưa ra một con trỏ gọi `"tcp"` cho tác nhân TCP, một đối tượng trong NS.

Câu lệnh `$ns attach-agent $n0 $tcp` định nghĩa nút nguồn kết nối TCP. Câu lệnh

```
Set sink [new Agent/TCPSink]
```

định nghĩa hoạt động của nút TCP đích và gán tới nó một con trỏ gọi là sink. Chúng ta lưu ý rằng trong TCP, nút đích có một vai trò tích cực trong giao thức tạo ra báo nhận để bảo đảm tất cả gói tin đến được đích.

Câu lệnh `$ns attach-agent $n4 $sink` định nghịa nút đích. Lệnh `$ns connect $tcp $sink` tạo kết nối TCP giữa các nút nguồn và nút đích.

TCP có nhiều tham số với các giá trị mặc định được cố định ban đầu có thể bị thay đổi nếu xét trong trường hợp cụ thể. Ví dụ, kích thước gói tin TCP là 1000 bytes. Giá trị này có thể thay đổi thành giá trị 552 byte bằng lệnh `$tcp set packetSize_ 552`.

Khi chúng ta có vài luồng dữ liệu, có thể ta muốn phân biệt chúng để ta có thể nhận ra chúng với các màu khác nhau trong phần hiển thị. Điều này được thực hiện bằng câu lệnh `$tcp set fid_ 1` gán cho kết nối TCP một số nhận dạng luồng bằng "1"; sau đó ta sẽ gán số nhận dạng luồng dòng cho kết nối UDP là "2".

Khi kết nối TCP được định nghĩa thì ứng dụng FTP có thể định nghĩa qua nó.

```
#Thiết đặt một kết nối UDP
 Set udp [new Agent/UDP]
 $ns attach-agent $n1 $udp
 Set null [new Agent/Null]
 $ns attach-agent $n5 $null
 $ns connect $udp $null
 $udp set fid_ 2

#Thiết đặt một CBR qua kết nối UDP
 Set cbr [new Application/Traffic/CBR]
 $cbr attach-agent $udp
 $cbr set packetSize_ 1000
 $cbr set rate_ 0.01Mb
 $cbr set random_ false
```

Bảng 3.3: Định nghĩa một ứng dụng CBR sử dụng tác nhân UDP

Điều này được thực hiện ở 3 dòng cuối trong bảng 3.2 .

Chú ý rằng cả tác nhân TCP cũng như ứng dụng FTP đều được cho các con trỏ: chúng ta gọi con trỏ cho tác nhân TCP là ‘‘tcp’’ (tuy nhiên có thể sử dụng tên khác) và con trỏ cho FTP là ‘‘ftp’’.

3.3.2. CBR qua UDP

Tiếp theo chúng ta sẽ định nghĩa kết nối UDP và ứng dụng CBR qua nó, xem bảng 3.3 . Một nguồn UDP (Agent/UDP) và đích (Agent/Null) được định nghĩa theo 1 cách tương tự như trường hợp của TCP. Về ứng dụng CBR sử dụng UDP, bảng 3.3 cũng chỉ ra cách định nghĩa tốc độ truyền dẫn và kích thước gói tin.

Thay vì định nghĩa tốc độ trong câu lệnh `$cbr set rate_ 0.01Mb`, ta có thể định nghĩa khoảng thời gian giữa các gói truyền đi bằng lệnh

```
$cbr set interval_ 0.005
```

Các tham số khác của CBR là `random_` , là một cờ sẽ chỉ ra mô phỏng có đưa ra “tạp âm” ngẫu nhiên trong thời gian truyền đã lập lịch hay không. Mặc định nó là “off”, và có thể được đặt thành “on” bằng lệnh

```
$cbr set random_ 1
```

Kích thước gói có thể được đặt một giá trị nào đó bằng lệnh

```
$cbr set packetSize_ <packet size>
```

3.3.3. UDP với các nguồn lưu lượng khác

Chúng ta có thể mô phỏng các kiểu ứng dụng lưu lượng sử dụng giao thức UDP khác:nguồn lưu lượng dùng và không dùng phân bố theo hàm mũ, nguồn dùng và không dùng phân bố pareto, và một nguồn điều khiển dựa trên bám vết (trace-driven). Nguồn có phân bố hàm mũ và nguồn có phân bố pareto được khai báo lần lượt là

```
Set source [new Application/Traffic/Exponential]
Set source [new Application/Traffic/Pareto]
```

Các nguồn này yêu cầu các tham số: `packetSize_` (theo byte); `burst_time_` định nghĩa thời gian "bật" trung bình; `idle_time` định nghĩa thời gian "tắt" trung bình và `rate_` xác định tốc độ truyền dẫn trong suốt chu kỳ "bật". Trong nguồn pareto bật/tắt chúng ta cũng định nghĩa "hình dạng" bằng tham số `shape_`. Một ví dụ của một Pareto bật/tắt được đưa ra:

```
Set source [new Application/Traffic/Pareto]
$source set packetSize_ 500
$source set burst_time_ 200ms
$source set idle_time_ 400ms
$source set rate_ 100k
$source set shape_ 1.5
```

(Các biến ngẫu nhiên được bàn trong phần 2.7)

Ứng dụng điều khiển dựa trên bám vết được định nghĩa như sau. Đầu tiên chúng ta khai báo file trace:

```
Set tracefile [new Tracefile]
$tracefile filename <file>
```

Sau đó chúng ta định nghĩa ứng dụng điều khiển dựa trên bám vết và gắn nó vào file đó

```
set stc [new Application/Traffic/Trace]
$src attach-tracefile $tracefile
```

File sẽ ở dạng nhị phân và chứa thời gian giữa các gói theo ms và kích thước gói theo byte.

3.4. Lập lịch sự kiện

NS là một mô phỏng dựa trên sự kiện rời rạc. Tập lệnh Tcl định nghĩa khi nào sự kiện xảy ra. Câu lệnh khởi tạo set ns `[new Simulator]` tạo một chương trình lập lịch sự kiện và các sự kiện được lập lịch bằng cách sử dụng dạng sau:

```
$ns at <time> <event>
```

Bộ lập lịch bắt đầu khi ta chạy ns, thông qua lệnh `$ns run`

Trong ví dụ đơn giản của chúng ta, ta sẽ lập lịch quá trình bắt đầu và kết thúc của ứng dụng FTP và CBR. Điều này có thể được thực hiện thông qua các lệnh sau:

```
$ns at 0.1 ‘‘$cbr start’’
$ns at 1.0 ‘‘$ftp start’’
$ns at 124.0 ‘‘$ftp stop’’
$ns at 0.1 ‘‘$cbr stop’’
```

Như vậy FTP sẽ hoạt động trong thời gian từ 1.0 đến 124.0 và CBR sẽ hoạt động trong thời gian 0.1 đến 124.5 (tất cả đơn vị là giây).

Bây giờ chúng ta đã sẵn sàng để chạy mô phỏng của nó. Nếu các câu lệnh được viết trong một file gọi là “ex1.tcl” (xem bảng **??**) thì đơn giản ta chỉ cần gõ `‘‘ns ex1.tcl’’`

Chú ý trong bảng **??** chúng ta đã thêm ở cuối một thủ tục khác ghi một file đầu ra với kích thước của sổ TCP tức thời ở các khoảng thời gian 0.1 giây. Trong ví dụ, tên file đầu ra là “WinFile”. Thủ tục là đệ quy nên cứ sau 0.1 giây nó sẽ gọi lại chính nó. Nó chuyển nguồn TCP và file tới đầu ra để ghi mhư một tham số mà ta muốn ghi ở đầu ra.

```
set ns [new Simulator]
# Định nghĩa các màu của các luồng khác nhau
$ns color 1 Blue
$ns color 2 Red
#Mở file bám vết
set tracefile1 [open out.tr w]
set winfile [open WinFile w]
#ns trace-all $tracefile1
#Mở file bám vết NAM
set namfile1 [open out.tr w]
#ns namtrace-all $namfile
#Định nghĩa thủ tục ‘‘finish’’
proc finish {} {
glogal ns tracefile1 namefile
#ns flusj-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
```

```
# Tạo 6 nút
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
#Tạo các kênh truyền giữa các nút
$ns duplex-link $n0 $n2 2Mb 10ms Droptail
$ns duplex-link $n1 $n2 2Mb 10ms Droptail
$ns simplex-link $n2 $n3 0.3Mb 100ms Droptail
$ns simplex-link $n3 $n2 0.3Mb 100ms Droptail
$ns duplex-link $n3 $n4 0.5Mb 40ms Droptail
$ns duplex-link $n3 $n5 0.5Mb 30ms Droptail
#Đặt vị trí nút (cho NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
#Đặt kích thước hang đợi cho kênh truyền (n2-n3) thành 20
$ns queue-limit $n2 $n3 20

#Thiết lập các phiên kết nối TCP
set tcp [new Agent/TCP]
$ns attack-agent $n0 $tcp
set tcp [new Agent/TCPSink]
$ns attack-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packetSize_ 552

$Thiết lấp một phiên kết nối FTP dựa trên TCP
set ftp [new Application/FTP]
$ftp attach-agent $tcp

#Thiết lập phiên kết nối UDP
set tcp [new Agent/UDP]
$ns attack-agent $n1 $udp
set null [new Agent/Null]
```

```
$ns attack-agent $n5 $null
$ns connect $udp $null
$tcp set fid_ 2

$Thiết lấp một phiên kết nối CBR dựa trên UDP
set cbr [new Application/Trafic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1000
$cbr set rate_ 0.01Mb
$cbr set random_ false

$ns at 0.1 ‘‘$cbr start’’
$ns at 1.0 ‘‘$ftp start’’
$ns at 124.0 ‘‘$ftp stop’’
$ns at 124.5 ‘‘$cbr stop’’

# Thủ tục vẽ kĩch thước của sổ. Lấy tên của nut tcp
nguồn và # của các file đầu ra như các tham số/vector
proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now {$ns now}
set cwnd {$tcpSource set cwnd_}
put $file ‘‘$now $cwnd’’

$ns at [expr $now+$time] ‘‘plotWindow $tcpSource $file
}
$ns at 0.1 ‘‘plotWindow $tcp $winfile’’

$ns at 125.0 ‘‘finish’’
$ns run
```

Bảng 3.4: Chương trình ex.tcl

3.5. Hiển thị: dùng NAM

Khi chúng ta chạy ví dụ 3.1 , công cụ hiển thị nam sẽ hiển thị một mạng 6 nút. Vị trí của các nút có thể được chọn ngẫu nhiên. Để thủ tục tái lập vị trí ban đầu của các nốt như trong hình 3.2 , chúng ta thêm vào tập lệnh tcl đoạn sau:

```
#đưa ra vị trí nút (cho NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient left
$ns simplex-link-op $n3 $n2 orient right
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
```

Chú ý: nếu một vị trí ngẫu nhiên của các nút được chọn và nó không thõa mãn, ta có thể nhấn vào nút "`re-layout`" và một vị trí khác sẽ được chọn. Ta cũng có thể sửa vị trí bằng cách nhấp chuột vào nút Edit/View và "kéo rê" mỗi nút tới vị trí yêu cầu (với sự trợ giúp của con chuột).

Cần lưu ý rằng trình hiển thị nam hiển thị cho chúng ta với hình ảnh các gói CBR (dòng tin từ nút 1 đến nút 5) màu đỏ và các gói TCP (từ nút 0 đến nút 4) màu xanh. Các ACK của TCP (các báo nhận) đi theo các hướng ngược và cũng có màu xanh tuy nhiên chậm hơn bởi vì một ACK có kích thước 40 byte trong khi các gói TCP có kích thước 552 byte. Để nhận được các màu, chúng ta cần định nghĩa trong phần bắt đầu của tập lệnh ex1.tcl

```
$ns color 1 Blue
$ns color 2 Red
```

Chú ý rằng nếu đã có một file nam thì không cần phải chạy ns để xem nó, mà phải gõ trực tiếp lệnh nam `<file name>`

"Snapshots" từ các trình hiển trị nam có thể được in (vào trong một máy in hoặc thành một file) bằng cách vào tùy chọn "File' ở menu trên cùng.

Một vài điều khác có thể thực hiện trong NAM

- Tô màu nút, ví dụ như nếu muốn n0 xuất hiện với màu đỏ chúng ta viết `$n0 color red`
- Hình dạng các nút: ở mặc định chúng là vòng, tuy nhiên có thể hiển thị khác đi. Ví dụ ta có thể gõ `$n1 shape box` (hoặc thay vì "box" ta có thể dùng "hexagon" hay "circle")
- Tô màu liên kết: như ví dụ

  ```
  $ns duplex-link-op $n0 $n2 color ‘‘green’’
  ```

- Thêm và xóa các dấu (mark): Chúng ta có thể đánh dấu một nút ở thời điểm đang xét (ví dụ ở cùng thời điểm khi chúng ta kích hoạt một nguồn lưu lượng nào đó ở thời điểm đó). Ví dụ chúng ta có thể gõ

  ```
  $ns at 2.0 ‘‘$n3 add-mark m3 blue box’’
  $ns at 30.0 ‘‘$n3 delete-mark m3’’
  ```

  Đoạn mã này tạo ra dấu màu xanh bao quanh nút 3 trong khoảng thời gian [2,30]

- Thêm các nhãn: một nhãn có thể xuất hiện trên màn hình từ thời điểm đang xét trở đi, như là đưa ra nhãn “active node” tới một nút n3 từ thời điểm 1.2.

  ```
  $ns at 1.2 "$n3 label \" active node\""
  ```

  Và đưa một nhãn "TCP input link" cho liên kết n0-n2 gõ

  ```
  $ns duplex-link-op $n0 $n2 label ‘‘TCP input link’’
  ```

- Thêm chữ: ở cuối khung của cửa số NAM, ta có thể tạo chữ hiện ra ở thời điểm đang xét. Chữ này có thể được dùng để mô tả một sự kiện nào đó được lập lịch ở thời điểm đó. Ví dụ.

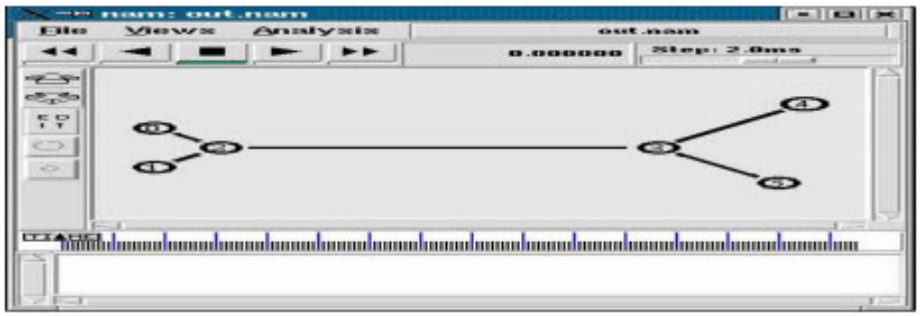
  ```
  $ns at 5 "$ns trace-annotate \"packet drop\""
  ```

- Ngoài ra trong NAM có thể giám sát kích thước hàng đợi. Ví dụ, giám sát hàng đợi đầu vào của liên kết giữa n2-n3, bằng cách gõ

  ```
  $ns  simplex-link-op $n2 $n3 queuePos 0.5
  ```

(Tất cả các ví dụ đều tham chiếu tới các đối tượng được định nghĩa trong ex1.tcl)

Giao diện đồ họa của NAM được chỉ ra trong hình 3.3

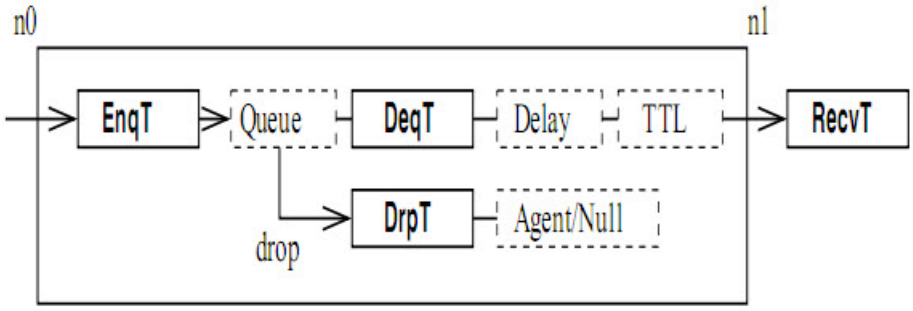


Hình 3.3: Giao diện đồ họa NAM

3.6. Bám vết

3.6.1. Bám các đối tượng

Mô phỏng NS có thể tạo ra trace hiển thị (cho NAM) cũng như một file bám vết ASCII tương ứng với các sự kiện đăng ký trong mạng. Khi chúng ta sử dụng bám vết (như đã đề cập trong phần 3.1) thì ns sẽ chèn 4 đối tượng vào trong liên kết: EngT, DeqT, RecvT và DrpT như chỉ ra trong hình 3.4 EnqT đăng ký thông tin liên quan



Hình 3.4: Bám các đối tượng trong một kênh truyền đơn công

đến một gói đến và được xếp hàng đợi ở hàng đợi đầu vào của kênh truyền. Nếu gói tin tràn thì thông tin liên quan đến gói bị hủy sẽ được điều khiển bởi DrpT. DeqT đăng ký thông tin ngay lúc gói tin bị đẩy ra khỏi hàng đợi. Cuối cùng, RecvT đưa thông tin của chúng ta về các gói nhận được ở đầu ra kênh truyền.

NS cho phép chúng ta nhận nhiều thông tin hơn là thông qua bám vết ở trên. Một phương pháp là sử dụng giám sát hàng đợi. Phương pháp này được mô tả ở cuối phần 7.3

3.6.2. Cấu trúc của các file bám vết

Khi bám vết trong một file ascii đầu ra, trace được tổ chức thành 12 trường như sau (trong hình 3.5). Ý nghĩa của các trường là:

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id

Hình 3.5: Các trường xuất hiện trong một trace

1. Trường đầu tiên là kiểu sự kiện. Được đưa ra bằng một trong 4 biểu tượng `r,+,-,d` lần lượt tương ứng với nhận (ở đầu ra của kênh truyền), đã xếp vào hàng, đã ra khỏi hàng, và bị loại.

2. Trường thứ 2 đưa ra thời điểm xảy ra sự kiện.

3. Đưa ra nút đầu vào của kênh truyền mà sự kiện xảy ra ở đó.

4. Đưa ra nút đầu ra của kênh truyền mà sự kiện xảy ra ở đó.

5. Đưa ra kiểu gói tin (ví dụ như CBR hay TCP. Kiểu tương ứng với tên mà chúng ta đã gán cho các ứng dụng đó. Ví dụ, ứng dụng TCP ở bảng 3.2 được gọi là "tcp".

6. Đưa ra kích thước gói tin

7. Một vài cờ (chúng ta sẽ xem ở sau)

8. Đây là mã nhận dạng luồng (fid) Ipv6 mà một người sử dụng có thể đặt cho mỗi dòng ở tập lệnh Otcl đầu vào.

9. Đây là địa chỉ nguồn đưa ra dưới dạng `"node.port"`

10. Đây là địa chỉ đích, có dạng như trên.

11. Đây là số thứ tự gói tin của giao thức lớp mạng. Mặc dù các thực thi UDP trong một mạng thực tế không sử dụng số tuần tự này, tuy nhiên ns vẫn giữ vết đi của số tuần tự gói UDP cho mục đích phân tích.

12. Trường cuối cùng chỉ ra mã nhận dạng duy nhất của gói tin.

Một ví dụ, xét đoạn mã đầu của trace sinh ra bằng việc chạy tập lệnh ex1.tcl được đưa ra trong bảng 3.4

```
r 93.393900519 0 RTR 19 message        32 1 1 -1 0 -1
s 94.278068529 0 RTR 20 message        32 0 0 -1 0 -1
r 94.278504547 1 RTR 20 message        32 0 0 -1 0 -1
s 100.000000000 0 AGT 21 tcp            40 0 0 1 0 0
s 100.000000000 0 RTR 21 tcp            60 0 0 1 1 0
r 100.000936054 1 AGT 21 tcp            60 0 0 1 1 0
s 100.000936054 1 AGT 22 ack            40 0 1 0 0 0
s 100.000936054 1 RTR 22 ack            60 0 1 0 0 0
r 100.001984108 0 AGT 22 ack            60 1 1 0 0 0
s 100.001984108 0 AGT 23 tcp            1000 0 0 1 0 1
s 100.001984108 0 RTR 23 tcp            1020 0 0 1 1 1
```

```
s 100.001984108 0 AGT 24 tcp            1000 0 0 1 0 2
s 100.001984108 0 RTR 24 tcp            1020 0 0 1 1 2
r 100.006992162 1 AGT 23 tcp            1020 0 0 1 1 1
s 100.006992162 1 AGT 25 ack            40 0 1 0 0 1
s 100.006992162 1 RTR 25 ack            60 0 1 0 0 1
r 100.011920235 1 AGT 24 tcp            1020 0 0 1 1 2
s 100.011920235 1 AGT 26 ack            40 0 1 0 0 2
s 100.011920235 1 RTR 26 ack            60 0 1 0 0 2
r 100.013128291 0 AGT 25 ack            60 1 1 0 0 1
s 100.013128291 0 AGT 27 tcp            1000 0 0 1 0 3
s 100.013128291 0 RTR 27 tcp            1020 0 0 1 1 3
r 100.014811365 0 AGT 26 ack            60 1 1 0 0 2
s 100.014811365 0 AGT 28 tcp            1000 0 0 1 0 4
s 100.014811365 0 RTR 28 tcp            1020 0 0 1 1 4
r 100.019719421 1 AGT 27 tcp            1020 0 0 1 1 3
s 100.019719421 1 AGT 29 ack            40 0 1 0 0 3
s 100.019719421 1 RTR 29 ack            60 0 1 0 0 3
r 100.024687497 1 AGT 28 tcp            1020 0 0 1 1 4
s 100.024687497 1 AGT 30 ack            40 0 1 0 0 4
s 100.024687497 1 RTR 30 ack            60 0 1 0 0 4
r 100.025795554 0 AGT 29 ack            60 1 1 0 0 3
s 100.025795554 0 AGT 31 tcp            1000 0 0 1 0 5
s 100.025795554 0 RTR 31 tcp            1020 0 0 1 1 5
s 100.025795554 0 AGT 32 tcp            1000 0 0 1 0 6
s 100.025795554 0 RTR 32 tcp            1020 0 0 1 1 6
r 100.026883631 0 AGT 30 ack            60 1 1 0 0 4
s 100.026883631 0 AGT 33 tcp            1000 0 0 1 0 7
```

Bảng 3.5: Nội dung của một file bám vết (trace file)

3.7. Biến ngẫu nhiên

Các biến ngẫu nhiên (RVs) với phân bố khác nhau có thể được tạo trong ns. Do vai trò quan trọng của các biến ngẫu nhiên trong mô hình hóa lưu lượng và mô phỏng mạng nên chúng ta cần nhớ lại vắn tắt các định nghĩa và tầm quan trọng của các biến ngẫu nhiên chính trong phụ lục 11. Chi tiết hơn, có thể tra cứu thêm, tại http://www.xycoon.com.

3.7.1. Hạt nhân (hay giá trị ban đầu của một biến ngẫu nhiên)và bộ tạo

Ngoài sự phân bố các biến ngẫu nhiên, có một số vấn đề khác đó là chúng ta cần quan tâm tới khi nào mô phỏng một biến ngẫu nhiên:

- Có phải chúng ta muốn nhận đựoc cùng một giá trị biếng ngẫu nhiên khi chạy lại mô phỏng (có thể thay đổi một vài tham số khác của mô phỏng)? Điều này cho phép chúng ta so sánh trực tiếp, cho một tập sự kiện ngẫu nhiên đơn lẻ, các kết quả mô phỏng phụ thuộc vào một vài tham số vật lý (như là các độ trễ kênh truyền hay độ dài hàng đợi) như thế nào.
- Thường khi chúng ta cần các biến ngẫu nhiên độc lập với nhau.

Tạo các biến ngẫu nhiên sử dụng một hạt giống (là một số nào đó mà chúng ta viết trong tập lệnh tcl). Giá trị hạt giống là 0 gây ra việc tạo một biến ngẫu nhiên mới mỗi lần ta chạy mô phỏng, vì vậy, nếu ta muốn có cùng các biến ngẫu nhiên đã đựoc tạo cho các lần mô phỏng khác nhau, ta sẽ phải lưu các biến ngẫu nhiên đã tạo lại.

Trái lại, nếu chúng ta sử dụng các hạt giống khác thì mỗi khi chúng ta chạy mô phỏng, cùng một chuỗi biến ngẫu nhiên được tạo trong mô phỏng sẽ được tạo ra

Trong ns, nếu chúng ta sử dụng các bộ tạo khác nhau với cùng hạt giống và phân bố, chúng sẽ tạo các ra giá trị biến ngẫu nhiên giống nhau (trừ khi seed bằng 0). Chúng ta sẽ thấy điều này trong ví dụ dưới

3.7.2. Tạo các biến ngẫu nhiên

Đầu tiên chúng ta tạo một bộ tạo mới, và gán nó một seed là 2 với lệnh

```
Set MyRng [new RNG]
$MyRng2 seed 2
```

Sau đó tạo một biến ngẫu nhiên thực sự, chúng ta phải định nghĩa kiểu phân bố và các tham số của nó. Chúng ta đưa ra một vài ví dụ dưới đây: ta sẽ tạo các biến ngẫu nhiên với phân bố Pareto, hằng số, đồng nhất, hàm mũ và siêu mũ.

1. **Phân bố pareto:** Một biến ngẫu nhiên phân bố pareto nghĩa là $r1$ được xây dựng bằng cách cho biết kỳ vọng của nó và tham số hình dạng (shape) của nó β, giá trị mặc định tương ứng là 1.0 và 1.5.

   ```
   Set r1 [new RandomVariable/Pareto]
   $r1 use-rng $MyRng
   ```

```
$r1 set avg_ 10.0
$r1 set shape_ 1.2
```

2. **Hằng số:** Một biến ngẫu nhiên suy biến là hằng số và bằng giá trị của nó

```
set r2 [new RandomVatiable/Constant]
$r2 use-rng $MyRng
$r2 set val_ 5.0
```

3. **Phân bố đều:** Được định nghĩa thông qua điểm nhỏ nhất và lớn nhất trong khoảng hỗ trợ của nó:

```
set r3 [new RandomVariable/Uniform]
$r3 use-rng $MyRng
$r3 set min_ 0.0
$r3 set max_ 10.0


\item \textbf{Phân bố mũ:} Được định nghĩa qua giá trị trung bình của nó
\begin{quote}
\begin{verbatim}
set r4 [new RandomVariable/Exponential]
$r4 use-rng $MyRng
$r4 set avg_ 5
```

4. **Phân bố siêu mũ:** Được định nghĩa như sau:

```
set r5 [new RandomVariable/HyperExponential]
$r5 use-rng $MyRng
$r5 set avg_ 1.0
$r5 set cov_ 4.0
```

Tiếp theo chúng ta sẽ trình bày một chương trình nhỏ (rv1.tcl) kiểm tra các biến ngẫu nhiên phân bố Pareto với các seed và bộ tạo khác nhau nhưng với cùng phân bố Pareto. Được đưa ra trong Bảng 2.6. Cho mỗi seed (giá trị 0, 1 và 2) và bộ tạo, chúng ta tạo một chuỗi 3 biến ngẫu nhiên. Biến "count" được gán số biến ngẫu nhiên mà chúng ta tạo bằng cách sử dụng "test" cho mỗi seed và bộ tạo.

Khi chạy ví dụ này ta có thể thấy rằng với seed bằng 0, hai bộ tạo cho các giá trị biến khác nhau; như vậy ta đã đạt được 6 giá trị khác nhau (3 giá trị từ mỗi bộ tạo). Cho các seed khác, một bộ tạo tạo ra 3 giá trị khác nhau nhưng các giá trị này không phụ thuộc vào bộ tạo: giá trị thứ n được tạo bởi bộ tạo 1 cũng như được tạo bởi bộ tạo 2.

```
## Ví dụ đơn giản làm rõ cách dùng lớp RandomVariable từ tcl
Set count 3
For {set I 0} {$i<3}{incr i}{
Puts ‘‘===== I = $i ’’

Set MyRng1 [new RNG]
$MyRng1 seed $i

Set MyRng2  [new RNG]
$MyRng2 seed $i

Set r1 [new RandomVariable/Pareto]
$r1 use-rng $MyRng1
$r1 set avg_ 10.0
$r1 set shape_ 1.2
Puts stdout ‘‘KT phân bố Pareto, avg = [$r1 set avg_]’’ shape = [$r1 set shape_ ]’’

Set r2 [new RandomVariable/Pareto]
$r1 use-rng $MyRng2
$r1 set avg_ 10.0
$r1 set shape_ 1.2
Puts stdout ‘‘KT phân bố Pareto, avg = [$r2 set avg_]’’ shape = [$r2 set shape_ ]’’

}
```

Bảng 3.6: Kiểm tra các biến ngẫu nhiên phân bố Pareto với các seed khác nhau

Chương 4

Làm việc với file trace

Mục lục

Người dịch: *Nguyễn Mạnh Linh*
Biên tập: *Nguyễn Quốc Bình*

Bản gốc: *NS Simulator for beginners, Chapter 3* [2]

Chương trình mô phỏng NS có thể cung cấp rất nhiều dữ liệu chi tiết về các sự kiện xảy ra khi một mạng hoạt động. Nếu chúng ta muốn phân tích những dữ liệu đó chúng ta cần phải trích tách những thông tin liên quan ra từ những file vết (`trace files`) và xử lý chúng

Tuy rằng ta có thể viết những chương trình bằng bất cứ ngôn ngữ lập trình nào

để có thể xử lý được những file dữ liệu đó, nhưng đã có một số công cụ miễn phí phù hợp cho mục đích phân tích file trace đã được phát triển trên một số hệ điều hành (unix, linux, window, ...). Tất cả những gì người sử dụng cần làm là viết những đoạn script ngắn được dịch và thực thi mà không cần phải biên dịch chương trình.

4.1. Xử lý file dữ liệu với công cụ awk

Công cụ awk cho phép chúng ta thực hiện một số các chức năng xử lý đơn giản trên các file dữ liệu như là tính trung bình các giá trị đưa ra trên một cột, tính tổng hoặc tích từng giá trị giữa một vài cột, và tất cả nhưng công việc cần đến thay đổi cấu trúc dữ liệu (data-reformating)...

Trong hai ví dụ sau đây chúng ta sẽ học cách tính giá trị trung bình của một cột trong một file và sau đó tính độ lệch chuẩn.

```
BEGIN { FS - ‘‘\t’’} {nl++} {s=s+$4} END {print ‘‘average:’’ s/nl}
```

Bảng 4.1: awk script để tính giá trị trung bình của cột 4 của một file

(Chú ý: kí hiệu `"\t"` được sử dụng nếu các cột được ngăn cách với nhau bởi dấu tab (tabulated). Nếu không dùng đấu tab mà chỉ dùng ký tự trắng thì ta thay thế bằng `" "`.)

```
BEGIN {FS=’’\t’’}{ln++}{d=$4-t}{s2=s2+d*d} END {print "standev:" sqrt(s2/ln)}
```

Bảng 4.2: awk script để tính giá trị độ lệch chuẩn của cột 4 của một file

Để sử dụng script đầu tiên để tính toán giá trị trung bình của cột thứ 4 của một file có tên là `"Out.ns"` chúng ta sẽ gõ như sau trên hệ điều hành unix:

```
awk -f Average.awk Out.ns
```

Chúng ta sẽ nhận được kết quả trông giống như là: `average : 29.397` cho giá trị trung bình của cột 4 .

Bây giờ để tính độ lệch chuẩn của cột đó chúng ta gõ:

```
awk -v t=29.397 -f StDev.awk Out.ns
```

Câu lệnh này sẽ trả về giá trị: `standev : 33.2003`. Chú ý rằng trong đoạn mã trên, chúng ta phải copy giá trị trung bình thu được từ đoạn mã trước vào trong câu lệnh tính toán độ lệch chuẩn. Thí dụ này đã chỉ ra làm thế nào để đưa một tham số vào một đoạn mã awk.

Chú ý rằng nếu chúng ta không chia ở phần cuối của đoạn mã awk đầu tiên (Bảng 4.1) chon l, chúng ta sẽ nhận được tổng của tất cả các giá trị trong cột 4 thay vì giá trị trung bình của chúng.

Một cách tốt nhất để thu được giá trị trung bình và độ lệch chuẩn là sử dụng mảng:

```
BEGIN { FS = ‘‘\t’’} {val[nl]=$4} {nl++} {s=s+$4} END {
   Av=s/nl
   For (I in val) {
     d=val[i]-av
     s2=s2+d*d
   }
   Print ‘‘averate: ‘‘ av ‘‘ standev ‘‘ sqrt(s2/nl)}
```

Bảng 4.3: Một đoạn mã awk sử dụng mảng để tính trung bình và độ lệch chuẩn

Ví dụ tiếp theo lấy đầu vào là một file với 15 cột (từ 0 đến 14). Đầu ra sẽ là một file có 5 cột, cột thứ nhất chứ cột số 1 của file gốc, và cột thứ 2 tới 5 là tổng của lần lượt các cột 3-4, 6-8, 9-11 và 12-14 (12-14 tương ứng với 3 cột cuối cùng trong file gốc). Để sử dụng đoạn mã trên chúng ta làm như sau:

```
BEGIN {FS=’’\t’’} {l1=$3+$3+$5} {l2=$6+$7+$8} {d1=$9+$10+$11} \
 {d2=$12+$13+$14} {print $1"\t" l1’’\t’’ l2’’\t’’ d1’’\t’’ d2 } END {}
```

Bảng 4.4: Một đoạn mã tính tổng các cột

```
awk -f suma.awk Conn4.tr > outfile
```

File gốc ở đây là file `Conn4.tr` và đầu ra được ghi vào một file có tên là `outfile`

4.2. Sử dụng grep

Lệnh grep trong unix cho phép chúng ta “lọc” một file. Chúng ta có thể tạo một file mới chỉ chứa một số dòng chứa một chuỗi ký tự nhất định lấy từ file gốc. Ví dụ, file trace của ns có thể chứa tất cả các loại gói tin (packets) đi qua tất cả các đường truyền (links) và chúng ta có thể chỉ quan tâm đến dữ liệu liên quan đến các gói tcp gửi từ node 0 đến node 2. Nếu các dòng liên quan đến các sự kiện đó chứa chuỗi `"0 2 tcp"` thì tất cả những gì chúng ta phải làm là gõ:

```
grep ‘‘ 0 2 tcp ‘‘ tr1.tr > tr2.tr
```

Ở câu lệnh trên “tr1.tr” là file trace gốc và “tr2.tr” là file mới xuất ra. Nếu chúng

ta muốn thu được một file chứa tất cả các dòng của tr1.tr khởi đầu với kí tự r, chúng ta gõ lệnh:

```
grep ‘‘^r’’ tr1.tr > tr2.tr
```

Nếu chúng ta muốn tạo ra một file mà tất cả các dòng bắt đầu với “s “ và chữa chuỗi “tcp 1020” chúng ta gõ lệnh:

```
grep ‘‘^s’’ simple.tr | grep ‘‘tcp 1020’’ > tr3.tr
```

4.3. Xử lý các file dữ liệu với `perl`

PERL là từ viết tắt cho `"Practical Extraction and Report Language"`. (Ngôn ngữ trích tách và báo cáo thực hành”. Perl`[?]` cho phép dễ dàng lọc và xử lý các file dữ liệu dạng ASCII trong unix. Ngôn ngữ này được sáng tạo ra bởi Larry Wall với ý tưởng chính là đơn giản hóa các nhiệm vụ quản trị hệ thống. Perl đã phát triển rất nhiều và ngày nay là một ngôn ngữ đa dụng và là một trong những công cụ được sử dụng nhiều nhất cho quản lý dữ liệu Web và Internet.

Perl là một ngôn ngữ thông dịch có rất nhiều tác dụng, nhưng những lợi ích quan trọng nhất khi sử dụng Perl đó là tìm kiếm, trích tách và báo cáo. Một số ưu điểm của việc sử dụng Perl đó là:

- Triển khai dễ dàng các chương trình nhỏ được sử dụng như các bộ lọc, để trích tách thông tin từ các file văn bản (text files).
- Nó được sử dụng trong rất nhiều các hệ điều hành mà không cần phải thay đổi mã chương trình.
- Duy trì và debug các đoạn mã Perl đơn giản hơn các chương trình được viết bởi các ngôn ngữ khác.
- Perl rất phổ biến, do đó có rất nhiều những script miễn phí (được phân phối bởi gnu) trên mạng Internet.

Chúng ta sẽ làm quen trong phần này một vài đoạn mã Perl hữu dụng.

Ví dụ đầu tiên được đưa ra trong bảng 4.5 thực hiện việc tự động tính toán thông lượng của các kết nối TCP. Chương trình lấy trung bình của thông lượng trên những khoảng xác định bởi một tham số gọi là “granularity” . Đầu vào của chương trình nhận 3 giá trị: tên của trace file (ví dụ: out.tr), node mạng mà tại đó chúng ta muốn kiểm tra thông lượng của kết nối TCP, và giá trị granularity.

```
#type: perl throughput.pl <trace file> <required node> <granularity> > file

$infile=$ARGV[0];
$tonode=$ARGV[1];
$granularity=$ARGV[2];
#We compute how many bytes were transmitted during time interval
#specified by granularity parameter in seconds
$sum=0;
$clock=0;

   Open (DATA,''<$infile'')
While (<DATA>) {
     @x = split(‘ ‘);
#column 1 is time
If ($x[1]-$clock <= $granularity)
{
#checking if the event corresponds to a reception
If ($x[0] eq ‘r’)
{
#checking if the destination corresponds to 1st argument
If ($x[3] eq $tonode)
{
#checking if the packet type is TCP
If ($x[4] eq ‘tcp’)
{
     $sum=$sum+$x[5];
}
}
}
}
Else
{   $throughput=$sum/$granularity;
    print STDOUT ‘‘$x[1] $throughput\n'';
    $clock=$clock+$granularity;
    $sum=0;
}
}
    $throughput=$sum/$granularity;
    print STDOUT ‘‘$x[1] $throughput\n'';
    $clock=$clock+$granularity;
    $sum=0;

    Close DATA;
exit(0);
```

Bảng 4.5: Một đoạn mã tính thông lượng

4.4. Vẽ đồ thị với gnuplot

Gnuplot là một phần mềm mã nguồn mở miễn phí được sử dụng rộng rãi cho cả hệ điều hành chạy trên nền unix/linux và window.

Gnuplot có một lệnh yêu cầu giúp đỡ có thể được sử dụng để học chi tiết hơn về các tính năng của phần mềm này.

Cách đơn giản nhất để sử dụng gnuplot đó là gõ `"plot <fn>"`, ở đây file (chúng ta đặt tên là fn) có hai cột thể hiện các giá trị tọa độ x và y của các điểm. Các điểm có thể được nối bởi đường với nhiều kiểu thể hiện được định nghĩa bởi lệnh:

```
plot ‘fn’ w lines 1
```

(những chữ số khác nhau có thể được sử dụng thay cho "1", giá trị này thể hiện các kiểu đường (line) khác nhau). Ta cũng có thể sử dụng các kiểu điểm (points) khác nhau bằng cách viết lệnh như sau:

```
plot ‘fn’ w points 9
```

(một số kiểu điểm có thể được sử dụng phụ thuộc vào chỉ số xuất hiện sau "points"). Một số chức năng khác của gnuplot: hãy xem trong ví dụ dưới đây:

```
set size 0.6, 0.6
set pointsize 3
set key 100,8
set xrange [90.0:120.0]
plot ‘fn1’ w lines 1, ‘fn2’ w lines 8, ‘fn3’ with points 9
```

- Dòng 1: chỉ ra kích thước đồ thị vẽ được nhỏ hơn mặc định.
- Dòng 2: chỉ ra kích thước điểm vẽ được lớn hơn mặc định.
- Dòng 4: giới hạn khoảng giá trị trên trục x là 90-120.
- Dòng 5: vẽ ra 3 đường trên cùng một đồ thị, mỗi đường lấy số liệu từ 3 file khác nhau: `fn1, fn2, fn3.`
- Dòng 3: cho gnuplot biết chính xác vị trí để vẽ "key", là phần chú thích của đồ thị mô tả các đối tượng vẽ. Thông thường, nó chỉ ra mỗi đối tượng được vẽ sử dụng kiểu đường và điểm như thế nào. Thay cho việc mô tả chính xác một vị trí, ta cũng có thể sử dụng các từ khóa ‘left’, ‘right’, ‘top’, ‘bottom’, ‘outside’ và ‘below’ .v.v

  ```
  set key below
  ```

  (câu lệnh trên đặt bảng chú thích bên dưới đồ thị), hay ta chỉ cần gõ "set nokey" để bỏ đi bảng chú thích. Chú ý rằng tên mặc định cho mỗi đối tượng xuất hiện trong bảng chú thích chỉ đơn giản là tên của tệp từ đó vẽ đồ thị. Nếu chúng ta

muốn đặt cho đối tượng một cái tên hơn là sử dụng tên của tệp thì chúng ta phải mô tả bằng câu lệnh như sau:

```
plot ‘fn1’ t ‘‘expectation’’ w lines 1, ‘fn2’ t ‘‘variance’’ w lines 2
```

Ở đây, những tiêu đề như “expectation” và “variance” sẽ xuất hiện trong bảng chú thích.

Nếu một chuỗi các câu lệnh được sử dụng đi sử dụng lại nhiều lần, ta có thể viết chúng vào trong một file, giả sử có tên là “g1.com”, và sau đó chỉ cần load lại file đó mỗi lần chúng ta sử dụng bằng câu lệnh như sau:

```
load ‘g1.com’
```

gnuplot có thể được sử dụng để trích tách một vài cột từ một file nhiều cột. Điều đó được thực hiện như sau:

```
plot ’queue.tr’ using 1: ($4/1000) t "kbytes" w lines 1, \
     ’queue.tr’ using 1:5 t "packest" w lines 2
```

Câu lệnh trên có nghĩa là vẽ một đường đồ thị sử dụng số liệu của cột 1 của file “queue.tr” trên trục x và các giá trị của cột 4 chia cho 1000 trên trục y, và sau đó vẽ trên cùng đồ thị đó một đường khác sử dụng số liệu của cột 5 cho trục y và cùng số liệu ở cột 1 cho trục x. Chú ý: thứ tự của "using", “t” và “lines” trong câu lệnh là quan trọng !

4.5. Vẽ đồ thị với xgraph

Xgraph là một công cụ vẽ đồ thị được cung cấp bởi ns. (Đôi khi cần phải tiến hành biên dịch riêng rẽ cho xgraph trong thư mục xgraph sử dụng lệnh `./configure` và `make`. Xgraph có thể được đóng gói cùng ns trong các phiên bản `ns-allinone` hoặc có thể tải riêng rẽ xgraph về cài). Chú ý rằng công cụ này cho phép tạo ra các file postscript (eps), ảnh Tgif, và các định dạng khác bằng cách ấn vào nút `‘‘Hdcpy’’`. Câu lệnh gọi xgraph có thể được khai báo trong kịch bản tcl do đó có thể xuất ra đồ thị ngay sau khi kết thúc công việc mô phỏng.

Đầu vào của xgraph là một hoặc nhiều ascii file chứa mỗi cặp giá trị x-y trên một dòng (mỗi dòng sẽ vẽ được một điểm trên đồ thị). Ví dụ lệnh xgraph f1 f2 sẽ vẽ ra trên cùng một hình đồ thị của file 1 và file 2.

Một số lựa chọn được sử dụng trong xgraph là:

- Title: sử dụng –t "title"
- Kích thước: `-geometry xsize x ysize`.
- Tiêu đề cho các trục: `-x ‘‘xtitle’’` (khai báo tiêu đề cho trục x) và `-y ‘‘ytitle’’` (khai báo tiêu đề cho trục y).

- Màu của chữ và lưới: dùng với cờ -v.

Một ví dụ cho câu lệnh vẽ bằng xgraph:

```
xgraph f1 f2 -geometry 800x400 -t ‘‘Loss rates’’ -x ‘‘time’’ -y ‘‘Lost packets’’
```

4.6. Trích tách thông tin trong một kịch bản tcl

Chúng ta có thể tích hợp các câu lệnh unix như là ‘‘grep’’ hay ‘‘awk’’ vào trong các kịch bản tcl để có thể tiến hành xử lý dữ liệu trong khi ghi vào file. Ví dụ, một cách khác để giới hạn các file trace (hay để xử lý chúng trực tiếp trong khi chúng được ghi) đó là sử dụng các câu lệnh linux liên quan đến xử lý file ngay bên trong các lệnh tcl.

Ví dụ, chúng ta có thể thay thế lệnh `$set file 1 [open out.tr w]` bằng câu lệnh:

```
set file1 [open ‘‘| grep \’’tcp\’’ > out.tr’’ w]
```

Câu lệnh trên sẽ lọc các dòng viết ra file ‘‘out.tr’’ và chỉ để lại những dòng có chứa từ ‘‘tcp’’

4.7. Minh họa một số file awk và gnuplot

4.7.1. Tính thông lượng của mạng theo hai kiểu file trace

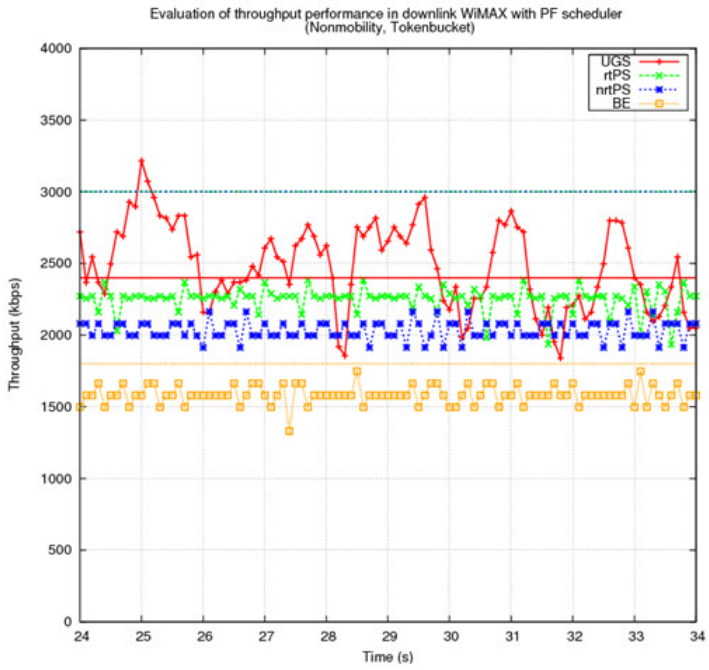
```
BEGIN {
recv = 0
k = 1
}
{
     # Trace line format: normal
     if ($2 != "-t") {
        event = $1
        time = $2
        if (event == "+" || event == "-") node_id = $3
        if (event == "r" || event == "d") node_id = $4
        flow_id = $8
        pkt_id = $12
        pkt_size = $6
        flow_t = $5
        level = "AGT"
```

```
    }
    # Trace line format: new
    if ($2 == "-t") {
       event = $1
       time = $3
       node_id = $5
       flow_id = $39
       pkt_id = $41
       pkt_size = $37
       flow_t = $45
       level = $19
    }

    # Calculate total received packets' size
    if (flow_id == flow && node_id == dst && event == "r" && level == "AGT")
    {
       if (time/tic >= k)
       {
           for (i=k; i<time/tic; i+=1)
           {
             if (i != k)
             recv = 0
             printf("%15g %18g\n",i*tic, (recv/tic)*(8/1000))
           }
           k = i
           recv = pkt_size
       }
       else recv += pkt_size
    }

}

END {
        printf("%15g %18g\n",k*tic, (recv/tic)*(8/1000))
        printf("%15g %18g\n",k*tic, 0)
}
```

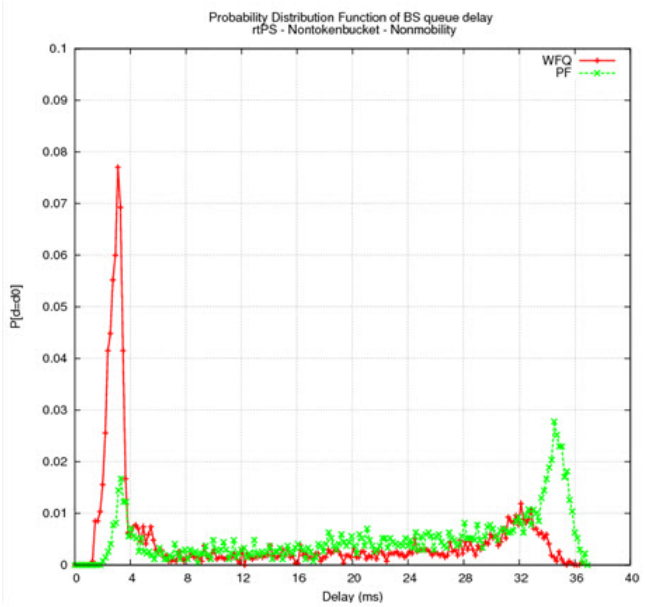
4.7.2. Mẫu vẽ đồ thị thông lượng vừa tính xong bằng file awk

```
Reset
set term postscript eps enhanced color
#set terminal png
#set output "test.png"
set output "throughput.eps"
set size 5/5.,4/3.

set key inside right top vertical Right noreverse
enhanced box linetype -1 linewidth 1.000

set grid
set xlabel "Time (s)"
set ylabel "Throughput (kbps)"
set xrange [40:40.5]
set yrange [0:5500]
set ytics 500
set xtics 0.05 rotate by -45
set style line 1 lt 1 lc rgb "red" lw 3
set style line 2 lt 2 lc rgb "green" lw 3
set style line 3 lt 3 lc rgb "blue" lw 3
set style line 4 lt 4 lc rgb "orange" lw 3

set title "Evaluation of throughput performance in
downlink WiMAX with PF scheduler \n (Nonmobility, tokenbucket scenario 2)"

set label 1 "UGS: 5625 bytes/3 frames \n rtPS: 5625 bytes/3
frames \n nrtPS: 7500 bytes/3 frames \n BE: unlimited" at 40.025,5250

plot "bw_ugs.tr" using 1:2 title "UGS" with lp ls 1, \
"bw_rtps.tr" using 1:2 title "rtPS" with lp ls 2, \
"bw_nrtps.tr" using 1:2 title "nrtPS" with lp ls 3,\
"bw_be.tr" using 1:2 title "BE" with lp ls 4
```

4.8. Một số file hình plot vẽ bằng gnuplot



Hình 4.1: Probability Distribution Fuction of BS queue delay rtPS - Nontokenbucket - Nonmobility



Hình 4.2: Evaluation of throughphut performance in downlink WiMAX with PF scheduler (Nonmonbility Tokenbucket)

Chương 5

NS Tutorial

Mục lục

Người dịch: *Trịnh Việt Thi*
Biên tập: *Hà Tất Thành, Trần Công Lý*

Bản gốc: *Tutorial for the Network Simulator "ns", part 4,5,6* [3]

5.1. Kịch bản Tcl đầu tiên

Ghi chú: Trong phần này, tôi sẽ hướng dẫn bạn sẽ viết một tập lệnh Tcl cho ns để mô phỏng cấu hình mạng đơn giản. Bạn sẽ biết được làm thế nào để tạo ra những nút (nodes)

và những liên kết (links), làm thế nào để gửi dữ liệu từ một node đến node khác, làm thế nào để điều khiển hàng đợi hay chạy NAM từ tập lệnh vừa viết để hiển thị quá trình mô phỏng.

5.1.1. Bắt đầu như thế nào

Chúng ta sẽ viết một "mẫu chung" để có thể sử dụng lại cho mỗi lần định viết tập lệnh Tcl mới. Bạn có thể viết tập lệnh Tcl bằng bất cứ trình soạn thảo nào như joe hoặc emacs. Tôi đặt tên cho ví dụ này là `'example1.tcl'`.

Đầu tiên, cần tạo ra đối tượng mô phỏng bằng câu lệnh:

```
set ns [new Simulator]
```

Tiếp theo tạo tập tin phục vụ cho chức năng bám vết dữ liệu (trace data) của NAM (network animator)

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Dòng lệnh đầu tiên sẽ mở tập tin 'out.nam' với định danh là 'nf'. Trong dòng lệnh thứ hai, truyền tham số là đối tượng được tạo ra ở trên để ns ghi tất cả dữ liệu mô phỏng có liên quan đến nam vào trong tập tin này.

Bước tiếp theo thêm thủ tục 'finish', để kết thúc các tập tin bám vết và bắt đầu nam, thủ tục này đóng file bám vết và khởi động network animator.

```
proc finish {} {
   global ns nf
   $ns flush-trace
   close $nf
   exec nam out.nam &
   exit 0
}
```

Bạn không cần phải hiểu tất cả đoạn mã trên, sẽ dễ hiểu hơn khi bạn nhìn thấy nó hoạt động.

Dòng tiếp theo bảo chương trình mô phỏng thực hiện thủ tục 'finish' sau 5.0 giây mô phỏng.

```
$ns at 5.0 "finish"
```

Nhìn dòng lênh bạn có thể hiểu ý nghĩa của nó. NS cung cấp cho bạn một cách đơn giản để lập lịch cho các sự kiện bằng câu lệnh 'at' .

Dòng cuối cùng là kết thúc việc mô phỏng

```
$ns run
```

Bây giờ bạn có thể lưu tập tin và chạy thử bằng lệnh 'ns example1.tcl'. Bạn có thể gặp thông báo lỗi như 'nam: empty trace file out.nam' bởi vì chúng ta chưa định nghĩa bất cứ đối tượng (node, liên kết...) hay sự kiện nào. Chúng ta sẽ học cách định nghĩa các đối tượng trong phần 5.2 và các sự kiện trong phần **??**.

Bạn sẽ phải sử dụng lại đoạn mã trong phần này ở các phần khác. Bạn có thể tải tại đây hoặc tại đây

5.1.2. Hai node, một liên kết

Trong phần này chúng ta sẽ định nghĩa một cấu hình mạng (topo) đơn giản với hai node được kết nối với nhau bởi một liên kết. Bên dưới có hai dòng lệnh định nghĩa hai node. (Chú ý: bạn phải chèn đoạn mã này trước dòng lệnh `'$ns run'` hay tốt hơn là trước dòng lệnh `'$ns at 5.0 "finish"'`).

```
set n0 [$ns node]
set n1 [$ns node]
```

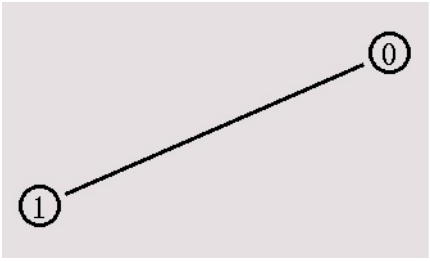
Lệnh `'$ns node'` là để tạo ra 1 node mới, như vậy 2 câu lệnh ở trên tạo ra hai node mới là n0 và n1.

Dòng tiếp theo kết nối hai node

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

Dòng này thiết lập liên kết song công với băng thông 1Megabit, độ trễ 10ms và hàng đợi DropTail giữa node n0 và n1.

Bây giờ bạn có thể lưu tập tin và bắt đầu đoạn mã với `'ns example1.tcl'`. Nam sẽ tự động bắt đầu và bạn có thể thấy kết quả tương tự như hình bên dưới.



Hình 5.1: Tạo kết nối giữa n0 và n1

Bạn có thể tải ví dụ đầy đủ tại đây hoặc tại đây nếu nó không hoạt động và bạn nghĩ mắc lỗi.

5.1.3. Gửi dữ liệu

Dĩ nhiên, ví dụ trên là chưa hoàn thiện vì bạn chỉ có thể thấy được mô hình topo và chưa có hoạt động xãy ra, vì thế bước tiếp theo là gửi dữ liệu từ node n0 đến node n1. Trong ns, dữ liệu luôn được gửi từ một tác nhân (agent) đến tác nhân khác. Do đó, cần phải tạo ra một tác nhân để gửi dữ liệu từ node n0, một tác nhân khác nhận được dữ liệu trên node n1.

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

Hai lệnh đầu tạo ra tác nhân UDP (User Datagram Protocol:giao thức dữ liệu ngắn người dùng) và gán nó cho node n0. Những câu lệnh tiếp theo tạo ra bộ tạo dữ liệu tốc độ không đổi CBR (Constant Bit Rate:tần số bit bất biến) mà có thể tự động gửi các gói tin có độ dài 500 byte cách nhau 0.005 giây (tức là 200 gói tin/giây). Bạn có thể tìm thấy các thông số thích hợp cho mỗi tác nhân ở ns manual page

Dòng tiếp theo là tạo một tác nhân Null với tác động làm suy giảm lưu lượng gán nó vào vào node nút n1.

```
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

Hai tác nhân được kết nối với nhau theo câu lệnh:

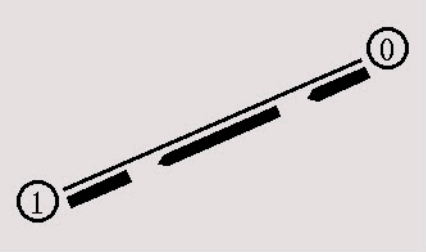
```
$ns connect $udp0 $null0
```

Và bây giờ, ta cần bảo trạm CBR khi nào thì gửi dữ liệu, khi nào thì dừng. Chú ý: Tốt nhất là đặt các dòng lệnh này ngay trước dòng `"$ns at 5.0 "finish"`

```
$ns at 0.5 "$cbr0 start"
```

Bây giờ bạn có thể lưu tập tin và bắt đầu mô phỏng lại. Khi bạn lick vào nút 'play' ở cửa sổ nam, bạn sẽ thấy 0.5 giây sau mô phỏng node 0 bắt đầu gửi gói dữ liệu tới node 1. Bạn có thể làm chậm nam bằng con trượt 'Step'

```
$ns at 4.5 "$cbr0 stop"
```



Hình 5.2: Gửi dữ liệu giữa n0 và n1

Như vậy bạn đã có một chút kinh nghiệm với nam và đoạn mã Tcl. Bạn có thể click bất kì gói dữ liệu trong cửa sổ nam để giám sát nó hay click trực tiếp lên các liên kết để có một số đồ thị thống kê. Bạn nên thay đổi tham số `'packetsize_'` và `'interval_'` trong đoạn mã Tcl để thấy được điều gì xảy ra.
Bạn có thể tải toàn bộ ví dụ tại đây hoặc tại đây
Hầu hết các thông tin cần thiết mà tôi cần để có thể viết đoạn mã Tcl này được lấy trực tiếp từ các tập tin ví dụ ở thư mục `'tcl/ex/'`, khi thiết lập các thông số CBR `(packetSize_, interval_)` tôi đã phải lấy từ ns manual page

5.2. Topo trong NS

Trong phần này chúng ta sẽ định nghĩa một mô hình với bốn node trong đó một node hoạt động như là một router chuyển dữ liệu mà hai node khác đang gửi đến node thứ tư. Tôi sẽ giải thích cách để phân biệt các luồng dữ liệu từ hai node này với nhau và tôi sẽ trình bày một hàng đợi có thể được giám sát như thế nào để có thể thấy được nó bị tràn như thế nào, và sẽ có bao nhiêu gói tin bị loại

5.2.1. Tạo topo trong NS

Như mọi khi, đầu tiên là xác định topo. Bạn nên tạo một ra tập tin 'example2.tcl' sử dụng mã này từ phần 5.1 như là một tập tin mẫu, Như tôi đã nói ở trên, đoạn mã này sẽ tương tự với mẫu trên. Bạn sẽ luôn cần phải tạo ra một đối tượng mô phỏng, luôn bắt phải đầu bằng cùng một câu lệnh và nếu muốn chạy nam tự động bạn sẽ luôn phải mở tập tin bám vết, khởi tạo nó, và định nghĩa thủ tục để đóng nó và khởi động nam.

Bây giờ chèn dòng lệnh vào trong mã để tạo ra bốn node.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

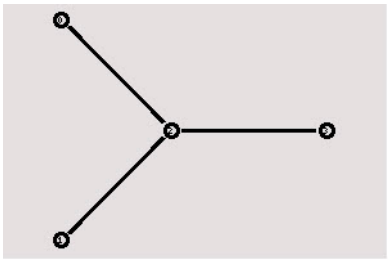
Bên dưới là ví dụ mã Tcl tạo ra ba liên kết song công giữa các nút.

```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
```

Bạn có thể lưu và khởi động đoạn mã bây giờ. Bạn có thể thấy rằng mô hình mạng có vẻ hơi rắc rối trong nam. Bạn có thể bấm vào nút 're-layout' để thấy rõ hơn nhưng sẽ tốt hơn nếu có thêm một chút điều chỉnh trong việc bố trí topo. Thêm 3 dòng sau vào đoạn mã Tcl của bạn và khởi động lại nó.

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

Bây giờ, bạn sẽ có thể hiểu đoạn mã này làm cái gì khi nhìn topo trong cửa sổ nam. Nó trông giống như hình bên dưới



Hình 5.3: Topo của đoạn mã gồm 4 nút

Chú ý rằng các phần liên quan đến bố trí tự động được bỏ qua, do nó đã bố trí theo ý mình. Tùy chọn cho hướng của các các liên kết là phải, trái, lên, xuống và sự kết hợp các hướng đó. Bạn có thể thử nghiệm với các cài đặt đó sau, nhưng bây giờ hãy thoát khỏi topo mạng theo đúng cách.

5.2.2. Tạo các sự kiện

Bây giờ chúng ta tạo ra hai tác nhân UDP với luồng tài nguyên CBR và gán chúng cho node n0 và n1. Rồi tạo ra tác nhân Null và gán nó cho node n3.

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
```

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

Hai tác nhân CBR cần phải được kết nối với tác nhân Null.

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

Chúng ta muốn tác nhân CBR thứ nhất bắt đầu gửi lúc 0.5 giây và kết thúc lúc 4.5 giây trong khi tác nhân CBR thứ hai bắt đầu lúc 1.0 giây và kết thúc lúc 4.0 giây.

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

Khi bạn bắt đầu đoạn mã 'ns example2.tcl' ngay bây giờ, bạn sẽ để ý rằng lưu lượng trên các kênh truyền từ n0 đến n2 và từ n1 đến n2 nhiều hơn lưu lượng mà kênh truyền từ n2 đến n3 có thể mang được. Một tính toán đơn giản để xác nhận việc này: Chúng tôi đang gửi 200 gói tin/giây trên mỗi kênh trong hai kênh truyền đầu tiên với kích thước gói tin là 500 byte. ĐIều này dẫn đến một băng thông 0.8 Mbps cho các kênh truyền từ n0 đến n2 và từ n1 đến n2.. Tổng băng thông là 1.6 Mb/giây nhưng kênh truyền giữa n2 và n3 chỉ có dung lượng là 1Mb/giây, vì thế hiển nhiên là vài gói tin bị loại ra. Nhưng gói tin nào bị loại? Cả hai luồng đều màu đen vì thế chỉ có thể thấy được điều gì đang diễn ra đối với gói

tin là giám sát chúng trong nam bằng cách click vào chúng. Tiếp theo của phần 2, tôi sẽ trình bày cách làm thế nào để phân biệt các luồng khác nhau và làm thế nào để thấy thật sự điều gì đang thực sự xãy ra ở hàng đợi tại kênh truyền từ n2 đến n3.

5.2.3. Đánh nhãn cho luồng dữ liệu

Thêm vào hai dòng lệnh bên dưới để xác định tác nhân CBR của bạn.

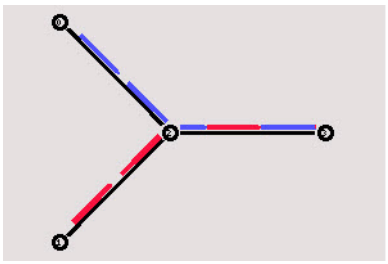
```
$udp0 set class_ 1
$udp1 set class_ 2
```

Tham số `'fid_'` là viết tắt của `'flow id'`.

Bây giờ thêm vài câu lệnh vào tập lệnh Tcl, tốt hơn là ở lúc bắt đầu, ngay sau khi tạo đối tượng mô phỏng, do đây là một phần của sự thiết lập mô phỏng

```
$ns color 1 Blue
$ns color 2 Red
```

Đoạn mã này cho phép thiết lập các màu sắc khác nhau của mỗi luồng.



Hình 5.4: Màu sắc các luồng

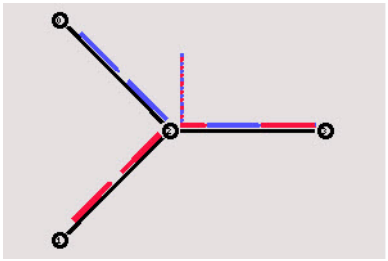
Bây giờ bạn có thể khởi động lại tập lệnh và một luồng có màu xanh, luồng kia màu đỏ. Khi theo dõi kết nối từ node n2 đến node n3 và bạn sẽ để ý rằng sau một thời gian truyền thì các phân bố giữa các gói tin màu xanh và màu đỏ không còn cân bằng nữa (chí ít là trên hệ thống của tôi). Trong phần tiếp theo tôi sẽ trình bày cho bạn làm thế nào để có thể theo dõi bên trong hàng đợi của liên kết này để tìm hiểu chuyện gì đang xãy ra ở đó.

5.2.4. Giám sát hàng đợi

Bạn chỉ cần thêm dòng lệnh bên dưới vào đoạn mã của bạn để giám sát hàng đợi cho kết nối từ n2 tới n3

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

Khởi động ns lại và bạn sẽ thấy một hình ảnh tương tự bên dưới sau một thời gian ngắn

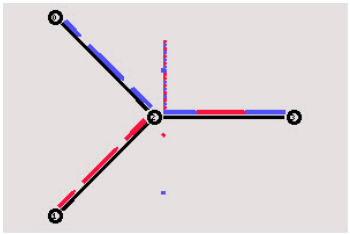


Hình 5.5: Giám sát hàng đợi

Bây giờ bạn có thể thấy các gói tin trong hàng đợi và sau đó có thể thấy các gói tin bị loại bỏ như thế nào và đi qua như thế nào, mặc dù chỉ có gói tin màu xanh bị loại (chí ít trong hệ thống của tôi, tôi đoán là có thể khác nhau trong phiên bản sau hoặc trước đó). Nhưng bạn không thể trong đợi quá nhiều vào "sự công bằng" từ một hàng đợi DropTail đơn giản. Vì thế hãy cố cải thiện bằng cách sử dụng hàng đợi SFQ (stochastic fair queueing: hàng đợi công bằng ngẫu nhiên) cho kết nối từ n2 đến n3. Thay đổi định nghĩa cho liên kết giữa n2 và n3 với dòng lệnh bên dưới

```
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
```

Bây giờ, việc xếp hàng có lẽ là "cân bằng" rồi. Số lượng màu xanh và màu đỏ bị loại là như nhau.



Hình 5.6: Cân bằng trong giám sát hàng đợi

Bạn có thể tải ví dụ đầy đủ ở đây hoặc ở đây

5.3. Mạng có tính chất động

Trong phần này tôi sẽ trình bày cho bạn một ví dụ về mạng động ở đó việc định tuyến chỉnh sửa lại các kênh truyền bị lỗi. Bằng cách này tôi sẽ chỉ cho bạn có thể giữ số lượng lớn các nút trong một mảng Tcl thay vì cấp cho mỗi nút một cái tên riêng.

5.3.1. Tạo một Topo lớn hơn

Tôi gợi ý cho bạn gọi tập lệnh Tcl trong ví dụ này là `'example3.tcl'`. Bạn có thể chèn vào mẫu tại đây hoặc tại đây, trong phần 5.1

Thông thường, trước tiên, một topo được tạo ra, mặc dù lần này chúng tôi có hướng tiếp cận khác mà bạn sẽ cảm thấy dễ dàng hơn khi muốn tạo các mô hình mạng lớn. Đoạn mã bên dưới tạo ra bảy nút và lưu chúng trong mảng `n()`

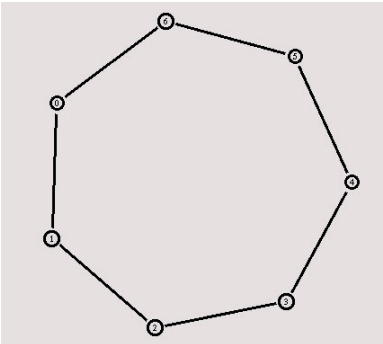
```
for {set i 0} {$i < 7} {incr i} {
  set n($i) [$ns nút]
}
```

Các bạn đã từng gặp vòng lặp "for" ở các ngôn ngữ lập trình khác trước đây, và tôi chắc rằng các bạn hiểu cấu trúc này một lần nữa. Để ý những mảng đó, cũng như các biến khác trong Tcl, chúng không cần phải được khai báo đầu tiên . Bây giờ chúng ta sẽ kết nối các nút thành một topo vòng tròn. Ban đầu dòng mã bên dưới trông có vẻ phức tạp hơn một chút.

```
for {set i 0} {$i < 7} {incr i} {
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
```

Vòng lặp "for" kết nối tất cả các nút với nút tiếp theo trong mảng trừ nút cuối cùng, nút này sẽ kết nối với nút đầu tiên.. Để thực hiện điều đó, tôi sử dụng toán tử %

Bây giờ, khi bạn cho chạy đoạn mã, đầu tiên topo có vẻ lạ trong NAM nhưng sau khi bạn bấm vào nút "re-layout" thì nó sẽ trông giống hình bên dưới.



Hình 5.7: Mô hình gồm 7 nút

5.3.2. Liên kết lỗi

Bước tiếp theo là gửi dữ liệu từ nút n0 đến nút n3.

```
#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

Giờ đây, đoạn mã ở trên có lẽ đã quen thuộc với bạn. Chỉ khác với các phần trước là bây giờ chúng ta sử dụng các thành phần mảng là các nút.

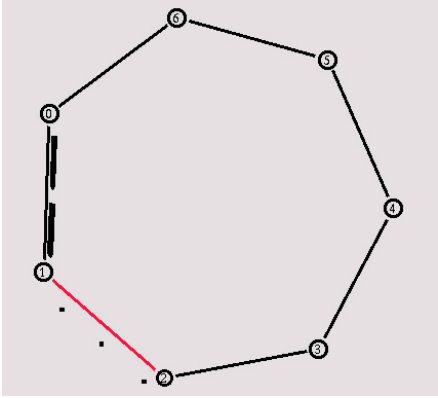
Nếu bạn khởi động tập lệnh, bạn sẽ thấy rằng lưu lượng sẽ chọn đường ngắn nhất từ nút 0 tới nút 3 đi qua nút 1 và 2 đúng như mong đợi. Bây giờ chúng tôi thêm một tính năng thú vị khác. Chúng ta hãy làm cho liên kết giữa nút 1 và 2 bị đứt trong 1 giây.

```
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
```

Có lẽ không quá khó để hiểu hai dòng lệnh trên. Bây giờ bạn chạy lại tập lệnh và bạn sẽ thấy rằng từ giây thứ nhất đến giây thứ 2 kênh truyền sẽ bị đứt và tất cả dữ liệu đã được gửi đi từ nút 0 bị mất.

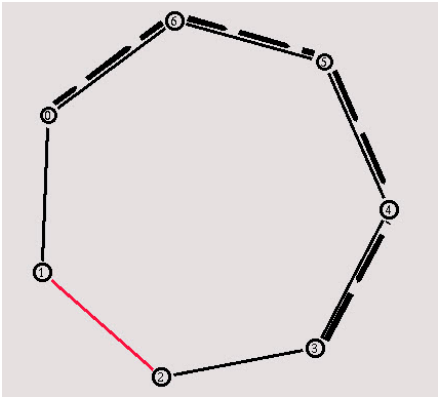
Bây giờ tôi sẽ cho bạn thấy làm thế nào để sử dụng định tuyến động để giải quyết vấn đề trên. Thêm vào dòng lệnh bên dưới vào phần đầu của đoạn mã Tcl của bạn sau khi đối tượng mô phỏng được tạo ra.

```
$ns rtproto DV
```



Hình 5.8: Minh họa liên kết bị lỗi

Bắt đầu mô phỏng lại, bạn sẽ thấy đầu tiên có rất nhiều gói tin nhỏ truyền qua mạng. Nếu bạn làm chậm nam xuống đủ để click vào một trong số chúng bạn sẽ thấy rằng các chúng là các gói tin `'rtProtoDV'` được sử dụng để thay đổi thông tin định tuyến giữa các nút. Khi kênh truyền bị đứt trở lại lúc 1 giây, việc định tuyến sẽ được cập nhật và lưu lượng sẽ được định tuyến lại đi qua nút 6, 5, và 4.



Hình 5.9: Lưu lượng truyền theo nút 6, 5, 4

Bạn có thể tải chương đầy đủ này tại đây hoặc tại đây để tham khảo

Chương 6

Mô phỏng và mô tả TCP/IP

Mục lục

Người dịch: *Nguyễn Thúy Vân*

Biên tập: *Hoàng Trọng Minh*

Bản gốc: *NS Simulator for beginners, Chapter 4* [2]

TCP (Transport Control Protocol) là một giao thức truyền tải chịu trách nhiệm truyền khoảng 90% lưu lượng dữ liệu trên Internet. Chính vì vậy, nắm vững và hiểu rõ TCP đóng

vai trò vô cùng quan trọng để định cỡ lưu lượng trong Internet. Mặc dù TCP đã được triển khai rộng khắp, nó vẫn phát triển không ngừng. IETF (Internet Engineering Task Force) là tổ chức tiêu chuẩn hóa chính liên quan tới giao thức TCP. Không giống như các tổ chức chuẩn hóa khác (ITU hoặc diễn đàn ATM) tất cả các tiêu chuẩn của IETF miễn phí và có sẵn trên mạng.

Ở phần đầu tiên, ta sẽ mô tả hoạt động của TCP. Còn ở các phần sau đó , ta xem xét một số đoạn mã NS mô tả phân tích TCP thông qua mô phỏng

6.1. Mô tả TCP

6.1.1. Các mục đích của TCP và điều khiển luồng theo cơ chế cửa sổ

TCP nhằm một vài mục đích sau:

- Điều chỉnh tốc độ truyền các gói dữ liệu sao cho phù hợp với băng thông khả dụng.
- Tránh hiện tượng nghẽn mạng.
- Tạo kết nối tin cậy bằng cách truyền lại các gói tin bị thất lạc.

Để điều khiển tốc độ truyền, số lượng các gói tin chưa được nhận (hay nói một cách chính xác hơn là những gói tin mà nguồn chưa nhận được thông tin phản hồi từ đích là đã nhận được) bị giới hạn bởi một tham số gọi là một cửa số tắc nghẽn. Chúng ta đặt tên là “W” , tuy nhiên trong ngữ cảnh của TCP nó ký hiệu là cwnd. Điều này có nghĩa là nơi truyền các gói tin buộc phải chờ và ngừng truyền các gói tin mà nó đã truyền trước đó và các gói tin ko được xác nhận này có tên là “W”. Để các gói tin này được xác nhận và có khả năng truyền lại các gói tin bị thất lạc, mỗi gói tin được truyền đi phải có một số thứ tự.

6.1.2. Các bản tin xác nhận

Mục tiêu của các bản tin xác nhận gồm

- Điều chỉnh tốc độ truyền của TCP, đảm bảo rằng các gói tin chỉ được truyền đi khi mà các gói tin khác đã được truyền ra khỏi mạng.
- Tạo đường kết nối đáng tin cậy bằng cách truyền tới nút nguồn các thông tin cần thiết để truyền lại các gói tin không đến được đích.

Làm thế nào mà đích đến của gói tin biết là gói tin đó có bị thất lạc ?
Làm thế nào mà ta biết được một gói tin bị mất?
Gói ACK chứa thông tin gì?

Gói tin ACK cho nguồn gửi tin biết số thứ tự của gói tin mà nó chờ đợi. Ví dụ sau sẽ minh họa cho điều này. Giả sử gói tin 1,2,...,6 đến đích theo thứ tự. Khi gói tin số 6 đến, đích gửi một gói tin ACK thông báo rằng nó đang đợi gói tin số 7. Nếu gói tin số 7 đến, đích yêu cầu gói tin số 8. Giả sử gói tin số 8 bị thất lạc, gói tin số 9 có thể đến đích được. Tại thời điểm đó, đích gửi gói tin ACK có tên là " ACK lặp" thông báo với nguồn gửi tin rằng nó chờ gói tin số 8. Thông tin trong gói tin ACK này tương tự như gói tin ACK đã gửi trước đó.

Phương pháp này có tên là "ACK ẩn" (implicit ACK) . Phương pháp này hiệu quả ngay cả khi mất các gói tin ACK. Hơn nữa, giả sử rằng gói tin ACK thông báo rằng đích đợi gói tin số 5 bị mất, Khi gói tin ACK tiếp theo đến và thông báo rằng đích đang đợi gói số 6, nút nguồn hiểu rằng đích đã nhận được gói số 5, vì vậy thông tin gửi bởi ACK đã mất được suy ra từ gói tin ACK kế tiếp.

Gói tin TCP bị coi là mất nếu như:

- Có 3 gói tin ACK giống nhau thông tin cho nguồn[1] về cùng một gói tin được gửi đi.
- Khi một gói tin được truyền đi, có một đồng hồ đếm thời gian. Nếu gói tin ACK của gói tin đã được truyền ko đến trong khoảng thời gian T_0 , có nghĩa là " quá hạn" (Time-out) và gói tin đó coi như bị mất.

Quá trình truyền lại sau khi lặp lại 3 gói tin ACK được gọi là "truyền lại nhanh" (fast retransmit)

Chọn thời gian T_0 như thế nào? Nguồn ước lượng thời gian quay vòng RTT trung bình, hay nói cách khác là thời gian cần thiết cho một gói tin đến đích cộng với thời gian cho gói tin ACK quay trở lại thông báo cho nguồn. Nó cũng đánh giá sự biến động của RTT. T_0 được tính toán như sau:

$$T_0 = \overline{RTT} + 4D$$

$\overline{RTT}$ là ước lượng hiện tại của RTT còn D là sự biến động của RTT. Để ước lượng RTT, ta tính toán sự chênh lệch M giữa thời gian truyền 1 gói tin và thời gian gói tin ACK của nó quay trở lại. Sau đó tính:

$$\overline{RTT} \leftarrow a \times \overline{RTT} + (1-a)M,$$

$$D \leftarrow aD + (1-a)\left|\overline{RTT} - M\right|$$

[1] ...

Để giảm số lượng gói tin ACK trong hệ thống, TCP thường xuyên sử dụng lựa chọn " ACK trễ" (delayed ACK) tại đó 1 gói tin ACK được truyền chỉ cho các gói tin d đến đích. Giá trị chuẩn của d là 2 (xem RFC 1122). Tuy nhiên, trễ 1 gói tin ACK đến tận khi $d > 1$ gói tin được nhận có thể dẫn đến việc đình trệ hoàn toàn trong trường hợp kích thước cửa sổ nhỏ. Vì vậy, nếu gói tin đầu tiên (nằm trong số d gói tin đang được chờ đợi) đến đích, sau một khoảng thời gian ngắt quãng (thường là 100ms) nếu d gói tin vẫn chưa đến thì một bản tin ACK được tạo ra mà không cần chờ đợi thêm nữa.

6.1.3. Cửa sổ chống tắc nghẽn động

Từ đầu những năm 80, trong suốt vài năm, TCP luôn có 1 cửa sổ chống tắc nghẽn cố định. Các mạng tại thời điểm đó ko ổn định, nhiều gói tin bị mất, Các chu kỳ nghẽn mạng lớn và rắc rối, trong suốt thời gian nghẽn thì thông lượng giảm đáng kể, nhiều gói tin phải truyền lại và thời gian trễ lớn. Để giải quyết vấn đề này, Van Jacobson [25] đã đề xuất sử dụng cửa sổ chống tắc nghẽn động. Kích thước của cửa sổ thay đổi tùy theo trạng thái của mạng. Về cơ bản, ý tưởng là: Khi cửa sổ nhỏ, kích thước của nó có thể tăng nhanh chóng , nhưng khi đạt đến một giá trị nhất định nào đó, kích thước của nó chỉ có thể tăng chậm. Khi hiện tượng nghẽn mạng được phát hiện, kích cỡ của cửa sổ giảm mạnh. Cơ chế động này có phép giải quyết tắc nghẽn nhanh chóng và sử dụng hiệu quả băng thông của mạng.

Chính xác hơn, định nghĩa ngưỡng W_{th} , hay tên gọi khác là "ngưỡng bắt đầu chậm" nó sẽ thể hiện đánh giá của chúng ta về dung lượng mạng. Cửa sổ bắt đầu ở giá trị 1. 1 gói tin đơn lẻ được truyền. Khi gói tin ACK của gói tin này quay trở lại, ta có thể truyền 2 gói tin. Đối với mỗi gói tin ACK của 2 gói tin này, cửa sổ tăng lên 1 giá trị, mục đích là khi các gói tin ACK của 2 gói tin này quay trở lại, ta có thể truyền 4 gói tin. Ta nhận thấy là cửa sổ tăng theo số mũ. Giai đoạn này có tên là "bắt đầu chậm" (slow start). Sở dĩ nó có tên như thế là bởi lẽ thay vì cửa sổ to dần lên nhanh chóng, nó chỉ lớn lên từ từ nếu ta bắt đầu với một giá trị $W = W_{th}$.

Nếu $W = W_{th}$, ta chuyển qua giai đoạn 2 có tên gọi là giai đoạn "tránh tắc nghẽn" . Khi đó, mỗi gói tin ACK quay lại sẽ làm cửa sổ W tăng lên $\lfloor 1/W \rfloor$. Sau khi truyền W gói tin, W tăng lên 1. Nếu ta truyền W gói tin cần t thời gian, thì truyền $W + 1$ cần $t + RTT$ và $W + 2$ cần $t + 2RTT$, ...v.v. Ta thấy là độ lớn của cửa sổ thay đổi tuyến tính.

6.1.4. Mất các gói tin và ngưỡng W_{th} động:

Không chỉ W động mà W_{th} cũng như vậy. Nó được cố định trong TCP khi một gói mất thì giá trị của W giảm đi một nửa.

Có một số phương pháp cải thiện TCP. phương pháp thứ nhất có tên là "Tahoe", tức là

bất kể khi nào phát hiện ra một gói tin bị mất thì cửa sổ giảm giá trị xuống 1 đơn vị và giai đoạn slow start bắt đầu. Điều này làm giảm kích cỡ cửa sổ nhanh chóng và cũng như vậy với tốc độ truyền tin.

Một phương pháp cải thiện khác có tên là Reno hay New- Reno, cửa sổ giảm xuống 1 nếu như phát hiện ra gói tin bị mất trong thời gian quá hạn. Khi một gói tin mất bị phát hiện ra thông qua các gói tin ACK bị lặp, cửa sổ giám sát tắc nghẽn giảm xuống 1/2. Giai đoạn slow start sẽ ko khởi tạo và ta vẫn ở trạng thái "tránh tắc nghẽn".

6.1.5. Khởi tạo kết nối

Để khởi tạo một kết nối TCP, nguồn gửi một gói tin "đồng bộ" (sync packet) dung lượng 40 byte đến đích. Sau đó đích gửi 1 gói tin ACK (cũng có độ dài 40 gói tin gọi là gói tin xác nhận đồng bộ (sync ACK)). Khi nguồn nhận gói tin ACK này, TCP có thể bắt đầu gửi dữ liệu. Chú ý rằng: nếu có bất kì gói tin nào nằm trong số những gói tin này bị mất sau thời gian quá hạn (thường là từ 3-6s) thì nó sẽ được truyền lại. Khi gói tin được truyền lại bị mất, thời gian quá hạn tăng lên gấp đôi và gói tin lại được gửi lại một lần nữa.

6.2. Quá trình bám vết và phân tích ví dụ Ex1.tcl

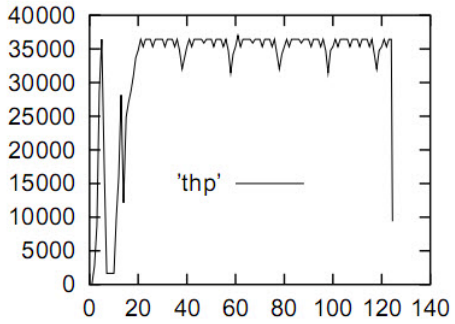
Hãy chạy file chương trình viết bằng ngôn ngữ perl " throughput.pl" (bảng **??**) trên file bám vết (trace file) tạo ra ở đoạn mã ex1.tcl (xem bảng 4.4). Ta phải gõ: perl throughput. Pl out.tr 4 1 > thp

Ta có một file đầu ra với thông lượng TCP trung bình đạt được (byte/giây) là một hàm của thời gian , trong trường hợp này, mỗi giây một lần, một giá trị mới của độ thông qua được tạo ra. File đầu ra có thể được thể hiện thông qua việc sử dụng lệnh gnuplot bằng cách gõ:

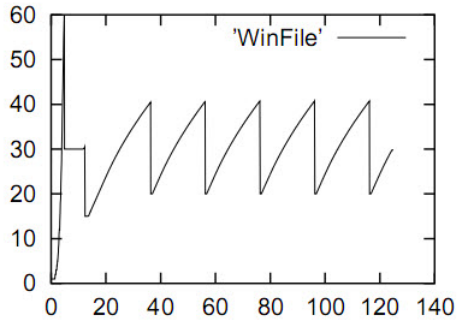
```
Gnuplot
Set size 0,4.0,4
Set key 60,15000
Plot 'thp' w lines 1.
```

Kết quả thể hiện ở hình 6.1

Để hiểu tốt hơn về hành vi của hệ thống, ta thể hiện kích thước cửa sổ bằng biểu đồ (hình 6.2). Đây là file "Winfile" tạo ra khi chạy đoạn mã ex1.tcl. Ta nhận thấy là từ thời điểm 20 trở đi, chế độ quay vòng trạng thái ko đổi của TCP được ổn định: TCP luôn ở giai đoạn tránh tắc nghẽn, và kích thước cửa sổ của nó tăng đều (theo qui luật hàm tuyến



Hình 6.1: Thông lượng kết nối TCP



Hình 6.2: Kích thước cửa sổ TCP

tính) cho đến khi xảy ra nghẽn mạng. Trước thời điểm 20, ta thấy TCP rơi vào giai đoạn slow-start trong thời gian rất ngắn. Ở thời điểm 4.2, tại giai đoạn slow-start xảy ra hiện tượng mất gói tin. Cửa sổ còn 1 nửa trong khi đó thông lượng gần như về 0. Ta giải thích hiện tượng này như thế nào? Lí do là tại thời điểm 4.2 có một khoảng thời gian quá hạn, chính vì lẽ đó, mặc dù kích thước của cửa sổ là 30 (gói tin) nhưng chẳng có hoạt động truyền tin nào diễn ra. Ở thời điểm 11, lại xảy ra hiện tượng mất gói tin trong giai đoạn slow-start.

6.3. TCP trên liên kết nhiễu và việc giám sát hàng đợi

Ở các ví dụ trước, gói tin bị mất là do tắc nghẽn. Thực tế, gói tin bị mất có thể là do nhiễu đường kết nối. Điều này đặc biệt đúng trong trường hợp sóng vô tuyến, ví dụ như đường truyền điện thoại hoặc đường truyền vệ tinh. Thực tế là đường kết nối có thể bị ngắt hoàn toàn trong giai đoạn nào đó. Ta sẽ xem xét vấn đề này sau đây tại mục 6.1. Hoặc cũng có thể kênh truyền bị nhiễu từ hiện tượng xuyên nhiễu cũng khiến cho gói tin bị lỗi và loại bỏ. Ở mục này, ta sẽ nêu cách đưa ra mô hình lỗi đơn giản nhất: Ta thừa nhận rằng các gói tin bị loại bỏ độc lập trên đg truyền theo một xác suất cố định nào đó.

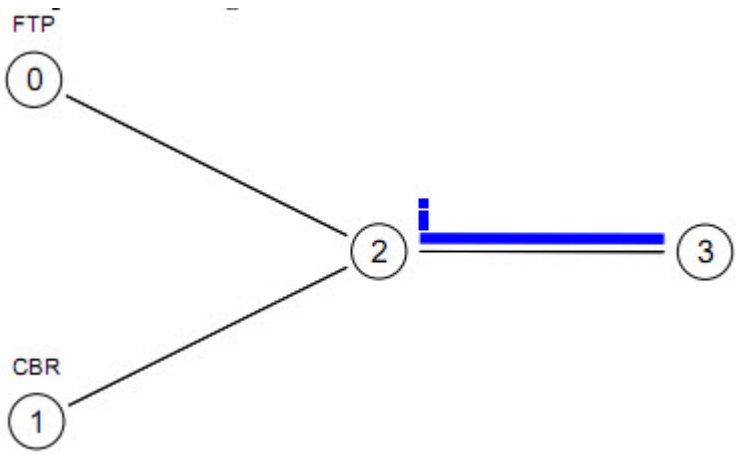
Mô hình lỗi liên kết được nêu ra ở nút kết nối n3 và n2 trong ví dụ ở hình 6.3 được tạo ra như sau:

```
#Set error model on link n2 to n3
Set loss_module [new ErrorModel]
$loss_module set rate _0.2
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
```

```
$ns lossmodel $loss_module $n2 $n3
```

Lệnh `$loss_module set rate_0.2` xác định tỉ lệ mất gói tin là 20% . Nó sử dụng 1 bộ tạo biến ngẫu nhiên được phân bố đều. Điều này được thể hiện ở dòng tiếp theo. Dòng cuối cùng quyết định xem liên kết nào bị ảnh hưởng.

Ta xem xét mạng được mô tả tại hình 6.3. Trong ví dụ này, kết nối TCP chia sẻ một kênh tạp âm với kết nối UDP



Hình 6.3: Ví dụ rdrop.tcl

Queue Mornitoring Một đối tượng quan trọng của ns là hàng đợi được giám sát. Đối tượng này cho phép thu thập các thông tin hữu ích về độ dài hàng đợi, về các gói tin đến, điểm đến và các gói tin bị mất. Để tiến hành việc giám sát hàng đợi giữa nút 2 và nút 3 ta gõ:

```
set qmon [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1];!
[$ns link $n2 $n3] queue-sample-timeout;
# [$ns link $n2 $n3] start-tracing!
```

Mục tiêu của quá trình giám sát hàng đợi gồm 4 tham số: 2 tham số đầu xác định kênh truyền chứa vị trí của hàng đợi, tham số thứ 3 là cho ra file bám vết và tham số cuối cùng cho biết xem mức độ thường xuyên mà ta mong muốn NS giám sát hàng đợi. Trong tình huống trên, đầu vào là hàng đợi ở đầu vào giữa nút 2 và nút 3 sẽ được giám sát 0.1s lần và kết quả đầu ra sẽ là file qm.out. File đầu ra gồm11 cột sau đây:

- Thời gian
- Nút đầu vào và đầu ra xác định hàng đợi

- Kích thước hàng đợi tính theo byte (tương ứng với thuộc tính `size_` của đối tượng monitor-queue).
- Kích thước hàng đợi tính theo gói tin (tương ứng với thuộc tính `pkt_`)
- Số lượng gói tin vừa đến (tương ứng với thuộc tính `parrivals_`)
- Số lượng gói tin rời khỏi liên kết (tương ứng với thuộc tính `pdeparture_`)
- Số lượng gói tin bị loại bỏ tại hàng đợi (tương ứng với thuộc tính `pdrop_`)
- Số lượng byte đến (tương ứng với thuộc tính `barrivals_`)
- Số lượng byte rời khỏi liên kết (tương ứng với thuộc tính `bdepartures_`)
- Số lượng byte bị loại bỏ (tương ứng với thuộc tính `bdrops_`)

Một cách khác để tìm hiểu trực tiếp các thuộc tính này được nêu ở mục 4.5 Ví dụ tiếp theo ở đoạn mã 4.1 mô tả toàn bộ đoạn mã tạo mẫu mô hình TCP với các gói tin bị loại bỏ do nhiễu.

```
# Tạo một ví dụ mô phỏng
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red
# Mở 1 file bám vết NAM
set nf [open out.name w]
$ns namtrace-all $nf
# Mở 1 file bám vết
set tf [open.tr w]
set windowVsTime2 [open WindowVsTimeNReno w]
$ns trace-all $nf
#Định nghĩa 1 quá trình "hoàn tất"
proc finish {}{
global ns nf tf
$ns flush-trace
close $nf
close $tf
exac nam out.nam &
exit 0
}
```

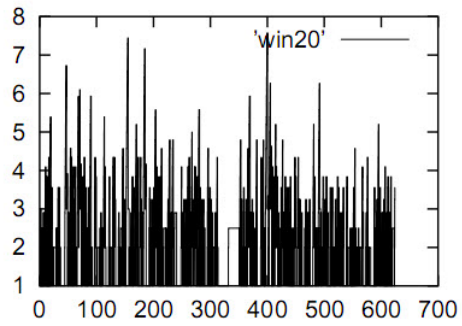
```
#Tạo 4 nút
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns at 0.1 “$n1 label \"CBR\""
$ns at 1.0 “$n0 label \"FTP\""
#Tạo đường link giữa các nút
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simpex-link $n2 $n3 0.07Mb 20ms DropTail
$ns simpex-link $n3 $n2 0.07Mb 20ms DropTail
# Đặt kích thước hàng đợi của liên kết (n2-n3) tới 10
$ns queue-limit $n2 $n3 10
# Giám sát hàng đợi cho liên kết giữa (ns-n3). (dành cho NAM).
$ns simplex-link-op $n2 $n3 queuePos 0.5

#Tạo mô hình lỗi trên liên kết giữa n3 và n2.
Set  loss_module [new ErrorModel]
$loss_module set rate_ 0.2
$loss_module ranvar [new  RandomVariable/Unionform]
$loss_module drop-target [new Agen/Null]
$ns lossmodel $loss_module $n2 $n3
#Thiết lập một kết nối TCP
Set  tcp [new Agen/TCP/Newreno]
$ns attach-agent $n0 $tcp
Set sink [new Agen/TCPsink/Delack]
$ns attach-agen $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Thiết lập một FTP thông qua kết nối TCP
Set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
#Thiết lập một kết nối UDP
Set UDP [new Agent/UDP]
```

```
$ns attach-agen $n1 $udp
Set null [new Agent/Null]
$ns attach-agent  $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Thiết lập một CBR thông qua kết nối UDP
Set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packetSize_ 1000
$cbr set rate_ 0.01Mb
$cbr set random_ false
#Lập lịch các sự kiện cho các tác nhân CBR và FTP
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns  at 624.0 ‘‘$ftp stop’’
$ns  at 624.5 ‘‘$fcbr stop’’

#in ra kích thước cửa sổ
Proc  plotWindow {tcpSource} {
Global ns
Set time 0.01
Set now [$ns now]
Set cwnd [tcpSource set cwnd_]
Puts $file ‘‘$now $cwnd’’
$ns at [expr $now+$time] ‘‘plotWindow $tcpSource $file’’}
$ns at 1.1 ‘‘plotWindo $tcp $windowVsTime2’’

# Lấy mẫu hàng đợi nút thắt cổ chai 0.1s một lần . Lưu file bám vết trong qm.out
Set qmon [$ns monitor-queue $n2 $n3 [opne qm.out w] 0.1];
[$ns link $n2 $n3] queue-sample-timeout;
# [$ns link $n2 $n3] start-tracing

#Tách các tác nhân tcp và sink (bộ chứa, bộ nhận)
$ns at 624.5 ‘‘$ns detach-agent $n0 $tcp’’
$ns detach-agent $n3 $sink’’
```

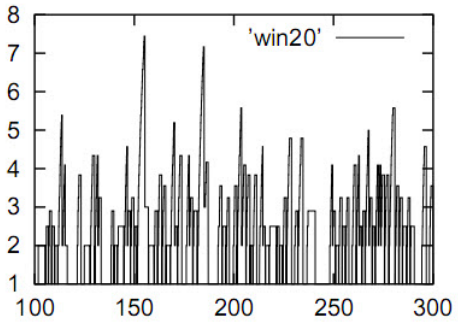
```
$ns at 625.0 ‘‘finish’’
$ns run
```

Bảng 6.1: Định nghĩa một ứng dụng FTP sử dụng tác nhân TCP

Trong hình 6.4 Ta sử dụng gnuplot để bám vết file do quá trình mô phỏng tạo nên. File này có tên gọi là WindowVsTimeNReno. Hình ảnh phóng ra của nó được mô tả ở hình 6.5 .

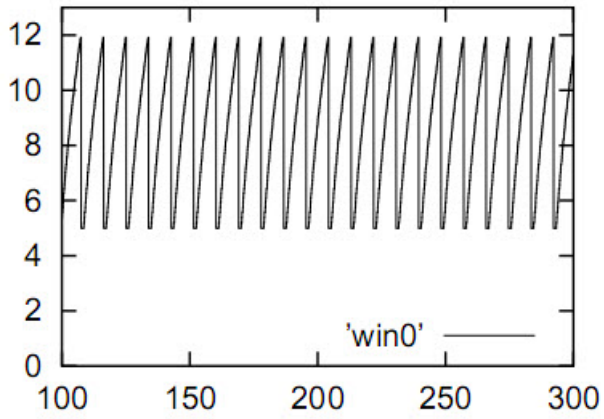


Hình 6.4: Kích thước cửa sổ TCP với 20% gói tin mất ngẫu nhiên



Hình 6.5: Kích thước cửa sổ TCP với 20% gói tin mất ngẫu nhiên: đã phóng to

Trong một vài trường hợp, Ta khảo sát thời gian quá hạn dài, khoảng 300. Để xem xem các gói tin ngẫu nhiên bị mất ảnh hưởng lớn như thế nào đến hiệu suất hoạt động của TCP, Ta thực hiện mô phỏng lại nhưng ko có các gói tin bị mất. Kết quả được mô tả tại hình 6.6



Hình 6.6: Cửa sổ TCP khi tỉ lệ mất gói là 0

Một cách đo hiệu năng quan trọng là thông lượng trung bình của TCP. Một cách đơn giản để tính toán là tìm trong file bám vết out.tr thời điểm mà đích nhận một gói tin TCP (tại nút 3). Khi thực hiện mô phỏng, Ta nhận thấy là thời điểm này là 624.08754 và file bám vết tương ứng là

```
R 624.0827 2 3 tcp 1000------- 1 0.0 3.0 1562 4350
```

Số trước số cuối cùng cho Ta biết rằng: gói tin mà đích nhận được là gói TCP số 1562. Theo đó, thông lượng TCP đơn giản là số bị chia ra bởi khoảng thời gian kết nối FTP (623s), nghĩa là `2.507 gói tin/s`, hoặc tương đương với `2.507 Kbyte/s` hay `20058 bps`) (khi một gói tin TCP mặc định có dung lượng là 1000 byte).

Lưu ý: Nếu ta nhìn vào dòng đầu tiên của file out.tr, Ta sẽ thấy là có các gói TCP khác (mỗi gói có kích cỡ 40 gói tin) mà Ta ko tính đến trong tổng số 1562 gói tin. Số seri của chúng là 0. Ta ko tính các gói tin này vì chúng tương ứng với các gói tin báo hiệu (signalling) liên quan đến lúc bắt đầu kết nối TCP.

Một điều cần chú ý nữa là Ta đã sử dụng phiên bản Ack bị trễ của TCP qua việc sử dụng lệnh:

```
Set sink [new Agent/TCPSink/DelAck]
Instead of simply set sink [new Agen/TCPSink]
```

6.4. Tạo nhiều kết nối với các đặc tính ngẫu nhiên

Để tạo nhiều kết nối, thay vì xác định từng nút, kênh truyền, kết nối hoặc các ứng dụng đơn lẻ, sẽ hữu ích hơn nếu xác định chúng như các vectơ (hay mảng) trong tcl (trong các câu lệnh lặp)

Hơn thế, việc lựa chọn các thông số kết nối (chẳng hạn như thời gian bắt đầu hoặc kết thúc hoạt động, trễ liên kết, vv) sẽ trở nên thú vị nếu ta chọn theo cách ngẫu nhiên. Ta sẽ xem xét cả 2 vấn đề trong mục này và sau đó sẽ đưa ra 1 ví dụ minh họa. Lưu ý rằng Ta cũng đã vừa xem xét các khía cạnh khác nhau của yếu tố ngẫu nhiên ở mục 6.3

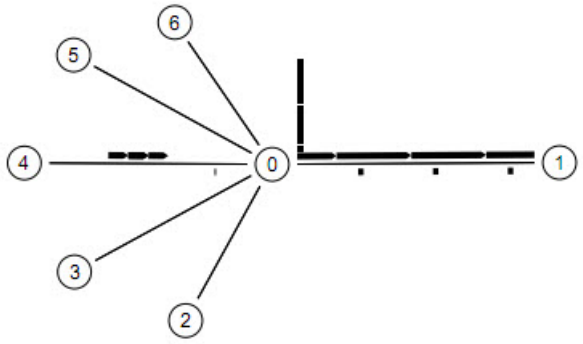
Ví dụ: Xem xét mạng ở hình 6.7. Đoạn mã được đưa ra ở bảng 6.2.

Ta tạo 5 kết nối FTP bắt đầu một cách ngẫu nhiên: thời gian bắt đầu là phân bố đồng dạng trong khoảng 0-7s. Toàn bộ quá trình mô phỏng diễn ra trong 10s. Ta tạo liên kết với độ trễ được chọn lựa ngẫu nhiên, phân bố đều trong khoảng từ 1-5s.

Cùng với file bám vết theo tiêu chuẩn được tạo ra, Ta cũng tạo thêm 1 file có tên là "win". File này cho biết sự thay đổi kích cỡ cửa sổ của tất cả kết nối với độ chi tiết là 0.03s.

Thực hiện điều này bằng cách dùng thủ tục plotWindow. Chú ý là file "win" được định địa chỉ bằng cách sử dụng con trỏ "windowVsTimes" Thủ tục này gọi môt cách đệ qui cho

từng kết nối một trong 5 kết nối đã tạo ra.



Hình 6.7: Ví dụ về mạng với một số kết nối TCP

```
#Create the simulator instance
set ns [new Simulator]

#Opening the trace files
set nf [open out.nam w]
$ns namtrace-all $nf

set tf [open out.tr w]
set windowVsTime [open win w]
set param [open parameters w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
global ns nf tf
$ns flush-tracert
close $nf
close $tf
exec nam out.nam &
exit 0
}

#Create bottleneck and dest nodes and link between them
```

```
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n2 $n3 0.7Mb 20ms DropTail

set NumbSrc 5
set Duration 10

#Source nodes
for {set j 1} {$j<=$numbSrc} {incr j} {
set S($j) [$ns node]
}

# Create a random generator for starting the ftp
# and for starting the ftp and for bottleneck link delays
set rng [new RNG]
$rng seed 0

#  parameters for random variables for delays
set RVdly [new RandomVariable/Uniform]
$RVdly set min_ 1
$RVdly set max_ 5
$RVdly use-rng $rng

# parameters for random variables for beginning of ftp connections
set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng
}
#We define two random parameters for each connection
for {set i 1} {$i<=$NumbSrc} { incr i } {
set startT ($i) [expr [$RVstart value]]
set dly)$i) [expr [$RVdly value]]
puts $param ‘‘dly($i) $dly($i) ms’’
puts $param ‘‘startT($i) $startT($i) sec’’ }

#Links between source and bottleneck
For {set j 1} {$j<=$NumbSrc} { incr j} {
```

```
$ns duplex-link $S($j) $n2 10Mb $dly($j)ms DropTail
$ns queue-limit $S($j) $n2 100
}

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#TCP Sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set tcp_src($j) [new Agent/TCP/Reno]
}

#TCP Destinations
For {set j 1} {$j<=$NumbSrc} { incr j} {
Set tcp_snk($j) [new Agent/TCPSink]
}

#Connections
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns attach-agent $S($j) $tcp_src($j)
$ns attach-agent $n3 $tcp_snk($j)
$ns connect $tcp_src($j) $tcp_snk($j)
}

#FTP sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
set ftp($j) [$tcp_src($j) attach-source FTP]
}

#Parametrisation of TCP sources
for {set j 1} {$j<=NumbSrc} { incr j } {
$tcp_src($j) set packetSize_ 552
}

#Schedule events for the FTP agents:
```

```
for {set i 1} P$i<=$NumbSrc} { incr i } {
$ns at $startT($i) ‘‘ftp($i) start’’
$ns at $Duration ‘‘$ftp($i) stop’’
}

proc plotWindow {tcpSource file k} {
global ns

set time 0.03
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file ‘‘$now $cwnd’’
$ns at [expr $now+$time] ‘‘plotWindow $tcpSource $file $k’’ }

# The procedure will now be called for all tcp sources
for {set j 1} {$j<=$NumbSrc} { incr j } {
$ns at 0.1 ‘‘plotWindow $tcp_src($j) $windowVsTime $j’’
}

$ns at [expr $Duration] ‘‘finish’’
$ns run
```

Bảng 6.2: Đoạn mã ex3.tcl đối với một số kết nói TCP cạnh tranh

6.5. Các kết nối TCP ngắn

Đa số lưu lượng trên Internet là truyền file. Dung lượng file truyền trung bình là khoảng 10Kbyte, nghĩa là một file trung bình có từ 10 gói TCP trở xuống thì kích cỡ của 1 gói TCp sẽ là 1Kbyte [10,36]. Điều này cho thấy hầu hết việc truyền file kết thúc trong giai đoạn “slow start”. Những file này thường được gọi là "con chuột" ("mice"). Tuy nhiên, điều đáng ngạc nhiên là hầu hết lưu lượng Internet được truyền bằng các file rất dài, được gọi là "con voi" ("elephants"). Một phân bố điển hình mô tả kích thước file có tên là Pareto [10], với tham số shape (shape parameter) nằm giữa 1 và 2 [10] (trung bình là 10KB). Trung bình kích thước file khoảng 2.5Kbyte . Như vậy, phân bố Pareto với giá trị kì vọng là 10Kbyte và kích thước bình quân là 2.5 Kbyte định nghĩa một phân bố Pareto với tham số shape

$\beta = 1.16$ và với kích thước nhỏ nhất là 1.37Kbytes. Phân bố các thời gian giữa hai sự kiện đến của các kết nối mới thường được lấy theo hàm mũ.

Trong phần này, ta sẽ xem xét các cách mô phỏng các phiên làm việc ngắn và để xác định phân bố thời gian truyền, số lượng kết nối đang diễn ra và lưu lượng.

Ta sẽ xem xét một mạng lưới có cùng topo như trong hình 6.2: một vài nguồn chia sẻ chung nút thắt cổ chai và cùng một đích chung. Thông số "NodeNb" cho biết số lượng nguồn. (trong ví dụ ta đang đề cập, thông số là 6). Nguồn TCP bây giờ được tham số hóa bởi 2 tham số: nút nguồn và số phiên từ nút nguồn đó. Đối với mỗi một trạm TCP ta định nghĩa một ứng dụng FTP. Các kết nối TCP mới đến theo một quá trình Poison. Vì vậy, ta khởi tạo một kết nối TCP mới bằng cách dùng các biến ngẫu nhiên được phân bổ theo hàm mũ. Giả sử liên kết nút thắt cổ chai là 2 Mbps, độ trễ là 1ms và kích thước 1 hàng đợi là 3000. Tất cả các liên kết đầu vào khác tham gia cùng liên kết này có băng thông là 100Mbps và độ trễ là 1ms. Ta sử dụng phương pháp New Reno với kích thước lớn nhất của cửa sổ là 2000.

Thời gian trung bình giữa những lần đến của các phiên TCP mới tại mỗi nút trong ví dụ đang xét là 45. Điều này có nghĩa là trung bình, 22.22 phiên mới đến mỗi nút, vì vậy tổng tốc độ đến của các phiên là 22.22 lần NodeNb mà trong ví dụ của ta là 133.33 phiên/s .Ta tạo ra các phiên làm việc có kích thước ngẫu nhiên (giá trị trung bình là 10Kbyte), tham số shape của phân bố Pareto là 1.5. Do vậy, tổng tốc độ đến của các bit là:

$$133.33 \times 10^4 \times 8 = 10.67 Mbps$$

Ta thấy tốc độ tạo ra của các bit lớn hơn dung lượng nút thắt cổ chai, vì vậy hiện tượng tắc nghẽn sẽ xảy ra. Tuy nhiên, TCP có khả năng tránh hiện tượng tắc nghẽn trong mạng (tại hàng đợi nút thắt cổ chai). Vì vậy, hiện tượng nghẽn mạng sẽ xảy ra theo những cách thức khác .

Điều khiển số phiên làm việc: Trong ngữ cảnh các phiên TCP ngắn, ta quan tâm đến cả số liệu thống kê gói lẫn số liệu thống kê phiên. Trong chương trình ns, Ta sẽ định nghĩa thủ tục đệ qui, được gọi là "Test". Thủ tục này kiểm tra mỗi phiên làm việc xem liệu nó kết thúc hay chưa. Thủ tục này gọi lại chính nó sau 0.1s. (Giá trị này được đặt trong biến "time"). Nếu một kết nối chấm dứt, Ta sẽ có một file đầu ra chứa các thông số sau:

- Đối tượng xác định kết nối i và j (trong đó (i, j) thể hiện cho lần kết nối thứ j từ nút i.
- Thời điểm bắt đầu và kết thúc kết nối.
- Thông lượng của kết nối.
- Kích thước của kênh truyền dữ liệu tính theo byte.

Sau một khoảng thời gian ngẫu nhiên, thủ tục lại xác định khởi đầu một quá trình truyền khác nữa. Trong đoạn mã sau đây, file đầu ra sẽ là Out.ns. Để kiểm tra xem liệu một phiên làm việc đã kết thúc hay chưa, Ta sử dụng lệnh:

```
If{$tcpsrc{$i,$j) set ack_]==[$tcpsrc($tcpsrc($i,$j) set maxseq_]} {
```

Một thủ tục đệ qui khác tên là "countFlows" được sử dụng để cập nhật số kết nối đang hoạt động từ mỗi nút (được lưu trữ dưới dạng vectơ "Cnts" có độ dài tương ứng với số kết nối đang diễn ra từ nút j[2] Thủ tục có 2 thông số: "ind" và "sign". Thông số "ind" cho biết nút nguồn nào được quan tâm còn thông số "sign"cho thủ tục biết phải làm gì: giá trị của "sign" là 0 nếu cuộc gọi kết thúc còn nó sẽ là 1 nếu cuộc gọi bắt đầu. Những thông số này được sử dụng khi việc gọi thủ tục xảy ra ở lúc bắt đầu và kết thúc kết nối. Thủ tục cũng tự gọi lại chính nó đều đặn sau 0.2 và sau đó nó in số lượng cuộc gọi đang diễn ra vào file Conn.tr. Để làm được điều đó, giá trị của thông số "sign" ko phải là 1 cũng ko phải là 0. (Ta đặt là 3)

Giám sát hàng đợi: Để tiếp tục với chương trình tcl, Ta xem xét một cách khác để thực hiện việc giám sát hàng đợi. Cách này phức tạp hơn phương pháp mà Ta đã xem ở mục 4.3 Ta sử dụng lại các câu lệnh:

```
Set qfile [$ns monitor-queue $N$D [open queue.tr.w] 0.05]
[$ns link $N $D] queue-sample timeout;
```

Tuy nhiên ta cũng có thể xóa dòng thứ 2. Thay vì phải giới hạn với lệnh này, ta làm việc trực tiếp với các thuộc tính của " giám sát hàng đợi"("monitor-queue") được mô tả ở mục 4.3. Việc này được tiến hành trong một thủ tục có tên là "record". Thủ tục này tự động lặp lại sau 0.05s. Ví dụ, chúng ta sao lưu băng thông đã sử dụng của hàng đợi (tính theo Kb/s) thành một file bằng cách chia số lượng số lượng gói tin đi trong một khoảng thời gian cho khoảng thời gian đó. Lưu ý là giám sát hàng đợi tổng số lượng các byte đến trong thuộc tính `bdepartures_`. Nếu như ta chỉ muốn tính đến số lượng departure trong một khoảng thời gian (mà ko phải trong suốt toàn bộ thời gian mô phỏng), ta phải chỉnh lại giá trị của `bdepartures_` ở cuối mỗi quá trình tính toán băng thông mới.

```
set ns [new Simulator]
#There are several sources of TCP sharing a bottleneck link
#and a single destination . Their number is given by the paramter NodeNb
#          S(1)  ----
```

[2]Mối quan tâm dành cho các bộ đếm khác nhau ở các nút khác nhau thể hiện ở chỗ ta cũng có thể sử dụng chương trình cho các liên kết ko đối xứng, trong trường hợp này ta có thể sẽ nghiên cứu sự phụ thuộc của hiệu năng vào độ trễ của kênh truyền và băng thông.

```
#           .       |
# . ----  ---- N -------- D(1) . . . D(NodeNb)
# .                    |
# S(NodeNb)        ----
#Next file will contain the transfer time of different connections
set out [open Out.ns w]
# Next file will contain the number of connections
set Conn [open Conn.tr w]
#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf
#We define three files that will be used to trace the queue size ,
#the bandwidth and losses at bottleneck.
Set qsize [open queuesize.tr  w]
Set qbw  [open queuebw.tr    w]
Set qlost [open queuelost.tr   w]
#defining the topology
Set  N  [$ns node]
Set  D  [$ns node]
$ns duolex-link $N  $D 2Mb 1ms Droptail
$ns queue-limit $N  $D 3000
#Number of souces
Set NodesNb 6
#Number of flows per suouce node
Set NumberFlows 530
#Nodes and links
For {set j 1} {$j<=$nodeNb} { incr j } {
Det S($j) [$ns node ]
$ns duplex-link $S($j) $N 100Mb 1ms DropTail
$ns queue-limit $S($j) $N 1000
}
#TCP sources and Destinations
For {set i 1} {$i<=$NodeNb} {incr I } {
For {set j 1} {$j<=$NumberFlows} {incr j } {
set tcpsrc($i,$j) [new Agent/TCP/Newreno]
set tcp_snk($i,$j) [new Agent/TCPSink]
$tcpsrc($i.$j) set window_ 2000
     }
```

```
}
#Connections
For {set i 1} {$i<=$NodeNb} {incr I } {
For {set j 1} {$j<=$NumberFlows} {incr j } {
$ns attach-agent $S($i) $tcpsrc($i,$j)
$ns attach-agent $D Stcp_snk($i,$j)
$ns connect $tcpsrc($i,$j) $tcp_snk($i,$j)
    }
}
#Generators for random size of files
Set rng1 [new RNG]
$rng1 seed 0
Set rng2 [new RNG]
$rng2 seed 0
# Random interarrival time of TCP transfers at each source i
Set RV [new RandomVariable/Exponential]
$RV set avg_ 0.045
$RV use-rng $rng1
#Random size of files to transmit
Set RVSize [new RandomVariable/pareto]
$RVSize set avg_ 10000
$RVSize set shape_ 1.5
$RVSize use-rng $rng2
#We now define the beginning times of transfers and the transfer sizes
#Arrivals of sessions follow a Poisson process.
#
For {set i 1} {$i<=$NumberFlows} { incr j } {
#set the beginning time of next transfer from source i
Ser t [expr $t + [$RV value]]
Set Conct($i,$j) $t
#set the size of next transfer from source i
Set Size($i,$j) [expr [$RVSize value]]
$ns at $Conct($i,$j) ‘‘$Sftp($i,$j) send $Size($i,$j)’’
#update the number of flows
$ns at $Conct($i,$j) ‘‘countFlows $i 1’’
} }
#Next is a recursive procedure that checks foreach session whether
#it has ended . The procedure calls itself each 0.1 sec ( this is
```

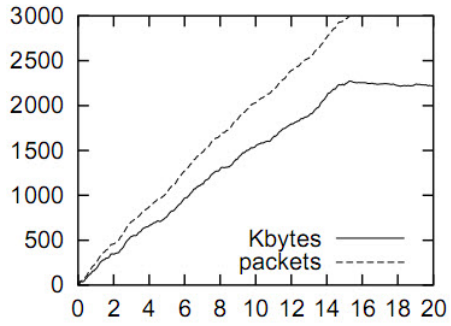
```
#set in the variable ‘‘time’’)
#If a connection has ended then we print in the file $Out
# * the connection identifiers i and j ,
# * the start and end time of the connection,
# *the throughput of the session ,
# *the size of the transfer in bytes
#and we further define another beginning of transfer after a random time .
Proc Test {} {
Global Conct tcpsrc Size NodeNb NumberFlows ns RV ftp Out tcp_snk RVSize
Set time 0.1
For {set i 1} {$i<=$NodeNb} {incr I } {
For {set j 1} {$j<=$NumberFlows} {incr j } {
#We now check if the transfer is over
if {[ cpsrc($i,$j) set ack_]==[$tcpsrc($i,$j) set maxseq_]} {
       if { $tcpsrc($i,$j) set ack_]>=0} {
# If the transfer is over , we print relevant information in $Out
put $Out ‘‘$i,$j\t$Conct($i,$j)\t[expr [$ns now]]\t\
[expr ($Size($i,$j))/(1000*([expr [$ns now]] - $Conct($i,$j)))]\t$Size($i,$j)’’
   countFlows $i 0
   $tcpsrc($i,$j) reset
   $tcp_snk($i,$j) reset
} } } }
$ns at [expr [$ns now]+$time] ‘‘Test’’
}
for {set j 1} {$j<=$nodeNb} {incr j } {
set Cnt($j) 0
}
#The following recursive procedure updates the number of connections
#as a function of time . Each 0.2 it prints them into $Conn. This
#is done by calling the procedure with the ‘‘sign’’ parameter equal
# 3 (in which case the ‘‘ind’’ parameter does not play a role). The
#procedure is also called by assingning the ‘‘sign’’ parameter 0 , or when
#from source I ends by assingning the ‘‘sing’’ parameter 0, or when
#it begins , by assingning it 1 (I is passed though the ‘‘ind’’ variable).

proc countFlows {ind sing} {
global Cnts Conn NodeNb
set ns [Simulator instance]
```

```
if {$sign==0 } { set Cnts($ind) [expr $Cnts($ind) - 1]
} else
if {$sign==1 } { set Cnts($ind) [expr $Cnts($ind0 + 1]
} else {
puts -nonewline $Conn ‘‘[$ns now] \t’’
set sum 0
for {set j 1} {$j<=$NodeNb} {incr j } {
set sum [expr $sum + $Cnts($j)]
}
puts $Conn ‘‘$sum’’
  $ns at [expr [$ns now] + 0.2] ‘‘countFlows 1 3’’
} }
#Define a ‘finish’ procedure
proc finish {} {
global ns tf qsize qbw qlost
$ns flush-trace
close $qsize
close $qbw
close $qlost
#Execute xgraph to display the queu size , queue bandwith and loss rate
exec xgraph queuesize.tr -geometry 800x400 -t  ‘‘Queue size’’ -x ‘‘secs’’ -y ‘‘#packets’’ &
exec xgraph queuebw.tr -geometry 800x400 -t ‘‘bandwith’’ -x ‘‘secs’’ -y ‘‘Kbps’’ -fg white &
exec xgraph queuelost.tr -geometry 800x400 -t ‘‘#Packets lost’’ -x ‘‘secs’’ -y ‘‘packets’’ &
exut 0
}
# QUEUE MONiTORiNG
set qfile [$ns monitor-queue $N $D [open queue,tr w] 0.05]
[$ns lick $N $D] queue-sample-timeout;
#The following procedure records queue size , bandwidth and loss rate
proc record {} {
global ns qfile qsize qbw alost N D
set time 0.0.5
set now [$ns now]
#print the current queue size in $qsize , the current used
# bandwidth is $qbw, and the loss rate in $qloss
$qfile instvar parrivals_ pdepartures_ bdrops_ bdepartures_ pdrops_
puts $qsize ‘‘$now [expr $parrivals_ -$pdepartures_ -$pdrops_]’’
puts $qbw  ‘‘$now [expr $bdepartures_*8/1024/$time]’’
```

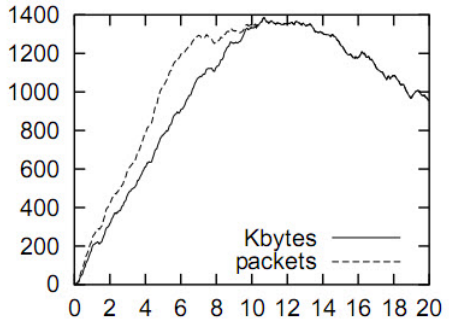
```
set bdepartures_ 0
puts $qlost ‘‘$now [expr $pdrops_/$time]’’
$ns at [expr $now+$time] ‘‘record’’
}
$ns at 0.0 ‘‘recoerd’’
$ns at 0.01 ‘‘Test’’
$ns at 0.5 ‘‘countFlows 1 3’’
$ns at 20 ‘‘finish’’
$ns run
```

Bảng 6.3: Đoạn mã shortTcp.tcl đối với một số kết nói TCP ngắn

Số lượng các phiên làm việc được tạo ra (530 phiên /nguồn) đảm bảo rằng các gói tin đến từ tất cả các nút tiếp tục cho đến tận khi kết thúc mô phỏng. Khi chạy đoạn mã ta thu được kích thước hàng đợi tính theo Kbyte và theo các gói tin như trong hình 6.8 Ta cũng thực hiện lại mô phỏng với số phiên làm việc tại mỗi nút giảm xuống còn 130 và kích thước của hàng đợi theo Kbyte và theo gói tin được mô tả ở hình 6.9.



Hình 6.8: Kích thước hàng đợi trong ví dụ shortTcp.tcl



Hình 6.9: Kích thước hàng đợi trong ví dụ shortTcp.tcl khi số kết nối bị giới hạn

Dưới đây là một số kết quả quan sát được

1. Trong cả 2 hình, số lượng gói tin tại hàng đợi nhiều hơn số lượng Kbyte được xếp hàng. Điều này dường như lạ vì một gói TCP có kích thước là 1Kbyte! Lí do là các phiên có số lượng gói tin lớn rất ít (Một phiên thường chỉ có 3 gói tin hoặc ít hơn). Vì vậy, số lượng gói tin đầu phiên có kích thước 40 byte (được gửi lúc bắt đầu kết nối TCP) cũng đáng kể. (khoảng 1/3) Nếu tính đến những gói tin ngắn này thì có nhiều gói tin hơn lượng Kbyte.

2. Ta quan sát thấy rằng ở hình 4.8 kích thước hàng đợi ổn định ở 3000. Đây là kích thước lớn nhất mà hàng đợi đạt tới. Tiếp theo thời điểm này sẽ xảy ra hiện tượng mất gói tin ở hàng đợi.

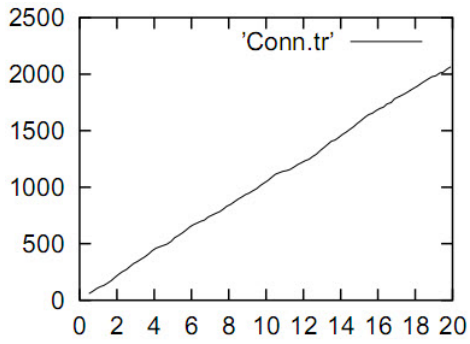
3. Như trên hình 4.8 thể hiện, số lượng gói tin luôn luôn lớn hơn số lượng Kbyte xếp hàng và trên hình 4.9, sau một khoảng thời gian nào đó, số lượng gói tin sẽ bằng với số lượng Kbyte. Tại thời điểm này, tất cả các gói tin ở hàng đợi là các gói tin dữ liệu TCP và ko có các gói tin 40 byte dành cho khởi tạo phiên. Điều này có được khi ta giới hạn số lượng gói tin tại mỗi nút xuống còn 130.

4. Nếu lấy tốc độ dữ liệu tạo ra trừ đi tốc độ ra của liên kết nút thắt cổ chai, ta sẽ thu được lượng dữ liệu được xếp hàng tại hàng đợi của liên kết nút thắt cổ chai lớn hơn rất nhiều. Lí do là dữ liệu cũng được lưu lại tại bộ nhớ đếm của người gửi.

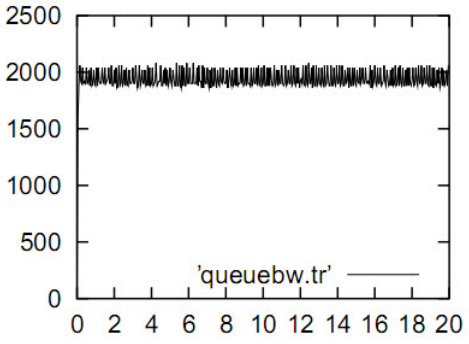
Tiếp theo ta quan sát sự phát triển số lượng kết nối đang diễn ra ở hệ thống, như trong hình 6.10 và lượng băng thông được sử dụng ở liên kết nút thắt cổ chai ở hình 6.11

6.6. Các công cụ giám sát tiên tiến:

Ở mục 6.5, ta đã kiểm tra sự kết thúc mỗi phiên làm việc TCP theo định kì bằng cách so sánh số thứ tự của bản tin ack hiện thời với số thứ tự lớn nhất của các kết nối. Qúa trình tìm kiếm này khá tốn kém. Ta xem xét 2 quá trình giám sát khác:



Hình 6.10: Số lượng kết nối



Hình 6.11: Băng thông được sử dụng trong giai đoạn nghẽn cổ chai

1. Thứ nhất là để để xác định những hoạt động cần làm liên quan đến việc hủy kết nối trong thủ tục "done", là một thủ tục tự động được gọi đến khi một kết nối kết thúc. Số nhận dạng của kết nối vừa kết thúc cũng như là các thuộc tính khác của kết nối (ví như thời gian khởi đầu của nó chẳng hạn) có thể được thủ tục này sử dụng nếu

nó được định nghĩa như các trạng thái của quá trình kết nối. Tiếp cận này được thể hiện ở đoạn mã tcl shortTcp2.tcl ở bảng **??**

2. Ta cũng có thể dùng quá trình giám sát trên mỗi luồng. Qúa trình này cung cấp các con số thống kê trên mỗi luồng về 1 số thông tin như số lượng gói tin đã truyền, các byte đã truyền, gói tin bị mất, vv. Ta sẽ để lại phần thảo luận tiếp cận này trong phần sau **??**

Định nghĩa "trạng thái"("state") của các kết nối TCP trong đoạn mã này được đưa ra cũng giống như cách định nghĩa kích thước cửa sổ lớn nhất của TCP, ngưỡng ban đầu của giai đoạn slow-start., vv. Trong đoạn mã, chúng ta định nghĩa thời gian bắt đầu phiên làm việc, nhận dạng nút và phiên làm việc cũng như là kích thước truyền như theo các trạng thái sau:

```
$tcpsrc($i, $j) set starts $t
$tcpsrc($i, $j) set sess $j
$tcpsrc($i, $j) set node $i
$tcpsrc($i, $j) set size [$RVSize value]]
```

Lưu ý là ta sử dụng các trạng thái khác của kết nối TCP:

- `Ndatapack_` là số lượng gói tin được truyền bởi kết nối (nếu 1 gói tin được truyền lại vài lần, nó chỉ được đếm 1 lần
- `Ndatapbytes_` là số lượng byte dữ liệu được truyền bởi kết nối
- `Nrexmitpackets_` là số lượng gói tin được kết nối truyền lại
- `Nrexmitbytes_` là số lượng byte được kết nối truyền lại

```
set ns [new Simulator]

# there are serveral source each generating many TCP session sharing a bootleneck
# link and a single destination. Their numner is given by the paramter NodeNb

#   S(1)     ----
#    .           |
#    .       ---- ---- N ---------D(1)...D(NodeNb)
#    .           |
#   S(1)     ----
```

```
#Next file will contain the trasfer time of diffent connections
  set Out [open Out.ns w]
#Next file will contain the number of connections
  set Conn [open Conn.tr w]
#Open the Trace file
  set tf [open out.tr w]
  $ns trace-all $tf

#defining the topolofy
  set N [$ns node]
  set D [$ns node]
  $ns duplex-link $N $D 2Mb 1ms DropTail
  $ns queue-limit $N $D 300

#Number of Nodes
  set NodeNb 6

#Number of flows per source node
  set NumberFlows 530

#Node and links
  for {set j 1} {$j<=$NodeNb} {incr j}{
  set S{$j} [$ns node]
  $ns duplex-link $S($j) $N 100Mb 1ms DropTail
  $ns queue-limit $S($j) $N 1000 }

#TCP Source, destinations, connections
  for {set i 1} {$i<=$NodeNb} {incr i} {
  for {set j 1} {$j<=$NumberFlows} {incr j }{
  set tcpsrc($i, $j) [new Agent/TCP/Newreno]
  set tcp_snk($i, $j) [new Agent/TCKSink]
  $tcpsrc($i, $j) set window_ 200
  $ns attach-agent $S ($i) $tcpsrc($i, $j)
  $ns attach-agent $D $tcp_snk($i, $j)
#ns connect $tcpsrc($i, $j) $tcp_snk($i, $j)
  set ftp($i, $j) [$tcpsrc($i, $j) attach-source FTP]
  }}
```

```
#Generator for random size of files.
  set rng1 [new RNG]
  $rng1 seed 0
  set rng2 [new RNG]
  $rng2 seed 0

#Random inter-arrival times of TCP rtasfer at each source i
  set RV [new Random Variable/Exponential]
  $RV set avg_ 0.045
  $RV use-rng $rng1

#Random size of files to transmit
  set RVSize [new RandomVariabl/Pareto]
  $RVSize set avg_ 10000
  $RVSize set shape_ 1.5
  $RVSize use-rng $rng2

#We now define the beginning times of transfer and the transfer sizes
#Arrivals of sessions follows a Poisson process.
#
  for {set i 1} {$i<=$NodeNb} {incr i}{
    set t [$ns now]
    for {set j 1} {$j<=$NumberFlows} {incr j}{
#Set the beginning time of next transfer from source and attributtes
  set t [expr $t + [$RV value]]
  $tcpsrc($i, $j) set starts $t
  $tcpsrc($i, $j) set sess $j
  $tcpsrc($i, $j) set node $i
     $tcpsrc($i, $j) set size [expr [$RVsize value]]

  $ns at [$tcpsrc($i, $j) set starts] "$ftp(
  $i,  $j) send [$tcpsrc($i, $j) ser size]"

# update the number of flows
  $ns at [$tcpsrc($i, &j) set starts] "contFlows $i 1"
  }}
```

```
  for {set j 1} {$j<=$NodeNb} {incr j }{
  set Cnts($j) 0
  }

#The following procedure is called whenever a connection ands
  Agen/TCP instproc done {}{
  global tcpsrc NodeNb NumberFlows ns RV ftp Out tcp_snk RVsize
#Print in $Out: node, session, start time, end time, duration,
#trans-pkts, trans-bytes, retrans-bytes, throughput
  set duration [expr [$ns now] - [$self set starts]]
  puts $Out "[$self set node] \t [$self set sess] \t [$self set starts]\t\
     [$ns now] \t $duration \t [$self set ndatapack_] \t\
     [$self set ndatabytes_] \t [$self set nrexmitbytes_] \t\
     [expr [$self set ndatabytes_]/$duration]"
    countFlows [$self set node] 0
  }

# The following recursive procedure updates the number of connections
# as a function of time. Each 0.2 sec it prints them into $Conn. This
# is done by calling the procedure with the "sign" parameter equal
# 3 (in which case the "ind" parameter does not play a role). The
# procedure is also called by the "done" procedure whenever a connection
# from source i ends by asssinging the "sign" parameter 0, or when
# it begins, by assinging it 1 (i is passed through the "ind" variable).

  proc countFlows {ind sign}{
  global Cnts Conn NodeNb
  set ns [Simulator instance]
     if {$sign==0} {set Cnts($ind) [expr $Cnts($ind) - 1]
  } elseif {$sign==1} {set Cnts($ind) [expr $Cnts($ind) + 1]
  } else {
    puts -nonewline $Conn "[$ns now] \t"
    set sum 0
    for {set j 1} {$j<=$NodeNb} {incr j} {
        puts -nonewline $Conn "$Cnts($j) \t"
        set sum [expr $sum + $Cnts($j)]
        }
      puts $Conn "$sum"
```

```
        $ns at [expr [$ns now] + 0.2] "countFlows 1 3"
    }}

#Define a 'finish' procedure
    proc finish {} {
       global ns tf
    close $tf
          $ns flush-trace
          exuit 0
    }

     $ns at 0.5 "countFlows 1 3"
     $ns at 20 "finish"

    $ns run
```

Bảng 6.4: Đoạn mã shortTcp.tcl đối với một số kết nói TCP ngắn

6.7. Bài tập

Bài tập 6.7.1. Giải thích tại sao kích thước cửa sổ dao động nhiều hơn so với thông lượng trong hình 6.1 và hình 6.2

Bài tập 6.7.2. Thông lượng trung bình và tỉ lệ mất gói tin của kết nối TCP trong ví dụ ex1.tcl là gì?

Bài tập 6.7.3. Kích thước hàng đợi trung bình trong ví dụ ex1.tcl là gì?

Bài tập 6.7.4. Nghiên cứu ảnh hưởng của xác suất mất gói tin trong mô hình nhiễu rdrop.tcl lên thông lượng TCP với xác xuất mất gói tin dao động từ 0 -40%

Bài tập 6.7.5. Sửa đổi đoạn mã rdrop.tcl để nghiên cứu ảnh hưởng xác suất mất của các gói tin (Cụ thể là gói tin ACK) trên kênh truyền ngược n3-n2. Vẽ đồ thị thông lượng như là một hàm xác suất mất gói tin đối với tỉ lệ mất gói tin dao động từ 0 - 40%. TCP nhạy cảm hơn với việc mất ngẫu nhiên các gói tin gửi đi hay nhận về các gói tin ACK?

Bài tập 6.7.6. Mô phỏng 2 liên kết TCP cạnh tranh đối xứng chia sẻ chung đường liên kết nút thắt cổ chai. Mỗi liên kết chiếm bao nhiêu phần băng thông nếu:

(i) chỉ 1 liên kết sử dụng lựa chọn ACK bị trễ và cả 1 liên kết đều là NewReno
(ii) cả 2 liên kết đều dùng lựa chọn ACK đơn giản, liên kết đầu dùng phiên bản Tahoe còn liên kết còn lại dùng phiên bản NewReno

Bài tập 6.7.7. Trong thủ tục plotWindow ở cuối đoạn mã ex3.tcl ở bảng 6.2 , chúng ta coi số liên kết như 1 tham số của thủ tục. Nhưng nếu ta coi nó như 1 biến chung thì điều gì sẽ xảy ra? (có nghĩa là chúng ta đã viết " global ns j")

Bài tập 6.7.8. Phân tích các quá trình mất gói tin trong ex3.tcl (xem bảng 4.2). Kích thược hàng đợi ở kênh n2-n3 sẽ là bao nhiêu để tránh hiện tượng mất gói tin?

Bài tập 6.7.9. Thêm vào đoạn mã shortTcp.tcl (Bảng 6.4) các gói tin bị mất ngẫu nhiên (i) ở kênh truyền đi (ii) ở kênh nhận về N-D. Tỉ lệ mất gói tin biến đổi trong khoảng từ 0 - 40 %. Phân tích thời gian trung bình truyền 1 file và độ lệch tiêu chuẩn của thời gian này như là một hàm của tỉ lệ mất gói tin. Giải thích kết quả thu được!

Lưu ý: trong trường hợp có nhiều người sử dụng, 1 người có thể mong đợi rằng nếu 1 số phiên làm việc có thông lượng thấp do bị tổn hao, sẽ có thêm thông lượng sẵn có dành cho các phiên làm việc khác để các phiên TCP ngắn ít nhạy cảm với tổn hao hơn các phiên dài. Các quá trình mô phỏng xác nhận điều này hay ko? Nếu ko, hãy giải thích điều gì xảy ra.

Chương 7

Định tuyến và mạng di động

Mục lục

Người dịch: *Trần Diệu Linh*

Biên tập: *Hoàng Trọng Minh*

Bản gốc: *NS Simulator for beginners, Chapter 5* [2]

Chúng ta sẽ xem lại trong chương này cả định tuyến unicast và multicast. Các giao thức định tuyến tập trung vào đường đi cố định (định tuyến tĩnh) sẽ được so sánh với định tuyến động. Tác dụng của kết nối động trong định tuyến sẽ được kiểm tra. Một tham chiếu tốt cho định tuyến qua Internet nằm trong tham khảo [6].

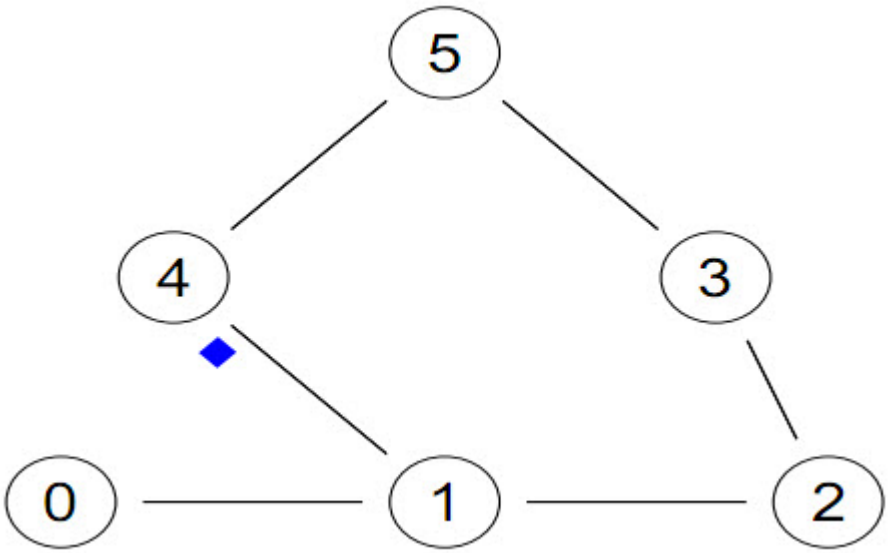
7.1. Bắt đầu như thế nào

Có một vài kiểu định tuyến qua Internet. Một kiểu đơn giản nhất là kiểu định tuyến tĩnh trong đó đường đi ngắn nhất (giới hạn bởi số các hop) được lựa chọn trong các kết nối.

Ns có thể mô phỏng các liên kết nhiễu (như chúng ta đã thấy trong phần 7.3) hoặc thậm chí các liên kết có thể bị ngắt kết nối. Để mô phỏng sự ngắt kết nối của một liên kết giữa nút n1 và nút n4 từ thời điểm 1 đến 4.5, ví dụ ta có thể dùng lệnh:

```
$ns rtmodel -at 1.0 down $ns1 $ns4
$ns rtmodel -at   4.5 up $ns1 $ns4
```

Bây giờ, chúng ta xét mạng được mô tả trong hình 7.1, trong đó có hai mạng khác nhau giữa nút nguồn 0 và nút đích 5. Định tuyến tĩnh mặc định được sử dụng bởi ns, sẽ lựa chọn tuyến 0-1-4-5 cho việc thiết lập kết nối.



Hình 7.1: Ví dụ về định tuyến

```
Set ns [new Simulator]
#Định nghĩa các mầu khác nhau cho các luồng dữ liệu (dành cho NAM)
$ns color 1 Blue
$ns color 2 Red
#Open the Trace file
Set file1 [open out.tr w]
```

```
$ns trace-all $file1
#mở file theo dấu NAM (open the NAM trace file)
Set file2 [open out.nam w]
$ns namtrace-all $file2
#định nghĩa thủ tục 'finish' (define a 'finish' procedure)
Proc finish {} {
global ns file1 file2
$ns flush-trace
close $file1
close $file2
exec nam out.nam &
exit 0
}

#dòng tiếp theo nên được ghi chú để chỉ rõ định tuyến tĩnh.
$ns rtprôt DV
#tạo 6 nút
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# Tạo các liên kết giữa các nút
$ns  duplex-link  $n0  $n1  0.3Mb  10ms  DropTail
$ns  duplex-link  $n1  $n2  0.3Mb  10ms  DropTail
$ns  duplex-link  $n2  $n3  0.3Mb  10ms  DropTail
$ns  duplex-link  $n1  $n4  0.3Mb  10ms  DropTail
$ns  duplex-link  $n3  $n5  0.3Mb  10ms  DropTail
$ns  duplex-link  $n4  $n5  0.3Mb  10ms  DropTail

#Đưa ra vị trí của nút (cho NAM)
$ns  duplex-link-op   $n0  $n1  orient  right
$ns  duplex-link-op   $n1  $n2  orient  right
$ns  duplex-link-op   $n2  $n3  orient  up
$ns  duplex-link-op   $n1  $n4  orient  up-left
$ns  duplex-link-op   $n3  $n5  orient  left-up
```

```
$ns  duplex-link-op   $n4  $n5  orient  right-up

#cài đặt kết nối TCP
Set tcp [new Agent/TCP/Newreno]
$ns  attach-agent  $n0  $tcp
Set sink [new Agent/TCPSink/DelAck]
$ns   attach-agent  $n5  $sink
$ns   connect  $tcp  $sink
$tcp  set  fid_  1

#Cài đặt FTP qua kết nối TCP
Set  ftp  [new Application/FTP]
$ftp  attach-agent  $tcp
$ftp  set  type_   FTP
$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 4.5 up $n1 $n4
$ns at 0.1 ‘‘$ftp start’’
$ns at 12.0 ‘‘finish’’
$ns run
```

Bảng 7.1: Kịch bản tcl cho định tuyến tĩnh và động(ex2.tcl)

Ngược với định tuyến tĩnh, Internet có thể tìm được đường đi khác khi nó thấy một tuyến bị ngắt kết nối. Tùy chọn này được sử dụng trong ns bằng cách thêm lệnh (xem thêm bảng 7.1)

```
$ns rtproto DV
```

Trong ví dụ ex2.tcl đưa ra trong bảng 7.1, liên kết kết nối 1-4 bị down trong suốt khoảng thời gian [1, 4-5]. Trong đồ thị NAM chúng ta có thể thấy liên kết này trở thành màu đỏ trong khoảng thời gian này. Một kết nối TCP được lập giữa nút 0 và 5. Khi chạy kịch bản, với định tuyến tĩnh (ghi chú lệnh `$ns  rtproto DV`) chúng ta thấy kể cả nếu kết nối được tiếp tục lại tại thời điểm 4.5, kết nối TCP chỉ kết nối lại tại thời điểm xấp xỉ 8. Lý do là timeout (thời gian quá hạn) đã xảy ra trong khi không có tín hiệu ACK trở lại với node 0, và khoảng thời gian đó bị nhân đôi với mỗi một timeout mới.

Trong đồ thị khảo sát NAM chúng ta có thể thấy trường hợp định tuyến động, các gói tin báo hiệu được sử dụng để quyết định đường đi không chỉ tại thời điểm bắt đầu mà còn tại thời điểm kết nối có thay đổi.

7.2. Mạng động

Chúng ta có thể thấy phần trước, chúng ta có thể xác định trạng thái liên kết hiện: liên kết có thể down và up tại các thời điểm chọn trước. Tuy nhiên, kết nối động có thể thay đổi theo các tiến trình ON-OFF hàm mũ, hoặc tiến trình ON-OFF xác định, hoặc theo một số dữ liệu khảo sát theo yêu cầu.

Các mô hình hàm mũ hoặc mô hình xác định có bốn tham số: thời gian bắt đầu (mặc định là 0,5 giây từ khi bắt đầu mô phỏng), chu kỳ up (mặc định là 10 giây), chu kỳ down (mặc định là 1giây) và thời gian kết thúc (mặc định là kết thúc mô phỏng). Trong trường hợp hàm mũ, các tham số up và down tương đương với khoảng thời gian dự tính. Ví dụ, cấu trúc cho mô hình quyết định được áp dụng cho liên kết n1-n2 là:

```
$ns rtmodel Deterministic  {0.8  1.0  1.0} $n1   $n2
```

(thời gian kết thúc là mặc định). Trong một câu lệnh theo dạng

```
$ns  rtmodel   Deterministic  {0.8  1.0}  $n1  $n2
```

Các thời điểm bắt đầu và kết thúc là mặc định và trong câu lệnh theo dạng

```
$ns  rtmodel  Deterministic  {-  1.0}  $n1  $n2
```

Chỉ có tham số không mặc định là chu kỳ down. Kết nối hàm mũ đưa ra ở trên bằng cách thay hàm xác định "Deterministic" bằng hàm mũ "Exponential"

Câu lệnh tương đương với kết nối dựa trên file theo vết là:

```
$ns  rtmodel  Trace  <config_file>  $n0  $n1
```

Cuối cùng, câu lệnh cũng có thể tạo thứ tự các trạng thái định tuyến là ns, và sử dụng nó như đầu vào (xem [1]).

Các lỗi nút: Có một khả năng là một nút đang down và up. Điều này giống như ta đã thấy trong các trường hợp lỗi liên kết, ngoại trừ trường hợp một nút xuất hiện tại điểm cuối.

7.3. Các giao thức định tuyến multicast (PIM)

Trong multicast, có thể có một vài nhóm thành viên multicast, các nhóm có thể chồng phủ lên nhau. Trong IP multicast, bên nhận phải yêu cầu là thành viên trong nhóm multicast

trong khi bên gửi có thể gửi mà không cần tham gia vào nhóm từ trước. Bên gửi không nhận được phản hồi từ mạng về việc bên nhận trong định tuyến IP multicast. Không phải tất cả các nút của mạng đều có thể xử lý multicast, trong ns ta có thể mô tả các nút có khả năng multicast một cách chi tiết.

Một giao thức định tuyến định nghĩa một cơ chế mà qua đó cây multicast được tính toán trong mô phỏng. Có hai lớp chính của các lớp định tuyến:

1. Kiểu "dense mode" thích hợp đối với trường hợp số lượng người dùng multicast lớn, trong trường hợp này, các cây multicast được xây dựng cho bất kỳ một cặp nguồn nào và nhóm multicast của nó. Việc xây dựng cây yêu cầu quảng bá đến tất cả các nút trong mạng.

2. "Sparse mode" được sử dụng trong trường hợp số các nút nhỏ. Do đó, định tuyến có thể được xử lý bằng việc sử dụng một cây đơn được chia sẻ.

Bốn giao thức định tuyến multicast được sử dụng trong ns: Dense Mode (DM), Centralised (CtrMcast), Share Tree Mode (ST), và Bi-directional Sahred Tree Mode (BST). Không may, cách ns mô phỏng các giao thức không có nhiều báo hiệu, đặc biệt trong quá trình khởi tạo. Giao thức DM chỉ là một giao thức có phiên bản động trong ns, được gọi là dynamicDM.

7.3.1. Chế độ Dense

Giao thức DM có hai chế độ khá giống nhau: Protocol pim DM (Protocol Independent Multicast – Dense Mode) và dvmrp (Distance Vector Multicast Routing Protocol) [40], pimDM đơn giản hơn một chút. Chúng đều dựa trên việc tràn lụt khởi tạo của mạng (sử dụng tiếp cận RFP) và sau đó dựa trên tính toán đường ngược lại ngắn nhất. Chúng ta giả sử rằng bảng định tuyến điểm tới điểm P2P là sẵn có. Điều này được thực hiện như sau.

- Nếu router nhận được gói tin đa điểm multipoint từ nguồn S đến nhóm G, đầu tiên nó kiểm tra (sử dụng bảng định tuyến P2P) mà giao diện nhận đầu vào tương ứng với gói tin S: điều này có nghĩa rằng router này là đường đi ngắn nhất từ nguồn (do đó, đây được gọi là "đường trở lại ngắn nhất"). Nếu kết quả là âm thì nó gửi thông điệp "delete (S.G)", ví dụ bản tin đến nguồn yêu cầu ngừng việc gửi đến nó gói tin từ S đến G.

- Nếu kết quả dương thì router gửi một bản sao của thông điệp đến tập T của tất cả các giao diện mà qua đó nó không nhận được yêu cầu "delete(S,G)". Nếu T rỗng, thì nó hủy gói tin và gửi thông điệp "delete(S,G)" đến giao diện mà qua đó nó nhận được thông điệp.

7.3.2. Định tuyến dựa trên điểm RP

Centralized mcast (mcast tập trung) (CtrMcast) tương tự như PIM-SM (Chế độ Sparse của PIM [11]). Đó là điểm gặp (Rendezvous Point – RP). Một cây chia sẻ được xây dựng cho một nhóm multicast có gốc tại điểm RP này. Trung tâm tính toán tập trung được sử dụng để tính toán việc chuyển (forward) các cây đi và cài đặt trạng thái chuyển multicast, S,G (trạng thái S tương ứng với nguồn của gói tin và G tương ứng với địa chỉ của nhóm multicast) tại các nút thích hợp khi bên nhận với tham gia vào nhóm. Các gói dữ liệu từ bên gửi đến nhóm là unicast đến RP. Việc multicast từ RP đến nhóm được thực hiện theo cây đường đi ngắn nhất.

Chế độ ST là phiên bản đơn giản hóa của giao thức định tuyến của chế độ Sparse ở trên. Giao thức này có phiên bản hai hướng trong ns được gọi là BST, mà được sử dụng trong phiên bản chuẩn CBT [7] và trong giao thức BGMP đối với multicast bên trong miền [8]

Trong các giao thức dựa trên điểm RP, tất cả các luồng lưu lượng multicast đều đi qua điểm RV và do đó gây ra hiện tượng thắt cổ chai. Nút này ảnh hưởng tới hiệu năng toàn mạng. Vấn đề khác với tiếp cận này là lưu lượng chuyển qua các đường không tối ưu. Ưu điểm của tiếp cận này là:

- Tính đơn giản của thông tin trạng thái: chỉ một entry cho một nguồn một nhóm.
- Báo hiệu không liên quan tới toàn bộ mạng.

Chú ý rằng, trong PIM-SM, có thể chuyển mạch để tối ưu các cây dựa trên nguồn (S,G) thay cho việc định tuyến qua điểm RV. Điều này xảy ra nếu tốc độ của dữ liệu nguồn vượt quá ngưỡng. Do đó, điểm RV có thể dừng hiện tượng thắt cổ chai nếu tốc độ lưu lượng lớn. Chế độ ST không mô phỏng các đặc tính này.

7.4. Định tuyến dựa trên điểm RP

Multicast yêu cầu cải tiến cho các nút và các liên kết của mạng. Do đó, ns có điều kiện rõ ràng từ lớp mô phỏng trước khi tạo topology. Do đó, chúng ta bắt đầu từ câu lệnh đặc trưng:

```
Set ns [new Simulator]
$ns  multicast
```

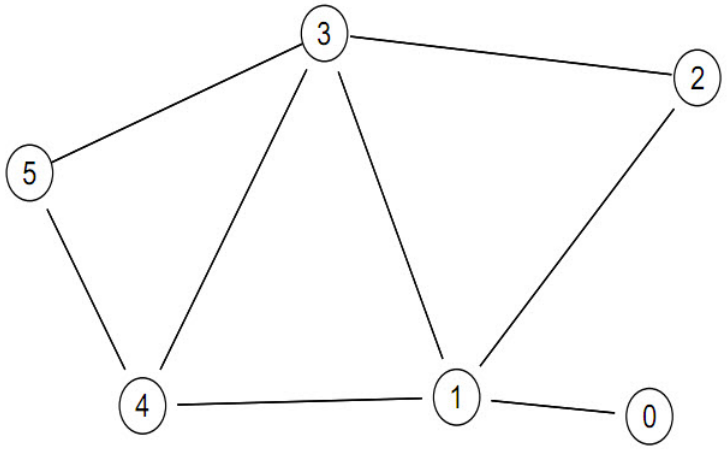
Trong kịch bản tcl, chúng ta định nghĩa địa chỉ nhóm qua sử dụng câu lệnh `"set group1 [Node allocaddr]"`.Sau đó, chúng ta định nghĩa một ứng dụng và một giao thức truyền tải, một mặt gán vào nút nguồn được đưa ra và mặt khác gán vào nhóm đích.

Chúng ta xét đến giao thức DM dưới đây. Khi nguồn S gửi đến một nhóm G trở thành active, nó bắt đầu tràn lụt thông tin qua mạng đến cây được gắn vào tương ứng với nhóm G. Khi một lá không tham gia vào nhóm multicast mà nhận một gói tin từ nhóm đó, nó gửi thông điệp đến giao diện đến để xóa nó từ cây (S,G) (gói "prune"). Sau đó, nó quảng bá trở lại nguồn: một nút nhận một thông điệp từ tất cả các liên kết đầu ra của nó trong cây của (S,G) yêu cầu xóa các liên kết này, gửi trở lại giao diện đến một thông điệp để xóa nó từ cây (S,G).

Một nguồn sẽ ngừng hoàn toàn việc gửi gói tin nếu không có kết nối đến bên nhận trong nhóm đó; nó sẽ tiếp tục lại việc gửi gói tin khi bên nhận kết nối.

Một ví dụ của cấu hình multicast với sáu nút mạng được mô tả trong hình 7.2

Chúng ta xét đến mạng được mô tả trong hình 7.1



Hình 7.2: Ví dụ về định tuyến multicast

```
Set ns  [new Simulator]
$ns multicast

Set  f  [open  out.tr  w]
$ns  trace-all   $f
$ns  namtrace-all  [open  out.nam  w]

$ns  color  1  red
#the nam colors for the prune packets
$ns  color  30  purple
#the nam colors for the graft  packets
```

```
$ns color 31  green
#allocate a multicast address;
Set group  [Node allocaddr]
#nod is the number of nodes
Set  nod  6
#creat multicast capanle nodes;
For  {set  i  1}  {$i <=  $nod}  {incr  i}   {
}
#creat links between the nodes
$ns  duplex-link  $n(1)  $n(2)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(2)  $n(3)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(2)  $n(4)  0.5Mb  10ms  DropTail
$ns  duplex-link  $n(2)  $n(5)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(3)  $n(4)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(4)  $n(5)  0.5Mb  10ms  DropTail
$ns  duplex-link  $n(4)  $n(6)  0.5Mb  10ms  DropTail
$ns  duplex-link  $n(5)  $n(6)  0.5Mb  10ms  DropTail
#configure multicast protocol;
Set  mproto DM
#all nodes will contain multicast protocol agents;
Set mrthandle  [$ns  mrtproto  $mproto]

Set  udp1  [new  Agent/UDP]
Set  udp1  [new  Agent/UDP]

$ns attach-agent  $n(1)  $udp1
$ns attach-agent  $n(2)  $udp2

Set  src1 [new  Application/Traffic/CBR]
$src1 attach-agent  $udp1
#udp1 set  dst_addr_   $group
$udp1  set  dst_port_   0
$src1 set random_  false
#creat  receiver  agents
Set  rcvr  [new  Agent/LossMonitor]
#joining and leaving the group;
$ns  at  0.6  ‘‘$n(3)  join-group  $rcvr  $group’’
$ns  at  1.3  ‘‘$n(4)  join-group  $rcvr  $group’’
```

```
$ns  at  1.6  ‘‘$n(5)  join-group  $rcvr  $group’’
$ns  at  1.9  ‘‘$n(4)  leave-group  $rcvr  $group’’
$ns  at  2.3  ‘‘$n(6)  join-group  $rcvr  $group’’
$ns  at  3.5  ‘‘$n(3)  leave-group  $rcvr  $group’’

$ns  at  0.4  ‘‘$src1  start’’
$ns  at  2.0  ‘‘$src2  start’’

$ns  at  4.0  ‘‘finish’’
Proc  finish {}  {
Global ns
$ns  flush-trace
Exec  nam out.nam  &
Exit 0
}
$ns  run
```

Bảng 7.2: Ví dụ cho multicast với mô hình DM: pimdm.tcl

Tác nhân điều khiển mất mát (Loss Monitor Agent): ở đây chúng ta sử dụng tác nhân điều khiển mất mát là một gói tin "sink agent" duy trì sự thống kê về lưu lượng nhận được, như thông tin về lượng thông tin nhận được cũng như bị mất. Cụ thể, chúng ta có thể truy cập các biến trạng thái sau: `nlost_` (số các gói tin mất mát), `npkts_` (số các gói tin nhận được), `bytes_` (số các byte nhận được), `lastPktTime_` (thời gian tại đó gói tin cuối cùng nhận được) và `expected_` (số trình tự mong muốn của gói tin tiếp theo). Ta có thể sử dụng thay cho tác nhân điều khiển mất mát, Null agent, như chúng ta đã làm trước đó, ví dụ: `set rcvr [new Agent/Null]` thay cho `set rcvr [new Agent/LossMonitor]`.

7.4.1. Chế độ DM

Câu lệnh set mproto DM chỉ ra rằng chúng ta sử dụng giao thức Dense Mode. Mặc định, pimDM được sử dụng. Để sử dụng chế độ dvmrp, phải thêm dòng sau

```
DM set CacheMissMode dvmrp
```

trước dòng `set mproto DM`

Trong chế độ DM, việc tràn lụt xảy ra theo chu kỳ, do đó phát hiện các nút được kết nối với nhóm. Giá trị bộ định thời cho chu kỳ được đưa ra trong biến được gọi là PruneTimeout

. Mặc định, giá trị này là 0.5 giây. Nếu ta yêu cầu giá trị khác , ví dụ như 0.8 giây, thì thêm vào kịch bản tcl câu lệnh:

```
DM set PruneTimeout 0.8
```

chỉ trước dòng set mproto DM.

7.4.2. Định tuyến với điểm RV tập trung

Đối với chế độ tập trung cần:

```
#configure  multicast protocol;
Set  mproto CtrMcast
# all nodes will contain multicast protocol agents;
Set mrthandle  [$ns  mrtproto  $mproto]
#set RV and bootstrap points
$mrthandle set_c_rp  $n(2)
```

Ở đây chúng ta chọn n(2) là điểm RP.

Trong cả chế độ tập trung cũng như chế độ ST, báo hiệu (các gói tin prune) không được mô phỏng.

Chúng ta thể hiện trong bảng 7.3 , cùng một ví dụ như trong pimdm.tcl (bảng 7.2) nhưng với giao thức định tuyến BST.

```
Set ns [new  Simulator  -multicast  on]
Set f [open out.tr  w]
$ns  trace-all  $f
$ns  namtrace-all [open out.nam  w]
$ns  color  1  red
# the nam colors for the prune packets
$ns  color  30  purple
#the nam colors for the graft packets
$ns color  31  green

#allocate a multicast address;
Set group  [Node  allocaddr]

#nod is the number of nodes
Set nod 6
```

```
#create multicast capable nodes;
For  {set  i  1}  {$i <= $nod}  {incr  i}  {
Set n($i) [$ns node]
}
#create link between the nodes
$ns  duplex-link  $n(1)  $n(2)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(2)  $n(3)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(2)  $n(4)  0.5Mb  10ms  DropTail
$ns  duplex-link  $n(2)  $n(5)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(3)  $n(4)  0.3Mb  10ms  DropTail
$ns  duplex-link  $n(4)  $n(5)  0.5Mb  10ms  DropTail
$ns  duplex-link  $n(4)  $n(6)  0.5Mb  10ms  DropTail
$ns  duplex-link  $n(5)  $n(6)  0.5Mb  10ms  DropTail
#configure multicast protocol;
BST set  RP_($group)  $n(2)
$ns mrtproto BST
Set udp1  [new Agent/UDP]
Set udp2  [new Agent/UDP]
$ns attach-agent  $n(1)  $udp1
$ns attach-agent  $n(1)  $udp1
$ns attach-agent  $n(2)  $udp2
Set src1  [new Application/Traffic/CBR]
$src1 attach-agent  $udp1
$udp1  set  dst_addr_  $group
$udp1  set  dst_port_  0
$src1  set  random_false
Set src2  [new Application/Traffic/CBR]
$src2 attach-agent  $udp2
$udp2  set  dst_addr_  $group
$udp2  set  dst_port_  1
$src2  set  random_false
#joining and leaving the group;
$ns  at  0.6  ‘‘$n(3)  join-group  $rcvr  $group’’
$ns  at  1.3  ‘‘$n(4)  join-group  $rcvr  $group’’
$ns  at  1.6  ‘‘$n(5)  join-group  $rcvr  $group’’
$ns  at  1.9  ‘‘$n(4)  leave-group  $rcvr  $group’’
$ns  at  2.3  ‘‘$n(6)  join-group  $rcvr  $group’’
$ns  at  3.5  ‘‘$n(3)  leave-group  $rcvr  $group’’
```

```
Proc finish {} {
Global ns
$ns  flush-trace
Exec  nam  out.nam &
Exit 0
}
$ns run
```

Bảng 7.3: Ví dụ cho multicast với mô hình điểm RV: bts.tcl

7.5. Khảo sát mô phỏng pimdm.tcl

Dense mode: pimdm và drmrp. Nếu chúng ta chạy mô phỏng và khảo sát bám vết, chúng ta sẽ thấy rằng trong việc thêm các gói tin CBR, có hai kiểu gói tin khác nhau: gói tin "prune" và gói tin "graft". Vai trò của gói tin prune được gửi bởi nút N là báo hiệu cho nút đó rằng đã gửi gói tin trước đó đến N để ngừng việc gửi các gói tin đến N. Gói tin "graft" là khởi tạo tín hiệu từ một nút muốn tham gia vào nhóm (sau khi nó bị ngắt kết nối). Trong hiển thị NAM trong mô phỏng của chúng ta, các gói tin graft là đèn xanh và prune là màu đỏ.

Chúng ta có thể thấy rằng tại thời điểm 0.4, nút 0 bắt đầy gửi các gói tin CBR tràn lụt qua mạng. Nhưng không có bên nhận tại nhóm multicast, do đó kết quả là, các gói tin prune trở về nguồn và việc truyền kết thúc (thời điểm 0.579). Tại thời điểm 0.6, một gói tin graft được gửi từ nút 2 (nút muốn tham gia vào nhóm) đến nút 1, và sau đó từ nút 1 đến nút 0. Nút 0 sau đó bắt đầu việc truyền. Tại thời điểm 0.9978, lại có việc thử kiểm tra xem có kết nối được với các nút bên nhận trong nhóm ngoài nút 2 và mạng được tràn lụt lại, các gói tin prune trở về để ngừng việc truyền đến các nút 3, 4, 5.

Chế độ tập trung: chúng ta thấy trong các gói tin bám vết được đóng gói được gửi từ nguồn đến điểm RV của kích thước 230 byte. Header sau đó được xóa bởi điểm RV mà sau đó chuyển gói tin (210 byte) đến các thành viên trong nhóm.

7.6. Bài tập

Bài tập 7.6.1. Chạy chương trình ex2.tcl (xem bảng 7.1) ghi chú lệnh `"$ns  rtproto DV"` và giải thích điều gì xảy ra.

Bài tập 7.6.2. Chạy chương trình ex2.tcl (xem bảng 7.1) với lệnh `"$ns  rtproto DV"` và giải thích sự khác nhau với định tuyến tĩnh trước đó.

Bài tập 7.6.3. Thay đổi và chạy mô phỏng ex2.tcl (xem bảng 7.1) trong khoảng 20 giây với định tuyến tĩnh nhưng với một kết nối hàm mũ ON-OFF động, với thời gian trung bình ON là 3 giây và thời gian trung bình OFF là 0.5 giây. Phân tích hành vi của kết nối TCP và hành vi của time-out.

Bài tập 7.6.4. Chạy kịch bản pimdm.tcl (xem bảng 7.2). Có bao nhiêu gói gin CBR được truyền từ mỗi một nguồn, và có bao nhiêu gói tin bị mất? Có bao nhiêu gói tin CBR được nhận tại các nút mà không cần chúng (chính xác hơn, có bao nhiêu gói tin prune được tạo)?

Bài tập 7.6.5. Xét dấu vết đạt được từ kịch bản pimdm.tcl. Tại thời điểm 1.8375 chúng ta bắt đầu có mất mát tại nút 0. Tại thời điểm 2.481 các gói tin bắt đầu mất cũng tại nút 1. Giải thích sự mất mát này.

Bài tập 7.6.6. Chạy chương trình pimdm.tcl với chế độ dvmrp của DM. Bạn có quan sát được sự khác biệt nào giữa dvmrp và phiên bản pimDM?

Bài tập 7.6.7. Chạy phiên bản tập trung của multicast. Giải thích điều gì xảy ra khi RP được thay đổi ở nút n(5) (trong NAM nó sẽ tương ứng với nút 4, khi NAM đếm từ 0). Giải thích vì sao nó ít hiệu quả hơn việc chọn nút RP là n(2). Chúng ta có thể đánh giá hiệu quả bằng cách nào?

Chương 8

Loại bỏ ngẫu nhiên sớm

Mục lục

Người dịch: *Hà Tất Thành*

Biên tập: *Nguyễn Nam Hoàng*

Bản gốc: *NS Simulator for beginners, Chapter 6* [2]

8.1. Mô tả RED

Cơ chế quản lí bộ đệm RED được giới thiệu vào năm 1993 by Floyd và Jacobson [10] , và được mô tả sâu hơn trong RFC 2309 [11] , Nhiều tài liệu tham khảo quan trọng về RED có thể được tìm thấy ở trang web http://www.icir.org/floyd/red.html. Ý tưởng cơ bản là không nên chờ tới khi bộ đệm bị đầy để phát hiện tắc nghẽn (loại bỏ gói tin), mà bắt đầu dò tìm sự tắc nghẽ trước khi bộ đệm tràn. Các dấu hiệu của sự tắc nghẽn có thể vẫn thông

qua việc loại bỏ gói tin, nhưng giờ đây, cũng có thể thông qua việc đánh dấu các gói tin mà không cần phải loại bỏ chúng.

Một số mục tiêu của quản lí bộ đệm RED là:

1. Ưu tiên các cụm dữ liệu (burst) ngắn thuộc loại delay sensitive (nhạy cảm với trễ), nhưng không cho phép kích thước hàng đợi trung bình tăng quá nhiều. Bằng cách dùng một việc lọc "low past" nào đó cho kích thước hàng đợi, mục đích này là để phát hiện sự tắc nghẽn mà có thời gian đủ dài.

2. Các cổng loại Drop Tail và Random Drop có sự thiên vị (bias) đối với lưu lượng cụm (bursty traffic). Thật vậy, trong những bộ đệm như vậy, càng nhiều lưu lượng của một kết nối loại cụm, hàng đợi càng có khả năng tràn tại thời điểm tới của các gói tin của kết nối đó.

3. Tránh đồng bộ: Ở bộ đệm loại drop-tail, nhiều kết nối có thể nhận dấu hiệu tắc nghẽn tại cùng thời điểm dẫn đến các dao động không mong muốn ở thong lượng. Các dao động này có thể làm cho thông lượng trung bình thấp hơn và jitter cao. Để tránh đồng bộ (là tránh việc các phiên kết nối nhận tín hiệu tắc nghẽn tại cùng 1 thời điểm), các tín hiệu tắc nghẽn được chọn một cách ngẫu nhiên.

4. Điều khiển kích thước hàng đợi trung bình. Để ý rằng điều đó cũng có nghĩa là điều khiển trễ hàng đợi trung bình.

Để đạt được các mục tiêu này, RED giám sát kích thước hàng đợi trung bình avg, và kiểm tra nó có nằm giữa ngưỡng cực tiểu minth và cực đại max_{th} nào đó hay không. Nếu có, thì một gói tin tới sẽ bị loại bỏ hay được đánh dấu với xác suất $p = p(avg)$, xác suất này là một hàm tăng theo kích thước gói tin trung bình. Tất cả các gói tin tới khi avg vượt quá max_{th} sẽ được đánh dấu/loại bỏ.

Xác suất $p(avg)$ được chọn như sau. Khi kích thước hàng đợi trùng bình thay đổi trong khoảng minth và max_{th}, một xác suất p_b thay đổi tuyến tính giữa 0 và giá trị max_p nào đó chẳng hạn.

$$p_b(avg) = \max_p \frac{avg - \min_{th}}{max_{th} - min_{th}}$$

Xác suất này được dùng như $p(avg)$ nếu tài thời điểm tới của gói tin trước đó $avg \geq min_{th}$. Nếu không $p(avg)/(1 + p(avg))$.

Kích thước hàng đợi trung bình này được giám sát như sau. Ban đầu, tham số avg được đặt = 0. Sau đó, với mỗi gói tin tới, avg được đặt giá trị mới:

$$(1 - \omega_q)avq + \omega q$$

Ở đây, q là kích thước hàng đợi thực tế và ω_q là một hằng số nào đó. Nếu hàng đợi trở nên rỗng, một công thức khác được dùng để cập nhật kích thước của nó, công thức này sẽ tính đến thời gian từ khi hàng đợi trở nên rỗng và một ước lượng số các gói tin có thể đã được gửi đi trong suốt thời gian rỗi này, xem thêm[11]. Để ước lượng các gói tin sau này, ta có lẽ sẽ cần đưa một ước lượng thô kích thước gói trung bình như một tham số trong ns.

Các ví dụ về các tham số RED đã nghiên cứu ở [10] là $\omega_q = 0.002, min_{th} = 5$ gói tin, $max_{th} = 15$ gói tin, $max_p = 1/50$ và kích thước hàng đợi là 100. Thông thường, người ta cũng khai thác giá trị min_{th} nằm trong dải từ 3 đến 50, và giữ $max_{th} = 3min_{th}$

Sự thực hiện red trong ns có thể được tìn thấy trong ns-allinone-2.XXX/queue/red.cc (XXX là phiên bản, ví dụ 1b9a).

8.2. Thiết đặt các tham số RED trong ns

Các tham số của RED trong ns được đưa ra trong các đối tượng sau:

1. `bytes_`: Lấy giá trị "true" nếu ta làm việc ở chế độ "byte mode" hoặc "fale" trong chế độ gói tin (chế độ mặc đinh.). Ở chế độ "byte mode", kích thước của một gói tin tới tác động lên khả năng đánh dấu nó.

2. `queue-in-bytes_`: Kích thước hàng đợi trung bình sẽ được đo bằng bytes nến nó được đặt là “true”. Trong trường hợp đó, các đối tượng `threst_` và `maxthres_` cũng được cân bằng bởi tham số kích thước gói tin trung bình đã được ước lượng `mean_pktsize_`. Nó có “false” theo mặc định.

3. `thres_`: Là ngưỡng kích thước hàng đợi cực tiểu min_{th}

4. `maxthres_`: Là ngưỡng kích thước hàng đợi cực đại max_{th} .

5. `mean_pktsize_` Là ước lượng của kích thước gói tin trung bình tính theo byte. Giá trị mặc định là 500.

6. `q_weight_`: Tham số độ nặng ω_{th} trong việc tính toán chiều dài hàng đợi đã được tính trung bình.

7. `wait_`: Là tham số cho phép duy trì khe thời gian giữa các gói tin bị loại bỏ khi khi được đặt thành "true" (là giá trị mặc định).

8. `linterm_`: Là nghịch đảo củamax_p. Giá trị mặc định là 10.

9. `setbit_`: Có giá trị "false” trong trường hợp RED được dùng để loại bỏ gói tin, và có giá trị "true" nếu RED đánh dấu gói tin với một bít tắc nghẽn. (phiên bản ECN của TCP tương tác với các bít tắc nghẽn này).

10. `drop-tail_`: Đây là tham số mà khi đặt giá giá trị "true" thì cho phép sử dụng chính sách drop-tail khi hàng đợi tràn hoặc kích thước hàng đợi trung bình vượt quá max_{th}.

Giá trị mặc định của các tham số `q_weight_, maxthres_, thres_` tương ứng là 0.002,15,và 5 cho tới cuối năm 2001. Trong những phiên bản gần đây, chúng được cấu hình tự động.

RED có các tham số và biến khác được triển khai trong ns. Cụ thể, S.Floyd khuyến cáo trong http://www.icir.org/floyd/gentle.html cho hoạt động tốt nhất của RED (trong mô phỏng và thực thi), để sử dụng, tham số `gentle_` được đặt thành "true" (đây là giá trị mặc định kể từ tháng 4, 2001). Trong các chỉnh sửa `gentle_` về RED trong ns, xác suất loại gói tin thay đổi từ max_p đến 1 khi kích thước hàng đợi trung bình thay đổi từ `maxthres_` đến 2 lần `maxthres_`. Tùy chọn này làm cho RED trở nên mạnh mẽ hơn để thiết lập các thông số `maxthres_` và `max_p`.

Một phiên bản khác là RED thích nghi, phiên bản này điều chỉnh sự lựa chọn tham số cho các lưu lượng mạng, như được mô tả ở [12] .

Để giám sát một bộ đệm red cho trước, chẳng hạn như một bộ đệm giữa hai nút mạng `$n2` và `$n3`, có thể gõ lệnh như sau:

```
set redq [[$ns link $n2 $n3]queue]
set traceq [open red-queue.tr.w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq
```

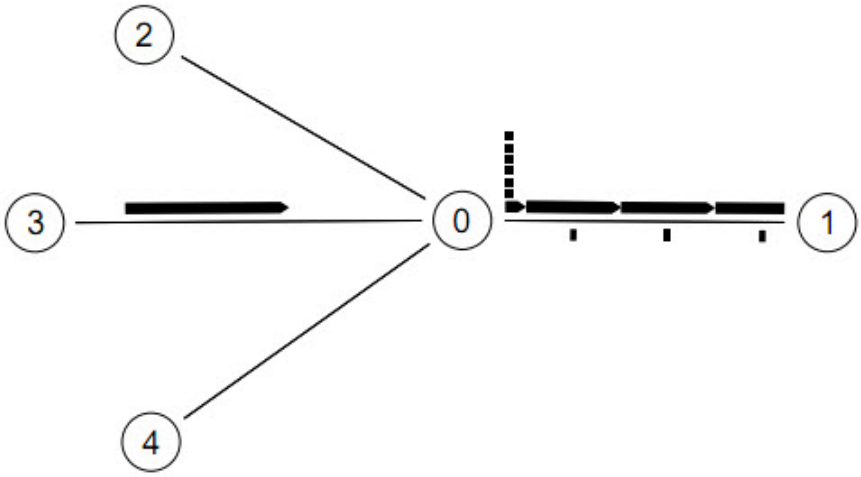
Ở đây, `curq_` là giá trị hàng đợi hiện thời và `ave_` là giá trị đã lấy trung bình. Điều này đem lại file một đầu ra (trong trường hợp của chúng ta là "red-queue.tr") với 3 cột. Cột đầu tiên chỉ ra nó là giá trị của kích thước hàng đợi hiện thời (bằng cách dùn cờ "Q") hay là kích thước hàng đợi trung bình (dùng cờ "a"). Sau đó đi đến thời điển hiệu tạo và cuối cùng là giá trị được giám sát.

8.3. Các ví dụ về mô phỏng

Ta xét mạng sau, được minh họa ở hình 9.1: Ta sẽ so sánh hoạt động của một vài cơ chế quản lý hàng đợi.

8.3.1. Bộ đệm loại Drop-Tail (Bỏ đàng đuôi)

Có cấu quản lí bộ đệm đầu tiên là cơ chế loại bỏ đằng đuôi đơn giản. Ta xét 3 kênh truyền đầu vào với độ trễn 1ms và băng thông là 10Mbps cho mỗi kênh. Kênh truyền cổ



Hình 8.1: Thiết lập mạng cho việc nghiên cứu RED

chai chung có độ trễ 20ms và găng thông 700kbps. Ta xét 3 phiên kết nối FTP sử dụng TCP và đặt kích thước cửa sổ cực đại là 8000. kích thước hàng đợi cổ chai alf 100. 3 phiên kết nối này khởi động ở các thời điểm ngẫu nhiên, được phân bố đều trong khoảng từ 0 đến 7 giây. Ở đó, trễ cho tới kênh cổ chai là 1ms. Ta chọn kích thước gói tin TCP là 552 bytes. Chú ý: trong phiên bản 2.1b9a của ns, khi ta gõ dòng lệnh

```
$tcp_src($j) set packetSize_552
```

Thì kích thước gói tin thực tế được tạo ra trong mô phỏng alf 592, do có 40 bytes tiêu đề (header) được thêm vào. Toàn bộ quá trình mô phỏng là 50s.

Bằng cách dùng tùy chọn monitor-queue mà ta đã thấy ở phần 7.3 , ta tạo một file gọi là queue.tr toàn bộ cột đầu tiên là thời gian và cột thứ 5 là kích thước hàng đợi đo bằng gói tin. Ta cũng sẽ dùng một thủ tục, goik là plotWindow, để giảm sát kích thước cửa sổ: nó tạo ra một file mà ở đó, cột đầu là thời gian, ba cột khác tương ứng với kích thước cửa sổ của 3 phiên kết nối.

```
set nt [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tf [open out.tr w]
set windowVsTime [open win w]
set param [open parameters w]
```

```
$ns trace-all $tf

#Định nghĩa thủ tục ‘finish’
proc finish {}{
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit 0
}

#Tạo các nút cổ chai và đích
set n2 [$ns node]
set n3 [$ns node]

#Tạo lien kết giữa các nút
$ns duplex-link $ns2 $ns3 0.7Mb 20ms DropTail

set NumSrc 3
set Duration 50

#Các nút nguồn
for {set j 1} {$j<=$NumSrc} {incr j} {
set S($j) [$ns node]
}

#Tạo một bộ phát ngẫu nhiên cho việc khởi động dịch vụ ftp và cho
#các trễ kênh truyền cổ chai
Set rng [new RNG]
$rnf seed 2

#Các tham số cho các biến ngẫu nhiên cho việc khởi động các kết
#nối ngẫu nhiên
#RVstart set nmin_ 0
#RVstart set nmax_ 7
#RVstart use-rng $rng
```

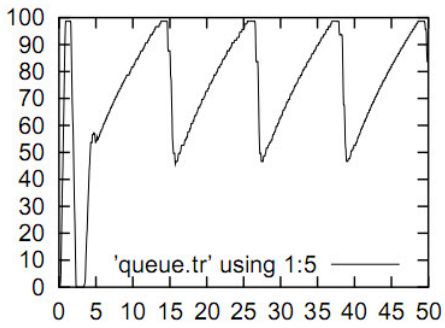
```
#Ta định nghĩa những thời điểm khởi động ngẫu nhiên cho
#mỗi phiên kết nối
for {set i 1} {$i<=$NumSrc} {incr i} {
set srartT($i) [expr [$RVstart value]]
set dly($i) 1
put $param ‘‘startT($i) $startT($i) sec’’
}

#Kênh truyền giữa nguồn và cổ chai
for {set j 1} {$j<=$NumSrc} {incr j} {
$ns duplex-link $S($j)$n2 10Mb $dly($j)ms DropTail
$ns queue-limit $S($j) $n2 20
}

#Đặt kích thước hàng đợi của kênh truyền (n2-n3) thành 100
$ns queue-limit $n2 $n3 100

set redq [[$ns link #n2 $n3]queue]
set traced [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq

#Nguồn TCP
for {set j 1} {$j<=$NumSrc} {incr j} {
set tcp_src($j) [new Agent/TCP/Reno]
$tcp_src($j) set window_ 8000
}
#Đích TCP
for {set j 1} {$j<=$NumSrc} {incr j} {
set tcp_snk($j) [new Agent/TCPSink]
}

# Các phiên kết nối
for {set j 1} {$j<=$NumSrc} {incr j} {
$ns attach-agent $S($j) $tcp_src($j)
$ns attach-agent $n3 $tcp_snk($j)
$ns $tcp_src($j) $tcp_snk($j)
```

```
}

# Các nguồn FTP
for {set j 1} {$j<=$NumSrc} {incr j} {
set ftp($j) [$tcp_src($j) attach-source FTP]
}
#Các tham số của nguồn TCP
for {set j 1} {$j<=$NumSrc} {incr j} {
$tcp_src($j) set packetSize_552
}

#Định trình các sự kiện cho các trạm FTP:
for {set i 1} {$i<=$NumSrc} {incr i} {
$ns at $srartT($i) ‘‘$ftp($i) start’’
$ns at $Duration ‘‘$ftp($i) stop’’
}

proc plotWindow {tcpSource file k}{
global ns NumSrc
set time 0.03
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
if {$k==1}{
puts -nonewline $file ‘‘$now \t $cwnd \t’’
} else {
If {$k < $NumSrc} {
puts -nonewline $file ‘‘$cwnd \t’’ }
}
if {$k == $NumSrc}{
puts -nonewline $file ‘‘$cwnd \n’’ }
$ns at [expr $now+$time] ‘‘plotWindow [$tcpSource $file $k’’}

# Thủ tục này bây giờ sẽ được gọi cho tất cả các nguồn tcp
for {set j 1} {$j<=$NumSrc} {incr j} {
$ns at 0.1 ‘‘plotWindow  $tcp_src($j) $windowVsTime $j’’
}

set qfile [#ns monitor-queue $n2 $n3 [open queue.tr w] 0.05]
```

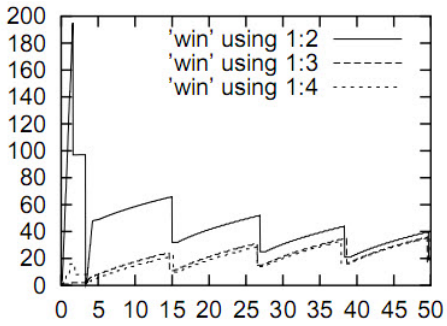
```
[$ns link $n2 $n3] queue-sample-timeout;
$ns at $Duration ‘‘$ftp($i) ‘‘finish’’
#ns run
```

Bảng 8.1: Đoạn mã tcl droptail.tcl

Trong suốt 50 giây mô phỏng, nguồn đã nhận được 7211 gói tin TCP. Tiếp theo, ta vẽ đồ thị kích thước hàng đợi (Hình 8.2) và kích thước cửa sổ (Hình 8.3)



Hình 8.2: Sự tăng kích thước hàng đợi



Hình 8.3: Kích thước cửa sổ của tất cả các phiên kết nối TCP

Ta thấy từ các hình này, có một sự động bộ cao giữa các kích thước cửa sổ: Chúng đều mất gói tin tại cùng một thời điểm. Hơn nữa ta có sự dao động cao trong các kích thước hàng đợi tương ứng với kích thước cửa sổ, và kích thước hàng đợi trung bình vào khoảng 75 gói tin. Điều này có nghĩa là có một độ trễ trung bình thêm vào, và bằng:

$$D_q = \frac{75x592x8}{700x10^3} = 507.42m\text{sec}$$

Chú ý 8.3.1 hàng đợi loại bỏ đằng đuôi có thể được mô phỏng bằng cách dùng một bộ đệm RED với$min_{th} = max_{th}$ được đặt trên kích thước hàng đợi cực đại và max_p được đặt về giá trị gần bằng 0, Điều đó sẽ cho phép ta sử dụng các công cụ giám sát độ dài hd trung bình và tức thời của RED. Tất nhiên, tham số `drop-tail_` có giá tị “true”.

8.3.2. Bộ đệm RED với cấu hình tham số tự động

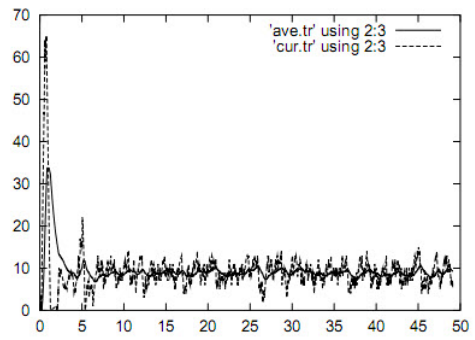
Ta chạy một mô phỏng thứ hai với các tham số như vậy. Chú ý rằng ta chọn cho độ trễ ngẫu nhiên một giá trị `seed` là 2 trong tất cả các lần mô phỏng, do khác với giá trị `seed` 0, nó sẽ đảm bảo cho các tham số ngẫu nhiên được dùng trong tất cả các lần mô phỏng.

Trong suốt 50 giây mô phỏng, nguồn tin đã nhận 7310 gói tin TCP, it hơn một chút so với trường hợp loại bỏ đằng đuôi (mà ở đó, ta có 7211 gói tin). Tiếp theo , ta vẽ kích thước hàng đợi (Hình 8.4 và 8.5)và kích thước cửa sổ (Hình 8.6).

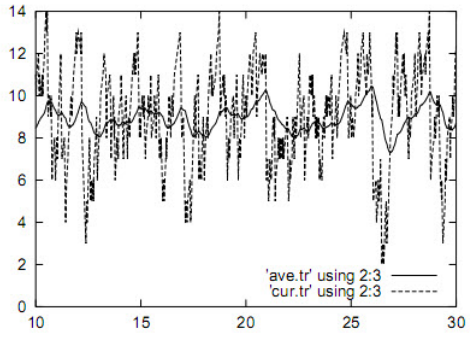
Ta thấy từ các hình này, không có sự động bộ giữa kích thước cửa sổ, và sự động bộ giữa kích thước hàng đợi trung bình thf nhỏ hơn nhiều so với trường hợp loại bỏ đằng đuôi: khoảng 10 (thay vì 75 trong trường hợp loại bỏ đằng đuôi). Do đó, trên trung bình của các phiên kết nối cũng nhỏ hơn nhiều:

$$D_q = \frac{10x592x8}{700x10^3} = 67.66msec$$

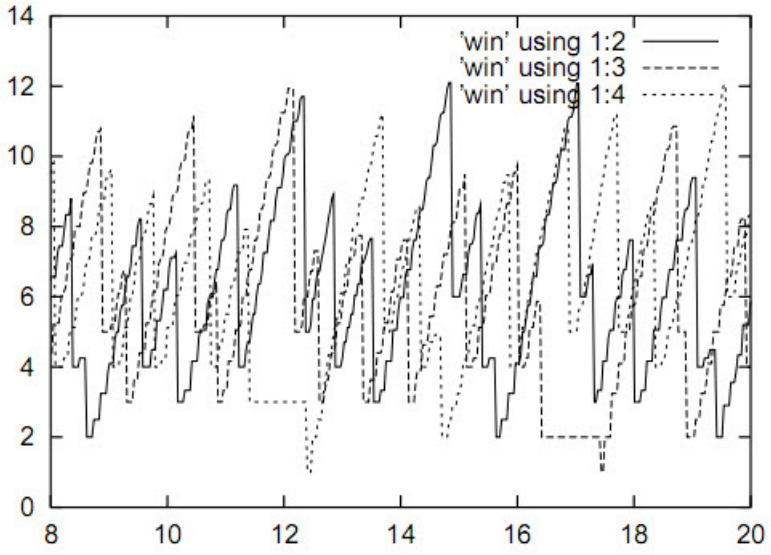
Ta quan sát thấy thay cho các dao động rộng trong kích thước hàng đợi và kích thước cửa sổ, bây giờ, ta đạt được sự biến động nhỏ hơn, nhanh hơn ở cả kích thước cửa sổ cũng như kích thước hàng đợi. Cuối cùng ta để ý rằng trong suốt thời gian mô phỏng, hàng đợi không bị tràn, không như ở trường hợp loại bỏ đằng đuôi RED đã cho phép hàng đợi lớn lên rất nhiều trong suốt khoảng thời gian ngắn lúc bắt đầu phiên kết nối,



Hình 8.4: Sự tiến triển của kích thước hàng đợi trung bình tức thời



Hình 8.5: Sự tiến triển của kích thước hàng đợi trung bình tức thời phóng to



Hình 8.6: Kích thước của sổ của tác cả các phiên kết nối TCP cho bộ đệm RED với các cấu hình tham số tự động

---

```
set nt [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf

set tf [open out.tr w]
set windowVsTime [open win w]
set param [open parameters w]
$ns trace-all $tf

#Định nghĩa thủ tục 'finish'
proc finish {}{
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exec grep ''a'' red-queue.tr > ave.tr
exec grep ''Q'' red-queue.tr > ave.tr
exit 0
}
```

```
#Tạo các nút cổ chai và đích
set n2 [$ns node]
set n3 [$ns node]

#Tạo lien kết giữa các nút
$ns duplex-link $ns2 $ns3 0.7Mb 20ms RED

set NumSrc 3
set Duration 50

#Các nút nguồn
for {set j 1} {$j<=$NumSrc} {incr j} {
set S($j) [$ns node]
}
#Tạo một bộ phát ngẫu nhiên cho việc khởi động dịch vụ ftp và cho
#các trễ kênh truyền cổ chai
Set rng [new RNG]
$rnf seed 2

#Các tham số cho các biến ngẫu nhiên cho việc khởi động các kết
#nối ngẫu nhiên
#RVstart set nmin_ 0
#RVstart set nmax_ 7
#RVstart use-rng $rng

#Ta định nghĩa những thời điểm khởi động ngẫu nhiên cho
#mỗi phiên kết nối
for {set i 1} {$i<=$NumSrc} {incr i} {
set srartT($i) [expr [$RVstart value]]
set dly($i) 1
put $param ‘‘startT($i) $startT($i) sec’’
}

#Kênh truyền giữa nguồn và cổ chai
for {set j 1} {$j<=$NumSrc} {incr j} {
$ns duplex-link $S($j)$n2 10Mb $dly($j)ms DropTail
$ns queue-limit $S($j) $n2 20
```

```
}

#Đặt kích thước hàng đợi của kênh truyền (n2-n3) thành 100
$ns queue-limit $n2 $n3 100

set redq [[$ns link #n2 $n3]queue]
set traced [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq

#Nguồn TCP
for {set j 1} {$j<=$NumSrc} {incr j} {
set tcp_src($j) [new Agent/TCP/Reno]
$tcp_src($j) set window_ 8000
}

#Đích TCP
for {set j 1} {$j<=$NumSrc} {incr j} {
set tcp_snk($j) [new Agent/TCPSink]
}



# Các phiên kết nối
for {set j 1} {$j<=$NumSrc} {incr j} {
$ns attach-agent $S($j) $tcp_src($j)
$ns attach-agent $n3 $tcp_snk($j)
$ns $tcp_src($j) $tcp_snk($j)
}

# Các nguồn FTP
for {set j 1} {$j<=$NumSrc} {incr j} {
set ftp($j) [$tcp_src($j) attach-source FTP]
}

#Các tham số của nguồn TCP
for {set j 1} {$j<=$NumSrc} {incr j} {
$tcp_src($j) set packetSize_552
```

```
}

#Định trình các sự kiện cho các trạm FTP:
for {set i 1} {$i<=$NumSrc} {incr i} {
$ns at $srartT($i) ‘‘$ftp($i) start’’
$ns at $Duration ‘‘$ftp($i) stop’’
}
proc plotWindow {tcpSource file k}{
global ns NumSrc
set time 0.03
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
if {$k==1}{
puts -nonewline $file ‘‘$now \t $cwnd \t’’
} else {
If {$k < $NumSrc} {
puts -nonewline $file ‘‘$cwnd \t’’ }
}
if {$k == $NumSrc}{
puts -nonewline $file ‘‘$cwnd \n’’ }
$ns at [expr $now+$time] ‘‘plotWindow [$tcpSource $file $k’’}
# Thủ tục này bây giờ sẽ được gọi cho tất cả các nguồn tcp
for {set j 1} {$j<=$NumSrc} {incr j} {
$ns at 0.1 ‘‘plotWindow  $tcp_src($j) $windowVsTime $j’’
}

$ns at $Duration ‘‘$ftp($i) ‘‘finish’’
#ns run
```

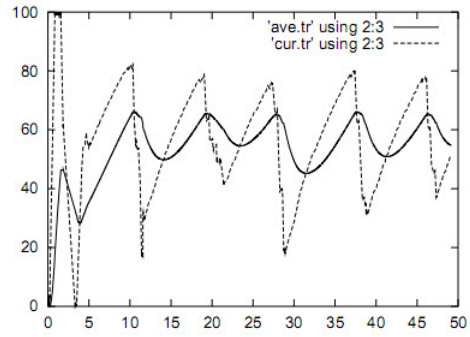
Bảng 8.2: Đoạn mã tcl droptail.tcl

8.3.3. Bộ đệm RED với các tham số khác

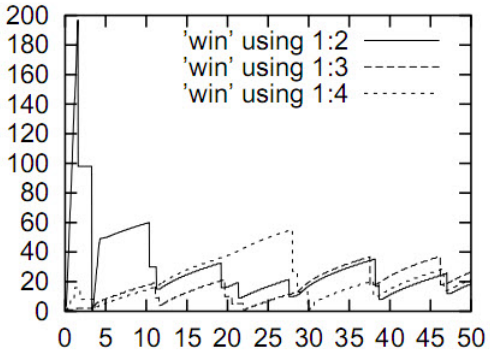
Giả sử ta muốn định nghĩa các tham số của chúng ta cho RED hơn là sử dụng các tham số mặc định. Ví dụ, giả sử ta muốn có trong ví dụ trước đây `max_th=60, min_th=40` và `q_weight_=0.02`. Sau đó ta sẽ thêm vào các câu lệnh:

```
Queue/RED set thresh_ 60
Queue/RED set maxthresh_ 60
Queue/RED set q_weight_ 0.002
```

Chú ý quan trọng: Những lệnh này nên được đặt ở lúc bắt đầu, trước khi các kênh truyền được định nghĩa! Các tiến trinh kích thước hàng đợi và cửa sở thu được được đưa ra ở Hình 8.8 và 8.7 tương ứng.



Hình 8.7: Sự thay đổi về kích thước hàng đợi



Hình 8.8: Kích thước hàng đợi của tất cả các phiên kết nối cho bộ đệm RED

Để ý rằng với các tham số mà ta chọn, độ dài hàng đợi được giữ quanh một giá trị trung bình là 50. Số các gói tin TCP nhận được trong suốt quá trình mô phỏng là 7212

8.4. Giám sát các luồng

Chúng tôi giới thiệu trong phần này bộ giám sát hàng đợi, là một cách hiệu quả để giám sát các đại lượng trên mỗi hàng đợi như lượng lưu lượng đã được truyền đi và bị mất. Chúng tôi sẽ sủa đoạn mã ns của chương trình `shortTcp2.tcl` (bảng **??**) để gộp một bộ đệm RED với chức năng giám sát.

Một luồng giám sát một kênh đơn công, vị vậy, trước hết ta định nghĩa kênh truyền ta muốn giám sát:

```
set flink [$ns simplex-link $N $D 2Mb 1ms RED]
```

và sau đó bộ giám sát hàng đợi được định nghĩa như sau đối với kênh truyền này:

```
set monfile [open mon.tr w]
set fmon [$ns makeflowmon Fid]
```

```
$ns attach-fmon $flink $fmon
#fmon attack $monfile
```

Khi chúng ta kích hoạt việc giám sát, ta nhận được các thống kê cho tới thời gian kích hoạt trong một file. Điều này được làm như sau:

```
$ns at $time "$fmon dump
```

Tiếp theo chúng tôi trình bày đoạn mã đầy đủ `shortRed.tcl` ở bảng **??** mà cho phép chúng ta nghiên cứu các phiên TCP ngắn tương tác với một bộ đệm RED.

```
set ns [new simulattor]
# Có nhiều nguồn, mà mỗi nguồn tao ra nhiều phiên TCP cùng chia sẻ một
#kênh truyền cổ chai và một đích đến chung. Số hiệu của chúng được cho
#bởi tham số NodeNb

# S(1)         ------
#   .               |
#   .          -----N----------D
#   .               |
# S(NodeNb)    ------

set Out [open Out.ns w];
set Conn [open Conn.tr w];
set tf [open out.ns w];
$ns trace-all $tf

ser NodeNb 6;
set NumberFlows 253;
set sduration 50;

#Khi các tham số sau đây được bình luận, hàng đời RED được
#cấu hình một cách tự động.
# Queue/RED set thresh_ 5
# Queue/RED set maxthresh_ 15
# Queue/RED set q_weight_ 0.002
#Định nghĩa topology
set N [$ns node]
```

```
set D [$ns node]
set flink [$ns simplex-linh $N $D 2Mb 1ms RED]
$ns simplex-link $N $D 1Mb 1ms Droptail
$ns queue-limit $N $D 50
#Giám sát hàng đợi, RED
set redq [[$ns linh $N $D] queue]
set traceq [open red-queue.tr w]
$redq trace curq_
$redq trace ave_
$redq attach $traceq

#nút mạng và kênh truyền
for {set j 1} {$j<=$NumSrc} {incr j} {
set S($j) [$ns node]
$ns duplex-link $S($j)$N 100Mb 1ms DropTail
$ns queue-limit $S($j) $N 100
}

#thiết lập bộ giám sát luồng
set monfile [open mon.tr w]
set fmon [$ns makeflowmon Fid]
$ns attach-fmon $flink $fmon
#fmon attack $monfile
#Các phiên kết nối, nguồn đích TCP
for {set i 1} {$i<=$NodeNb} {incr i} {
for {set j 1} {$j<=$NumFlows} {incr j} {
set tcpsrc($i,$j) [new Agent/TCP/newreno]
set tcp_snk($i,$j) [new Agent/TCPSink]
set k [expr $i*1000 + $j];
$tcpsrc($i,$j) set fid_ #k
$tcpsrc($i,$j) set window_ 2000
$ns attach-agent $S($i) $tcpsrc($i,$j
$ns attach-agent $D tcp_snk($i,$j)
$ns connect tcpsrc($i,$j tcp_snk($i,$j)
set ftp($i,$j) [$tcpsrc($i,$j attach-source FTP]
}
}
# Các bộ tạo các file có kích thước ngẫu nhiên
```

```
set rng1 [new RNG]
$rng1 seed 0
set rng2 [new RNG]
$rng2 seed 0
#Tạo hai kiểu phân bố ngẫu nhiên cho hai bộ tạo rng1 và rng2
# những khe thời gian đến ngẫu nhiên của bộ phát TCP ở mỗi nguồn i
set RV [new RandomVariable/Exponential]
$RV set avg_ 0.3
$RV use-rng $rng1
# Kích thước ngẫu nhiên của các file truyền đi
set RVSize [new RandomVariable/Pareto]
$RVSize set avg_ 10000
$RVSize set shap_ 1.5
$RVSize set use-rng $rng2

#Bây giờ ta định nghĩa thời gian bắt đầu của các phiên truyền và kích
#thước truyền
#Việc đến của một phiên tuân theo quá trinh Poison

for {set i 1} {$i<=$NodeNb} {incr i} {
set t [$ns now]
for {set j 1} {$j<=$NumFlows} {incr j} {
# đặt thời điểm khởi đầu cho phiên truyền kế tiếp từ nguồn và
# các thuộc tính
set t [expr $t + [$RV value]]
$tcpsrc($i,$j) set starts $t
$tcpsrc($i,$j) set sess $j
$tcpsrc($i,$j) set node $i
$tcpsrc($i,$j) set size [expr [$RVSize value]]
$ns at [$tcpsrc($i,$j) set starts] ‘‘$ftp($i,$j) send [$tcpsrc($i,$j) set size]’’
# Cập nhật sô luồng
$ns at [$tcpsrc($i,$j) set starts] ‘‘countFlows $i 1’’
}
}
for {set j 1} {$j<=$NodeNb} {incr j} {
set Cnt($j) 0
}
# Thủ tục sau đây được gọi tại bất kỳ thời điểm nào khi
```

```
# một phiên kết nối kết thúc
Agent/TCP instproc done {} {
global tcpsrc NodeNb NumberFlows ns RV ftp Out tcp_snk RVSiz
# In vào biến Out các tham sô: node, phiên,
# thời gian khởi đầu, thời gian kết thúc,
# giai đoạn làm việc, số gói tin truyền, số byte truyền, thông lượng.
Set duration [expr [$ns now]] - [$self set starts]]
puts $Out ‘‘[$self set node] \t [$self set sess] \t\
[$self set starts] \t [ns now] \t $duration \t\
[$self set ndatapack_] \t [$self set ndatabytes_] \t\
[$self set nrexmibytes_] \t [expr [$self set ndatabytes_]/$duration]’’
countFlows [$self set node] 0
}
# Thủ tục tính toán lặp sau đây cập nhật số kết nối như một hàm theo thời gian.
# Cứ 0.2s nó in ra $Com. Điều này được thự hiện bằng cách gọi thủ tục
# với tham số "sign" bằng 3 (trong đó, trường hợp tham số ‘‘ind’’ không
# đóng vai trờ gì cả), Thủ tục này cũng được gọi ra bởi thủ tục đã
# thực hiện khi một kết nối từ nguồn thứ i kết thúc bằng việc gán thám số
# "sign" giá trị 0, hay khi nó bắt đầu, bằng việc gán cho nó
# giá trị 1 (i được truyền qua biến ‘‘ind’’).
proc countFlows {ind sign}(
global Cnts Com NodeNb
ser ns [Simulator instance]
if {#sign==0} {set Cnts($ind)} [expr $Cnts($ind)-1]
} else {#sign==1} {set Cnts($ind)} [expr $Cnts($ind)+1]
} else {
  puts -nonewline $Conn ‘‘[$ns now] \t’’
  set sum 0
  for {set j 1} {$j<=$NodeNb} {incr j}{
  puts -nonewline $Conn ‘‘[$ns now] \t’’
  set sum [expr $sum + Cnts($j)]
}
puts $Conn ‘‘$sum’’
$ns at [expr [$ns now] + 0.2] ‘‘countFlows 1 3’’
} }
proc finish{}{
global ns tf
$ns flush-trace
```

```
close $tf
exec grep ‘‘a’’ red-queue.tr > ave.tr
exec grep ‘‘Q’’ red-queue.tr > cur.tr
exit 0
}
$ns at 0.5 ‘‘countFlows 1 3’’
$ns at [expr $sduration 0 0.01] ‘‘$fmon dump’’
$ns at $sduration finish
$ns run
```

Bảng 8.3: Kịch tcl shortRed.tcl

File giám sát luồng bao gồm thông tin cụ thể hơn về loại gói tin bị loại bỏ (drop type). Nó cho phép phân biệt giữa các kiểu loại bỏ sớm do việc từ chối các gói tin sớm, và loại bỏ thực sự do tràn bộ đệm. File này có dạng như sau:

1. Cột 1: Thời gian mà tại đó “dump” được thực hiện.
2. Cột 2 và 5: Đưa ra số nhận dạng luồng.
3. Cột 3: Trống (một trường vào 0)
4. Cột 4: Kiểu luồng.
5. Cột 6 và 7: Nguồn và đích của luồng.
6. Cột 8 và 9: Tổng số lần đến của luồng tính bằng gói tin và byte tương ứng.
7. Cột 10 và 11: Tổng số lần loại bỏ sớm của luồng tính bằng gói tin và byte tương ứng.
8. Cột 12 và 13: tổng số lần đến của tất cả các luồng tính bằng gói tin và byte tương ứng.
9. Cột 14 và 15: Tổng số lần loại bỏ sớm của tất cả các luồng tính bằng gói tin và byte tương ứng.
10. Cột 16 và 17: Tổng số lần loại bỏ của tất cả các luồng tính bằng gói tin và byte tương ứng.
11. Cột 18 và 19: Tổng số lần loại bỏ của một luồng cụ thể tính bằng gói tin và byte tương ứng.

Chú ý: để áp dùng bộ giám sát luồng, mỗi phiên kết nối TCP mà ta muố gián sát cần có một số nhận dạng luồn. Ở trường họp của chúng ta, ban đầu, ta nhận dạng một luồng bằng số thứ tự và nút nguồn của nó (ví dụ: Phiên kết nối thứ 3 khởi đầu ở nút 4). Ta chuyển điều đó thành một vector có hướng như sau:

```
set k [expr $i*1000 +$j];
$tcpsrc($i,$j) set fid_ $k
```

Quá trình mô phỏng tạo ra các file đầu ra sau:

1. Cur.tr và ave.tr giảm sát sự tiến hoá của kích thước hàng đợi và phiên bản đã lấy trung bình của nó.

2. Conn.tr cho việc giám sát số các phiên kết nối tích cực từ mỗi trong 6 nguồn (số 6 được đưa vào biến NumberFlows như là tham số trong đoạn mã) cũng như tổng các phiên kết nối tích cực, như một hàm theo thời gian.

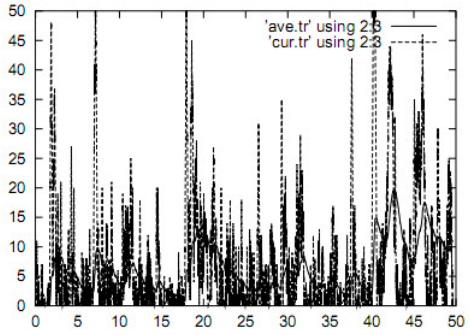
3. Out.ns dành cho việc giám sát mỗi phiên (được nhận dạng với nút nguồn và số phiên bắt nguồn từ nút đó), thời gian khởi đầu, thời gian kết thúc và thời gian của phiên két nối, số các gói tin truyền được, các byte truyền được và các byte phải truyền lại, và thông lượng mà phiên kết nối đã đạt được.

4. mon.tr là tập tin bám vết được tạo ra bởi bộ giám sát luồng, nó bao gồm số gói tin và byte truyền được và bị mất ở mỗi phiên kết nối.

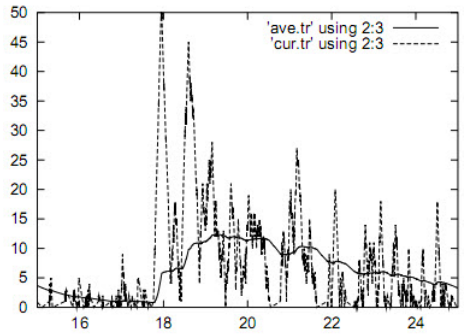
5. Out.tr là tập tin bám vết toàn cục cho tất cả các sự kiện.

Ta đã dùng ở đoạn mã trên với phiên bản RED với việc cấu hình tự động. Ta vẽ đồ thị về kích thước hàng đợi và sự biến động trung bình của nó ở hình 8.9 - 8.10. Ta thấy rằng độ dài hàng đợi thay đổi thất thường hơn nhiều so với ở trường hợp phiên kết nối TCP kiên nhẫn (mà ta đã thấy ở hình 8.4 và 8.5). Số các phiên kết nối hoạt động ở hình 8.11

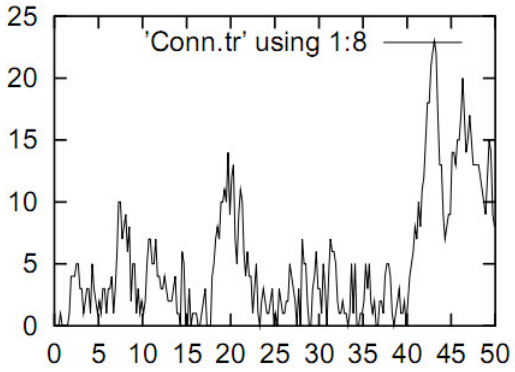
Ta bao gồm trong đoạn mã trên đây nhiều cách để giám sát. Cách giám sát trực tiếp số phiên truyền lại và các gói tin tới thông qua thủ tục "done": nó đem lại tất cả dữ liệu liên quan tới phiên kết nối. Bộ giám sát hàng đợi mặt khác đem lại thông tin cục bộ về sự mất mát ở một kênh truyền cụ thể. Nếu phiên kết nối đi qua một vài kênh truyền "cổ chai", phương pháp thứ nhất ưu việt hơn, Phương pháp thứ hai có ưu điểm là đem lại nhiều thông tin đã làm trễ mà có thể hữu ích trong việc hiểu sự đóng góp của mỗi phiên ở một vài nút gây ra tắc nghẽn ở một phiên nào đó.



Hình 8.9: Sự tiến triển của kích thước hàng đợi và giá trị trung bình của nó



Hình 8.10: Sự tiến hóa của kích thước hàng đợi và giá trị trung bình của nó được phóng to)



Hình 8.11: Số các phiên kết nối tích cực theo thời gian

8.5. Bài tập

Bài tập 8.5.1. Xem xét đoạn mã shortRED.tcl (bảng 8.3) và sửa lại chương trình để có sự điều chỉnh các tham số `thresh_, maxthresh_,q_weight_`, Những tham số này cũng như là kích thước hàng đợi trên kênh truyền N-D nên được chọn như thế nào để thông lượng đạt cực đại. Nghiên cứu điều này bằng mô phỏng và giải thích sự trả giá.

Bài tập 8.5.2. Một trong những mục đích của RED là cho phép công bằng hơn với các cụm ngắn. Phân tích thông lương và xác suất mất gói tin của một phiên kết nối như một hàn theo kích thước của nó với RED và so sánh nó với hành đợi Drop tail. Sử dụng đoạn mã `shortRED.tcl` (bảng 8.3). Thử các tham số khác nhau cho trường hợp dùng hàng đợi RED để đạt được sự công bằng tốt hơn, bài tập này dựa trên [13] .

Chương 9

Các dịch vụ phân biệt

Mục lục

Người dịch: *Nguyễn Chí Thành*

Biên tập: *Hà Tất Thành*

Bản gốc: *NS Simulator for beginners, Chapter 7* [2]

Trong mạng Internet truyền thống, tất cả các kết nối được xử lý như nhau trong mạng.

Đây là điểm trái ngược với các khái niệm mạng khác, như mạng ATM (chế độ truyền không đồng bộ), có thể đáp ứng chất lượng yêu cầu dịch vụ đối với các kết nối ở mức cao hơn về xử lý và báo hiệu liên quan đến việc chấp nhận các kết nối mới và duy trì các kết nối đang có. Hơn nữa, vì tài nguyên mạng bị giới hạn nên việc đảm bảo hiệu suất yêu cầu phải từ chối các kết nối mới nếu tài nguyên không có sẵn. Đây là điểm tương phản so với đặc tính best-effort của mạng Internet ngày nay, một nơi mà điều khiển truy nhập không được thực hiện.

Tuy nhiên, người ta đã nhận ra tầm quan trọng của việc phân biệt giữa các lớp kết nối và khả năng cấp phát tài nguyên cho các kết nối theo lớp của chúng. Do đó, một thuê bao sẽ sẵn sàng trả nhiều hơn để có được độ trễ nhỏ hơn và thông lượng lớn hơn. Đây cũng là yêu cầu đối với các ứng dụng thời gian thực trên Internet (thoại, video).

Vì lý do đó nên các dịch vụ phân biệt (Diffserv) đã được đưa ra. Diffserv dựa trên việc đánh dấu các gói tin tại rìa mạng (nơi gói tin bắt đầu đi vào mạng) theo mức hiệu suất cần cung cấp, sau đó dựa theo các dấu này, các gói tin này sẽ được xử lý khác nhau tại các nút mạng. Cách phổ biến để phân biệt gói tin là dùng bộ đệm RED, tức là dùng các tham số khác nhau cho các gói tin khác nhau. Modul NS có xử lý Diffserv được phát triển ở Nortel Networks, và chương này được dựa trên phần lớn các báo cáo của Nortel.

9.1. Mô tả chuyển tiếp có đảm bảo của Diffserv

Diffserv được thực hiện trong ns theo "Assured forwarding" (chuyển tiếp có đảm bảo ở [20] được chuẩn hóa. Một gói tin phụ thuộc vào một luồng có thể nhận 3 mức ưu tiên cùng với luồng đó, đôi khi được gọi là "drop precedence", được dùng để cung cấp xác suất mất gói thấp hơn cho các gói tin đồng bộ trong kết nối TCP, do ko giống như các gói tin khác, việc các gói tin đồng bộ bị mất có thể gây ra thời gian time-out (gián đoạn liên lạc) rất dài. Ngoài việc phân biệt mỗi luồng, thì tất cả các luồng được phân thanh các lớp (tối đa là 4 lớp), và các lớp khác nhau sẽ nhận được mức độ xử lí khác nhau.

Hơn thế nữa, có thể phân biệt giữa các luồng dịch vụ. Có 4 lớp dịch vụ được định nghĩa, các gói tin thuộc lớp nào thì được đặt vào hàng đợi của lớp đấy. Để phân biệt giữa các gói tin trong cùng một lớp, người ta thiết kế 3 hàng đợi ảo ở từng hàng đợi một trong số 4 hàng đợi trên. Với mỗi 12 tập hợp (của 4 lớp và 3 mức ưu tiên nội trong mỗi luồng tương ứng với một điểm mã mà gói tin được cho trước khi vào mạng. Trên thực tế không cần triển khai tất cả các hàng đợi và tất cả các nhóm ưu tiên.

Kiến trúc Diffserv gồm 3 thành phần:

1. Thành phần quản lý tài nguyên và chính sách: tạo ra các chính sách và phân phối chúng cho các router hỗ trợ diffserv. Một chính sách sẽ xác định gói tin được gán mức

dịch vụ nào trong mạng, việc này có thể phụ thuộc vào tính chất của tài nguyên của luồng đó (như tốc dộ trung bình và tính truyền theo loạt) và các thành phần mạng đặc biệt theo đó được thêm vào tại rìa mạng để xác định tính chất của tài nguyên. Trong mô phỏng ns, chính sách được xác định hoàn toàn trong đoạn mã tcl.

2. Các router biên: chịu trách nhiệm gán các điểm mã vào các gói tin theo chính sách được chỉ rõ bởi người quản trị mạng. Để làm được điều này, người quản trị phải đo các tham số của lưu lượng đầu vào của mỗi luồng.

3. Các router lõi: về cơ bản thì Diffserv để giữ các thông tin mật tại rìa mạng , router trong mạng chỉ đơn giản gán mức ưu tiên tương ứng với các gói tin theo mã đánh dấu của chúng. Mức ưu tiên này chuyển thanh các tham số quyết định việc lập lịch và hủy bỏ trong các router lõi.

9.2. Các router MRED

9.2.1. Mô tả chung

Thực tế có 3 bộ đệm RED ảo (gọi là MRED-Multi RED) trong mỗi hàng đợi vật lý cho phép tăng cường xử lý và tạo nên tính phụ thuộc trong hoạt động. Một cách để thực hiện là thông qua phiên bản RIO C (Rio Coupled) của MRED, trong đó xác suất loại bỏ các gói có mức ưu tiên thấp (gọi là gói tin ngoài luồng) dựa trên độ dài trung bình của tất cả các hàng đợi ảo, còn xác suất hủy gói có mức ưu tiên cao (gói trong luồng) dựa trên độ dài trung bình của hàng đợi ảo của chính hàng đợi ảo của gói tin đó.

Ngược lại, trong RIO-D (RIO De-coupled) xác suất loại bỏ mỗi gói tin dựa trên kích thước hàng đợi ảo của chính nó. Phiên bản khác là WRED (Weight Red) trong đó tất cả xác suất đều dựa trên một chiều dài đơn. Cũng có thể dùng hàng đợi dropTail

9.2.2. Cấu hình MRED trong ns

Để xác định số hàng đợi vật lý, ta dùng lệnh

```
$dsredq set numQueues_ $m
```

trong đó m có thể lấy giá trị từ 1 đến 4.

Cấu hình hàng đợi 0 là RIO-C bằng lệnh `$dsredq setMREDMode RIO-C 0`

Nếu tham số cuối cùng không được cho trước thì tất cả các hàng đợi được đặt là RIO-C. Tương tự có thể định nghĩa các RIO-C khác. Để chỉ rõ số `$n` của hàng đợi ảo chúng ta dùng

lệnh:

```
$dsredq setNumPrec $
```

Tham số Red được cấu hình dùng lệnh

```
$dsredq configQ $queueNum $virtualQueueNum $minTh $maxTh $maxP
```

Có 5 tham số: số hàng đợi, số hàng đợi ảo, $min_{th}, max_{th}, vmax_p$. Tham số qw có thể được cho (như tham số thứ 6) và nếu nó không được xác định thì sẽ nhận giá trị mặc định là 0.002.

Hàng đợi droptail dùng lệnh

```
$dsredq setMREDMode DROP
```

Cấu hình sau đó được cho trước với duy nhất 3 tham số đầu tiên:

```
$dsredq configQ $queueNum $virtualQueueNum $minTh
```

Tất cả gói tin đến bị hủy khi đạt giá trị min_{th} Như ta thấy trong chương nói về RED, để tính xác suất hủy gói tin cần ước lượng được kích thước gói tin. Với gói tin kích thước 1000 byte dùng lệnh `$dsredq meanPktSize 1000`

Lập lịch: Các chế độ lập lịch cụ thể được định nghĩa. Ví dụ như weighted round robin với trọng số hàng đợi là 5 và 1 tương ứng, được định nghĩa qua lệnh

```
$dsredq setSchedularMode WRR
$dsredq addQueueWeights 1 5
```

Cơ chế lập lịch khác là Weighted Interleaved Round Robin (WIRR), Round Robin (RR) là cơ chế lập lịch được mặc định, và có mức ưu tiên hạn chế (PRI).

Bảng PHB: một tập 4 hàng đợi cùng với các hàng đợi ảo được bổ sung với bảng PHB (cách xử lý trên 1 hop) (Per Hop Behavior). Các mục trong bảng được định nghĩa bởi: (1) điểm mã và (2) lớp (hàng đợi vật lý) và (3) "quyền ưu tiên" (hàng đợi ảo). Một mục được gán với câu lệnh có dạng:

```
$dsredq addPHBEntry 11 0 1
```

có nghĩa là điểm mã 11 được ánh xạ tới hàng đợi ảo 1 trong hàng đợi vật lý số 0.

9.2.3. Truy vấn TCL

Ba lệnh sau in ra các kết quả tương ứng (1) trong bảng PHB, (2) số hàng đợi vật lý và hàng đợi ảo và (3) kích thước trung bình của trọng số RED của các hàng đợi vật lý cụ thể (trong trường hợp này là hàng đợi 0):

```
$dsredq printPHBTable
$dsredq printStats
$dsredq getAverage 0
```

9.3. Định nghĩa các chính sách

9.3.1. Định nghĩa

Tất cả các luồng có cùng địa chỉ nguồn và đích đều lệ thuộc vào một chính sách chung. Một chính sách định nghĩa loại các chính sách, tôcs độ mong muốn, và các chỉ tiêu kĩ thuật chính sách khác. Nó chỉ rõ ít nhất 2 điểm mã. Việc lựa chọn điểm mã nào phụ thuộc vào sự so sánh giữa mục đích của luồng tin và tốc độ gửi gói tin hiện thời, và cũng có thể dựa trên các tham số phụ thuộc vào chính sách (như burstiness chẳng hạn). Chính sách quy định loại bộ đo được dùng để đo các tham số lưu lượng đầu vào liên quan. Một gói tin đến thiết bị biên sẽ làm cho bộ đo cập nhật trạng thái thay đổi tương ứng với luồng gói tin, và gói tin sau đó được đánh dấu theo chính sách. Gói tin có một điểm mã khởi đầu ứng với mức độ dịch vụ yêu cầu; việc đánh dấu gói tin có thể làm giảm mức dịch vụ so với mức dịch vụ ban đầu của gói tin.

Một bảng các chính sách được dùng trong ns để lưu trữ loại chính sách của mỗi luồng. Không phải tất cả các mục trong bảng đều được dùng thật sự. Bảng bao gồm:

1. Địa chỉ nút nguồn
2. Địa chỉ nút đích
3. Loại chính sách
4. Loại thiết bị đo
5. Điểm mã khởi đầu
6. CIR (tốc độ thông tin cho phép)
7. CBS (kích thước cụm cho phép)
8. C bucket (kích thước hiện thời của dung lượng cho phép)
9. EBS (kích thước cụm vượt mức)
10. E bucket (kích thước hiện thời của dung lượng vượt quá)
11. PIR (tốc độ thông tin tối đa)
12. PBS (kích thước cụm tối đa)
13. P bucket (kích thước hiện thời của dung lượng tối đa)
14. Thời gian đến của gói tin cuối cùng
15. Tốc độ gửi tin trung bình

16. Độ dài cửa sổ TSW (TSW là chính sách dựa trên tốc độ truyền trung bình và lấy bình quân theo kích thước cửa sổ dữ liệu tính theo giây). Giá trị mặc định là 1 giây.

Sau đây là các loại chính sách:

1. **TSW2CM (TSW2CMPolicer):** Dùng CIR và 2 thứ tự ưu tiên hủy gói. Cái nào thấp hơn thì được dùng theo xác suất khi CIR vượt mức cho phép.
2. **TSW3CM (TSW3CMPolicer):** Dùng một CIR, một PIR và 3 thứ tự ưu tiên hủy gói. Mức ưu tiên trung bình được dùng theo xác suất khi CIR vượt mức cho phép, và cái thấp nhất được dùng theo xác suất khi PIR vượt mức cho phép.
3. **Token Bucket (TokenBucketPolicer):** Dùng CIR và CBS và 2 thứ tự ưu tiên hủy gói
4. **Single Rate Three Color Marker (srTCMPolicer):** Dùng CIR, CBS, EBS, và PBS để chọn từ 3 drop precedences.
5. **Two Rate Three Color Marker (trTCMPolicer):** Dùng CIR, CBS, EBS, và PBS để chọn ra từ 3 drop precedences.

Mỗi loại chính sách trên đây định bộ đo mà nó sử dụng. Một bảng chính sách định nghĩa cho mỗi loại chính sách điểm mã khởi đầu cũng như là 1 hay 2 điểm mã giảm mức. Điểm mã khởi đầu thường được gọi là "green code" và điểm mã giảm mức thấp nhất là "red". Nếu có một điểm mã khác ở giữa, nó được gọi là "yellow".

9.3.2. Cấu hình

Để cập nhật bảng chính sách dùng lệnh "addPolicyEntry", trong đó gồm biến hàng đợi biên bao hàm hàng đợi biên, nút nguồn và nút đích của luồng, loại chính sách, điểm mã khởi đầu của nó, và sau đó là giá trị các tham số mà nó dùng như CIR, CBS, PIR và PBS như trên. CIR và PIR có đơn vị là bps, còn CBS, EBS, và PBS có đơn vị byte. Ví dụ như:

```
$edgeQueue addPolicerEntry [$n1 id][$n8 id] trTCM 10 200000 1000 300000 1000
```

Ở đây chúng ta thêm vào 1 chính sách cho luồng xuất phát từ nút `$n1` và kết thúc ở nút `$n8`. Nếu dùng chính sách TSW thì có thể thêm vào cuối chiều dài cửa sổ TSW. Nếu không được thêm vào thì lấy giá trị mặc định là 1s. Sau đó một lệnh `''addPolicyEntry''` khác chỉ rõ chính sách và điểm mã khởi đầu (với một luồng không cụ thể) sẽ định nghĩa điểm mã giảm mức với tất cả các luồng sử dụng chính sách đó với cùng một điểm mã khởi đầu. Ví dụ:

```
$edgeQueue addPolicerEntry srTCM 10 11 12
```

Ba lệnh sau đây in ra (1) toàn bộ bảng chính sách, (2) toàn bộ bảng giám sát và (3) kích thước hiện thời đo bằng byte của khối C tương ứng:

```
$edgeQueue printPolicyTable
$edgeQueue printPolicerTable
$edgeQueue getBucket
```

9.4. Mô phỏng Diffserv: bảo vệ các gói tin dễ bị tấn công

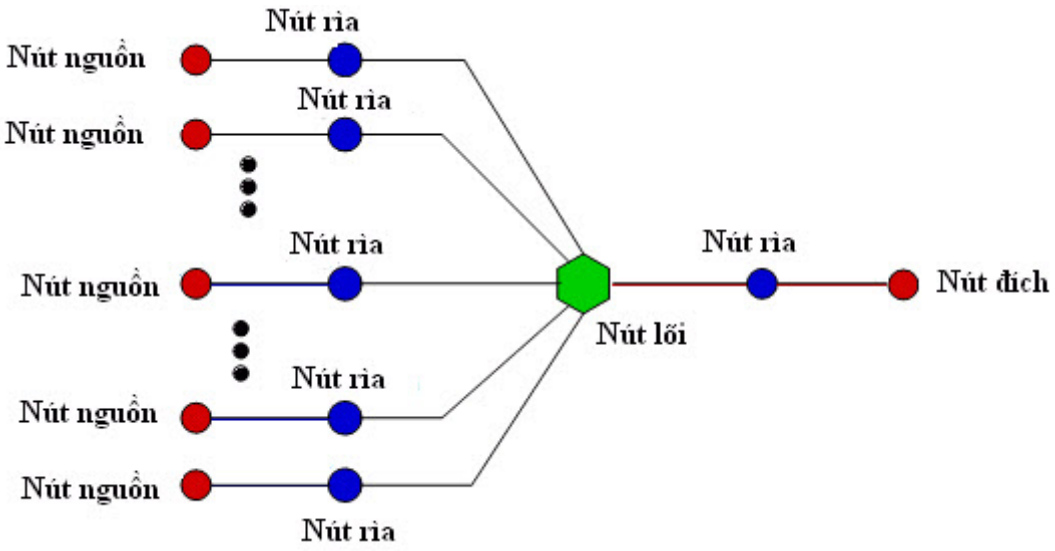
Trong kết nối TCP, việc mất một số đoạn gây ảnh hưởng nhiều hơn lên hiệu năng của phiên kết nối so với các đoạn khác. Những đoạn này là (i) các đoạn thiết lập kết nối, (ii) đoạn được gửi đi khi kết nối có một cửa sổ nhỏ, và (iii) đoạn được gửi đi sau 1 khoảng thời gian timeout hay lần truyền lại gấp. Ta gọi đó là những đoạn hay gói dễ bị tấn công. Trong một tài liệu gần đây, các tác giả có chỉ ra việc đánh dấu những đoạn này với mức ưu tiên cao hơn và thực hiện ưu tiên dùng kiến trúc Diffserv sẽ làm hiệu suất kết nối TCP được cải thiện đáng kể.Tuy nhiên, việc đánh dấu này yêu cầu các thành phần lớp mạng biết về thông tin lớp vận chuyển, ví dụ như trạng thái kết nối TCP. Mục đích của ví dụ mô phỏng mà chúng ta giới thiệu là để chỉ ra rằng có thể đánh dấu ưu tiên các đoạn nhạy cảm mà không cần bất kỳ thông tin nào của lớp vận chuyển, vì vậy sẽ đơn giản hóa việc thực hiện đánh dấu ưu tiên các gói TCP.

9.4.1. Kịch bản mô phỏngĐịnh nghĩa

Mở đầu về phân loại dịch vụ: Hai mức ưu tiên được định nghĩa. Mức cao hơn "gói tin vào" hay "green packets" và mức thấp hơn "gói tin ra" hay "red packets". Chúng ta tập chung vào chính sách đơn giản nhất có sẵn trong ns: cửa sổ thời gian trượt (TSW2CM). Một tốc độ cho phép CIR được định nghĩa cho mỗi router biên. Miễn là tốc độ kết nối dưới mức CIR thì tất cả gói tin được đánh dấu là mức ưu tiên cao. Khi tốc độ vượt mức CIR thì các gói tin được đánh dấu theo xác suất bình quân, mà tốc độ của gói tin được đánh dấu với mức ưu tiên cao ứng với CIR. Tốc độ truyền được tính theo tốc độ trung bình trên "cửa sổ TSW"; trong phần mô phỏng thì khoảng thời gian cửa sổ là 20ms.

Với kinh nghiệm của mình chúng tôi thay đổi mức CIR tại nút biên nguồn và nghiên cứu ảnh hưởng của nó đến hiệu suất.

Chúng ta sẽ xem xét một topo mạng đơn giản với 1 nút thắt cổ chai, được mô tả trên hình 9.1



Hình 9.1: Topo mạng

Mỗi nút nguồn được kết nối tới nút biên tương ứng nơi lưu lượng được đánh dấu theo tham số sẽ được chỉ rõ. Router biên được kết nối tới router lõi tại nút thắt cổ chai, và sau đó qua một router khác để tới đích.

Có 20 nút nguồn, mỗi nút tạo ra các kết nối TCP. Chúng ta sẽ thí nghiệm với mạng LAN có tốc độ cao (trễ truyền dẫn nhỏ) với đường truyền hoàn toàn đối xứng

- Đường truyền giữa nút biên và nút nguồn tương ứng có độ trễ $10\mu sec$ và băng thông 6Mbps.
- Đường truyền giữa nút biên và nút đích tương ứng có độ trễ $10\mu sec$ và băng thông 10Mbps.
- Đường truyền giữa nút trung tâm và nút biên gắn với nút nguồn có độ trễ $0.1\mu sec$ và băng thông 6Mbps.
- Đường truyền giữa nút trung tâm và nút biên gắn với nút đích có độ trễ $1\mu sec$ và băng thông 10Mbps.

Mô hình lưu lượng: Một file được truyền có phân bố xác suất Pareto với tham số shape là 1.25 và kích thước trung bình 10kbytes.

File được truyền đến mỗi nút nguồn theo tiến trình Poisson với tốc độ trung bình 5 file/giây. Nhiều phiên truyền từ cùng 1 nút nguồn có thể được kích hoạt đồng thời.

Tham số quản lý việc xếp hàng: Hàng đợi chỉ có thể được xây dựng tại router nút cổ chai, ví dụ như tại đường truyền giữa nút trung tâm và nút biên nối tới đích. Ta chọn kích thước của nó là 100 gói tin. Vì thế nên các tham số quản lý hàng đợi tại các nút khác không

ảnh hưởng đến kết quả. Trong hàng đợi thắt cổ chai tại nút trung tâm, quản lý hàng đợi multi-RED được dùng với phiên bản RIO-D; ta chọn tham số giống nhau cho cả 2 mức ưu tiên (chi tiết hơn ở phần sau). Mục đích của chúng ta trong việc chọn ra các tham số không phải là để nhận được hiệu năng cần thiết và tối ưu mà đúng ra là để tạo ra các điều kiện cho phép ta nghiên cứu ảnh hưởng của Diffserv khi giảm xác suất mất các đoạn dễ bị mất và tác động của nó đến hiệu suất của các hoạt động trên TCP (độ trễ, thông lượng). Vì lý do đó, chúng ta chọn tham số giống nhau cho 2 mức ưu tiên (điều này sẽ được giải thích ở dưới).

Kích thước hàng đợi trung bình được giám sát, với mỗi loại gói tin (đỏ, xanh) (điều này được thực hiện bằng cách dùng hàm mũ chuẩn lấy trung bình với tham số $w_q = 0.01$). Các gói tin có màu cho trước bắt đầu bị loại bỏ khi số hàng đợi trung bình của gói tin vượt quá giá trị min_w; ta chọn $min_w = 15$; xác suất hủy gói này tăng tuyến tính với kích thước hàng đợi trung bình cho đến khi đạt tới giá trị $max_w = 45$, trong đó xác suất hủy gói lấy giá trị $max_p = 0.5$. Khi vượt quá giá trị này, thì xác suất hủy gói bằng 1.

Lưu ý rằng việc phân biệt giữa các mức ưu tiên được thực hiện bằng cách dùng các tập tham số khác nhau: như việc hủy gói tin màu xanh tại kích thước hàng đợi lớn hơn. Chúng ta thường không sử dụng cách này, vì sự loại bỏ khi kích thước cửa sổ lớn sẽ gây trễ lớn hơn, trong đó một số tham số thử nghiệm cho kết quả thông lượng thấp hơn với gói tin màu xanh hơn là gói tin màu đỏ và sự suy giảm chung về hiệu suất. Với việc cho cùng tham số giống nhau với cả 2 mức ưu tiên, chúng ta có thể nghiên cứu về tác động trực tiếp của việc bảo vệ các gói tin dễ bị tấn công trên đường truyền TCP. Sự đánh dấu được thực hiện bằng cách dùng RIO-D, trong đó xác suất từ chối mỗi loại mầu phụ thuộc vào số lượng gói tin trung bình của loại đó. Vì vậy, để có ít gói tin màu xanh bị hủy hơn gói màu đỏ, chúng ta chọn thông lượng của chúng thấp hơn (cho nên cũng phụ thuộc vào kích thước hàng đợi trung bình); điều này được thực hiện bằng cách chọn giá trị CIR thích hợp mà giá trị này xác định phần gói tin bị đánh dấu màu xanh.

Mô phỏng kéo dài 80s. Tốc độ đến của các bit ở nút cổ chai là

$$\frac{20 \times 1.04 \times 10^4 \times 8}{0.22} = 7.563 Mbps$$

Kết quả trên thu được như sau: Một gói tin kích thước trung bình là 1040 bytes trong đó 1000 bytes là dữ liệu và 40 bytes là mào đầu phụ. Một file ftp trung bình được giả thiết là có 104 bytes dữ liệu, có nghĩa là tổng kích thước trung bình (bao gồm cả mào đầu phụ) khoảng 1.04x104x8 bits. Kết quả này đạt được khi nhân số nút nguồn và chia cho thời gian trung bình giữa các file đến tại 1 nút. Đoạn mã ns được cho trong bảng 7.1.

```
Set ns [new Simulator]
```

```
# Có nhiều nguồn và mỗi nguồn tạo nhiều phiên TCP
# chia sẻ cùng 1 đường truyền nút cổ
# chai và 1 đích. Số của chúng được cho bởi tham số NodeNb

# S(1) -------- E(1) ---------
#  .             |
#  . -------- E(i)   ------Core------- Ed ------------D
#  .             |
# S(NodeNb)-E(NodeNb)-

set cir0 100000; #policing parameter
set cir1 100000; #policing parameter
set pktSize 1000
set NodeNb     20;  # số nút nguồn
set NumberFlows    360;# số luồng trên một nút nguồn
set sduration 60  ;# thời gian mô phỏng

# định nghĩa các màu khác nhau cho luồng dữ liệu (cho NAM)
$ ns color 1 Blue
$ ns color 2 Red
$ ns color 3 Green
$ ns color 4 Brown
$ ns color 5 Yellow
$ ns color 6 Black

#file bao gồm thời gian truyền của các kết nối khác nhau
Set Out [open Out.ns w];

#file bao gồm số lượng kết nối
Set Conn [open Conn.tr w];

#mở file lưu vết
Set tf   [open out.tr w];
$ns trace-all $tf

#Open file lưu vết NAM
Set file2 [open out.nam w]
# $ns namtrace-all $file2
```

```
# Định nghĩa topo
Set D  [$ns node]
Set Ed  [$ns node]
Set Core  [$ns node]

Set flink [$ns simplex-link $core $Ed 10Mb 1ms dsRED/core]
$ns queue-limit $Core $Ed 100
$ns simplex-link $Ed $Core 10Mb 1ms dsRED/edge
$ns duplex-link $Ed $D 10Mb 1ms 0.01ms DropTail

For {set j 1} {$j<=$NodeNb} {incr j} {
  Set S($j) [$ns node]
  Set E($j) [$ns node]
  $ns duplex-link $S($j) $E($j) 6Mb  0.01ms  DropTail
  $ns simplex-link $E($j) $Core  6Mb  0.1ms  dsRED/edge
  $ns simplex-link $Core $E($j)  6Mb  0.1ms  dsRED/edge
  $ns queue-limit $S($j)  $E($j)  100
}

#cấu hình Diffserv
Set qEdC [[$ns link $Ed $Core] queue]
$qEdC meanPktSize 40
$qEdC set numQueues_ 1
$qEdC setNumPrec 2
For {set j 1} {$j<=$NodeNb} {incr j} {
  $qEdC addPolicyEntry [$D id] [$S($j) id] TSW2CM 10
  $cir0 0.02
}
$qEdC addPolicerEntry TSW2CM 10 11
$qEdC addPHBEntry 10 0 0
$qEdC addPHBEntry 11 0 1
$qEdC configQ 0 0 10 30 0.1
$qEdC configQ 0 1 10 30 0.1

$qEdC printPolicyTable
$qEdC printPolicerTable
```

```
Set qCEd [[$ns link $Core $Ed] queue]
# set CEd [$flink queue]
$qCEd meanPktSize $pktSize
$qCEd set numQueues_ 1
$qCEd set NumPrec 2
$qCEd addPHBEntry 10 0 0
$qCEd addPHBEntry 11 0 1
$qCEd setMREDMode RIO-D
$qCEd configQ 0 0 15 45 0.5 0 0.1
$qCEd configQ 0 1 15 45 0.5 0 0.1
For {set j 1} {$j<=$NodeNb} {incr j} {
Set qEC($j) [[$ns link $E($j) $Core] queue]
$qEC($j) meanPktSize $pktSize
$qEC($j) set numQueues_ 1
$qEC($j) setNumPrec 2
$qEC($j) addPolicyEntry [$S($j) id] [$D id] TSW2CM 10
$cir1 0.02
$qEC($j) addPolicerEntry TSW2CM  10  11
$qEC($j) addPHBEntry 10 0 0
$qEC($j) addPHBEntry 11 0 1
#$qEC($j) configQ 0 0 20 40 0.02
$qEC($j) configQ 0 0 10 20 0.1
$qEC($j) configQ 0 1 10 20 0.1

$qEC($j) printPolicyTable
$qEC($j) printPolicerTable

Set qCE($j) [[$ns link $Core $E($j)] queue]
$qCE($j) meanPktSize 40
$qCE($j) set numQueues_ 1
$qCE($j) setNumPrec 2
$qCE($j) addPHBEntry 10 0 0
$qCE($j) addPHBEntry 11 0 1
# $qCE($j) configQ 0 0 20 40 0.02
$qCE($j) configQ 0 0 10 20 0.1
$qCE($j) configQ 0 1 10 20 0.1
}
```

```
# set flow monitor
Set monfile [open mon.tr w]
Set fmon [$ns makeflowmon Fid]
$ns attach-fmon $flink $fmon
$fmon attach $monfile

#TCP Sources, destinations, connections
For {set i 1} {$i<=$NodeNb} {incr i} {
For {set j 1} {$i<=$NumberFlows} { incr j } {
Set tcpscr($i,$j) [new Agent/TCP/Newreno]
Set tcp_snk($i,$j)[new Agent/TCPSink]
Set k [expr $i*1000 +$j];
$tcpsrc($i,$j) set fid_$k
$tcpsrc($i,$j) set window_ 2000
$ns attach-agent $S($i)  $tcpsrc($i,$j)
$ns attach-agent $D $tcp_snk($i,$j)
$ns connect $tcpsrc($i,$j) $tcp_snk($i,$j)
Set ftp($i,$j) [$tcpsrc($i,$j) attach-source  FTP]
}}

# khởi tạo kích thước ngẫu nhiên cho file
Set rng1 [new  RNG]
$rng1 seed  22

# Random inter-arrival times of TCP transfer at each source i
Set RV [new RandomVariable/Exponential]
$RV set avg_  0.22
$RV use-rng  $rng1

#dummy command
Set t [$RVSize  value]

#Bây giờ chúng ta định nghĩa thời gian bắt đầu truyền và
#kích thước gói tin đến sau tiến trình Poisson

For {set i 1} {$i<=$NodeNb} {incr i}  {
Set t [$ns now]
For {set i 1} {$i<=$NumberFlows} {incr i} {
```

```
# set the beginning time of next transfer from source and attributes
$tcpsrc($i,$j) set sess  $j
$tcpsrc($i,$j) set node $i
Set t [expr $t + [$RV value]]
$tcpsrc($i,$j) set starts  $t
$tcpsrc($i,$j) set size [expr [$RVSize value]]
$ns at [$tcpsrc($i,$j) set starts]  ‘‘$ftp($i,$j) send  [$tcpsrc($i,$j)  set  size]’’
$ns at [$tcpsrc($i,$j) set starts]  ‘‘countFlows  $i  1’’
}}

For {set j 1} {$j<=NodeNb} {incr j} {
Set Cnts($j)  0
}

# the following procedure is called whenever a connection ends
Agent/TCP instproc done {} {
Global tcpsrc NodeNb NumberFlows ns RV ftp Out tcp_snk RVSize
# print in $Out:  node, session, start time, end time, duration,
# trans-pkts, transm-bytes, retrans-bytes, throughput
   Set duration [expr [$ns now] - [$self set starts]]
   Set i [$self set node]
   Set j [$self set sess]
   Set time [$ns now]
   Puts $Out ‘‘$i \t $j \t $time \t\
$time  \t  $duration  \t  [$self  set  ndataback_]  \t\
[$self set ndatabytes_]  \t [$self set nrexmitbytes_] \t\
[expr [$self set ndatabytes_] /$duration]’’
   #update the number of flows
countFlows [$self  set  node]  0
}

# Thủ tục đệ quy sau đây sẽ cập nhật số kết nối như là một
# hàm thời gian. Mỗi 0.2 nó sẽ in kết quả ra $biến $Conn.
# Điều này được thực hiện bằng cách gọi thủ tục với tham số
# ‘‘đóng dấu’’ bằng 3 (trong trường hợp tham số ‘‘ind’’ không
# đóng vai trò gì cả). Thủ tục cũng được gọi bất cứ khi nào
# một kết nối từ nguồn i và kết thúc bằng cách gán tham số
#‘‘ đóng dấu’’ 0, hay khi nó bắt đầu, bằng cách gán nó bằng 1
```

```
#(i được chuyển qua biến  '' ind'').

Proc countFlows {ind sign} {
Global Cnts Conn NodeNb
Set ns [Simulator instance]
if {$sign==0} {set Cnts($ind) [expr $Cnts($ind) - 1]}
 elseif {$sign==1} {set Cnts($ind) [expr  $Cnts($ind) + 1]}
elseif {
  Puts -nonewline $Conn ''[$ns now] \t''
  Set sum 0
  For {set j 1} {$j<=$NodeNb} {incr j} {
Puts -nonewline $Conn ''$Cnts($j)\t''
Set sum [expr $sum + $Cnts($j) ]
  }
  Puts $Conn ''$sum''
  $ns at [expr [$ns now] + 0.2] ''countFlows 1 3''
} }

$Define a 'finish' procedure
Proc finish {} {
Global ns tf qsize qbw qlost file2
$ns flush-trace
Close $file2
Exit 0
}

$ns at 0.5 ''countFLows 1 3''
$ns at [expr $sduration - 0.01] ''$fmon dump''
$ns at [expr $sduration - 0.001] ''$qCEd printStats''
$ns at $sduration ''finish''
$ns run
```

Bảng 9.1: Đoạn mã Diffs.tcl

CIR	10kbps	30kbps	100kbps	200kbps	300kbps	1Mbps	10Mbps
Các gói tin đồng bộ bị mất	120	95	53	45	17	78	114
Các gói tin đầu bị mất	125	119	90	56	37	73	115
Tổng	1699	1612	1476	1286	1088	1290	1577

Bảng 9.2: Bảo vệ các gói tin dễ bị tổn thương như một hàm của CIR

9.5. Kết quả mô phỏng

Mất gói : chúng ta kiểm tra ảnh hưởng của tốc độ đánh dấu CIR đến xác suất mất gói của gói tin SYN và của gói tin dữ liệu đầu tiên trong kết nối, như trước đây.

Ta thấy rằng cần giảm việc mất gói tin SYN bằng một thừa số của 7, và mất gói dữ liệu đầu tiên của một kết nối bằng thừa số của 3.4, để cả hai đạt được tốc độ CIR là 320kbps.

Throughput và goodput: số gói tin dữ liệu được truyền thành công trong mô phỏng hoàn toàn độc lập với tốc độ CIR: mức trung bình là 58285, với độ lệch chuẩn là 395 gói tin. Đó là vì trên thực tế tỉ lệ gói đến không phụ thuộc vào tốc độ CIR. Chú ý rằng khi xác suất mất gói thấp, thì thông lượng cũng thấp, nó luôn không đổi giống như một hàm của CIR.

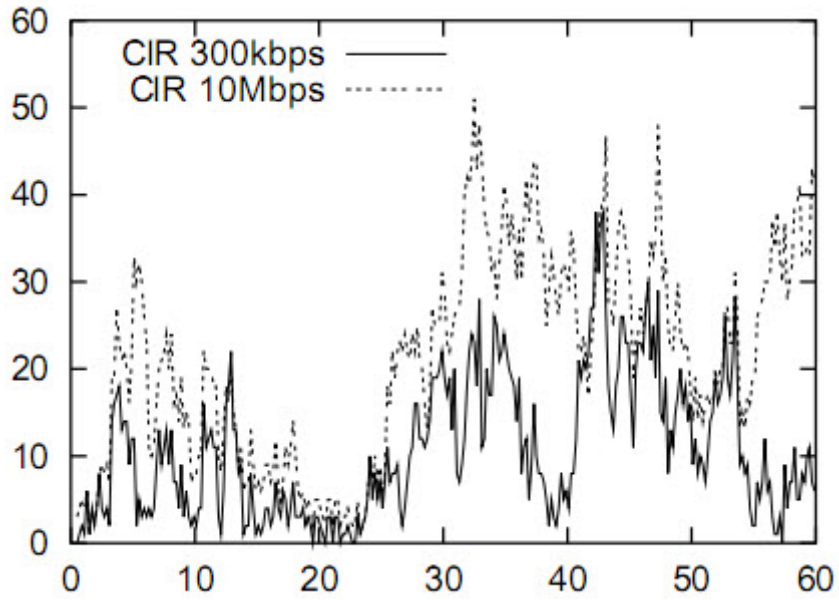
Số phiên kết nối : tổng số phiên là một hàm của thời gian được cho trong hình 9.2 cho tốc độ CIR là 200kbps (giá trị tối ưu) và cho trường hợp không có ưu tiên (CIR là 10Mbps). Chúng ta sẽ thấy từ kết quả mô phỏng rằng lược đồ đánh dấu với CIR 300kbps cho hiệu suất tốt hơn:

giảm số kết nối kích hoạt hiện có bị đánh dấu. Liên hệ với thực tế, thời gian của phiên trong đánh dấu ngắn hơn, chúng ta sẽ làm rõ trong phần sau. Thật vậy, vì tỉ lệ gói tin đến trong 1 phiên kết nối là giống nhau và không phụ thuộc vào CIR số phiên trung bình nên tỉ lệ với thời gian trung bình của 1 phiên (hệ số tỉ lệ là tỉ lệ gói đến trong phiên).

Thời gian phiên kết nối: trong bảng 9.3 ta thể hiện thời gian trung bình của một phiên như là 1 hàm của CIR. Trong khoảng tốc độ 100kbps đến 300kbps thời gian trung bình giảm bởi hệ số 1/3 (cho 100kbps) và 1/2 (cho 300kbps) với trường hợp không có ưu tiên.

CIR	10kbps	30kbps	100kbps	200kbps	300kbps	1Mbps	10Mbps
sess. duaration	0.252616	0.231077	0.167948	0.149478	0.119509	0.203202	0.225075

Bảng 9.3: Thời gian trung bình của môt phiên theo hàm CIR



Hình 9.2: Sự tiến hóa của tổng số phiên

9.6. Thảo luận và kết luận

Có 1 vài hạn chế khi thực hiện việc đánh dấu. Sự cải thiện đáng kể mà chúng ta nhận được không phải sẽ có được trong mọi hoạt cảnh, và chúng tôi đưa ra một vài chỉ dẫn mà chúng tôi đã đính chính trong những mô phỏng tiếp theo, để mô tả những hạn chế của nó.

1. Những gói tin dễ bị tấn công làm giảm hiệu suất đáng kể vì chúng gây ra thời gian gián đoạn (time-out) dài. Đặc biệt là trong trường hợp mất gói tin đồng bộ sẽ dẫn đến thời gian time-out 3s hay 6s. Trong mạng tốc độ cao thời gian truyền file rất ngắn (thường thì tổng thời gian truyền ngắn hơn nhiều thời gian time-out), vì thế ta mong muốn đạt hiệu suất cao hơn bằng cách loại trừ thời gian time-out này. Trong mạng tốc độ thấp, thì việc loại trừ thời gian time-out là không cần thiết.

2. Trong mô phỏng của chúng ta, một file có kích thước trung bình 10kbytes, là kích thước đã được tính trung bình trên mạng Internet. Điều này có nghĩa là 10% gói tin là gói SYN và có thể hơn thế, 10% khác là gói đầu trong một lần truyền. Khoảng 20% gói bị mất tương ứng với loại gói tin dễ bị tấn công này,vì thế loại trừ mất gói sẽ dẫn đến cải thiện đáng kể hiệu suất. Nếu chúng ta áp dụng với các file lớn hơn, thì phần gói tin dễ bị tấn công sẽ nhỏ hơn, để giá trị thêm vào nhỏ hơn.

9.7. Bài tập

Mô phỏng của chúng ta đã hạn chế giới hạn với lưu lượng FTP. Trong bài tập này chúng ta xem xét lưu lượng HTTP: thời gian từ lúc kết thúc truyền file đến lúc bắt đầu phiên truyền mới được phân phối theo hàm mũ với giá trị trung bình 0.1s. Cái này được gọi là "thời gian suy nghĩ". Vì thế từ mỗi nút nguồn chỉ có duy nhất 1 file được truyền tại cùng thời điểm (nhiều nhất là 1 phiên kích hoạt). Viết chương trình tcl cho mô hình lưu lượng và kiểm tra hiệu suất của nó như là một hàm của CIR, so sánh với lưu lượng FTP.

Chương 10

Mô phỏng mạng LAN

Mục lục

Người dịch: *Lê Thị Nga*
Biên tập: *Hà Tất Thành*
Bản gốc: *NS Simulator for beginners, Chapter 8* [2]

10.1. Cở sở

Mục đích của các mạng cục bộ là chia sẻ tài nguyên đáng giá hoặc không bị độc quyền sử dụng bởi một người dùng. Ví dụ như bộ nhớ ổ đĩa cứng và máy in. Mạng LAN làm đơn giản hóa việc trao đổi bản tin, chia sẻ file và tính toán phân tán. Mạng cho phép chia sẻ chi phí cài đặt và hoạt động.

Tồn tại bốn kiểu topo chính của mạng LAN: mạng hình sao, mạng lưới, mạng vòng và bus.

Trong cấu hình sao, các thiết bị đầu cuối được kết nối với hub trung tâm nhờ liên kết điểm-điểm. Điều này cho phép điều khiển tập trung các tranh chấp và xử lý quảng bá một cách dễ dàng. Hạn chế của mô hình này là sự tin cậy trong các hoạt động của nó bị giới hạn bởi sự phụ thuộc vào hub trung tâm. Hơn nữa, khả năng của hub trung tâm hạn chế số lượng trạm có thể kết nối.

Cấu hình lưới cho phép có một vài tuyến giữa hai người dùng. Ví dụ như hệ thống điện thoại.

Mạng vòng bao gồm các bộ lặp được kết nối nhờ các liên kết đơn hướng điểm-điểm để tạo thành một vòng kín. Ví dụ như IBM token ring, IEEE 802.5 và FDDI. Mạng vòng đơn giản và cho phép truyền trên liên kết này trong khi đang nhận từ một liên kết khác. Để tránh sự tranh chấp chỉ có một trạm có thể truyền tại một thời điểm, điều đó có nghĩa là việc sử dụng tài nguyên không hiệu quả.

`indexEthernet` cấu hình bus gồm có các đoạn cáp đơn được kết nối đến các trạm, các kết nối này là kiểu kết nối đa điểm: tất cả các trạm được kết nối tới bus, và nếu hai trạm cùng truyền tại một thời điểm thì có tranh chấp . Ví dụ như token bus (IEEE 802.4) và Ethernet (IEEE 802.3, IEEE 802.9 và IEEE 802.4). Bản thân bus là thụ động (không phục hồi hoặc khuếch đại tín hiệu) dẫn đến phạm vi giới hạn. Để mở rộng phạm vi cần sử dụng một bộ lặp có thể kết nối nhiều bus đến một cáp.

Trong các cấu hình mạng khác hình sao, giao thức MAC phân tán (đa truy nhập) đến các kênh cần được thực hiện. Các giao thức truy nhập này được chia làm 3 loại: chia sẻ tĩnh, truy nhập ngẫu nhiên, và truy nhập theo yêu cầu.

Trong **các giao thức chia sẻ tĩnh**, các nguồn tài nguyên được phân bổ cố định giữa các người sử dụng. Các giao thức chính là:

1. TDMA (đa truy nhập phân chia theo thời gian): thời gian được chia thành các khe khác nhau và phân bổ cho nhiều người sử dụng.

2. FDMA (đa truy nhập phân chia theo tần số): nhiều người dùng có thể truyền đồng thời nhờ sử dụng các tần số khác nhau. Việc truy nhập đến vệ tinh thường dựa trên cơ sở kết hợp TDMA và FDMA.

3. CDMA (Đa truy nhập phân chia theo mã): nhiều người sử dụng có thể truy nhập tại cùng một thời điểm và sử dụng cùng tần số, nhưng sử dụng các mã khác nhau. Các mã thường trực giao với nhau nên làm giảm nhiểu. Được sử dụng trong các mạng di động thế hệ thứ 3 như là hệ thống viễn thông di động toàn cầu (UMTS).

Trong **giao thức truy nhập ngẫu nhiên**, truy nhập ngẫu nhiên có thể gây ra xung đột và cần truyền lại. Ví dụ Aloha được sử dụng trong truyền thông vệ tinh và Ethernet. Để giảm xung đột, một trạm sẽ lắng nghe trước, nếu không có tín hiệu nào khác đang được truyền trên kênh chung và nếu có thì việc truyền hoãn lại. Điều này được gọi là "đa truy nhập cảm nhận sóng mang" (CSMA). Hơn nữa, (one can avoid transmission a long tram) Có thể tránh việc truyền lên đường truyền nếu trong suốt quá trình truyền dẫn phát hiện có xung đột. Điều đó được gọi là "phát hiện xung đột" (Collision Dectection-CD). Phiên

bản này của CSMA bao gồm cả CSMA/CD và CSMA được triển khai(implemented) trong ns, đặc biệt là CSMA/CD.

Ethernet là một giao thức MAC dựa trên CSMA/CD, và các phiên bản khác đã được chuẩn hóa, ví dụ IEEE 802.3 (10Mbps), IEEE 802.9 (đa phương tiện), IEEE 802.11 (Ethernet không dây), IEEE 802.12 ("AnyLan" tốc độ cao 100Mbps, thích hợp với các kiểu mạng LAN khác nhau) và IEEE 802.14 (tốc độ cao). Ethernet gồm có các thuật toán backoff để truyền lại. Một số M được lựa chọn ngẫu nhiên, $0 \leq M < 2^k$ với $k = min(n, 10)$, và n là tổng số xung đột đã xảy ra với các gói. Thời gian trước khi truyền lại được lấy M lần "cửa sổ xung đột" (bằng hai lần thời gian truyền tối đa của tín hiệu trong mạng cục bộ). Khi $n = 16$, phiên truyền bị hủy.

Chuẩn IEEE802.3 được triển khai rong ns. Cửa sổ xung đột được giới hạn 51,2us và mạng cục bộ giới hạn trong 5km. Nó có một vài phiên bản, ví dụ như 10base5, 10base2, 10broad36 và một số phiên bản khác. Số đầu tiên chỉ thông lượng Mbps, số thứ hai cho điều chế ("broad" có nghĩa là không điều chế). Khoảng thời gian rỗi giữa các khung truyền dẫn trong 10base5 là 9,6us. Khi sử dụng Ethernet tốc độ cao, khoảng thời gian rỗi này cũng như thời gian lan truyền cực đại giảm đi, và điều đó tăng tối đa phạm vị của mạng

Tồn tại công nghệ Ethernet khác dựa trên chuyển mạch nơi số lượng lớn đầu vào có thể tồn tại và có ít xung đột hơn.

10.2. Mô phỏng mạng LAN với ns:

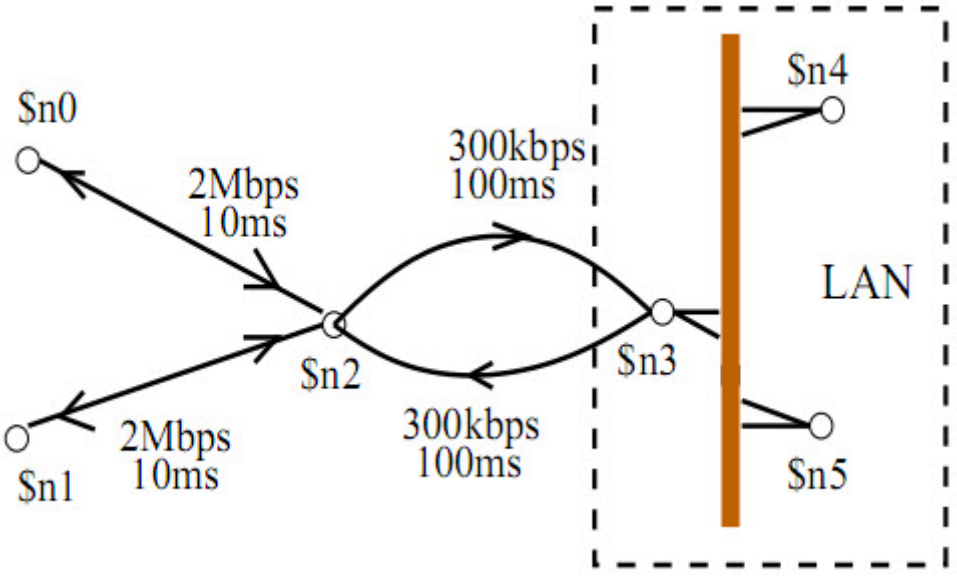
Bộ mô phỏng Ns mô phỏng 3 mức liên quan đến một mạng cục bộ: các giao thức lớp liên kết (như là ARQ (giao thức yêu cầu lặp lại tự động), giao thức MAC (ví dụ như Ethernet hoặc token ring) và kênh vật lý. Cách cơ bản để định nghĩa một mạng LAN nhờ một nhóm các node được liên kết bởi lệnh:

```
Set lan [$ns newLan <arguments>]
```

Có 7 tham số:

1. Nhóm các node, ví dụ: ‘‘$n3 $n4 $n5’’,!
2. Độ trễ
3. Độ rộng băng tần
4. Kiểu lớp liên kết (ví dụ "LL")
5. Kiểu hàng đợi nhiễu , ví dụ "QueueDrop Tail"
6. Kiểu MAC (ví dụ: "Mac/Csma/Cd" hoặc "Mac/802_3")

7. Kiểu kênh (ví dụ: "Channel")



Hình 10.1: Ví dụ mạng LAN

Theo một ví dụ, Xét mạng trong hình 11.1 là một kiểu biến đổi của mạng đã được nghiên cứu ở chương 2 (hình **??**) với các node n3, n4, và n5 được đặt trong một mạng LAN chung. Điều này có nghĩa là các gói TCP được hướng đến node 4 cũng đến node 5 (Và bị loại bỏ tai đây), TCP được gửi bởi node n4 tới node n0 cũng đến node n5 (và bị loại bỏ ở đây) và các gói UDP được gửi đến node n5 cũng đến node n4 (và bị loại bỏ ở đây).

Tập tin mã sau đó giống như ex1.tcl (bảng **??**) nhưng chúng ta thay thế các lệnh

```
$ns duplex-link $n3 $n4 0.5Mb 40Ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 30Ms DropTail
```

bằng lệnh:

```
set lan [$ns newLan ‘‘$n3 $n4 $n5’’ 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel ]
```

Chương 11

Mô phỏng mạng cho MPLS (MNS)

Mục lục

Người dịch: *Đặng Thành Chương*

Biên tập: *Nguyễn Nam Hoàng.*

Bản gốc: *Design and Implementation of MPLS Network Simulator* [4]

11.1. Giới thiệu

MNS là viết tắt của MPLS Network Simulator

Tài liệu này sẽ trình bày vấn đề thiết kế và thực thi bộ phần mềm mô phỏng MNS, hỗ trợ việc thiết lập kênh chuyển mạch nhãn ràng buộc CR-LSP đối với lưu lượng QoS, cũng như các chức năng cơ bản về MPLS, như giao thức phân bố nhãn LDP và chuyển mạch nhãn. Để có thể hỗ trợ các chức năng đó, MNS bao gồm rất nhiều thành phần; đó là CR-LDP, MPLS Classifier, Service Classifier, Admission Control, Resource Manager, và Packet Scheduler. Để đảm báo tính chính xác và hiệu quả của MNS, hai ví dụ sẽ được mô phỏng và thực hiện. Một là mô phỏng đối với các luồng lưu lượng với các QoS khác nhau, và hai là mô phỏng sự giành trước (preemtion) tài nguyên.

Cấu trúc của tài liệu này như sau: phần 2 sẽ đưa ra cái nhìn tổng quát về MNS, vấn đề thiết kế và thực thi chi tiết hơn sẽ được trình bày trong phần 3. Các ví dụ mô phỏng sẽ được giới thiệu trong phần 4, và cuối cùng là phần kết luận.

11.2. Giới thiệu

Phần này sẽ trình bày mục đích, phạm vi, kiến trúc và khả năng thực hiện của MNS. MNS được thực hiện bằng cách mở rộng phần mềm NS2, là bộ mô phỏng dựa trên IP, được bắt đầu bằng bộ mô phỏng mạng REAL bởi UC Berkeley, 1989. Hiện tại NS ver 2 là kết quả của dự án VINT.

11.2.1. Mục đích và phạm vi

Mục đích chính của phần mềm này là phát triển để có thể mô phỏng các ứng dụng khác nhau của MPLS trong điều kiện không có mạng MPLS thực tế. Bộ mô phỏng MPLS được thiết kế dựa trên các lý do sau:

- *Tính mở rộng* (Extensibility)
- *Tính dễ sử dụng* (Usability)
- *Tính "nhỏ gọn"* (Portability)
- *Tính "tái sử dụng"* (Reusability)

Phạm vi thực hiện của bộ mô phỏng MPLS:

- Chuyển mạch nhãn: các thao tác chuyển mạch nhãn, giảm (decrement) TTL, hop biên (áp chót)

- LDP: các thông điệp LDP (Request, Mapping, Withdraw, Release, Notification)
- CR-LDP: các thông điệp CR-LDP

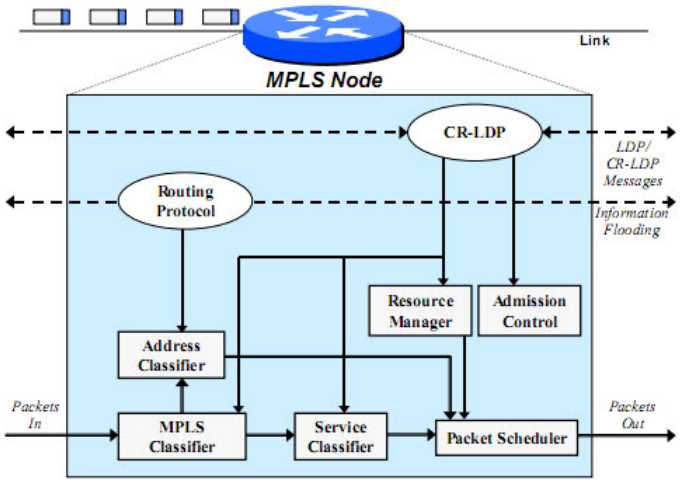
Khả năng của bộ mô phỏng MPLS liên quan đến việc thiết lập kênh LSP như sau:

- Trong LSP Trigger Strategy: hỗ trợ khởi động (trigger) theo điều khiển (control-driven) và theo dữ liệu (data-driven)
- Trong Label Allocation and Distribution Scheme: chỉ hỗ trợ lược đồ downstream trong khởi động control-driven, hỗ trợ cả 2 lược đồ upstream và downstream-on-demand trong khởi động data-driven
- Trong Label Distribution Control Mode: chỉ hỗ trợ mode độc lập trong khởi động control-driven, và cả 2 mode độc lập và ordered trong khởi động data-driven.
- Trong Label Retention Mode: chỉ hỗ trợ trong chế độ duy trì (conservative mode)
- ER –LSP based on CR-LDP: thiết lập dựa trên thông tin đường đi xác định trước bởi user.
- CR –LSP based on CR-LDP: thiết lập dựa trên các tham số như tốc độ lưu lượng, kích thước vùng đệm.
- Đặt trước tài nguyên (Resource preemption): giành trước tài nguyên của CR-LDP đã tồn tại với setup-priority và holding-priority.
- Flow Aggregation (Gộp luống): gộp cỏc luồng "mịn" (fine flows) vào 1 luồng "thụ" (coarse flow)

11.2.2. Mô hình khái niệm của MNS hỗ trợ QoS

Các hàm chức năng được mô tả như sau:

- LDP/CR-LDP: tạo ra hoặc xử lý các thông báo LDP.CR-LDP
- MPLS Classifier: thực hiện các phép toán xử lý nhãn như: push, pop và swap đối với các gói MPLS
- Service Classifier: phân loại phục vụ để áp dụng với các gói đến bằng cách sử dụng nhãn và thông tin giao tiếp hoặc trường CoS của MPLS shim header, và kết hợp mỗi gói với sự đặt trước thích hợp.
- Admission Control – xem xét Traffic Parameter của CR-LDP, và xác định node MPLS có đủ tài nguyên khả dụng hay không để đáp ứng yêu cầu QoS.



Hình 11.1: Mô hình khái niệm của MNS

- Resource Manager: tạo/xóa các hàng đợi yêu cầu, cũng như quản lý thông tin về tài nguyên.
- Packet Scheduler: quản lý các gói trong hàng đợi sao cho chúng có thể nhận được các yêu cầu phục vụ.

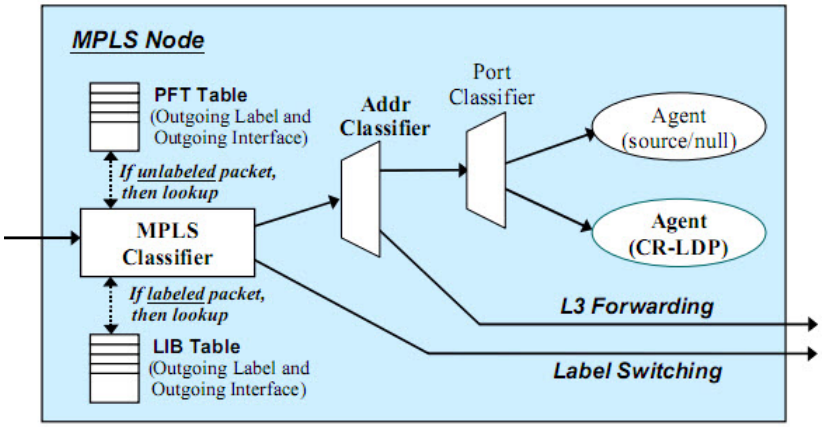
11.3. Thiết kế và thi hành với MNS

11.3.1. Chuyển mạch nhãn

Bộ mô phỏng MPLS thực hiện bằng cách mở rộng NS là bộ mô phỏng dựa trên IP. Trong NS, một node bao gồm các agent (tác nhân) và classifier. Một agent là đối tượng gởi/nhận (sender/receiver) của giao thức, và classifier là đối tượng chịu trách nhiệm đối với sự phân loại gói. Đối với mục đích tạo ra node MPLS mới, MPLS Classifier, và LDP agent được chèn vào trong node IP.

Kiến trúc node MPLS với chuyển mạch nhãn được chỉ ra trong hình 2. Khi node MPLS nhận 1 gói, nó sẽ thực hiện như sau:

- `MPLS Classifier` xác định gói nhận được có gán nhãn hay không. Nếu nó được gán nhãn, `MPLS Classifier` thực hiện chuyển mạch nhãn đối với gói. Nếu nó không được gán nhãn nhưng thuộc một LSP, nó sẽ được điều khiển như một gói được gán nhãn. Ngược lại, `MPLS Classifier` gởi nó đến `Addr Classifier`
- `Addr Classifier`: thực hiện chuyển tiếp L3 đối với gói tin



Hình 11.2: Kiến trúc nút MPLS với chuyển mạch nhãn

- Nếu hop tiếp theo của gói cũng là chính nó, gói sẽ được gởi đến `Port Classifier`

Đối với chuyển mạch nhãn, hai bảng được xác định như sau:

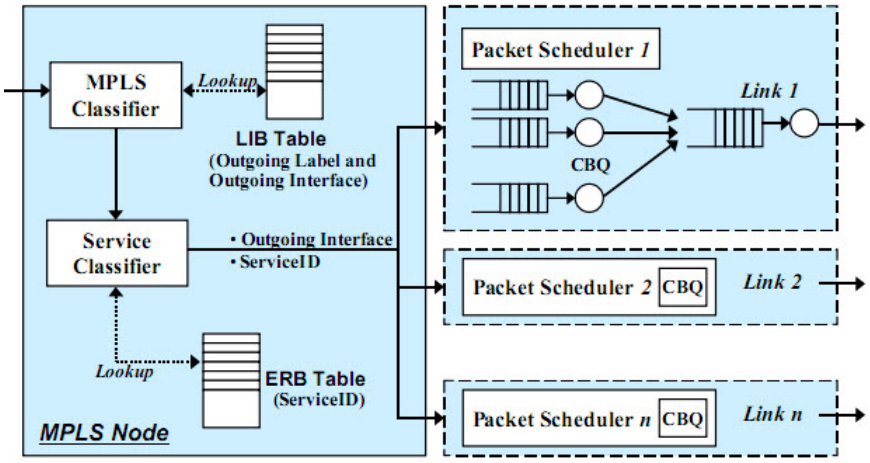
- `Partial Forwarding Table (PFT)` – nó là tập con của Forwarding Table, và được sử dụng để ánh xạ một gói IP thành LSP tại LSR vào (Ingress). Nó bao gồm `FEC`, `FlowID`, và `LIBptr`. `LIBptr` là con trỏ cho biết một điểm vào của bảng LIB.
- `Label Information Table (LIB)` – Chứa các thông tin về việc thiết lập các LSP, và được sử dụng cung cấp chuyển mạch nhãn đối với gói được gán nhãn. Nó bao gồm các nhãn in/out và giao diện in/out.

11.3.2. Chuyển mạch nhãn

Để hỗ trợ lưu lượng MPLS thời gian thực, `Service Classifier` là thành phần được thiết kế và thi hành. CBQ, là công cụ đã được thực thi trong NS, được lựa chọn đối với thành phần `Packet Scheduler`. Trong thực tế, thành phần `Packet Scheduler` sẽ được thực hiện tại các node. Tuy nhiên, NS thực hiện chức năng này tại các Link.

Trong MNS, một bảng được xác định để duy trì các thông tin cho việc thiết lập ER-LSP; đó là `Explicit Route information Base` (ERB). ERB chứa thông tin cho việc thiết lập ER-LSP , như LSPID và `ServiceID`

Hình 11.3 chỉ ra quá trình xử lý lưu lượng MPLS QoS của node và link MPLS. Khi một gói dữ liệu MPLS đi đến tại node MPLS, nó được đưa đến `MPLS Classifier`. Node MPLS sẽ tìm (indexes) trong bảng LIB đối với các nhãn ra (outgoing-label) và giao diện ra (outgoing-interface) để thực hiện các phép toán nhãn và bảng ERB đối với ServiceID để đưa ra hàng đợi dịch vụ. Tùy theo lớp mà gói thuộc vào, nó sẽ được xếp vào vùng đệm

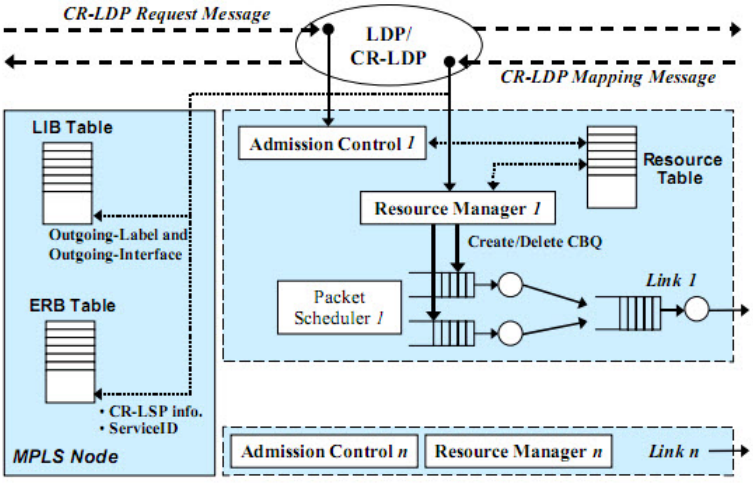


Hình 11.3: Xử lý lưu lượng QoS MPLS của nút và link MPLS

tương ứng trong CBQ. Và rồi nó sẽ được đáp ứng bởi CBQ và được truyền ra giao diện cổng ra.

11.3.3. Sự giành trước tài nguyên

Các thành phần Admission Control và Resource Manager được thiết kế và thi hành đối với việc quản lý tài nguyên. Thành phần Resource Manager có chức năng tạo/xóa bỏ các hàng đợi CBQ, và quản lý các thông tin tài nguyên (tức là bảng Resource).



Hình 11.4: Quá trình giành trước tài nguyên của nút và link MPLS

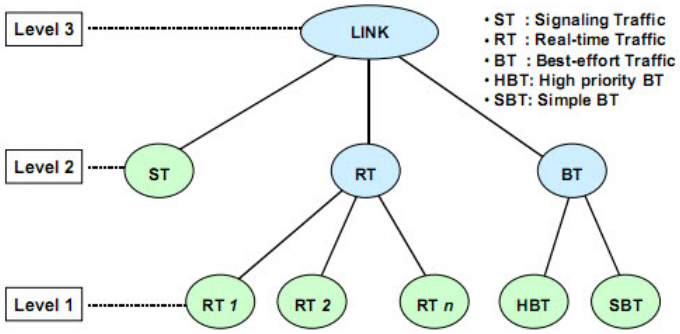
Hình 11.4 chỉ ra quá trình xử lý việc dành trước tài nguyên của node và link MPLS. Khi thành phần CR-LDP nhận bản tin yêu cầu CR-LDP, nó gọi Admission Control để kiểm tra

node có được yêu cầu tài nguyên. Nếu có đủ tài nguyên khả dụng, Admission Control sẽ giành trước tài nguyên bằng cách cập nhật bảng Resource. Sau đó, bản tin yêu cầu LDP được cho phép đi qua để đến node MPLS tiếp theo.

Khi thành phần CR-LDP nhận bản tin CR-LDP Mapping, nó sẽ lưu giữ nhãn và thông tin giao tiếp trong bảng LIB và các thông tin CR-LSP được yêu cầu (tức là LSPID) trong bảng ERB. Và cuối cùng, bản tin LSP Mapping sẽ được cho phép đi qua để đến node MPLS trước đó.

11.3.4. Mức lớp - Class Level

Có 3 mức lớp và 4 dịch vụ (ST, RT, HBT, SBT), được chỉ ra trong hình 11.5. Người sử dụng có thể cấu hình các tham số của CBQ, tức là tốc độ lưu lượng, kích thước vùng đệm, quyền ưu tiên. Các hàng đợi của ST, HBT và SBT được tạo ra theo cách cố định khi môi trường mô phỏng được cấu hình. RT thì được tạo ra/xóa bỏ theo cách động khi khi có một sự kiện mô phỏng, ví dụ bản tin CR-LDP đến tại node MPLS.



Hình 11.5: Mức lớp trong MPLS

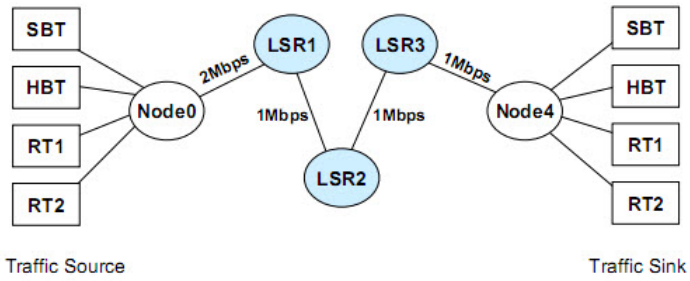
11.3.5. Môi trường thực thi

Bộ mô phỏng MNS được thực thi trên hệ thống Sun Unix bởi chương trình ns-2.1b6, NS version 2.1

11.4. Các ví dụ mô phỏng

11.4.1. Mô phỏng lưu lượng QoS

Trong phần này, chúng ta khảo sát một vài loại của lưu lượng MPLS với mỗi loại QoS khác nhau, được chỉ ra trong hình 11.6. Theo cách này, độ chính xác và tính hiệu quả của MNS sẽ được kiểm định.



Hình 11.6: Mạng MPLS

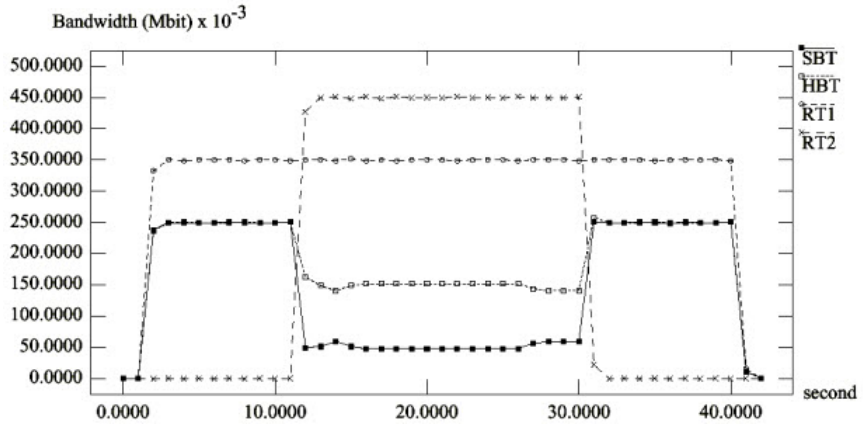
Trong hình 11.6 , node0 và node4 là các node IP. LSR1, LSR2 và LSR3 là các node MPLS. Có 4 cặp tải làm việc: một cặp là Simple Best-effort Traffic, được gọi là SBT, một cặp là High priority Best-effort Traffic, được gọi là HBT, và hai cặp Real-time Traffic, tương ứng là RT1 và RT2. Lưu lượng được đưa vào tại node0 và đi ra từ node4. Băng thông kết nối (link) là 1Mbit/s, ngoại trừ kết nối giữa Node0 và LSR1. SBT và HBT phát ra lưu lượng tốc độ bit không thay đổi (CBR) tại 250 Kbit/s, theo thứ tự lần lượt. RT1 và RT2 phát ra lưu lượng CBR tại 350 Kbit/s và 450 Kbit/s, một cách tương ứng. Do đó, tổng tốc độ dữ liệu tới của tất cả các tải làm việc lớn hơn băng thông kết nối của mạng MPLS.

Hình 11.7 chỉ ra mã lập lịch các sự kiện đối với mô phỏng. Có 4 LSP được thiết lập đối với lưu lượng; đó là hai ER-LSP đối với SBT và HBT, và hai CR-LSP đối với RT1 và RT2. Tại thời điểm 0.1 giây, 2 ER-LSP được thiết lập đối với SBT và HBT, và 1 CR-LSP đối với RT1. Tại 1.0, SBT, HBT và RT1 phát ra các lưu lượng của chúng. Tại 10.0 giây, một CR-LSP được thiết lập đối với RT2, và RT2 phát ra lưu lượng của nó tại 11.0 giây. Tại 30.0 giây, RT2 dừng việc phát ra lưu lượng của nó và CR-LSP đối với RT2 được giải phóng tại 31.0 giây. Cuối cùng, SBT, HBT và RT1 dừng việc phát lưu lượng của chúng tại 40.0 giây.

Kết quả mô phỏng được chỉ ra trong hình **??**. Như trong hình **??**, tổng băng thông giữa khoảng 11 giây và 30 giây gần bằng với băng thông kết nối, tức là hiệu quả sử dụng băng thông là rất cao. Trong khoảng thời gian đó, RT1 và RT2 có thể đạt được băng thông đã yêu cầu, trong khi SBT và HBT tận dụng băng thông còn lại của kết nối. Và HBT được phục vụ nhiều hơn SBT.

```
#                    setup-er-lsp {FEC   ER(Explicit-Route)    LSPID}
$ns at 0.1 "$LSR1 setup-er-lsp    3          1_2_3              1000"
$ns at 0.1 "$LSR1 setup-er-lsp    3          1_2_3              1100"
#                    setup-cr-lsp {FEC   ER   LSPID  Bandwidth  BufferSize PacketSize
#                                                    SetupPrio HoldingPrio}
$ns at 0.1  "$LSR1 setup-cr-lsp   3   1_2_3  1200    350K       400        200   7  3"
$ns at 1.0  "$SBT start"
$ns at 1.0  "$HBT start"
$ns at 1.0  "$RT1 start"
$ns at 10.0  "$LSR1 setup-cr-lsp  3  1_2_3  1300  450K  400 200 7 3"
$ns at 11.0  "$RT2 start"
$ns at 30.0  "$RT2 stop"
$ns at 31.0  "$LSR1 release-lsp-using-release  1300"
$ns at 40.0  "$SBT stop"
$ns at 40.0  "$HBT stop"
$ns at 40.0  "$RT1 stop"
```

Hình 11.7: Mã lập lịch sự kiên



Hình 11.8: Sự biến đổi lưu lượng băng thông
fig:h118

11.4.2. Mô phỏng sự ưu tiên trước tài nguyên

Trong phần này, chúng ta mô phỏng sự ưu tiên trước tài nguyên được xác định trong chuẩn CR-LDP. Đối với mô phỏng này, tốc độ lưu lượng của RT1 và RT2 được ghi lại. RT1 và RT2 phát ra lưu lượng CBR lần lượt tại 600 Kbit/s và 700 Kbit/s. Do đó, khi RT1 và RT2 phát ra lưu lượng tại thời điểm giống nhau thì tổng tốc độ của tất cả các khối lượng làm việc sẽ lớn hơn băng thông kết nối của mạng MPLS.

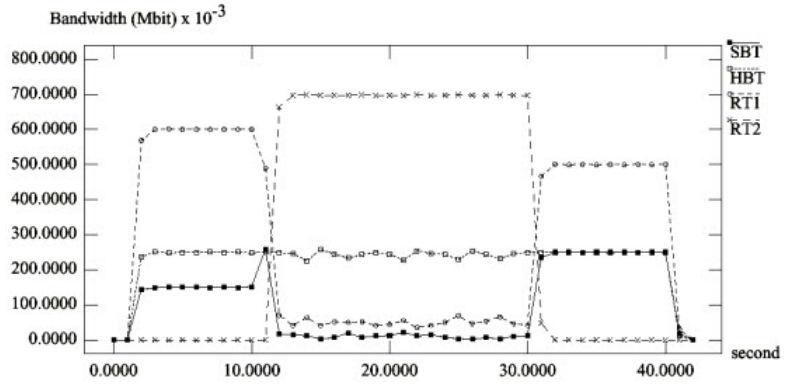
Hình 11.9 chỉ ra mã lập lịch các sự kiện đối với mô phỏng. Việc thiết lập độ ưu tiên (setup-priority) và giữ độ ưu tiên (holding-priority) đối với CR-LDP của RT1 lần lượt là 7 và 4. Việc thiết lập độ ưu tiên (setup-priority) và giữ độ ưu tiên (holding-priority) đối với CR-LDP của RT2 lần lượt là 3 và 2. Vì vậy, RT2 có thể giành được tài nguyên trước RT1

```
Sns at 0.1 "SLSR1 setup-er-lsp 3 1_2_3 1000"
Sns at 0.1 "SLSR1 setup-er-lsp 3 1_2_3 1100"

# setup-priority= 7, holding-priority= 4 for RT1 Traffic
Sns at 0.1 "SLSR1 setup-cr-lsp 3 1_2_3 1200 600K 400 200 7 4"

Sns at 1.0 "SSBT start"
Sns at 1.0 "SHBT start"
Sns at 1.0 "SRT1 start"

# setup-priority= 3, holding-priority= 2 for RT2 Traffic
Sns at 10.0 "SLSR1 setup-cr-lsp 3 1_2_3 1300 700K 2000 200 3 2"
Sns at 11.0 "SRT2 start"

Sns at 30.0 "SRT2 stop"
Sns at 31.0 "SLSR1 release-lsp-using-release 1300"

Sns at 40.0 "SSBT stop"
Sns at 40.0 "SHBT stop"
Sns at 40.0 "SRT1 stop"
```

Hình 11.9: Mã lập lịch sự kiện

bởi vì giá trị setup-priority của RT2 nhỏ hơn giá trị holding-priority của RT1.

Kết quả mô phỏng được chỉ ra trong hình 11.10, trong đó chỉ ra tài nguyên của RT1 được giành trước RT2 tại giây thứ 11. Từ giây thứ 11, RT1 được đáp ứng với lưu lượng cao nhất. Trong khoảng thời gian này, chỉ RT1 có thể đạt được băng thông theo yêu cầu, trong khi trong khi SBT, HBT và RT2 tận dụng băng thông còn lại của kết nối. Và HBT được phục vụ nhiều hơn SBT và RT2.



Hình 11.10: Kết quả của mã lập lịch sự kiện

11.5. Kết luận

MPLS được biết đến như là công nghệ quan trọng nhất đối với việc giải quyết nhiều bài toán trên Internet. Rất nhiều kỹ thuật liên quan đến MPLS đã được cung cấp. MNS cung

cấp một nền tảng mô phỏng với nhiều kỹ thuật đã được xem xét.

Bằng cách sử dụng MNS, chúng ta đã mô phỏng 2 ví dụ trong tài liệu này. Với các sự kiện đơn giản, các kết quả được chứng minh là có tính khả thi và hiệu quả đối với MNS. Cần nhiều kết quả mô phỏng hơn để chứng tỏ MNS được thực hiện trong tương lai.

Chương 12

Mạng di động

Mục lục

Người dịch: *Nguyễn Hoàng Minh Phúc*
Biên tập: *Nguyễn Anh Tôn*
Bản gốc: *NS Simulator for beginners, Chapter 9* [2]

Có hai cách để liên lạc không dây giữa hai thiết bị: thứ nhất là qua mạng tế bào tập trung trong đó mỗi một thiết bị di động được kết nối tới một hoặc nhiều trạm cơ sở đặt cố định (mỗi trạm cơ sở chịu trách nhiệm một ô), do đó liên lạc giữa hai thiết bị di động đòi hỏi có sự tham gia của một hoặc nhiều trạm cơ sở. Kiểu thứ hai là kiểu không tập trung bao gồm cơ sở của một mạng ad-hoc giữa những người dùng mong muốn liên lạc với nhau. Do tăng thêm giới hạn phạm vi hoạt động của một thiết bị di động (đối với một trạm cơ sở đặt cố định), kiểu liên lạc này đòi hỏi các thiết bị đi động không chỉ phải đóng vai trò là nguồn hoặc đích của các gói tin được truyền trong qua trình liên lạc của chúng mà còn đảm nhiệm thêm vai trò đẩy các gói tin cho quá trình liên lạc giữa các thiết bị khác nữa. Trạm tế bào có phạm vi lớn hơn các mạng ad-hoc. Tuy nhiên các mạng ad-hoc lại có ưu điểm là có thể triển khai nhanh chóng bởi vì chúng không đòi hỏi hạ tầng cơ sở sẵn có.

Trong các mạng tế bào, các thành phần không dây bị hạn chế chỉ truy cập vào một mạng và trong mạng đó các giao thức định tuyến cổ điển có thể được sử dụng. Trái lại, mạng ad-hoc dựa vào các giao thức định tuyến đặc biệt để có thể đáp ứng được sự thay đổi thường xuyên của cấu trúc liên kết mạng.

Để mô hình hóa tốt các mạng tế bào, các công cụ mô phỏng phức tạp như kênh vô tuyến vật lý cũng như các công cụ để mô phỏng các kỹ thuật điều khiển công suất cần được sử dụng. NS không phải là một module cao cấp để mô phỏng lớp vật lý (mặc dù nó có chứa một số các chức năng mô phỏng kênh vô tuyến đơn giản).

Trong các mạng ad-hoc, trái lại, các giao thức định tuyến thường tập trung. NS cho phép mô phỏng các giao thức định tuyến chính cũng như sự truyền tải và các ứng dụng sử dụng các giao thức đó. Ngoài ra, nó còn cho phép tính toán đối với lớp MAC, tính di động và một vài đặc điểm của lớp vật lý.

Các giao thức định tuyến thông dụng có trong NS bao gồm:

- DSDV- Destination Sequenced Distance Vector.
- DSR- Dynamic Source Routing (Định tuyến nguồn động).
- TORA/IMPE-Temporally Ordered Routing Algorithm / Internet MANET Encapsulation Protocol.
- AODV- Ad-hoc On Demand Distance Vector.

12.1. Các thuật toán định tuyến

Có một vài thuật toán định tuyến dùng trong các mạng có dây truyền thống và một vài ý tưởng nảy sinh từ các thuật toán đó đã được sử dụng trong các mạng ad-hoc. Chúng ta sẽ chú trọng đến các vấn đề sau:

1. **Định tuyến theo trạng thái kết nối.** Mỗi nút mạng đều lưu trữ toàn bộ cấu trúc kết nối của mạng cùng với tham số của mỗi đường kết nối. Mỗi nút mạng gửi quảng bá theo định kỳ giá trị tham số của các đường kết nối bắt đầu từ chúng tới tất cả các nút mạng khác. Mỗi nút cũng sẽ tự cập nhật cấu trúc kết nối mới của mạng và áp dụng một thuật toán tìm đường ngắn nhất để chọn ra điểm tới tiếp theo cho ứng với từng đích đến.

2. **Định tuyến theo Vector khoảng cách.** Mỗi nút mạng chỉ quan tâm đến giá trị tham số của các kết nối đi ra từ nó. Thay vì quảng bá thông tin tới tất cả các nút, nó quảng bá theo định kỳ tới các nút lân cận của nó một giá trị ước lượng khoảng cách ngắn nhất tới mọi nút trong mạng. Các nút lân cận sử dụng thông tin này để tính toán lại bảng định tuyến của mình bằng thuật toán tìm đường ngắn nhất. Phương pháp này có hiệu quả tính toán cao hơn, dễ dàng triển khai hơn và cũng đòi hỏi dung lượng lưu trữ ít hơn so với định tuyến trạng thái kết nối (Link Sate).

3. **Định tuyến nguồn.** Các quyết định định tuyến được thực hiện tại nguồn và các gói hoàn toàn được chuyển theo các tuyến đó.

4. **Định tuyến tràn**. Một nguồn sẽ gửi thông tin tới tất cả các nút lân cận của nó, các nút này sẽ chuyển tiếp các thông tin đó tới các nút lân cận của chúng và việc chuyển tin cứ tiếp tục như vậy. Và bằng cách đánh số thứ tự cho các gói tin, một nút mạng có thể sắp xếp gói tin vào đúng vị trí của nó.

Tiếp theo chúng ta sẽ nói về các giao thức định tuyến đặc biệt có trong NS.

12.1.1. Vector khoảng cách theo thứ tự đích– DSDV

DSDV là một giao thức định truyến vector khoảng cách. Mỗi một nút đều có một bảng định tuyến để chỉ ra hướng đi ứng với mỗi địa chỉ đích, tham số của bảng bao gồm địa chỉ của điểm đến tiếp theo và số các điểm trung gian cần đi qua trước khi đến đích. Mỗi nút theo định kỳ sẽ gửi quảng bá bản tin cập nhật bảng định tuyến của nó. Một chuỗi số theo thứ tự được sử dụng để đánh dấu cho từng đường định tuyến. Các con số này thể hiện đường định tuyến nào là mới hơn: đường định tuyến có số thứ tự cao hơn sẽ tốt hơn, còn nếu có hai đường định tuyến có cùng số thứ tự thì đường nào có số lượng điểm trung gian

ít hơn sẽ tốt hơn. Nếu một nút phát hiện ra một đường định tuyến bị hỏng nó sẽ đặt giá trị vô cùng cho số lượng điểm trung gian đồng thời số thứ tự sẽ được cập nhật (tăng lên) nhưng sẽ là một số lẻ, điều này có nghĩa là số thứ tự chẵn là tượng trưng cho việc đường định tuyến đó vẫn còn sử dụng được.

12.1.2. Vector khoảng cách theo yêu cầu đặc biệt – AODV

AODV là một kiểu định tuyến vector khoảng cách. Nó không đòi hỏi các node phải duy trì các đường định tuyến tới các đích bởi vì không phải lúc nào chúng cũng được sử dụng. Chỉ cần các điểm đầu nút của một kết nối có một đường định tuyến phù hợp để đi được tới nhau là được. Giao thức này sử dụng các bản tin khác nhau để tìm ra và duy trì các kết nối, chúng gồm có: bản tin yêu cầu định tuyến (RREQs), bản tin phản hồi yêu cầu định tuyến (RREPs) và bản tin báo lỗi định tuyến (RERRs). Các loại bản tin này được nhận thông qua giao thức UDP và normal IP header processing applies.

AODV gán số thứ tự cho từng đường định tuyến. Số thứ tự của đích đếnđược tạo ra bởi các đích đến như là một tham số trong thông tin định tuyến mà nó gửi đi tới các nút đang yêu cầu định tuyến tới nó. Sử dụng các số thứ tự này đảm bảo sẽ không xuất hiện hiện tượng lặp (loop) và cho phép chúng ta biết đường định tuyến nào là tốt hơn. Để chọn lựa giữa hai đường định tuyến tới cùng một đích, nút gửi yêu cầu định tuyến luôn luôn chọn đường số thự tự tương ứng cao nhất. Các nút mà là một phần của một đường định tuyến vẫn dùng được có thể đưa ra thông tin kết nối bằng cách quảng bá định kỳ ra hàng xóm xung quanh nó bản tin Hello (là một bản tin RREP đặc biệt). Nếu bản tin Hello không còn nhận được từ một hàng xóm nào đó sau một khoảng thời gian định trước thì có nghĩa là có thể đã mất kết nối tới vị hàng xóm đó.

Khi một nút phát hiện ra đường đi tới một nút hàng xóm không còn dùng được nữa nó sẽ xóa thông tin đường này và gửi đi một bản tin RERR tới các hàng xóm còn lại, việc này giúp duy trì danh sách các hàng xóm vẫn còn liên lạc được. Thủ tục này sẽ được lặp lại tại các nút nhận được bản tin RERR. Một nút mà nhận được một bản tin RERR có thể lại gửi đi một bản tin RREQ.

AODV không cho phép điều khiển các kết nối theo một hướng duy nhất.

12.1.3. Định tuyến nguồn động – DSR

Được thiết kế cho mạng di động đặc biệt có số node có thể lên tới 200 với mức độ lưu động cao. Giao thức này làm việc theo kiểu "theo yêu cầu" nghĩa là không có hành động cập nhật theo định kỳ.

Các gói tin được chuyển hoàn toàn theo các đường đã chọn. Nhờ vậy giảm được lượng

đáng kể các bản tin cập nhật trong mạng. Các nút lưu trữ trong bộ nhớ tạm thời của chúng tất cả các đường định tuyến mà chúng biết. Giao thức này bao gồm việc tìm ra các đường định tuyến và duy trì chúng.

Tại quá trình tìm kiếm đường định tuyến, một nguồn có yêu cầu gửi tin tới một đích sẽ gửi quảng bá bản tin yêu cầu định tuyến RREQ. Các nút nhận được bản tin RREQ này sẽ tìm kiếm trong bộ nhớ của nó xem có đường nào đi đến địa chỉ đích được ghi trong RREQ không. Nếu nó không tìm thấy thì bản tin RREQ sẽ được gửi tiếp đi xa hơn đồng thời nó sẽ thêm địa chỉ của nó vào danh sách chuỗi điểm trung gian. Thủ tục này sẽ tiếp tục cho tới khi tới được đích cần đến hoặc đến được một node nào đó có đường định tuyến tới đích cần đến. Đường định tuyến sẽ được xác định bằng cách đảo ngược danh sách chuỗi điểm trung gian. Vì các đường định tuyến không cần phải đối xứng , nên khi DSR kiểm tra bộ nhớ lưu trữ bảng định tuyến tạm thời của node cuối cùng trong thủ tục tìm kiếm và nếu một đường được tìm thấy nó sẽ được sử dụng để thay thế. Do đó kết nối một chiều có thể được điều khiển.

Duy trì đường định tuyến: Khi tạo ra hoặc đẩy đi một gói tin sử dụng một đường định tuyến, mỗi nút mà gói tin đi qua có trách nhiệm xác thực rằng dữ liệu có thể được chuyển qua theo đường kết nối từ nó đến điểm trung gian tiếp theo. Một gói tin xác nhận có thể cung cấp sự xác thực cho một kết nối rằng có thể chuyển dữ liệu trên nó. Các gói tin xác nhận thường là một bộ phận của giao thức MAC (như là khung xác nhận lớp liên kết được định nghĩa trong IEEE 802.11), hoặc là "xác nhận thụ động" nghĩa là một nút có thể xác định được rằng gói tin có nó đã được nhận bởi một nút trung gian vì nó có thể biết được nút trung gian đó đã đẩy gói tin đó đi xa hơn. Nếu những bản tin xác nhận này chưa được gửi đi thì một nút có thể yêu cầu một bản tin xác nhận (bản tin xác nhận này có thể được gửi trực tiếp tới nút đó qua một đường định tuyến khác). Bản tin xác nhận có thể bị yêu cầu nhiều lần (trong một giới hạn đã đặt trước), và nếu liên tục không nhận được bản xác nhận, đường định tuyến đó sẽ bị loại bỏ khỏi bộ nhớ định tuyến tạm thời và một bản tin "Định tuyến lỗi" sẽ được gửi tới các nút mà đã từng gửi tin qua kết nối đó kể từ thời điểm cuối cùng nhận được bản tin xác nhận. Các nút nghe ngóng hoặc chuyển tiếp các gói tin sẽ sử dụng tất cả các thông tin định tuyến đã tích trữ để cập nhật cho gói tin định tuyến của nó.

12.1.4. Thuật toán định tuyến đặt chỗ tạm thời – TORA

Giao thức này thuộc dòng họ các giao thức đảo chiều kết nối. Nó có thể cung cấp nhiều đường định tuyến giữa 2 điểm. TORA bao gồm ba phần: tạo ra, duy trì và xóa đường. Tại mỗi nút mạng, một bản sao chép riêng biệt của TORA được chạy với từng đích. TORA xây dựng một đồ thì không tuần hoàn trực tiếp hướng tới một đích. Nó gán một tham số

cho từng nút trong mạng (đối với các đích thông thường). Các tin nhắn sẽ đi từ các nút có tham số lớn hơn tới những nút có tham số thấp hơn. Các đường định truyến được phát hiện ra bằng cách sử dụng các gói tin Truy vấn (QRY) và Cập nhật (UPD).

Khi một nút mạng không có đường kết nối xuống cần có đường định tuyến tới một đích, nó gửi quảng bá bản tin QRY và sẽ truyền đi cho tới khi nó tìm thấy được một nút có đường định tuyến tới đích cần đến hoặc truyền tới được đích cần đến. Đích này sẽ đáp lại bằng cách gửi quảng bá bản tin UPD có chứa giá trị tham số của nút. Một nút mạng khi nhận được gói tin UPD sẽ cập nhật giá trị của tham số tương ứng của nó và gửi quảng bá ra một gói tin UPD khác. Việc này có thể tạo ra một số lượng các đường trực tiếp từ nguồn tới đích.

Nếu một nút mạng phát hiện ra một đích nào đó mà nó không thể nào tìm được đường tới, nó sẽ đặt cho điểm đích này giá trị tham số lớn nhất. Trong trường hợp nút mạng này không thể tìm thấy được bất kỳ một hàng xóm nào có giá trị tham số là hữu hạn tới node đích đó, nó sẽ cố gắng tìm một đường định tuyến mới. Nếu cũng không có đường định tuyến nào tới được đích này (ví dụ như đích đến thuộc một phần bị chia cắt của mạng), nút mạng này sẽ gửi quảng bá gói tin Xóa (CLR) để thiết lập lại tất cả các trạng thái định tuyến và xóa bỏ đi các định tuyến không còn đúng từ các thành phần khác của mạng.

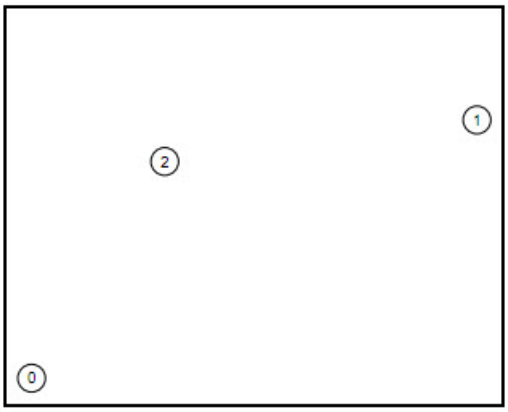
TORA hoạt động phía bên trên giao thức IMEP (Giao thức đóng gói MANET internet), đây là giao thức cung cấp khả năng phân phối ổn định các bản tin định tuyến và cung cấp các giao thức định tuyến liên quan đến sự thay đổi các kết nối cho các hàng xóm của nó. IMEP cố gắng tập hợp các bản tin IMEP và TORA lại thành một gói tin (gọi là khối) để có thể giảm lưu lượng. Để có được thông tin về trạng thái của các kết nối tới hàng xóm, IMEP theo sẽ gửi định kỳ các bản tin BEACON được trả lời lại bởi các bản tin HELLO

12.2. Mô phỏng mạng di động

12.2.1. Kịch bản mô phỏng

Chúng ta bắt đầu bằng việc đưa ra một kịch bản đơn giản: một kết nối TCP đơn qua 3 node mạng với phạm vi 500 x400m như được miêu tả trong hình 12.1.

- Vị trí khởi tạo của node 0, 1 và 2 lần lượt là (5,5), (490,285) và (150,240) (chúng ta giả sử rằng tọa độ trục z bằng 0).
- Vào điểm thời gian 10, node 0 bắt đầu di chuyển tới điểm (250,250) với vận tốc 3m/s
  Vào điểm thời gian 15, node 1 bắt đầu di chuyển tới điểm (45,285) với vận tốc 5m/s
  Vào điểm thời gian 10, node 0 bắt đầu di chuyển tới điểm (480,300) với vận tốc 5m/s
  Node 2 không tham gia mô phỏng.



Hình 12.1: Ví dụ về một mạng ad-hoc 3 điểm

Mô phỏng này kéo dài 150s. Tại điểm thời gian 10, một kết nối TCP được khởi tạo giữa node 0 và node 1.

Chúng ta sẽ sử dụng ở dưới đây giao thức định tuyến đặc biệt DSDV và giao thức IEEE802.11 MAC.

12.2.2. Viết một tcl script

Chúng ta bắt đầu bằng việc xác định rõ một vài tham số cơ bản cho quá trình mô phỏng, cung cấp thông tin cho các layer khác nhau. Nó được làm như dưới đây:

```
Set val(chan)    Channel/WirelessChannel;    # channel type
Set val(prop)    Propagation/TwoRayGround;   # radio-propagation model
Set val(netif)   phy/WirelessPhy;            # network interface type
Set val(mac)     Mac/802.11;                 # MAC type
Set val(ifq)     Queue/DropTail/PriQueue ;   # interface queue type
Set val(ll)      LL;                         # link layer type
Set val(ant)     Antenna/OmniAtenna;         # antenna model
Set val(ifqlen)  50;                         # max packet in ifq
Set val(nn)      3;                          # number of mobilenodes
Set val(rp)      DSDV;                       # routing protocol
Set val(x)       500;                        # X dimention of topography
Set val(y)       400;                        # Y dimention of topography
Set val(stop)    150;                        # time of simulation end
```

Những thông số này được sử dụng trong quá trình cấu hình các node với các lệnh như sau

```
$ns node-config -adhocRouting $val(rp) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType &val(ifq) \
                -ifqLen $val(ifqLen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -channelType $val(chan) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTtrace ON \
                -macTrace OFF \
                -movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
}
```

Bốn tùy chọn cuối cùng trong phần cấu hình nút mạng có thể nhận giá trị ON hoặc OFF. Tùy chọn agentTrace sẽ lần theo đường của TCP, routerTrace sẽ cung cấp đường đi của các gói tin trong quá trình định tuyến, macTrace thì lần theo các gói tin giao thức MAC, và movementTrace được sử dụng để lần theo sự di chuyển của các node

Vị trí khởi tạo của nút 0 là:

```
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0
```

Sự di chuyển tuyến tính của một nút mạng được tạo ra bằng cách xác định rõ thời gian nó bắt đầu chuyển động, giá trị tọa độ x và y của đích đến và tốc độ của nó. Ví dụ, sự di chuyển của nút số 1 được xác định như sau:

```
$ns at 15.0 ‘‘$node_(1) setdest 45.0 285.0 5.0’’
```

Chúng ta cần tạo ra vị trí ban đầu của nút mạng cho NAM (Network AniMator) bằng cách sử dụng :

Tương tự như vậy chúng ta khởi tạo ví trí ban đầu cho các nút còn lại.

```
For {set i 0} {$i < $val(nn)} { incr i } {
```

```
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}
```

Chúng ta chỉ ra các nút mạng khi quá trình mô phỏng kết thúc với:

```
For {set i 0} {$i < $val(nn)} { incr i} {
        $ns at $val(stop) ‘‘$node_($i) reset’’;
}
```

Sau đó chúng ta tạo kết nối TCP và tạo ứng dụng ftp qua kết nối này như bình thường. Kết thúc quá trình mô phỏng như thường lệ ngoại trừ việc thêm lệnh để kết thúc NAM.

12.3. Định dạng file vết

Một ví dụ về một đường trong dấu vết đầu ra là

```
r 40.649943289 _1_  AGT --- 1569 tcp 1032 [a2 1 2 800]  ------- [0:0 1:0 32 1] [35 0] 2 0
```

- Tham số đầu tiên là một chữ cái giá trị của nó có thể là r, s, f , D thay cho “received”, “sent”, “forward” và “dropped”. Nó cũng có thể là M để ám chỉ một địa điểm hoặc một sự di chuyển, nó sẽ được miêu tả sau.
- Tham số thứ hai là thời gian
- Thứ ba là số thứ tự của nút mạng
- Thứ tư là MAC để biểu thị rằng gói tin có liên quan đến lớp MAC, AGT thể hiện đó là một gói tin lớp Transport (chẳng hạn như TCP), hoặc RTR nếu đó là một gói tin được đính tuyến. Giá trị này cũng có thể là IFQ để chỉ ra các sự kiện liên quan đến hàng đợi ưu tiên (như gói tin bị rớt chẳng hạn).
- Ở phía sau các dấu gạch ngang là số thứ tự của gói tin (là global sequence number chứ không phải là tcp sequence number).
- Tại trường tiếp theo là các thông tin bổ sung về gói tin (chẳng hạn như tcp, ack hoặc udp)
- Tiếp đến là kích cỡ của gói tin (tính theo bytes)
- Trong ngoặc vuông là các thông tin về lớp MAC. Số thập lục phân đầu tiên a2 (tương đương với 162 trong hệ thập phân) miêu tả số thời gian tính theo giây được mong đợi

là sẽ gửi xong gói tin qua kênh vô tuyến. Số thứ hai có giá trị 1 là để thay cho MAC-id của node đang gửi gói tin, số thứ ba có giá trị 2 – đó chính là MAC-id của nút mạng đang nhận gói tin. Trường cuối cùng, 800, để chỉ ra loại MAC là `ETHERTYPE_IP`.

- Trong ngoặc vuông thứ 2 là địa chỉa IP nguồn và đích, tiếp theo là ttl (Time To Live) của gói tin (trong trường hợp này là 32)
- Ngoặc vuông thứ ba đề cập đến thông tin tcp: số thứ tự của nó và số chứng thực (acknowledgement number)

Có nhiều kiểu định dạng khác nhau liên quan đến các kỹ thuật định tuyến khác và các loại gói tin khác nhau.

Một câu lệnh di chuyển có dạng như sau:

```
M 10.00000 0 (5.00, 5.00, 0.00), (250.00, 250.00), 3.00
```

Con số đầu tiên là thời gian, thứ hai là số node, sau đó là vị trí nguồn và đích, cuối cùng là tốc độ.

```
# Một ví dụ gồm 3 nút cho mô phỏng ad-hoc với DSDV
# Định nghĩa các tùy chọn
Set val(chan)   Channel/WirelessChannel;   # loại kênh
Set val(prop)   Propagation/TwoRayGround;  # radio-propagation model
Set val(netif)  Phy/WirelessPhy;           # network interface type
Set val(mac)    Mac/802.11;                # loại MAC
Set val(ifq)    Queue/DropTail/PriQueue;   # interface queue type
Set val(ll)     LL;                        # link layer type
Set val(ant)    Antenna/OmniAtenna;        # antenna model
Set val(ifqlen) 50;                        # max packet in ifq
Set val(nn)     3;                         # number of mobilenodes
Set val(rp)     DSDV;                      # routing protocol
Set val(x)      500;                       # X dimention of topography
Set val(y)      400;                       # Y dimention of topography
Set val(stop)   150;                      # time of simulation end
Set ns                  [new Simulator]
Set tracefd             [open simple.tr w]
Set windowVsTime2       [open win.tr w]
Set namtrace            [open simwrls.nam w]
$ns trace-all $tracefd
@ns namtrace-all-wireless $namtrace $val(x) $val(y)
```

```
# set up topography object
Set topo                [new Topography]

$topo load_flatgrid $val(x) $val(y)

Creat-god $val(nn)

# Creat nn mobilenodes  [$val(nn)] and attach them to the channel
# configure the nodes
$ns node-config -adhocRouting $val(rp) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType &val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -channelType $val(chan) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTtrace ON \
                -macTrace OFF \
                -movementTrace ON
for {set i 0} {$i < $val(nn) } { incr i } {
             set node_($i) [$ns node]
}

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0
```

```
# Generation of movements
$ns at 10.0 ‘‘$node_(0) setdest 250.0 250.0 3.0’’
$ns at 15.0 ‘‘$node_(1) setdest 45.0 285.0 5.0’’
$ns at 110.0 ‘‘$node_(0) setdest 480.0 300.0 5.0’’

# Set a TCP connection between node_(0) and node_(1)
Set tcp [new Agent/TCP/Newreno]
$tcp set class_2
Set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
Set ftp [nw Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 ‘‘$ftp start’’

# Printing the window size
Proc plotWindow {tcpSource file} {
Global ns
Set time 0.01
Set now [$ns now]
Set cwnd [$tcpSource set cwnd_]
Puts $file ‘‘$now $cwnd’’
$ns at [expr $now+$time] ‘‘plotWindow $tcpSource $file’’ }
$ns at 10.1 ‘‘plotWindow $tcp $windowVsTime2’’

# Define node initial position in nam
For {set i 0} {$i < $val(nn)} { inct i } {
# 30 define the node size for nam
$ns initial_node_pos $node_($i) 30
}
# Telling nodes when the simulation ends
For {set i 0} {$i < $val(nn)} { inct i } {
$ns at $val(stop) ‘‘$node_($i) reset’’ ;
}
# ending nam and the simulation
$ns at $val(stop) ‘‘$ns nam-end-wireless $val(stop)’’
$ns at $val(stop) ‘‘stop’’
```

```
$ns at 150.01 ‘‘puts \’’end simulation\’’ ; $ns halt’’
Proc stop {} {
Globla ns tracefd namtrace
$ns flush-trace
Close $tracefd
Close $namtrace
}

$ns run
```

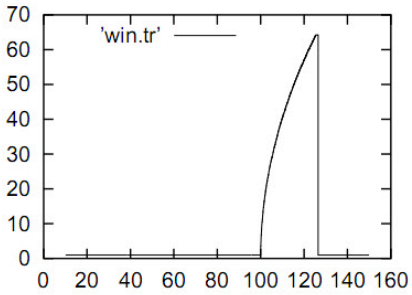
Bảng 12.1: Kịch bản "wrls-dsdv.tcl" cho TCP ttrên một mạng ad-hoc

12.4. Phân tích kết quả mô phỏng

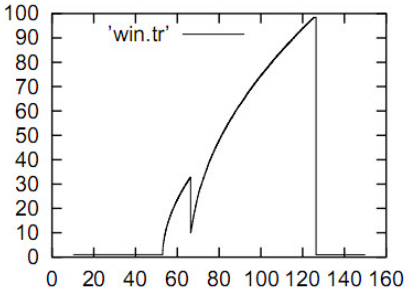
Tại lúc bắt đầu các node ở cách nhau quá xa và không thể tạo được kết nối đến nhau. Gói tin TCP đầu tiên được truyền tại điểm thời gian 10 nhưng kết nối không thể mở được. Trong lúc ấy các node 0 và node 1 bắt đầu di chuyển về phía node 2. Sau 6 giây (timeout) cố gắng tạo kết nối lần thứ hai nhưng vẫn không được và giá trị timeout được nhân đôi thành 12 giây. Tại điểm thời gian 18 một cố gắng kết nối được thực hiện. Giá trị timeout lại được nhân đôi thành 24 giây và tiếp tục như vậy lại nhân đôi thành 48 giây. Chỉ đến thời điểm 100 giây kết nối mới được thiết lập. Node 1 và node 0 là đủ gần nhau nên đã tạo được một kết nối trực tiếp. Các điểm di động tiếp tục di chuyển và lại xa nhau thêm cho đến khi kết nối trực tiếp bị đứt. Giao thức định tuyến là quá chậm để tạo ra một đường định tuyến khác. Tiến trình có thể xem ở trong hình 12.2 và một cái chụp nhanh của nam(Network AniMator) tại thời điểm 124.15 giây được ghi tại hình 12.4

Tiếp theo chúng ta thay đổi một chút các tham số mô phỏng. Việc thay đổi duy nhất thực ở đây là chúng ta thay đổi thời điểm truyền ftp từ 10 thành 12. Việc này sẽ làm cho nút số 0 1 sẽ nằm trong vòng phủ sóng của nút số 2 khi thời gian vào khoảng 53 giây, tại lúc này kết nối tcp cố gắng kết nối sẽ thành công giữa nút 0 và 1. Điều này được miêu tả trong hình 12.5 . Tại điểm thời gian 66 node 0 và node 1 là đủ gần nhau để tạo được một kết nối trực tiếp. Tiến trình được này có thể thấy ở hình 12.3 . Tại thời điểm đường định tuyến thay đổi có một gói tin TCP bị mất nên đã làm cho kích thước cửa sổ giảm xuống.

Tại điểm thời gian 125.5 nút 0 và 1 là ở quá xa nhau để có thể duy trì kết nối nên kết nối bị đứt.



Hình 12.2: Kích thước cửa sổ TCP trong kịch bản 3 nút mạng với giao thức định tuyến DSDV



Hình 12.3: Kích thước cửa sổ TCP trong kịch bản 3 nút mạng với giao thức định tuyến DSDV với kết nối qua và không qua trung gian)

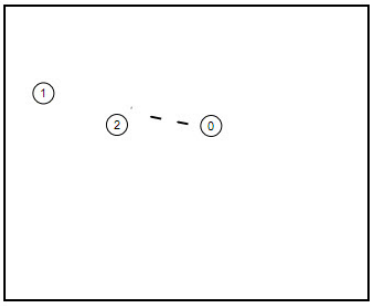
12.5. So sánh với định tuyến ad-hoc khác

12.5.1. TCP qua DSR

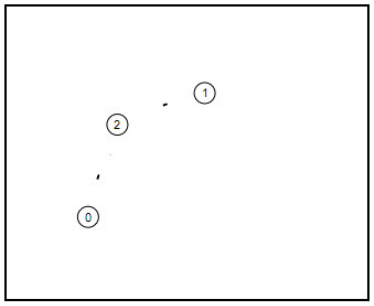
Trước hết chúng ta thay đổi giao thức định tuyến sang DSR bằng cách thay đổi wrls-dsdv.tcl thành :

```
Set val(rp)  DSR  ;# routing protocol
```

Khi thực hiện mô phỏng, chúng ta sẽ quan sát 5 giai đoạn. Trong giai đoạn đầu tiên và cuối cùng, các nút mạng ở quá xa nhau nên sẽ không có kết nối. Giai đoạn 2 và 4 kết nối giữa nút 0 và 1 theo đường thông qua nút 2 , trong khi đó ở giai đoạn 3 kết nối trực tiếp giữa 2 nút mạng.



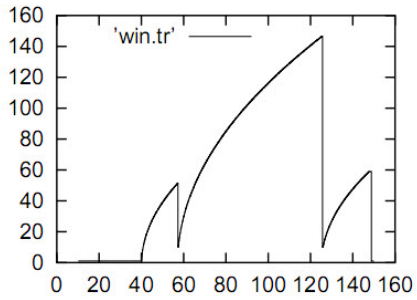
Hình 12.4: TCP trong kịch bản 3 nút mạng với giao thức định tuyến DSDV, thời điểm 124.14 giây, kết nối trực tiếp



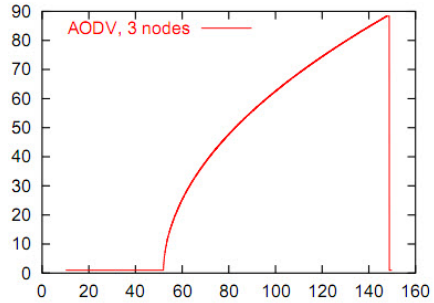
Hình 12.5: TCP trong kịch bản 3 nút mạng với giao thức định tuyến DSDV, thời điểm 58 giây, kết nối qua 1 trung gian

Giai đoạn 2 bắt đầu vào khoảng thời gian 40. Giai đoạn 3 bắt đầu vào khoảng 60 giây.

Tại thời điểm 125.50 giai đoạn 4 bắt đầu và kết thúc tại điểm 149 giây đồng thời cũng kết thúc tất cả kết nối. Các giai đoạn được mô tả trong hình 12.6



Hình 12.6: Kích thước cửa sổ của kết nối TCP qua DSR



Hình 12.7: Kích thước cửa sổ của kết nối TCP qua AODV

Từ kết quả thu được ta có thể thấy được các điểm sau:

- Trong DSDV, hệ thống không thể cung cấp được giai đoạn 4 do vậy kết nối bị đứt sớm hơn.
- Tổng số gói tin TCP được truyền sử dụng DSR nhiều hơn DSDV. Với DSR có 6770 gói tin TCP (dữ liệu) được nhận, trong khi DSDV với cùng các tham số thì số gói tin là 2079. (Chúng ta có thể thu được thông tin này bằng cách sử dụng:

  `Grep ‘‘^r’’ simple.tr | grep ‘‘tcp’’ | grep ‘‘_1_AGT’’ > tcp.tr`

  và sau đó đếm số dòng. Hoặc chúng t cũng có thể có được bằng cách nhìn vào số thứ tự của gói tin tcp cuối cùng.)

Nếu chúng ta lần theo dấu vết của một gói tin TCP, chẳng hạn với gói tin có số thứ tự 5, chúng ta sẽ thấy nó xuất hiện nhiều lần:

```
s 40.298003207 _0_ AGT   --- 1507 tcp 1040 [0 0 0 0] ...
r 40.298003207 _0_ RTR   --- 1507 tcp 1040 [0 0 0 0] ...
s 40.298003207 _0_ RTR   --- 1507 tcp 1060 [0 0 0 0] ...
f 40.310503613 _2_ RTR   --- 1507 tcp 1060 [13a 2 0 800] ...
r 40.310503613 _2_ RTR   --- 1507 tcp 1060 [13a 2 0 800] ...
f 40.310503613 _2_ RTR   --- 1507 tcp 1068 [13a 2 0 800] ...
r 40.348863637 _1_ RTR   --- 1507 tcp 1068 [13a 1 2 800] ...
r 40.348863637 _1_ RTR   --- 1507 tcp 1040 [13a 1 2 800] ...
```

Gói tin đầu tiên được gửi bởi lớp TCP của nút số 0 sau đó được nhận bởi giao thức định tuyến của cùng nút đó và được gửi đi từ cùng với một header đã thêm vào. Tiếp đó

nó được nhận và chuyển tiếp bởi nút số 2, cuối cùng nó được nhận tại nút số 1 tại lớp định tuyến, cuối cùng là lớp TCP. Dấu vết ở trên thu được bằng cách mở chức năng ghi dấu vết của agentTrace và routerTrace. 4 dòng còn lại đề cập đến cùng một gói tin sẽ xuất hiện nếu chúng ta mở chức năng macTrace.

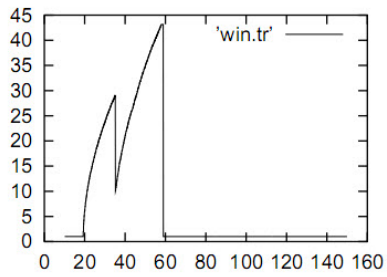
12.5.2. TCP qua AODV

Việc mô phỏng với cùng tham số như trên được lặp lại với AODV. Tiến trình có thể thấy ở hình 12.7 . Kết nối đã truyền đi 3924 gói tin dữ liệu TCP. Từ đầu đến cuối nó ở trong một giai đoạn dài với đường định truyến qua node 2 trung gian.

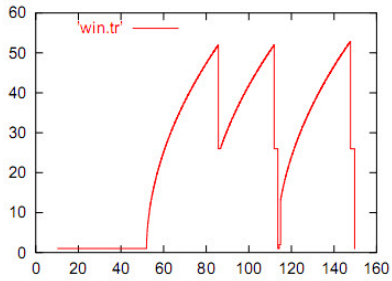
Thực tế là việc thay đổi đường định tuyến đã bị ngăn, không có sự mất mát gói tin do vậy kích thước cửa sổ vẫn cao. Tuy nhiên, chúng ta thấy rằng giá trị của nó vẫn thấp hơn DSR. Điều này là do thực tế thời gian đi một vòng (tham số cần thiết để tăng window một đơn vị) là dài hơn bởi vì ở đây không sử dụng định tuyến trực tiếp. Điều này giải thích tại sao nó truyền ít dữ liệu hơn DSR. Chúng ta nhận thấy ở đây rằng việc tìm kiếm quãng đường ngắn hơn dẫn đến hiệu suất TCP tốt hơn.

12.5.3. TCP qua TORA

Với cùng tham số như wrls-dsdv.tcl, TORA không truyền được gói tin nào cả. Để tăng tính kết nối, chúng ta thêm vào một node cố định khác tại tọa độ (250,240) với vai trò chỉ là để chuyển tiếp các gói tin. Tiến trình thu được như trong hình 12.8.



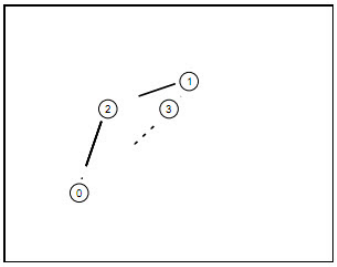
Hình 12.8: Kích thước cửa sổ của kết nối TCP qua TORA với 4 nút mạng



Hình 12.9: TCP qua AODV với giá trị lớn của kích thước cửa sổ lớn nhất

Tại thời điểm ban đầu không có kết nối nào cả. Khi kết nối hình thành, một đường định tuyến được tạo ra bằng cách sử dụng các nút mạng: 0-2-3-1 (xem hình 12.10 miêu tả thời điểm 33). Tại điểm thời gian 34.5 một đường ngắn hơn được hình thành: 0-2-1 nhưng đường cho ACK vẫn không thay đổi. Sau đó vào thời điểm 44 đường cho ACK thay đổi thành 1-3-0 (hình 12.11).

Hình 12.10: TCP qua TORA với 4 nút mạng, điểm thời gian 33



Hình 12.11: TCP qua Tora với 4 nút mạng. Điểm thời gian 56

12.5.4. Một vài bình luận

Trong các ví dụ chúng ta đã thấy, việc mất gói tin xảy ra khi khoảng cách vật lý là quá xa hoặc khi đường định tuyến thay đổi, và gói tin sẽ không bị mất khi có bộ đệm bị tràn. Điều này là do thực tế rằng chúng ta đã sử dụng một giá trị mặc định của kích cỡ lớn nhất của cửa sổ TCP là 20. Do đó cửa sổ thực tế được sử dụng là nhỏ nhất giữa độ dài cửa sổ nghẽn và 20. Trong hình 9.9 chúng ta trình bày một tiến trình kích thước cửa sổ của TCP qua AODV dưới các điều kiện giống hệt như cái chúng ta đã sử dụng để có được hình 9.7 nhưng giá trị cửa sổ lớn nhất là 2000. Chúng ta thấy rằng chúng ta sẽ bị mất gói tin do tràn bộ đệm.

12.6. Sự tác động của TCP tới giao thức MAC

12.6.1. Bối cảnh

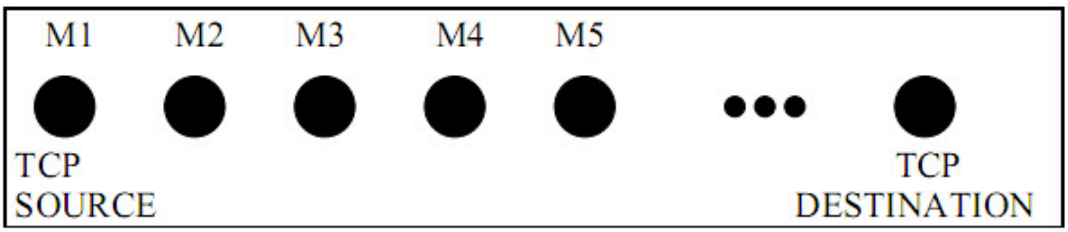
Trong phần trước chúng ta đã đề cập tới một lượng nhỏ các điểm di động và đã thấy được ảnh hưởng của việc di động tới hiệu suất của TCP. Khi chúng ta có một số lượng lớn các điểm di động, hiện tượng riêng biệt mới do các lớp MAC và lớp vật lý có thể có một tác động nghiêm trọng tới hiệu suất TCP. Để có thể hiểu được tác động này chúng ta trước tiên sẽ miêu tả một vài khía cạnh liên quan đến sự vận hành của lớp MAC IEEE802.11 và lớp vật lý.

Mỗi một truyền dẫn của gói tin DATA tại lớp MAC là một phần của một giao thức bắt tay 4 bước. Điểm di động muốn gửi một gói tin, chúng ta gọi nó là M1, đầu tiên sẽ gửi một gói RTS (Request to Send). Nếu điểm di động đích (gọi là M2) có thể nhận được gói tin này nó sẽ gửi một gói tin CTS (Clear to Send). Nếu M1 nhận được CTS nó có thể gửi gói tin DATA (có thể là dữ liệu TCP hoặc gói ACK). Cuối cùng M2 gửi một gói tin ACK (lớp MAC) để M1 hiểu rằng gói dữ liệu đã được nhận hoàn chỉnh.

Giao thức bắt tay được dùng để giảm khả năng xung đột. Các xung đột có thể xảy ra khi một điểm di động (giả sử là M3) cũng muốn gửi gói tin đến cho M2 tại cùng thời điểm với M1; M3 có thể không biết được rằng M1 cũng đang truyền dữ liệu đến M2, khi đó một sự xung đột giữa các gói tin của M1 và M3 có thể xảy ra ở M2. Hiện tượng này được gọi là "hiện tượng điểm cuối bị dấu". Với giao thức bắt tay, M3 sẽ không cố gắng gửi gói tin khi nó biết được gói tin CTS gửi từ M2 tới M1.

Nếu M1 không nhận được gói tin CTS nó sẽ dừng việc truyền tin lại và sau một khoảng thời gian nó sẽ gửi lại một gói tin RTS. Node gửi tin sẽ hủy gói tin DATA nếu nó đã gửi tin RTS 7 lần mà vẫn không nhận được gói tin phản hồi CTS nào. Một gói tin DATA cũng sẽ bị hủy đi sau 4 lần truyền lại mà vẫn không nhận được gói tin ACK (lớp MAC).

Mặc dù bắt tay sẽ giảm khả năng xuất hiện các xung đột "dấu điểm cuối", nhưng nó không hoàn toàn chấm dứt được nó. Để hiểu được cách mà các xung đột như vậy vẫn có thể xảy ra, chúng ta phải xem xét đến phạm vi địa lý của nhiễu và sự tiếp nhận. Phần cứng hiện nay chỉ ra rằng phạm vi truyền dẫn là khoảng 250 m và phạm vi nhiễu là khoảng 550m. Theo như cấu hình chuỗi trong hình 12.12 , khoảng cách giữa các node là 200m. Although nodes that are two hops apart are not hidden from each other, nodes that are three hops apart are, and may creat collisions. Thực tế, nếu node M4 muốn truyền tin đến M5 thông qua một đường truyền từ M1 đến M2, nó không thể nghe thấy CTS từ node M2 vì khoảng cách từ nó đến M1 lớn hơn 550m. Do đó M4 có tể tạo ra một đường truyền đến M5 và nó sẽ va chạm tại node M2 với đường truyền trừ M1. Chúng ta sẽ nghiên cứu trong phần này sự va chạm cả loại xung đột này trên hiệu suất TCP bằng cách sử dụng mô phỏng ns, giới hạn với topo chuỗi. Chúng ta sẽ không đề cập đến vấn đề di động ở đây. Chúng ta xem xét [3, 19, 37] để thấy chi tiết hơn.



Hình 12.12: Chuỗi topo

Hiện tượng mà chúng ta đã mô tả giới hạn của số lượng gói tin có thể truyền đồng thời trong mạng ad-hoc mà không gặp xung đột. Và chính Sự thúc ép về không gian trở thành một nhân tố chính làm giới hạn hiệu quả của TCP chứ không phải là tràn bộ đệm. Được thể hiện trong [19] với cấu hình chuỗi của chúng ta, sẽ có ích nếu chúng ta giới hạn kích cỡ lớn nhất của cửa sổ TCP xuống khoảng $n/4$; hơn nữa việc tăng kích cỡ lớn nhất của cửa sổ còn gây ra nhiều sự xung đột hơn và là giảm giá trị thời gian. Trong phần này chúng ta sẽ kiểm tra khẳng định này bằng cách mô phỏng. Hơn nữa, vì số lượng gói tin có thể

truyền đồng thời bị giới hạn nên chúng ta sẽ thử tăng số gói tin TCP truyền thành công bằng cách giảm đường truyền ACK,sử dụng ACK trễ (Delayed ACK). Ns cho phép chúng ta mô phỏng ACK trễ với $d = 2$. Chúng ta sẽ xem xa hơn cách vận hành trong trường hợp $d > 2$ bằng cách tạo ra thay đổi trong trình mô phỏng ns.

12.6.2. Kịch bản mô phỏng

Chúng ta sử dụng mô hình truyền sóng mặt đất 2 tia, IEEE892.11 MAC, và mô hình anten đẳng hướng của ns. Chúng ta sử dụng thuật toán định tuyến AODV, một hàng đợi giao diện dài 50 tại mỗi nút mạng. Chúng ta sẽ kiểm tra phiên bản NewReno của TCP, là phiên bản đã được triển khai nhiều nhất. Chúng ta làm với 4 kịch bản: 3, 9, 20 và 30. Trong trường hợp của 3 và 9 đòi hỏi 150 giây cho mỗi mô phỏng (để có được sự vận hành ổn định). Các trường hợp còn lại đòi hỏi 1500 giây cho mỗi mô phỏng. Một gói tin dữ liệu TCP được đặt kích cỡ 1040 bytes (đã bao gồm header). Kịch bản cho trường hợp ACK bị trễ (với $d = 2$) được cho trong bảng 12.2. Ở dưới, khi cấu hình các node chúng ta sẽ sử dụng tùy chọn "macTrace ON" để có được dấu vết chi tiết của gói tin giao thức MAC. Điều này cho phép chúng ta phân tích lý do của mỗi gói tin TCP bị thất lạc.

```
# Định nghĩa các tùy chọn
Set val(chan)     Channel/WirelessChannel;    # loại kênh
Set val(prop)     Propagation/TwoRayGround;   # radio-propagation model
Set val(netif)    Phy/WirelessPhy;            # network interface type
Set val(mac)      Mac/802.11;                 # loại MAC
Set val(ifq)      Queue/DropTail/PriQueue;    # interface queue type
Set val(ll)       LL;                         # link layer type
Set val(ant)      Antenna/OmniAtenna;         # antenna model
Set val(ifqlen)   50;                         # max packet in ifq
Set val(nn)       9;                          # number of mobilenodes
Set val(rp)       AODV;                       # routing protocol
Set val(x)        2200;                       # X dimention of topography
Set val(y)        500;                        # Y dimention of topography
Set val(stop)     150;                        # time of simulation end
Set ns             [new Simulator]
Set tracefd        [open simple.tr w]
Set windowVsTime2  [open win.tr w]

$ns trace-all $tracefd
```

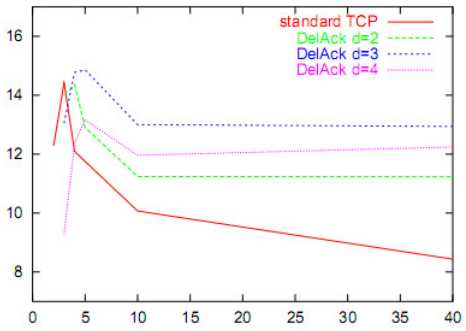
```
# set up topography object
Set top             [new Topography]

$topo load_flatgrid $val(x) $val(y)

Creat-god $val(nn)

# Creat nn mobilenodes [$val(nn)] and attach them to the channel

# configure the nodes
   $ns node-config -adhocRouting $val(rp) \
                   -llType $val(ll) \
                   -macType $val(mac) \
                   -ifqType &val(ifq) \
                   -ifqLen $val(ifqlen) \
                   -antType $val(ant) \
                   -propType $val(prop) \
                   -phyType $val(netif) \
                   -channelType $val(chan) \
                   -topoInstance $topo \
                   -agentTrace ON \
                   -routerTtrace ON \
                   -macTrace ON \
                   -movementTrace OFF
for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
}
# Provide initial location of mobilenodes

For {set i 0} {$i < $val(nn)} { incr i } {
$node_($i) set X_ [expr  ($i+1)*200.0]
$node_($i) set Y_ 250.0
$node_($i) set Z_ 0.0
}

# Set a TCP connection between node_ (0) and node_ (8)
Set tcp [new Agent/TCP/Newreno]
```

```
$tcp set class_2
$tcp set window_ 2000
Agent/TCPSink/DelAck set interval_ 100ms
Set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_ (8) $sink
$ns connect $tcp $sink
Set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 ‘‘$ftp start’’

# In ra kích thước cửa sổ
Proc plotWindow {tcpSource file} {
Global ns
Set time 0.1
Set now [$ns now]
Set cwnd [$tcpSource set cwnd_]
Puts $file ‘‘$now $cwnd’’
$ns at [expr $now+$time] ‘‘plotWindow $tcpSource $file’’ }
$ns at 1.1 ‘‘plotWindow $tcp $WindowVsTime2’’

# Telling nodes when the simulation ends
For {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) ‘‘$node_($i) reset’’:
}

$ns at $val(stop) ‘‘stop’’
$ns at [expr $val(stop)+0.1] ‘‘puts \’’end simulation\’’ ; $ns halt’’
Proc stop {} {
   Global ns tracefd
   $ns flush-trace
   Close $tracefd
}
$ns run
```

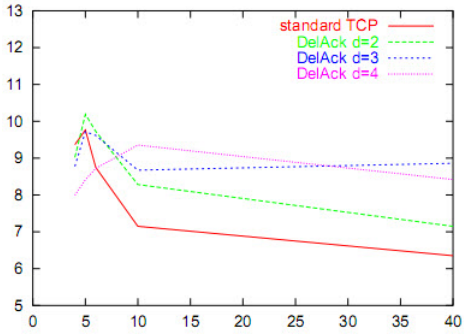
Bảng 12.2: Kịch bản tcpwD.tcl cho TCp qua một mạng ad-hoc tĩnh với chuỗi topo

12.6.3. Các kết quả mô phỏng

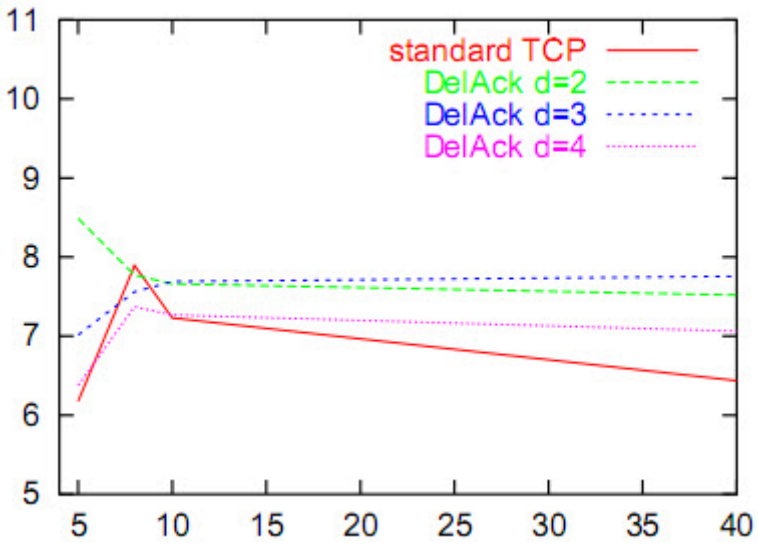
Kết quả mô phỏng đối với n = 9, 20 và 30 nút mạng được ghi lại ở các hình 12.13 đến hình 12.15



Hình 12.13: Số gói tin/giây đối với $n = 9$ như một chức năng của kích cỡ cửa sổ lớn nhất



Hình 12.14: Số gói tin/giây đối với $n = 20$ như một chức năng của kích cỡ cửa sổ lớn nhất



Hình 12.15: : Số gói tin/giây đối với $n = 30$ như một chức năng của kích cỡ cửa sổ lớn nhất

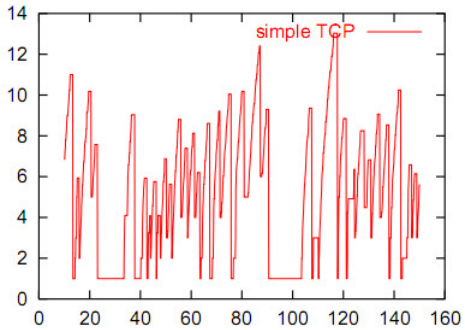
Chúng ta thấy rằng tùy chọn trễ ACK (Delayed ACK) chuẩn ($d = 2$) làm tốt hơn một chút TCP chuẩn (nhưng không tốt hơn với các giá trị lớn nhất khác của kích thước cửa sổ) đối với $n = 9$, và làm tốt hơn nhiều (hơn 10%) TCP chuẩn với $n = 30$. Sự tiến bộ này thu được trễ ACK với $d = 3$ (đối với $n = 9$ và $n = 20$). Nhưng sự tiến bộ lớn nhất mà chúng ta thấy chính là tất cả các phiên bản trễ ACK đều tốt hơn TCP chuẩn đối với các kích thước

cửa sổ lớn nhất lớn hơn 10, với các tùy chọn $d = 3$ hoặc $d = 4$ thực hiện tốt hơn tùy chọn trễ ACK chuẩn. Với $n = 9$, phiên bản trễ ACK với $d = 3$ chúng ta thấy mang lại từ 30% đến 40% tiến bộ hơn so với TCP chuẩn với bất kỳ giá trị kích thước lớn nhất của cửa sổ lớn hơn 10 nào; trong phạm vi đó nó cũng làm tốt hơn TCP chuẩn $20\% - 30\%$ với $n = 20$ và $6\% - 20\%$ với $n = 30$. Phiên bản $d = 4$ thậm chí còn làm tốt hơn với $n = 20$ cho các cửa sổ có kích cỡ lớn nhất mang giá trị từ 10 đến 25. Hiệu suất tốt hơn của trễ ACK còn có thể thu được bằng cách tối ưu khoảng thời gian của các tùy chọn trễ ACK

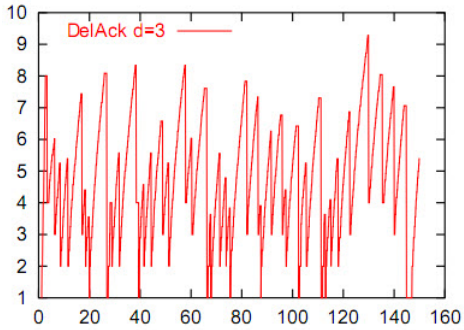
Kết luận quan trọng nhất của các đường cong là tính mạnh mẽ của các tùy chọn trễ ACK. Trong thực tế, khi chúng ta không biết số lượng nút mạng, không có lý do gì để giới hạn kích thước cửa sổ xuống một giá trị nhỏ, vì vậy điều này có thể làm giảm số lượng gói tin gửi được một cách đáng kể. Khi lựa chọn giá trị lớn cho kích cỡ lớn nhất của cửa sổ, các phiên bản trễ ACK làm tốt hơn một cách đáng kể so với TCP chuẩn. Chúng nhận được hầu hết các giá trị tối ưu mà TCP chuẩn có thể nhận được nếu nó biết số lượng nút mạng và có thể chọn kích cỡ lớn nhất của cửa sổ một cách tương ứng.

Đối với một giá trị nhỏ cố định của kích cỡ lớn nhất của cửa sổ, tùy chọn trễ ACK không làm tốt hơn TCP chuẩn vì trong hầu hết thời gian, kích cỡ của cửa sổ giới hạn số lượng gói tin TCP được truyền xuống nhỏ hơn d, điều đó có nghĩa là tùy chọn trễ ACK phải chờ cho đến khi bộ đếm thời gian (mặc định là $100ms$) kết thúc trước khi tạo ra một ACK; trong suốt khoảng thời gian này node nguồn không thể truyền được tin.

Tiếp theo, chúng ta vẽ đồ thị tiến trình cho kích cỡ cửa sổ với $n = 9$ cho TCP chuẩn và cho TCP với tùy chọn Delayed ACK với $d = 3$. Kích cỡ cửa sổ được lấy mẫu với chu kỳ 0.1 giây. Chúng ta thấy rằng mặc dù kích cỡ lớn nhất cửa sổ là 2000, cửa sổ nghẽn thực tế không vượt quá 13. Từ các hình chúng ta thấy rằng với TCP chuẩn, số gói tin mất là thường xuyên hơn và nghiêm trọng hơn (làm tăng timeout) trong khi với phiên bản d=3 của trễ ACK không hề làm tăng timeout.



Hình 12.16: Tiến trình của kích thước cửa sổ cho TCP chuẩn với kích thước cửa sổ lớn nhất là 2000

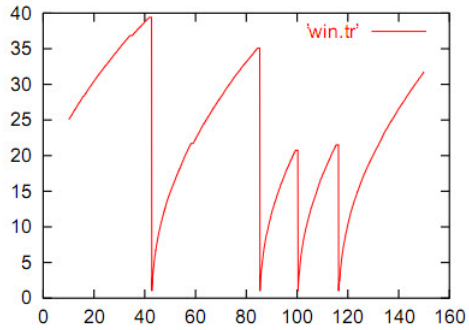


Hình 12.17: Tiến trình của kích thước cửa sổ cho DelAck TCP với $d = 3$ và kích thước cửa sổ lớn nhất là 2000

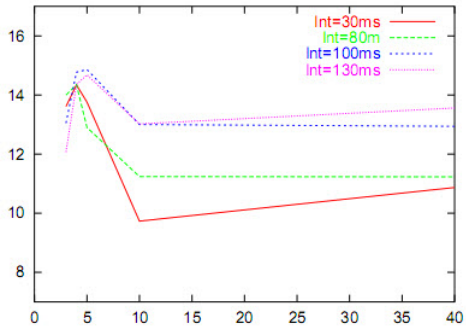
Trong hình 12.18 chúng ta thấy tiến trình của kích cỡ cửa sổ nghẽn của TCP chuẩn với kích cỡ lớn nhất của cửa sổ là 3 trong trường hợp 9 nút mạng. Chúng ta đã biết rằng một kích cỡ lớn nhất giữa 2 và 3 sẽ thật sự mang đến hiệu suất tối ưu (và điều này đã được xác minh lần nữa ở hình 12.13). Chúng ta thấy trong hình 12.18 rằng hầu như không có sự mất mát gói tin. Để ý là kích cỡ cửa sổ thực tế là nhỏ nhất giữa cửa sổ nghẽn (được miêu tả trong hình) và kích cỡ lớn nhất của cửa sổ (giá trị của nó ở đấy là 3).

Trong các hình trước đó, tất cả các phiên bản sử dụng trễ ACK đều đã có khoảng thời gian mặc định là 100ms (như đã giải thích trong phần giới thiệu). Tiếp theo, chúng ta thay đổi độ dài của khoảng thời gian và kiểm tra sự tác động của nó với số lượng gói tin, xem hình 12.19. Chúng ta xem xét phiên bản trễ ACK với $d = 3$. Chúng ta thấy rằng giá trị mặc định thực hiện khá tốt, mặc dù với các cửa sổ có kích thước lớn nhất nhỏ, khoảng thời gian nhỏ thực hiện tốt hơn một chút, trong khi đó với cửa sổ có kích thước lớn nhất lớn, một khoảng thời gian lớn (130ms) sẽ tốt hơn. Chúng ta đã cố gắng tăng khoảng thời gian trên 130ms nhưng sau đó số gói tin đã giảm xuống.

Cuối cùng, chúng ta xem xét trường hợp $n = 3$ nút mạng. Trong trường hợp này hiện tượng dấu điểm cuối không còn xảy ra nữa, vì thế chúng ta không thấy việc mất mát gói tin TCP nào cả, cho dù với bất cứ kích cỡ cửa sổ nào. Thậm chí sau đó, trễ ACK có thể được sử dụng để tăng đáng kể hiệu suất. Việc này được chứng minh trong bảng 12.3 và mang tới số lượng gói tin TCP được nhận thành công trong 149 giây với $n = 3$. Sự cải tiến đã làm tăng từ 10% đến 15% khi d tăng từ 2 lên 4, không phụ thuộc vào kích cỡ lớn nhất của cửa sổ (chỉ cần nó lớn hơn d).



Hình 12.18: Tiến trình của kích thước cửa sổ cho TCP chuẩn với 9 nút mạng và kích thước cửa sổ lớn nhất là 3



Hình 12.19: Tác động của khoảng thời gian trễ ACK tới số gói tin TCP gửi được, như là một chức năng của kích thước cửa sổ lớn nhất. d=3

Tuy nhiên với $d = 4$ như mong đợi, chúng ta có thể thất rằng chúng ta nhận được một hiệu suất tồi với kích cỡ lớn nhất của cửa sổ là 3, bởi vì đích luôn phải chờ đợi cho đến khi khoảng thời gian 100ms của tùy chọn trễ ACK hết để gửi một gói tin ACK (vì các cửa sổ

	Standard TCP	*Delayed Ack Versions*		
WinMax	Standard	$d = 2$	$d = 3$	$d = 4$
3	6068	6602	6763	2699
2000	6094	6565	6779	6888

Bảng 12.3: Số lượng gói tin gửi được trong khoảng thời gian 149 giây với n=3 như một chức năng của kích thước cửa sổ lớn nhất

chỉ cho phép gửi 3 gói dữ liệu).

12.6.4. Thay đổi cho NS với trường hợp $n > 2$

Chương 13

Hàng đợi cổ điển

Mục lục

Người dịch: *Lê Thị Nga*
Biên tập: *Nguyễn Anh Tôn*
Bản gốc: *NS Simulator for beginners, Chapter 10* [2]

Ns2 có thể được sử dụng để mô phỏng các mô hình hàng đợi cổ điển. Trong các mô hình cổ điển đơn giản nhất, thời gian đến giữa các gói là ngẫu nhiên và phân bố xác suất chung nào đó, và thời gian để truyền một gói là cũng là ngẫu nhiên theo các phân bố khác nào đó. Thực tế thời gian truyền dẫn thay đổi có thể phản ánh sự không đổi của tốc độ truyền nhưng nhưng có sự biến đổi kích thước gói.

13.1. Mô phỏng hàng đợi

Ví dụ hàng đợi đơn giản nhất cho phân tích toán học là hàng đợi $M/M/1$: λ là khoảng thời gian giữa hai lần đến phân bố theo hàm mũ phụ thuộc tham số và μ là tổng thời gian truyền của một gói cũng có phân bố mũ với các tham số khác. Một gói có thể được truyền đi tại một thời điểm và kích thước bộ đệm là vô hạn. Nếu như chúng ta chỉ rõ $\rho = \mu/\lambda$,

thời gian trung bình của gói trong hệ thống là:

$$E[Q] = \frac{\rho}{1-\rho} \tag{13.1}$$

Bảng 13.1 giới thiệu mô phỏng của hàng đợi này. Mô phỏng này tạo ra file `out.tr` với tất cả các sự kiện, and also a monitor-queue trace called qm.out, và cũng tạo ra một file truy vết dung để giám sát hang đợi gọi là qm.out,như đã được đề cập ở mục 7.3. Bằng cách vẽ cột 5 (các gói trong kích cỡ hàng đợi) dựa vào cột 1 (thời gian) Bằng cách vẽ đồ thị cột 5 theo cột 1, chúng ta nhận được sự mở rộng chiều dài hàng đợi (xem hình 13.1). Kích cỡ hàng đợi được mô phỏng trung bình trên 1000sec là 9.69117, giá trị xấp xỉ của 10 theo 13.1 . Chú ý rằng chúng ta sử dụng cách đơn giản hơn để trình bày và vận dụng các biến ngẫu nhiên so với cách được thể hiện ở mục 2.7: chúng ta không trình bày generators và seeds.

```
set ns [new Simulator]
    set tf [open out.tr w]
$ns trace-all $tf
set lambda 30.0
set mu 33.0
set n1 [$ns node]
set n2 [$ns node]
# kích thước các gói được làm tròn thành số nguyên (bytes),
# chúng ta phải có kích thước các gói lớn hơn và lỗi do làm tròn nhỏ,
# để băng thông lớn
set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]
$ns queue-limit $n1 $n2 10000
# thiết lập kích thước gói và thời gian đến giữa các gói ngẫu nhiên
set InterArrivalTime [new RandomVariable/Exponential]
$InterArrivalTime set avg_ [expr 1/$lambda]
set pktSize [new RandomVariable/Exponential]
$pktSize set avg_ [expr 100000.0/(8*$mu)]
set src [new Agent/UDP]
$ns attach-agent $n1 $src
# quản lý hàng đợi
set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]
$link queue-sample-timeout
proc  finish {} {
global ns tf
```

```
$ns flush-trace
close $tf
exit 0
}
proc sendpacket {} {
global ns src InterArrivalTime pktSize
set time [$ns now]
$ns at [expr $time + [$InterArrivalTime value]] ‘‘sendpacket’’
set bytes [expr round ([$pktSize value])]
$src send $bytes
}
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $src $sink
$ns at 0.0001 ‘‘sendpacket’’
$ns at 1000.0 ‘‘finish’’
$ns run
```

Bảng 13.1: Kịch bản tcl mm1.tcl để mô phỏng hàng đợi MM1

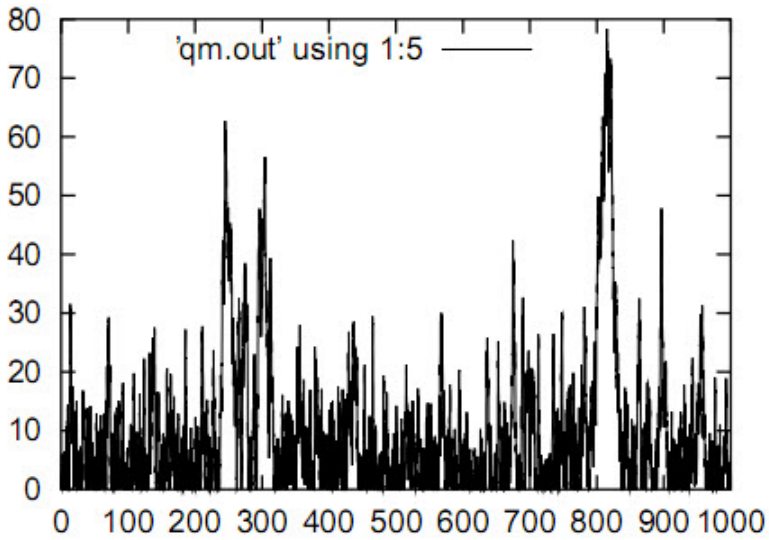
Khá hấp dẫn khi phân tích kết quả mô phỏng và cố gắng tìm lý do hợp lý cho sự khác biệt. Chúng ta có thể tìm được vài lý do cho the simulation's impressisions (và sử dụng các kết luận để cải thiện mô phỏng).

- Công thức 13.1 tính toán toàn bộ một gói đang được truyền, trong khi sự mô phỏng chỉ tính tới phần còn lại của gói dữ liệu đã được truyền mà vẫn còn nằm trong hang đợi. Sự khác biệt này có thể tạo nên kết quả mô phỏng thấp hơn kết quả chính xác khoảng 0,5 gói.

  *Cách khác, các gói mô phỏng được rút ngắn tại giá trị 1KB, giá trị kích cỡ mặc định của một gói UDP. Như vậy thời gian truyền nhỏ hơn so với ý định của chúng ta. Để sửa lại, có thể thay đổi giá trị cực đại mặc định của gói, ví dụ như 100000, bằng cách thêm dòng lệnh: `$src set packetSize_ 100000`
  sau lệnh: `src [new Agent/UDP]`

- Thời gian mô phỏng trên không đủ dài. Với thời gian 20000, ta nhận được giá trị chính xác hơn.

Hàng đợi M/D/1 có thời gian đến phân bố mũ nhưng thời gian truyền các gói là không đổi. Để mô phỏng, đơn giản thay thế biến ngẫu nhiên pktSize bằng giá trị trung bình của biến. Cũng như thế, hàng đợi D/M/1 có thời gian truyền phân bố mũ và thời gian giữa hai gói đến là không đổi. Để mô phỏng, chúng ta có thể thay thế biến ngẫu nhiên InterarrivalTime bởi giá trị trung bình.



Hình 13.1: Sự mở rộng kích cỡ của hàng đợi $M/M/1$

Hàng đợi $M/D/1$ có thời gian đến phân bố mũ nhưng thời gian truyền các gói là không đổi. Để mô phỏng, đơn giản thay thế biến ngẫu nhiên `pktSize` bằng giá trị trung bình của biến. Cũng như thế, hàng đợi $D/M/1$ có thời gian truyền phân bố mũ và thời gian giữa hai gói đến là không đổi. Để mô phỏng, chúng ta có thể thay thế biến ngẫu nhiên InterarrivalTime bởi giá trị trung bình.

13.2. Hàng đợi hữu hạn

Trong mô phỏng ở trên, ta đã sử dụng bộ đệm rất lớn để tránh mất gói dữ liệu. Có thể sử dụng các bộ đệm nhỏ hơn và quan sát mất mát. Với hàng đợi M/M/1 và K bộ đệm, xác suất mất gói là:

$$P(loss) = \frac{\rho^K}{\sum_{i=0}^{k} \rho^i}$$

Cách đơn giản để tính toán xác suất mất gói từ mô phỏng là chia tổng số gói bị mất cho tổng số gói đến, hai giá trị này nhận được ở dòng cuối cùng của file giám sát hàng đợi.

```
set ns [new Simulator]
```

```
set tf [open out.tr w]
$ns trace-all $tf
set lambda 30.0
set mu 33.0
set qsize 2
set duration 2000
set n1 [$ns node]
set n2 [$ns node]
set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]
$ns queue-limit $n1 $n2 $qsize
# thiết lập kích thước gói và thời gian đến giữa các gói ngẫu nhiên
set InterArrivalTime [new RandomVariable/Exponential]
$InterArrivalTime set avg_ [expr 1/$lambda]
set pktSize [new RandomVariable/Exponential]
$pktSize set avg_ [expr 100000.0/(8*$mu)]
set src [new Agent/UDP]
$ns attach-agent $n1 $src
# quản lý hàng đợi
set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]
$link queue-sample-timeout
proc  finish {} {
global ns tf
$ns flush-trace
close $tf
exit 0
}
proc sendpacket {} {
global ns src InterArrivalTime pktSize
set time [$ns now]
$ns at [expr $time + [$InterArrivalTime value]] ‘‘sendpacket’’
set bytes [expr round ([$pktSize value])]
$src send $bytes
}
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $src $sink
$ns at 0.0001 ‘‘sendpacket’’
$ns at 1000.0 ‘‘finish’’
```

```
$ns run
```

Bảng 13.2: Kịch bản tcl mm1k.tcl mô phỏng hàng đợi MM1.

Thêm lệnh `$src set packetSize_100000` như đã được đề cập ở mục trước, ta có được sự phù hợp giữa mô phỏng và công thức. Ví dụ, với $K = 2$, ta có $P(loss) = 0.298$ với thời gian mô phỏng 2000 giây và $P(loss) = 0.3025$ ở công thức trên. Với $K = 5$, ta thu được kết quả tương ứng là 0.131 và 0.128. Các lệnh được thể hiện ở bảng 12.2 Chú ý: với $K = 1$, mô phỏng không thực hiện tốt; trong trường hợp này tất cả các gói đến đều bị mất

Chương 14

Mạng vệ tinh trong NS

Mục lục

Người dịch: *Trương Thị Hiền*
Biên tập: *Nguyễn Nam Hoàng*
Bản gốc: *The ns Manual, Chapter 3* [1]

Chương này tập trung mô tả các tính năng mở rộng để dễ dàng mô phỏng mạng vệ tinh trong ns. Cụ thể, các tính năng mở rộng này cho phép ns có thể mô hình hóa các phần sau:

- Các vệ tinh địa tĩnh loại chuyển tiếp (bent-pipe) truyền thống với đa người dùng trên các tuyến lên và tuyến xuống và các tuyến không đối xứng.
- Các vệ tinh địa tĩnh với các khối xử lý tín hiệu trên vệ tinh (processing payload) gồm hai loại: – hoặc là khối xử lý loại khôi phục tín hiệu (regenerative payload) hoặc là khối xử lý chuyển mạch gói hoàn toàn (full packet switching)
- Các chòm vệ tinh LEO quỹ đạo cực như Iridium hoặc Teledesic. Những mô hình vệ tinh này phần lớn được nhắm đến việc sử dụng ns cho nghiên cứu về các khía cạnh của hệ thống vệ tinh, như là các giao thức MAC, lớp liên kết dữ liệu, giao thức định tuyến, và giao thức giao vận.

14.1. Tổng quan về các mô hình vệ tinh

Muốn mô phỏng mạng vê tinh một cách chính xác đòi hỏi phải liệt kê đầy đủ bản chất của tần số sóng mang RF(như nhiễu, hiệu ứng fading), các tương tác lẫn nhau của giao thức (như sự tương tác giữa các lỗi cụm thặng dư trên đường truyền và mã kiểm tra lỗi), tác động thứ cấp của quỹ đạo (sự tiến động- precession, sự bất thường của lực hấp dẫn ...).Tuy nhiên, để nghiên cứu các đặc điểm cơ sở mạng vệ tinh qua các khía cạnh về kết nối mạng, các đặc tính nhất định có thể đơn giản hóa đi. Ví dụ như, hiệu năng của giao thức TCP qua đường truyền vệ tinh bị thay đổi đôi chút qua việc sử dụng một phép gần đúng sovới việc dùng mô hình kênh chi tiết - dẫn đến hiệu năng có thể được mô tả dựa vào toàn bộ xác xuất mất gói tin. Đây là cách tiếp cận dẫn đến kiểu mô phỏng này: đó là tạo ra một framework để nghiên cứu sự giao vận, định tuyến, và MAC trong môi trường vệ tinh bao gồm cả vệ tinh địa tĩnh lẫn chòm sao vệ tinh LEO (quỹ đạo thấp) quỹ đạo cực. Dĩ nhiên là người dùng có thể dựa vào các mô hình mở rộng này để thay đổi các chức năng tại một lớp nào đó.

14.1.1. Vệ tinh địa tĩnh

Quỹ đạo vệ tinh địa tĩnh của trái đất nằm trên đường xích đạo cách khỏang 36000 km so với mặt nước biển. Trên thực tế, vệ tinh địa tĩnh có thể bị trôi đi so với vị trí thiết kế ban đầu bởi sự thay đổi của trường hấp dẫn- những tác động này bị bỏ qua trong mô phỏng hệ thống ns.

Có hai mô hình vệ tinh địa tĩnh được đưa ra: Mô hình vệ tinh địa tĩnh chuyển tiếp "bent-pipe" truyền thống – nó đơn thuần chỉ là các bộ lặp lại trên quỹ đạo- tất cả các gói tin nhận được bởi vệ tinh loại này trên kênh tuyến lên được truyền dọc theo tần số sóng mang RF đến các đường tuyến xuống tương ứng (do vậy nó gọi là truyền theo đường ống). Những vệ tinh mới hơn sẽ tăng hiệu quả xử lý băng cơ sở, cả về phục hồi tín hiệu số lẫn thực hiện việc chuyển mạch gói nhanh tích hợp trên tàu không gian. Trong chương trình mô phỏng, các vệ tinh này có thể được mô hình hóa như là nút mạng ns thông thường với các khối chức năng thực hiện phân loại và định tuyến.

Trước đây, người dùng có thể mô phỏng đường truyền vệ tinh địa tĩnh bằng cách mô phỏng đơn giản một đường truyền trễ dài dùng đường truyền và nút mạng ns thông thường. Sự nâng cấp chính trong phần mở rộng so với các vệ tinh địa tĩnh là ở tính năng mô phỏng giao thức MAC. Người dùng có thể thiết lập nhiều thiết bị đầu cuối ở các vị trí khác nhau trên bề mặt trái đất và kết nối chúng đến đường truyền kênh tuyến lên và tuyến xuống của cùng một vệ tinh. Trễ truyền sóng trong hệ thống (có sự khác nhau chút ít với mỗi người dùng) sẽ được mô hình hóa một cách chính xác. Thêm vào đó, kênh tuyến lên và kênh tuyến xuống cũng có thể được định nghĩa khác nhau (có thể là ở băng thông khác nhau hoặc ở các kiểu mô hình lỗi khác nhau)

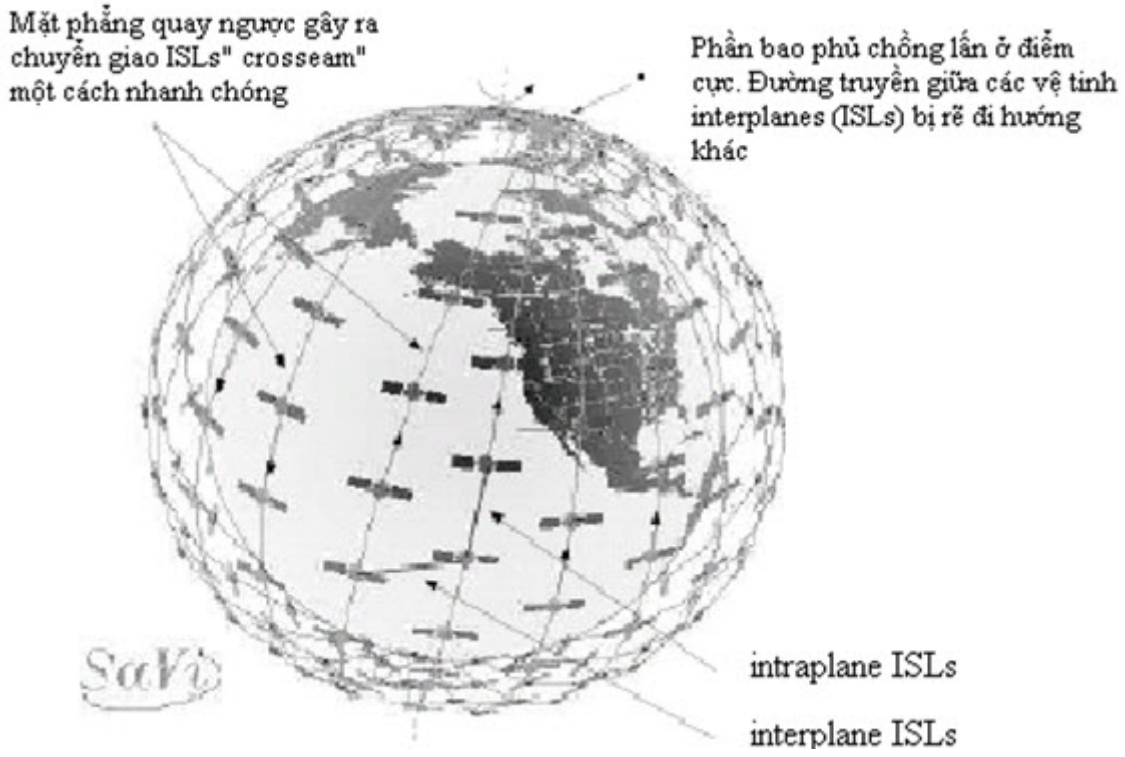
14.1.2. Các vệ tinh LEO (Các vệ tinh quỹ đạo thấp)

Hình minh họa này được tạo ra từ gói phần mềm Savi tại trung tâm nghiên cứu vệ tinh của đại học Minnesota

Các hệ thống vệ tinh quỹ đạo cực, chẳng hạn như hệ thống Iridium và hệ thống Teledesic vừa được đề xuất có thể được mô hình hóa trong ns. Ngoài ra cũng có các cấu hình cho các chòm sao vệ tinh phi địa tĩnh khác (ví dụ như chòm sao Walker). Người dùng có thể phát triển các lớp dành cho loại chòm sao mới để.

Các tham số của các chòm sao vệ tinh có thể mô phỏng được gồm có:

- **Các thông số cơ bản của chòm sao:** bao gồm độ cao của vệ tinh, số lượng vệ tinh, số mặt phẳng quỹ đạo và số vệ tinh trên mỗi mặt phẳng quỹ đạo.



Hình 14.1: Ví dụ của một chòm sao LEO quỹ đạo cực

- **Quỹ đạo:** độ nghiêng của quỹ đạo có thể biến đổi giá trị liên tục trong khoảng từ 00 đến 1800 (góc nghiêng lớn hơn 900 tương ứng với quỹ đạo ngược của hành tinh). Độ lệch tâm quỹ đạo thường không được tính đến. Sự tiến động nút cũng vậy. Khoảng cách giữa các vệ tinh trong cùng một mặt phẳng quỹ đạo được giữ cố định. Sự đồng bộ tương đối giữa các mặt phẳng quỹ đạo cũng được giữ nguyên (mặc dù là nhiều hệ thống có thể không điều khiển sự đồng bộ giữa các mặt phẳng quỹ đạo).

- **Đường truyền giữa các vệ tinh (ISL):** đối với các chòm sao quỹ đạo cực, các ISLs trong cùng mặt phẳng (intraplane ISL), giữa hai mặt phẳng cùng chiều lân cận (interplane ISL) và giữa hai nửa mặt phẳng ngược chiều (crossseam ISLs) đều có thể định rõ. Intraplane ISLs tồn tại giữa các vệ tinh trong cùng mặt phẳng và sẽ không bao giờ ngừng hoạt động hoặc chuyển giao. Interplane ISLs thì tồn tại giữa các vệ tinh lân cận với nhau xoay quanh mặt phẳng. Những đường kết nối như thế không được kích hoạt ở gần các điểm cực (ở trên ngưỡng vĩ độ ISLs như trong bảng) bởi vì cơ cấu vị trí ăntен không thể bám theo những đường truyền này trong vùng cực. Giống như intraplane ISLs, interplane ISLs cũng không bao giờ chuyển giao. Còn Crossseam ISLs thì có thể tồn tại trong một chòm sao giữa các vệ tinh trong mặt phẳng có hướng đối lập (nơi các mặt phẳng tạo nên cái gọi là "đường phân giới" trong các topo mạng). Đường truyền vệ tinh GEO cũng có thể được định rõ cho chòm sao của vệ tinh địa tĩnh.

	Iridium	Teleesic
Độ cao so với mặt nước biển	780km	1375km
Số mặt phẳng quỹ đạo	6	12
Số vệ tinh trên mỗi mặt phẳng	11	24
Độ dốc (inclination)(deg)	86.4	84.7
Khoảng cách giữa hai mặt phẳng (deg)	31.6	15
Chia đường phân giới(seam)(deg)	22	15
Ngưỡng góc ngẩng (deg)	8.2	4
Đồng bộ cùng mặt phẳng	Có	Có
Đồng bộ khác mặt phẳng	Có	Không
ISLs trên mỗi vệ tinh	4	8
Băng thông của ISL	25Mb/s	15Mb/s
Băng thông tuyến lên xuống	1.5Mb/s	1.5Mb/s
ISLs giữa hai mặt phẳng ngược chiều	Không	Có
Ngược vĩ độ ISL (deg)	60	60

- **Các đường truyền từ trái đất đến vệ tinh (GSL):** nhiều thiết bị đầu cuối có thể được kết nối đến một kênh vệ tinh GSL. Đường truyền GSL cho vệ tinh GEO là không đổi, trong khi đường truyền GSL cho kênh LEO thì định kì chuyển giao như mô tả ở sau.
- **Ngưỡng góc ngẩng thu tín hiệu (elevation mask):** Góc ngẩng mà mỗi đường truyền GSL có thể hoạt động. Hiện nay, nếu vệ tinh LEO đang phục vụ một thiết bị đầu cuối mà rơi khỏi ngưỡng goc ngẩng này, thiết bị đầu cuối sẽ tìm kiếm một vệ tinh khác mà đang nằm phía trên ngưỡng góc ngẩng. Các thiết bị đầu cuối vệ tinh xác định thời cơ chuyển giao theo các khoảng thời gian trễ đặc trưng bởi mỗi người dùng. Mỗi thiết bị đầu cuối khởi tạo chuyển giao không đồng bộ. Cũng có thể định rõ một hệ thống trong đó mỗi lần chuyển giao xảy ra một cách đồng bộ trong hệ thống.

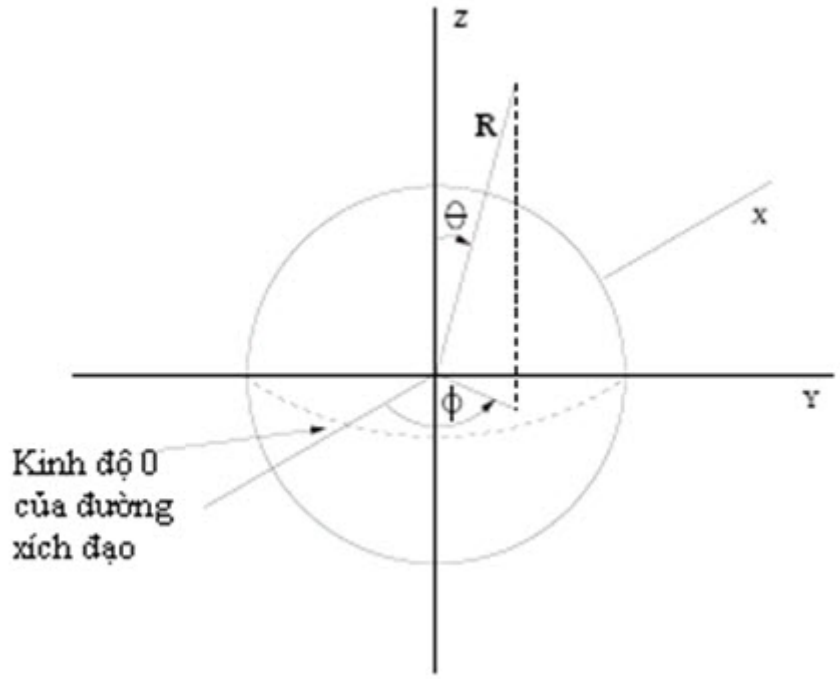
Bảng dưới đây liệt kê các thông số sử dụng mô phỏng cho hệ thống Iridium và Teledesic

14.2. Sử dụng các tính năng mở rộng cho vệ tinh

14.2.1. Nút mạng và vị trí của nút mạng

Các nút mạng vệ tinh có hai loại cơ bản : *nút mạng của vệ tinh địa tĩnh* hoặc *nút mạng của các vệ tinh không phải là địa tĩnh*. Thêm vào đó, các nút thuộc thiết bị đầu cuối có

thể đặt trên bề mặt trái đất. Có 3 loại nút mạng khác nhau được bổ sung với cùng lớp đối tượng Sat Node nhưng với các lớp đối tương khác nhau dành vị trí, quản lý chuyển giao và kênh truyền. Lớp đối tượng vị trí (position objects) theo dõi vị trí của vệ trinh trên hệ quy chiếu như là một hàm thời gian mô phỏng đã xảy ra. Thông tin vị trí này được dùng để quyết định trễ truyền kết nối và xấp xỉ thời gian cho chuyển giao kết nối.



Hình 14.2: Hệ thống tọa độ hình cầu dùng các nút mạng vệ tinh

Hình 14.2 minh họa một hệ thống tọa độ hình cầu, và hệ thống tọa độ Đề-các tương ứng.

Hệ thống tọa độ coi trái đất là trung tâm, và trục z trùng kít với góc quay của trục trái đất. $(R, \theta, \phi) = 6378km, 90^o, 0^o$ tương ứng với kinh độ 0 (kinh tuyến gốc) trên đường xích đạo.

Đặc biệt, có một lớp của nút vệ tinh là `Class Node/SatNode`, là một trong 3 loại đối tượng `Position` có thể gắn vào. Mỗi đối tượng `SatNode` và `Position` là đối tượng do OTcl/C++ phân ra riêng biệt, nhưng hầu hết mã chương trình nằm trong C++. Có các loại đối tượng Position sau:

- *Position/Sat/Term:* Mỗi thiết bị đầu cuối được xác định bằng tọa độ kinh độ và vĩ độ của nó. Miền vĩ độ từ [-90; 90] và miền kinh độ từ [-180; 180], với giá trị âm tương ứng cho hướng Nam và hướng Tây. Khi thời gian mô phỏng tăng, thiết bị đầu cuối di chuyển trên bề mặt Trái đất. Bộ tạo nút có thể sử dụng để tạo ra một thiết bị đầu cuối với đối tượng vị trí được gắn vào như sau:

```
$ns node-config -satNodeType terminal n \
(Các nút khác có thể thiết lập cấu hình ở đây)
set n1 [$ns node]
$n1 set-position $lat $lon; # độ tính theo thập phân
```

- *Position/Sat/Geo:* Một vệ tinh địa tĩnh GEO cũng có thông số đặc trưng là vị trí kinh độ trên xích đạo. Khi thời gian mô phỏng tăng, vệ tinh địa tĩnh di chuyển qua hệ thống tọa độ với chu kỳ quỹ đạo giống như chu kỳ quay của trái đất. Kinh độ thay đổi từ [-180; 180] độ. Có hai loại nút vệ tinh địa tĩnh quen thuộc, đó là GEO (cho vệ tinh có xử lý tín hiệu) và "bộ lặp geo" (cho vệ tinh chuyển tiếp). Bộ tạo nút có thể dùng để tạo ra vệ tinh địa tĩnh với đối tượng vị trí gắn vào như sau:

```
ns node-config -satNodeType geo (or ‘‘geo-repeater’’) n
(other node config commands go here...)
set n1 [$ns node]
$n1 set-position $lon; # in decimal degrees
```

- *Position/Sat/Polar* Vệ tinh quỹ đạo cực có quỹ đạo tròn theo mặt phẳng quỹ đạo cố định trong hệ thống tọa độ. Trái đất quay ở phía dưới của mặt phẳng quỹ đạo này, vì vậy vùng phủ song (footprint) của vệ tinh trên bề mặt trái đất sẽ theo các hướng đông - tây và bắc - nam. Cụ thể hơn, loại "đối tượng vị trí cực" này có thể sử dụng để mô hình sự chuyển động của bất kì một quỹ đạo tròn nào trong một mặt phẳng cố định. Chúng ta dùng thuật ngữ " cực" ở đây bởi vì sau này chúng ta dùng các vệ tinh để mô hình chòm sao vệ tinh quỹ đạo cực.

  Quỹ đạo vệ tinh được đặc trưng bởi 6 tham số: *độ cao so với mặt nước biển, trục chính, độ lệch tâm, góc lên đúng của điểm lên (right ascension of ascending Node), độ dốc và thời gian cận điểm trái đất nhất.* Vệ tinh quỹ đạo cực trong ns chủ yếu có quỹ đạo tròn, vì vậy chúng ta đơn giản hoá tính chất của quỹ đạo chỉ còn 3 tham số: đó là *độ cao so với mặt nước biển, độ dốc và kinh độ*, với tham số thứ tư *alpha* đặc trưng cho vị trí ban đầu của vệ tinh trên quỹ đạo, như miêu tả dưới đây.

  Độ cao so với mặt nước biển tính bằng km. **Độ dốc** nằm trong khoảng [0; 180] độ, với 90 tương ứng là quỹ đạo thuận và góc lớn hơn 90 tương ứng với quỹ đạo ngược. Điểm lên (ascending node) tương ứng với vị trí của vệ tinh mà footprint của vệ tinh ngang qua xích đạo khi đang di chuyển từ nam sang bắc. Trong chương trình mô phỏng này, kinh độ của điểm lên là kinh độ trên trái đất, tại đó điểm đáy (nadir point) của vệ tinh đi qua đường xích đạo khi di chuyển từ nam tới bắc [1]. *Kinh độ của điểm lên* nằm trong khoảng [-180; 180] độ.

[1] ...

Tham số thứ tư, **alpha**, xác định vị trí ban đầu của vệ tinh dọc theo quỹ đạo, bắt đầu từ điểm lên. Ví dụ, giá trị alpha 180 độ chỉ ra rằng vệ tinh đang ở phí trên đương xích đạo, di chuyển từ bắc đến nam. Alpha có giá trị từ [0; 360] độ. Cuối cùng, thông số thứ 5, **mặt phẳng**, được định nghĩa khi tạo ra nút vệ tinh cực. Tất cả các vệ tinh trong cùng một mặt phẳng được đưa vào cùng một chỉ mục mặt phẳng. Bộ tạo nút mang được dùng để tạo ra một vệ tinh cực với đối tượng vị trí gắn vào như sau:

```
$ns node-config -satNodeType polar n
(câu lệnh cấu hình nút mạng khác ở đây )
set n1 [$ns node]
$n1 set-position $alt $inc $lon $alpha $plane
```

14.2.2. Đường truyền vệ tinh

Mỗi nút vệ tinh có một hoặc nhiều khối giao diện mạng vệ tinh, mà mỗi kênh sẽ kết nối đến đối tượng của lớp vật lý trong khối đó. Hình 14.3 minh họa các thành phần chính này. Đường truyền vệ tinh khác đường truyền vô tuyền ns ở hai khía cạnh chính:

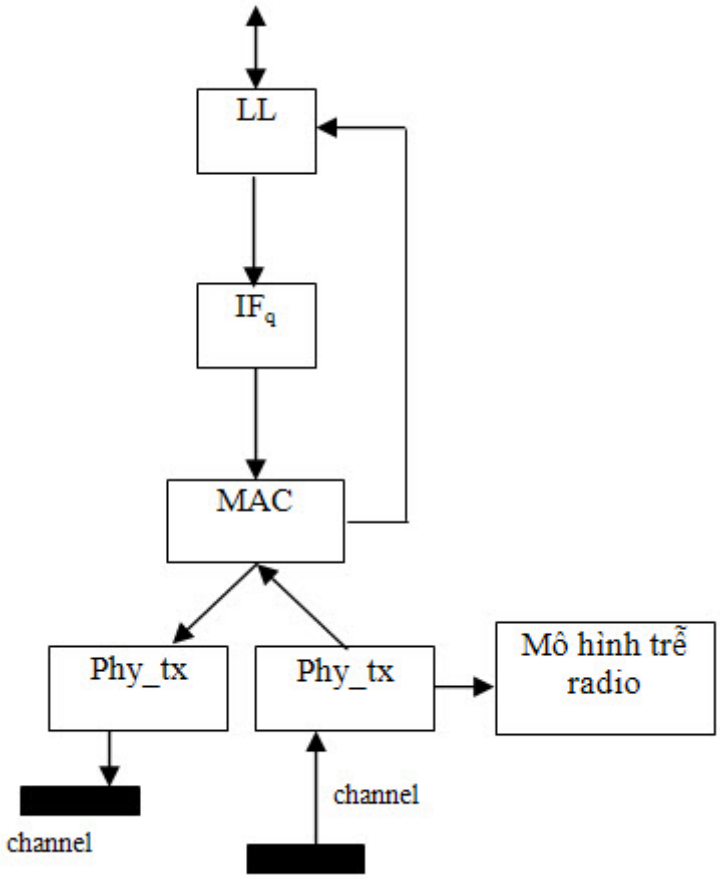
- Giao diện truyền và nhận tín hiệu phải được kết nối đến các kênh khác nhau.
- Không có chức năng ARP(giao thức phân giải địa chỉ). Mô hình truyền sóng radio là một mô hình cho người dùng thêm các mô hình lỗi chi tiết nếu có. Các mã lệnh hiện tại không dùng mô hình truyền sóng.

Giao diện mạng có thể thêm vào theo câu lệnh sau của `Class Node/SatNode::`

```
$node add-interface $type $ll $qtype $qlim $mac $mac_bw $phy
```

Câu lệnh `add-interface` trả lại kết quả của giá trị chỉ mục mà có thể được dùng để truy cập vào khối giao diện mạng sau này trong quá trình mô phỏng. Theo quy ước, giao diện đầu tiên được tạo ra trên một nút mạng sẽ gắn vào kênh truyền tuyến lên hoặc tuyến xuống của vệ tinh hoặc thiết bị đầu cuối. Tham số sau phải được đưa vào:

- **type:** Chỉ ra loại kênh truyền, có các loại sau: geo hoặc polar dùng cho các đường truyền từ một thiết bị đầu cuối đến GEO hoặc đến vệ tinh cực tương ứng, gsl hoặc gsl-repeater dùng cho đường truyền từ vệ tinh đến một thiết bị đầu cuối, intraplane, interplane và crossseam ISLs. Trường type này được dùng bên trongchuwowng trình mô phỏng để xác định rõ các loại đường truyền khác nhau, nhưng về mặt cấu trúc thì chúng cũng tương đương nhau.
- **ll:** Loại lớp đường truyền (chỉ có duy nhất một lớp LL/Sat).



Hình 14.3: Các thành phần chính của giao diện mạng vệ tinh

- **qtype:** Loại hàng đợi (ví dụ lớp `Queue/DropTail`). Bất kì loại hàng đợi nào cũng có thể sử dụng, tuy nhiên nếu các tham số thêm vào lớn hơn độ dài của hàng đợi yêu cầu thì sau đó instproc có thể cần được sửa đổi để bao gồm nhiều đối số (agruments) hơn nữa.
- **qlim:** Độ dài của hàng đợi giao diện, tình theo số gói dữ liệu.
- **mac:** Loại giao thức MAC. Hiện tại, có hai loại được định nghĩa: lớp `Mac/Sat-` cho đường truyền chỉ có một máy thu (tức là giả thuyết không có xung đột xảy ra) và lớp `Mac/Sat/UnslottedAloha-` sự triển khai phương pháp ALOHA không phân khe .
- **mac_bw:** Băng thông của đường truyền được đặt bởi tham số này, nó điều khiển thời gian truyền dẫn: Giao thức MAC gửi đi nhanh như thế nào. Kích cỡ gói dữ liệu sử dụng để tính toán thời gian truyền là tổng giá trị của hàm size() trong phần mào đầu gói dữ liệu chung và `LINK_HDRSIZE` là chỉ kích cỡ của bất kì phần mào đầu của lớp liên kết dữ liệu nào. Giá trị mặc định cho `LINK_HDRSIZE` là 16 bytes(giá trị mặc định trong thư viện satlink.h). Thời gian truyền được mã hóa vào phần mào đấu gói

dữ liệu để dùng cho giao thức MAC nhận (để mô phỏng đợi cả gói dữ liệu đến).

- **phy:** Lớp vật lý- hiện tại chỉ có hai lớp vật lý (lớp `Phy/Sat` và lớp Phy/Repeater) được định nghĩa. Lớp `Phy/Sat` chuyển thông tin lên và xuống các khối- như mã hóa trong mạng không dây đã miêu tả ở chương trước, mô hình truyền dẫn radio được gắn vào điểm này. Lớp Phy/Repeater truyền các gói dữ liệu nhận được theo kiểu đường ống trên giao diện nhận truyền thẳng tới giao diện truyền.

Một ISL có thể thêm vào giữa hai nút mạng dùng câu lệnh sau:

```
$ns add-isl $ltype $node1 $node2 $bw $qtype $qlim
```

Câu lệnh này tạo ra hai kênh (loại `Channel/Sat`), và giao diện mạng thích hợp ở cả hai nút mạng, và gắn các kênh truyền vào giao diện mạng. Băng thông của đường truyền được đặt là bw. Lọai đường truyền (`ltype`) phải được thiết lập là loại `intraplane, interplane, hoặc crossseam`.

Một GSL bao gồm việc them vào các giao diện mạng và một kênh truyền trên vệ tinh (điều này thông thường được hoàn thành dùng phương pháp phủ ngoài được mô tả ở đoạn sau), và sau đó định nghĩa giao diện phù hợp trên nút ở mặt đất và gắn chúng vào đường truyền vệ tinh, như sau:

```
$node add-gsl $type $ll $qtype $qlim $mac $bw_up $phy n
[$node_satellite set downlink_] [$node_satellite set uplink_]
```

Ở đây, tham số type phải là geo hoặc polar, và chúng ta thường dùng biến `downlink_` và `uplink_` của vệ tinh, do đó, đường truyền tuyến lên và tuyến xuống của vệ tinh phải được tạo ra trước khi gọi câu lệnh này. Mặc định rằng bộ tạo nút đối với các nút mạng vệ tinh (đã mô tả ở phần 5.3 cuốn *The ns Manual-The VINT Project*) sẽ tạo ra các loại nút mạng theo loại vệ tinh GEO hay polar , giao diện đường truyền tuyến lên tuyến xuống của chúng, tạo ra và gắn vào kênh đường truyền, dựa trên việc lựa chọn giao diện được quy định.

14.2.3. Chuyển giao

Mô hình hóa sự chuyển giao vệ tinh là thành phần chủ chốt của mô phỏng mạng vệ tinh LEO. Rất khó để dự đoán chính xác chuyển giao diễn ra như thế nào trong một hệ thống LEO thực bởi vì vấn đề này chưa được quan tâm kĩ trong các tài liệu hiện có. Trong những chức năng mở rộng của vệ tinh, chúng ta thiết lập các tiêu chí nhất định cho việc chuyển giao và cho phép các nút mạng điều khiển trạng thái đòi hỏi chuyển giao một cách độc lập. Một cách thức khác có thể thực hiện là các sự kiện chuyển giao được đồng bộ trong suốt

quá trình mô phỏng - không khó để thay đổi chương trình mô phỏng hoạt động theo cách như vậy.

Không có chuyển giao kênh truyền nào liên quan đến vệ tinh địa tĩnh. Nhưng có hai loại kênh truyền đến vệ tinh quỹ đạo cực cần được chuyển giao: kênh truyền GSLs trong vệ tinh quỹ đạo cực và các kênh truyền crossbeam ISLs. Các kênh interplance không chuyển giao nhưng chúng không hoạt động ở các vĩ độ cao như chúng ta miêu tả ở dưới đây.

Mỗi thiết bị đầu cuối kết nối đến vệ tinh quỹ đạo cực khởi động bộ định thời, nếu quá thời hạn sẽ dẫn đến Quản lý CHuyển giao `(HandoffManager)` kiểm tra vệ tinh này có ở nằm trong vùng hoạt động của thiết bị đầu cuối hay không. Khi đó thì quản lý chuyển giao sẽ tách thiết bị đầu cuối từ đường truyền của vệ tinh, và tìm kiếm dọc theo đường truyền số trạm vệ tinh cho các vệ tinh khác có thể. Đầu tiên, vệ tinh "tiếp theo" trong mặt phẳng quỹ đạo hiện tại được kiểm tra, vị trí đến vệ tinh này sẽ được lưu trong đối tượng Position của mỗi trạm vệ tinh cực, và được đặt thông số trong suốt cấu hình mô phỏng dùng `Node/SatNode` câu lệch: `"$node set_next $next_node''`. Nếu vệ tinh tiếp theo không phù hợp, quản lý chuyển giao sẽ tìm kiếm các vệ tinh còn lại. Nếu tìm thấy một vệ tinh cực thích hợp thì nó sẽ kết nối giao diện mạng của nó đến kênh truyền tuyến lên và tuyến xuống của vệ tinh, và bắt đầu bộ đếm thời gian chuyển giao. Nếu không tìm thấy vệ tinh nào, nó bắt đầu bộ đếm giờ và lại thử lại sau đó. Nếu bất kì sự thay đổi đường kết nối nào diễn ra thì bộ phận định tuyến sẽ nhận được thông báo.

Vùng hoạt động và bộ đếm khoảng thời gian chuyển giao được đặt thông số qua OTcl:

```
HandoffManager/Term set elevation_mask_ 10; # theo độ
HandoffManager/Term set term_handoff_int_ 10; # theo giây
```

Thêm vào đó, chuyển giao có thể được thực hiện một cách ngẫu nhiên để tránh nhiễu pha bằng cách đặt giá trị biến sau:

```
HandoffManager set handoff_randomization_ 0; # 0 là false, 1 là true
```

Nếu biến `handoff_randomization` là true, thì khoảng thời gian chuyển giao tiếp theo là một biến ngẫu nhiên được chọn từ phân bố đều trong dải *(0:5*_term_handoff_int_, 1.5*term_handoff_int_).*

Crossseam ISLs là loại ISLs duy nhất có thực hiện chuyển giao. Tiêu chuẩn cho chuyển giao Crossseam ISLs ở đó có tồn tại một vệ tính trong mặt phẳng gần đó không và vệ tinh này gần với vệ tinh đã cho hơn mặt phẳng mà nó đang đc nối tới. Một lần nữa, một bộ định thời chuyển giao chạy trong bộ quản lí chuyển giao ở vệ tinh cực quyết định khi nào thì chòm sao vệ tinh được duyệt cho các cơ hội chuyển giao. Các chuyển giao kiểu Crossseam ISL được khởi đầu bởi các vệ tinh trong mặt phẳng có số thứ tự thấp hơn 2. Do đó, có khả

năng sinh ra một điều kiện tạm thời mà một vệ tinh cực có 2 crossseam ISLs trong đó (tới các vệ tính khác nhau). Khoảng thời gian giữa hai lần chuyển giao vệ tinh có thể được đặt lại từ OTcl và cũng có thể được ngẫu nhiên hóa.

```
HandoffManager/Sat set sat_handoff_int_ 10; # theo giây
```

Interplane và crossseam ISLs không hoạt động khi vệ tinh ở gần điểm cực, bởi vì yêu cầu vị trí cho đường truyền rất khắt khe bởi vì vệ tinh di chuyển lại đến vệ tinh khác. Việc ngắt những đường kết nối này được giám sát bở tham số sau:

```
HandoffManager/Sat set latitude_threshold_ 70; # theo độ
```

Giá trị của tham số này trong ví dụ mô tả chỉ là dự đoán, giá trị chính xác là độc lập nhờ vào phần cứng của vệ tinh. Khối quản lý chuyển giao kiểm tra vĩ độ và quan sát thời gian trễ chuyển giao vệ tinh của nó, nếu một hoặc cả các vệ tinh khác nằm trên vĩ độ `latitude_threshold_` tính bằng độ (phía bắc hoặc phía nam) thì đường truyền không hoạt động cho đến khi các vệ tinh đều nằm dưới ngưỡng này.

Cuối cùng, nếu crossseam ISLs tồn tại, sẽ xảy ra tình huống chắc chắn mà trong đó các vệ tinh kéo đến một vệ tinh khác quá gần ở chính giữa vĩ độ, (nếu quỹ đạo không gần với quỹ đạo cực ban đầu). Chúng ta kiểm tra hiện tượng xảy ra cho quỹ đạo của hai vệ tinh bị chồng lấn lên nhau với tham số sau:

```
HandoffManager/Sat set longitude_threshold_ 10; # theo độ
```

Một lần nữa khẳng định lại, giá trị cho tham số trong ví dụ này chỉ là suy đoán, nếu có hai vệ tinh gần kinh độ nhau hơn ngưỡng kinh độ cho trước `longitude_threshold_` thì đường truyền giữa chúng bị ngắt. Tham số này không thể đặt mặc định (đặt là 0), tất cả các mặc định cho các biến thay đổi liên quan giữa các vệ tinh có thể tìm thấy trong thư mục `~ns/tcl/lib/ns-sat.tcl`.

14.2.4. Định tuyến

Định tuyến trong mạng vệ tinh tương tự với định tuyến trong ns2 đã có ngoại trừ việc nó được thực hiện hoàn toàn bằng C++.

Với mỗi sự thay đổi của topo mạng, một phần tử định tuyến trung tâm sẽ xác định tôp mạng toàn cục, sẽ tính toán các tuyền mới cho mọi nút mạng và sử dụng các tuyến này để dựng lên một bảng chuyển tiếp (forwarding table) cho mỗi nút mạng. Hiện nay, bảng chuyển tiếp này được duy trì bởi một phần tử định tuyến trên mỗi nút mạng, và các gói dữ liệu không gửi đến phần tử nào trên nút mạng thì sẽ được gửi mặc định đến phần tử định tuyến. Ở đích nhận tin có một đường đi, bảng định hướng chứa điểm đầu đường truyền đi tương ứng. Người dùng được cảnh báo rằng loại định tuyến trung tâm này có thể dẫn đến vi phạm quan hệ nhân quả.

Định tuyến có trong lớp `SatRouteObject` và được tạo ra và thi hành bằng câu lệnh OTcl sau:

```
set satrouteobject_ [new SatRouteObject]
$satrouteobject_ compute_routes
```

trong đó cuộc gọi đến `compute_routes` được biểu diễn sau tất cả các kết nối và nút mạng trong mô phỏng đã được minh họa. Giống như `Scheduler`, có một trường hợp của `SatRouteObject` trong mô phỏng, và nó được chấp nhận bởi một biến trung gian ví dụ trong C++. Xét ví dụ, cuộc gọi để tính lại đường đi sau một topo thay đổi là:

```
SatRouteObject::instance().recompute();
```

Thay vì sử dụng định tuyến trung tâm hiện nay thì thiết kế của phần tử định tuyến trên mỗi nút mạng được thực hiện với cách thức định tuyến phân bố. Các gói dữ liệu dùng cho định tuyến có thể gửi đến cổng 255 của mỗi nút mạng. Chìa khóa để định tuyến phân bố làm việc chính xác là cho phần tử định tuyến có quyền quyết định đường truyền gói dữ liệu đến. Điều này được thực thi bởi lớp đối tượng `NetworkInterface` trong mỗi liên kết, mà chỉ có các nhãn đường truyền trên mỗi gói đến. Hàm trợ giúp `NsObject*intf_to_target` (int label) có thể sử dụng để trả lại kết quả phần đầu của đường truyền tương ứng với mỗi nhãn đang có. Việc sử dụng các phần tử định tuyến tương đương với các phần mở rộng của phương thức di động.

Việc tính đường đi định tuyến ngắn nhất dùng trễ truyền hiện nay như là một tham số định tuyến. Có thể tính toán các tuyến bằng cách sử dụng việc đếm các kết nối trên tuyền (hop count) và không sử dụng trễ truyền dẫn. Để thực hiện điều này, ta phải đặt biến mặc định sau đây là "false":

```
SatRouteObject set metric_delay_ "true"
```

Với topo mạng lớn như ví dụ Teledesic ở trên thì mã định tuyến trung tâm sẽ làm chậm thời gian xử lý bởi vì nó thực thi thuật toán tìm đường ngắn nhất trên tất cả các cặp mỗi topo thay đổi thậm chí cả các topo không có dữ liệu truyền. Để tăng tốc mô phỏng khi không có nhiều dữ liệu để truyền nhưng có nhiều vệ tinh và ISLs, có thể sử dụng tính toán tuyến theo sự xuất hiện của dữ liệu (data driven) chứ không tính toán tuyến theo sự xuất hiện của chuyển giao (handoff driven). Với việc tính toán tuyến theo sự xuất hiện của dữ liệu, đường đi được tính chỉ khi có một gói dữ liệu gửi đi, và hơn nữa, thuật toán đường đi ngắn nhất từ nguồn đơn (chỉ cho nút mạng với một gói dữ liệu gửi đi) được thực thi thay cho thuật toán tìm đường ngắn nhất của tất cả các cặp. Biến OTcl có thể cấu hình như sau (đặt là false mặc định) :

```
SatRouteObject set data_driven_computation_ "false"
```

14.2.5. Hỗ trợ bám vết

Đối tượng đặc biệt SatTrace (lớp SatTrace được tách từ lớp Trace) được dùng để ghi lại tọa độ địa lý là kinh độ và vĩ độ của nút mạng đang ghi dấu vết này (trong trường hợp một nút mạng vệ tinh, vĩ độ và kinh độ tương ứng với điểm thấp nhất của vệ tinh). Ví dụ, một gói dữ liệu trên một đường truyền từ nút mạng 66 đến nút mạng 26 có thể được ghi lại như sau:

```
+ 1.0000 66 26 cbr 210 ------- 0 66.0 67.0 0 0
```

nhưng trong mô phỏng vệ tinh, thông tin vị trí được gắn vào như sau:

```
+ 1.0000 66 26 cbr 210 ------- 0 66.0 67.0 0 0 37.90 -122.30 48.90 -120.94.
```

Trong trường hợp này, nút mạng 66 ở vĩ độ 37.900^0, kinh độ -122.300^0, trong khi nút mạng 26 là một vệ tinh LEO mà các điểm thuộc vệ tinh là ở vĩ độ 48.900^0, và kinh độ -120.300^0 (vĩ độ âm tương ứng với hướng nam, trong khi kinh độ âm tương ứng với hương tây). Thêm vào đó lớp `Trace/Sat/Error` tìm vết bất kì gói dữ liệu nào bị lỗi thông qua một mô hình lỗi. Tìm vết các lỗi truy cập gói giảm đi do lỗi sau, ví dụ:

```
e 1.2404 12 13 cbr 210 ------- 0 12.0 13.0 0 0 -0.00 10.20 -0.00 -10.00
```

Có thể xảy ra ở một nút mạng sinh ra một gói mà không thể truyền đi (như trong `sat-mixed.tcl`). Điều này sẽ chỉ ra sự suy giảm trong file truy vết với trường đích đến đặt là –2, và đến tận –999.00:

```
d 848.0000 14 -2 cbr 210 ------- 1 14.0 15.0 6 21 0.00 10.00 -999.00 -999.00.
```

Điều này chỉ ra rằng nút mạng 14 đang cố gắng gửi một gói dữ liệu đến nút mạng 15, nhưng không thể tìm thấy đường đi thuận lợi nhất.Để dễ dàng tìm vết tất cả các đường truyền vệ tinh trong mô phỏng dùng câu lệnh *before* sau đây để minh họa cho các nút mạng và đường truyền:

```
set f [open out.tr w]
$ns trace-all $f
```

Sau đó dùng dòng lệnh sau đây sau khi tạo ra tất cả các nút mạng và đường truyền (chèn tất cả kiểu lỗi vào nếu có thể), để dễ dàng truy vết các đường truyền vệ tinh:

```
$ns trace-all-satlinks $f
```

Đặc biệt, điều này sẽ đặt sự bám vết xung quanh các hàng đợi lớp liên kết dữ liệu trong các đường truyền vệ tinh sẽ nhận được truy vết giữa lớp MAC và lớp liên kết dữ liệu cho gói dữ liệu nhận được. Để có khả năng truy vết chỉ trên một đường truyền xác định trên một nút mạng xác định thì chỉ có thể dùng câu lệnh sau:

```
$node trace-inlink-queue $f $i $node trace-outlink-queue $f $i
```

trong đó i là chỉ mục giao diện được truy vết. Sự bổ sung của đối tượng truy vết vệ tinh có

thể tìm thấy trong thư mục *...ns/tcl/lib/ns-sat.tcl và ...ns/sattrace.cc,h*

14.2.6. Các mô hình lỗi

Các mô hình lỗi trong ns đã được trình bày ở chương trước. Những mô hình lỗi này có thể được thiết lập đê tạo ra các gói dữ liệu bị lỗi theo các hàm phân bố xác suất khác nhau. Những mô hình lỗi này đơn giản và không cần phải tương ứng với các lỗi có thể có trên kênh vệ tinh thực (cụ thể như kênh LEO). Người dùng được tự do thiết lập các mô hình kiểu lỗi phức tạp hơn mà giống với môi trường vệ tinh cụ thể hơn. Mã sau đưa ra ví dụ cách thức thêm một mô hình lỗi vào một đường truyền:

```
set em_ [new ErrorModel]
$em_ unit pkt
$em_ set rate_ 0.02
$em_ ranvar [new RandomVariable/Uniform]
$node interface-errormodel $em_
```

Nó sẽ thêm một mô hình lỗi vào đường nhận tín hiệu của giao diện đầu tiên được tạo trên nút mạng $node (cụ thể là giữa lớp MAC và lớp liên kết dữ liệu)- giao diện đầu tiên nói chung tương ứng với giao diện đường truyền tuyến lên và tuyến xuống cho vệ tinh hoặc cho thiết bị đầu cuối (nếu chỉ có một đường truyền tuyến lên hoặc đường truyền tuyến xuống tồn tại). Để thêm vào mô hình lỗi này vào một stack khác (đánh chỉ mục bởi i) thì dùng mã lệnh:

```
$node interface-errormodel $em_ $i
```

14.2.7. Các lựa chọn cấu hình khác

Đưa ra cấu hình ban đầu của vệ tinh theo lý thuyết từ lần thứ 0, rất dễ để bắt đầu cấu hình vệ tinh từ bất cứ điểm tùy ý bằng tham số `time_advance_parameter` (điều này thực sự hữu ích cho mô phỏng LEO). Trong suốt quá trình chạy mô phỏng, đặt vị trí của đối tượng đến vị trí thời gian:`Scheduler::instance().clock + time_advance_ seconds.`

```
Position/Sat set time_advance_ 0; # tính bằng giây (seconds)
```

14.2.8. Mô phỏng hỗ trợ NAM

NAM không được hỗ trợ trong mô phỏng này.

14.2.9. Tích hợp với mã hữu tuyến và vô tuyến

Phần trợ giúp đã được thêm vào để kết nối các nút hữu tuyến (wires) dựa trên OTcl truyền thống với các nút mạng vệ tinh. Phần này mô tả về khả năng và hạn chế của mã này

Các mã lệnh vệ tinh (và mã vô tuyến) thông thường hực hiện định tuyến trong C++, trong khi mã ns gốc thì dùng đan xen giữa OTcl và C++. Do các lý do về tính tương thích nghịch (backward) thì khó để tích hợp đủ cho cả mã hữu tuyến và mã vô tuyến. Dêt hực hiện tích hợp mã hữu tuyến và mã vô tuyến, ns tạo ra một nút cổng đặc biệt (gọi là trạm cơ sở- base station) để dùng định tuyến phân cấp và định vị nút trạm gốc đơn trong mạng không dây Bởi vì định tuyến không được tích hợp hoàn toàn, cho nên topo của mạng mô phỏng bị giới hạn chỉ có một nút cổng trên mỗi mạng vô tuyền thành phần (subnet) không dây (ví dụ một gói dữ liệu không thể đi vào mạng vô tuyến từ một cổng vật lý hữu tuyến và chuyển đi thông qua một cổng khác). Tích hợp mã hữu tuyến và vệ tinh được thức hiện theo cách khác. Bằng cách chọn chức năng cấu hình nút mạng `$ns node-config -wiredRouting ON` định tuyến C++ trong mã vệ tinh được tắt đi,và thay vào đó, tất cả các thay đổi của topo vệ tinh chuyển thành theo mã OTcl.

Kết quả là, mảng `link_array` trong OTcl được thao tác theo các topo thay đổi, và định tuyến cơ bản OTcl- có thể xảy ra. Giá phải trả cho việc này là cần nhiều thời gian thi hành cho các mô phỏng lớn hơn (như là Teledesic), nhưng với các mô phỏng nhỏ hơn, sự khác biệt là không đáng kể.

Ví dụ chi tiết cách dùng của chức năng mới này được trình bày trong thư mục *...ns/tcl/ex /sat-wired.tcl*, và một tính năng kiểm tra tương tự trong vệ tinh kiểm tra bài tập phù hợp với mã này. Thêm vào đó, tất cả các ví dụ vệ tinh trong thư mục *...ns/tcl/ex* có thể chuyển đổi thành định tuyến OTcl bằng cách dùng chức năng `$ns node-config -wiredRouting ON`. Tuy nhiên, có một vài chú ý:

- Chức năng định tuyến hữu tuyến cho vệ tinh đã chỉ được kiểm tra với định tuyến tĩnh (được mặc định sẵn): `$ns rtProto Static`. Mã lệnh này kích hoạt cập nhật bảng định tuyến toàn cục dựa trên bất cứ thay đổi topo vệ tinh nào.
- Chức năng `data_driven_computation_` không thể đặt là "true" khi định tuyến hữu tuyến là ON.
- Trong file truy vết, khi một gói dữ liệu bị mất bit do "không có tuyến đến host" (như khi có thay đổi một topo mạng), truy vết tìm một bít khác nhau phụ thuộc vào wiredRouting đang để ON hay OFF. Trong trường hợp thông thường, chỉ có một dòng trên mỗi drop, với nhãn đích là "–2". Trường hợp sau có ba trường hợp có thể

xảy ra “enque "+", deque "-", and drop "d" tương ứng đến một gói dữ liệu giống nhau, và đích là “-1”

- Trong các trường hợp ít xảy ra, có thể có các bản tin cảnh báo trong suốt quá trình thực hiện biểu thị việc "nằm ngoài phạm vi của nút mạng". Điều này có thể xảy ra nếu một nút mạng bị đứt kết nối trong topo và sau đó một nút mạng khác gửi một gói dữ liệu đến đó. Ví dụ, `wiredRouting` trong file *...ns/tcl/ex/sat-mixed.tcl*. Điều này xảy ra bởi vì bảng định tuyến có kích thước thay đổi phụ thuộc vào sự thay đổi của topo mạng. Nếu một nút mạng bị đứt kết nối nó sẽ không có bất cứ sự tiếp nhận được chèn vào bảng định tuyến.(và do đó bảng định tuyến không thể tăng lên để phù hợp với số nút mạng). Cảnh báo này không nên tác động đến đầu ra truy vết thực.
- Không có sự tương tác nào với các mã vô tuyến hoặc IP di động.

14.2.10. Các tập lệnh ví dụ

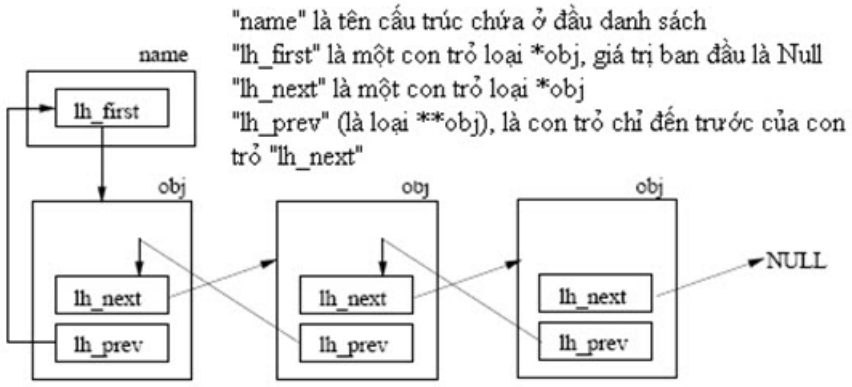
Các tập lệnh ví dụ này có thể tìm thấy trong thư mục *...ns/tcl/ex*, bao gồm:

- `sat-mixed.tcl`: Mô phỏng cả hai loại vệ tinh cực và vệ tinh địa tĩnh.
- `sat-wired.tcl`: tương tự như tập lệnh trước, nhưng nó chỉ ra cách kết nối các nút mạng hữu tuyến trong mô phỏng vệ tinh.
- `sat-repeater.tcl`: biểu thị cách sử dụng đơn giản của vệ tinh địa tĩnh loại chuyển tiếp, bao gồm cả các mô hình lỗi.
- `sat-aloha.tcl`: mô phỏng một trăm thiết bị đầu cuối trong cấu hình vệ tinh VSAT dùng giao thức MAC aloha không khe cắm với một vệ tinh địa tĩnh chuyển tiếp. Thiết bị đầu cuối lắng nghe kênh truyền của chúng (sau một khoảng trễ) và nếu nó không nhận được gói dữ liệu của nó với mỗi khoảng thời gian trễ, nó sẽ thực hiện cơ chế back-off theo hàm mũ và sau đó truyền lại gói dữ liệu.Tồn tại ba hàm khác nhau: `basic, basic_tracing, và poisson`, các hàm khác nhau này đã được miêu tả ở đầu mỗi chú thích của tập lệnh. `sat-iridium.tcl`: Mô phỏng chòm sau LEO với tham biến tương tự chòm sao của Iridium (tập lệnh hỗ trợ là `sat-iridium-links.tcl, sat-iridium-linkswithcross.tcl, và sat-iridium-nodes.tcl`)
- `_at-teledesic.tcl`: Mô phỏng chòm sao LEO có kênh truyền rộng với tham số tương tự những đề xuất cho chòm sao vệ tinh 288 Teledesic (cũng hỗ trợ các tập lệnh là `sat-teledesic-links.tcl` và `sat-teledesic-nodes.tcl`). Thêm vào đó, có một tập lệnh để kiểm tra tính phù hợp nhiều cấu hình một cách đông thời, có thể tìm thấy ở thư mục *...ns/tcl/test/testsuite-sat.tcl*.

14.3. Thực hiện phần mở rộng mô phỏng vệ tinh

Mã chương trình của phần mở rộng mô phỏng vệ tinh có thể tìm thấy ở thư mục `...ns/sat.h, sathandoff.cc,h, satlink.cc,h, satnode.cc,h, satposition.cc,h, satroute.cc,h, sattrace.cc,h, and ns/tcl/lib/ns-sat.tcl`. Hầu hết chúng được thực hiện trong C++.

Trong phần này, chúng ta tập trung vào bốn thành phần chính của sự thực hiện gồm: tên, danh sách đã liên kết, kiến trúc nút và "detailed look" tại kiến trúc liên kết vệ tinh



Hình 14.4: Bổ sung danh sách đường truyền trong ns

14.3.1. Sử dụng các danh sách liên kết

Các danh sách liên kết sau được sử dụng trong quá trình thực hiện:

- `class Node` duy trì một danh sách tĩnh của tất cả các đối tượng của lớp Node trong mô phỏng. Biến `Node::nodehead_` lưu phần tử đầu tiên của danh sách. Danh sách lien kết này được sử dụng cho định tuyến trung tâm, tìm vệ tinh để chuyển giao, hoặc truy vết.

- `class Node` chứa danh sách các đường truyền vệ tinh trên mỗi nút mạng. Cụ thể, danh sách các đối tượng của lớp LinkHead. Biến `linklisthead_` lưu phần tử đầu tiên của danh sách. Danh sách lien kết của LinkHead được dùng để kiểm tra các đường truyền chuyển giao, và khôi phục các topo mạng liền kề.

- `class Channel` chứa danh sách các đối tượng của lớp Phy trên mỗi kênh. Phần đầu của danh sách được lưu trong biến có tên là `if_head_`. Danh sách này được sử dụng để xác định giao diện trên kênh để nhận một bản sao của gói tin.

Hình 14.4 cung cấp một giản đồ liệt kê các cách thức tổ chức kết nối được sắp xếp. Mỗi đối tượng trong danh sách liệt kê được kết nối qua "LINK_ENTRY" có thành viên được bảo vệ của lớp. Các lối vào này chứa con trỏ để chỉ đến các thành phần trong danh sách và một con trỏ nữa để đánh địa chỉ của con trỏ "next" ở phía trước ở đối tượng có trước.

14.3.2. Cấu trúc nút mạng

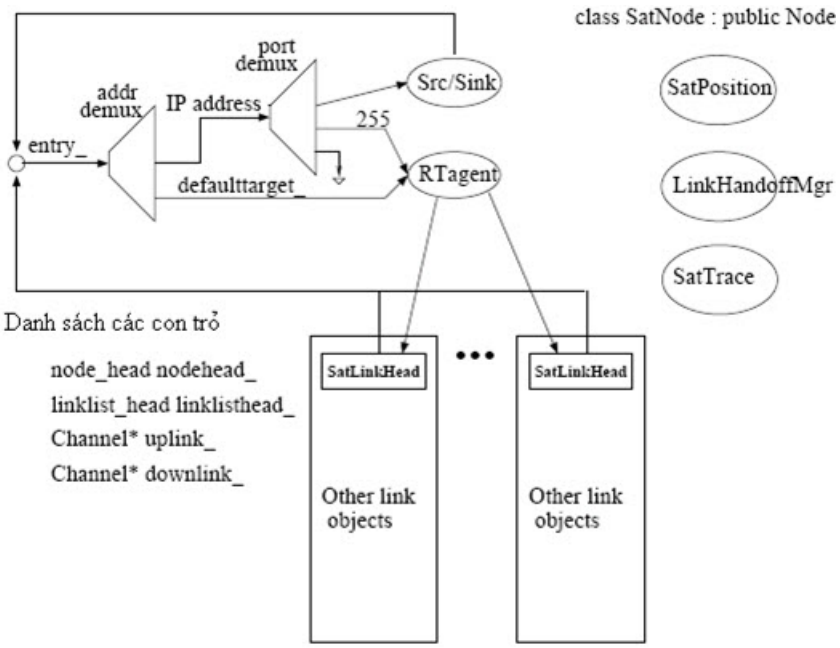
Hình 14.5 là một giản đồ liệt kê các thành phần thiết bị chính của một `SatNode`. Cấu trúc này có sự tương đồng với `MobileNode` trong phần mở rộng mạng không dây, nhưng cũng có một vài khác biệt. Giống như các nút mạng ns, SatNode có một điểm "entry" đến một loạt các `"classifiers". "Classifier"` cho địa chỉ bao gồm một bảng định tuyến cho gói dữ liệu truyền thẳng đến nút ngoài mạng. Nhưng vì định tuyến OTcl không được sử dụng, tất cả các gói dữ liệu không dành riêng cho nút mạng này (và do đó sẽ truyền thẳng đến phân loại cổng), sẽ được gửi đến đích mặc định. Gói dữ liệu dành cho nút mạng cổng 255 được phân loại như gói dữ liệu định tuyến và truyền thẳng đến phần tử định tuyến.

Mỗi nút mạng chứa một hoặc nhiều "network stack" bao gồm một `SatLinkHead` chung tại điểm "entry" của đường truyền. `SatLinkHead` được dùng như một API để lấy một đối tượng khác trong cấu trúc đường truyền, vì vậy nó chứa một số con trỏ (mặc dù ở đây API chưa hoàn thành). Gói dữ liệu khi rời khỏi "network stack" thì có trường `iface_` trong phần tiêu đề của gói dữ liệu được mã với chỉ số `NetworkInterface` duy nhất tương ứng với mỗi đường truyền. Giá trị này có thể sử dụng để hỗ trợ định tuyến phân tán như miêu tả ở dưới.

Phần tử định tuyến cơ bản là `SatRouteAgent`; nó có thể sử dụng dung với định tuyến trung tâm. `SatRouteAgent` chứa bảng định hướng thẳng đến phân tích địa chỉ của gói dữ liệu đến đích LinkHead cụ thể- nó là một nhiệm vụ của `SatRouteObject` để cư trú ở bảng này chính xác. SatRouteAgent thêm các trường nhất định ở tiêu đề của gói dữ liệu và sau đó gửi gói dữ liệu xuống đến đường truyền thích hợp.

Để thực hiện một giao thức định tuyến phân tán, một `SatRouteAgent` mới được định nghĩa, điều này sẽ học về topo bằng cách đánh dấu chỉ số giao diện trong mỗi gói dữ liệu như nó đến ngăn xếp, một hàm trợ giúp trên nút mạng `intf_to_target()` cho phép nó phân tích một giá trị chỉ số đến một LinkHead cụ thể. Có nhiều con trỏ đến ba đối tượng thêm vào trong một SatNode. Đầu tiên, mỗi SatNode chứa một `LinkHandoffMgr` điều khiển để tận dụng cơ hội chuyển giao và ngang hàng chuyển giao. Nút mạng vệ tinh và nút mạng thiết bị đầu cuối mỗi cái có phiên bản `LinkHandoffMgr` thiết kế riêng.

Cuối cùng, số các con trỏ đến đối tượng chứa trong một SatNode. Chúng ta đã thảo luận `linklisthead_` và `nodehead_` trong phần trên. Con trỏ `uplink_` và `downlink_` được dùng một cách thuận tiện giả định rằng trong hầu hết các mô phỏng, một vệ tinh hoặc một



Hình 14.5: Cấu trúc của lớp SatNode

thiết bị đầu cuối chỉ có một kênh tuyến lên và một kênh tuyến xuống.

14.3.3. Các chi tiết về đường truyền vệ tinh

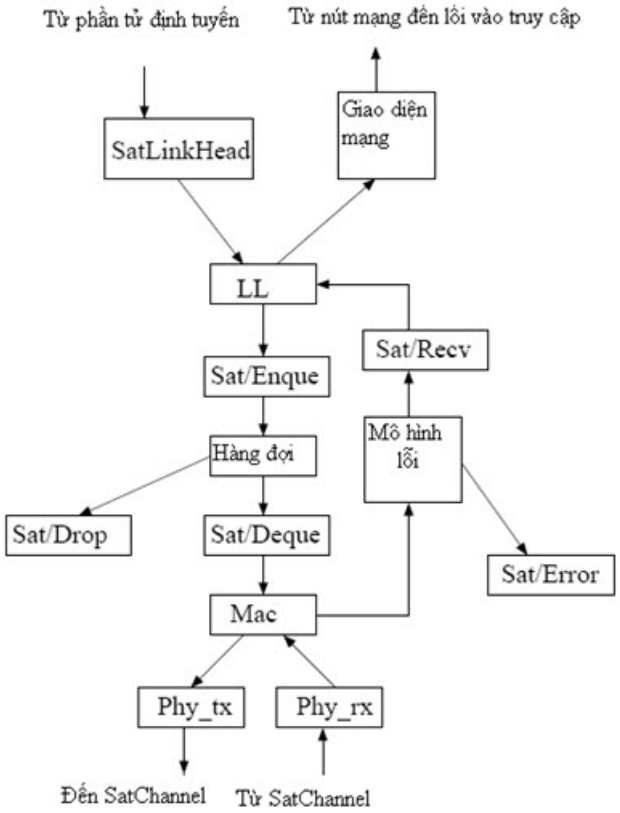
Thông tin chi tiết về đường truyền vệ tinh được minh họa trong Hình 14.6. Tập tin *...ns/tcl/lib/ns-sat.tcl* chứa các instprocs OTcl khác nhau dung ghép các liên kết theo Hình 14.6. Hầu hết các mã lệnh dung cho các thành phần khác nhau của đường truyền nằm trong thư viện *...ns/satlink.cc,h.*

Điểm "entry" của một "network stack" là đối tượng `SatLinkHead` .Đối tượng `SatLinkHead` xuất phát từ Class LinkHead. Các đối tượng đầu đường truyền dung để cung cấp một API đồng nhất đối với mọi network stacks [2]. Mã lệnh hợp lệ cho trường `type_` thể tìm thấy trong thư viện *ns/sat.h.* SatLinkHead lưu biến kiểu logic `linkup_` mà nó chỉ ra đường truyền đến ít nhất một nút mạng khác trên một kênh kết thúc. Lệnh thực thi C++ của SatLinkHead được tìm thấy ở thư viện *ns/satlink.cc,h.*

Các gói dữ liệu rời khỏi một nút mạng sẽ truyền thẳng đến `SatLinkHead` đến đối tượng của `class SatLL`. Lớp SatLL thuộc lớp liên kết dữ liệu LL.Giao thức lớp liên kết dữ liệu (như giao thức ARQ) có thể được định nghĩa ở đây. SatLL hiện tại sẽ cấp địa chỉ MAC đến gói dữ liệu. Chú ý rằng trong trường hợp vệ tinh, chúng ta không dùng giao thức phân giải địa chỉ (ARP); thay vào đó, chúng ta chỉ đơn giản dùng biến MAC `index_` như địa chỉ

[2]In the author's opinion, all network stacks in ns should eventually have a LinkHead object at the front– the class SatLinkHead would then disappear

của nó, và chúng ta sử dụng chức năng trợ giúp để tìm địa chỉ MAC của giao diện tương ứng của nút mạng bước truyền tiếp theo.Bởi vì lớp liên kết dữ liệu LL xuất phát từ lớp LinkDelay, tham số `delay_` của LinkDelay có thể dùng để mô hình hóa bất cứ độ trễ xử lý trong lớp liên kết dữ liệu, với giá trị mặc định trễ này là không.



Hình 14.6: Chi tiết đặc tả ngăn xếp giao diện của mạng

Đối tượng tiếp theo gói dữ liệu tiến tới là hàng đợi giao diện. Tuy nhiên, nếu truy vết được kích hoạt, phần tử truy vết có thể bao quanh hàng đợi, như chỉ ra ở hình 14.6 . Phần này của một đường truyền vệ tinh hoạt động như một đường truyền ns thông thường. Lớp thấp hơn tiếp theo là lớp MAC. Lớp MAC lấy gói dữ liệu từ đối tượng hàng đợi (hoặc truy vết deque)- một sự móc nối gữi lớp MAC và hàng đợi cho phép MAC kéo gói dữ liệu khỏi hàng đợi nếu nó cần chúng. Thời gian truyền của một gói dữ liệu được mô hình ở lớp MAC, MAC tính toán trễ truyền dẫn của gói (dựa trên việc đếm trường `LINK_HDRSIZE` được định nghĩa trong satlink.h và trường size ở phần đầu chung của gói dữ liệu), và yêu cầu một gói khác cho đến khi gói hiện tại được gửi xuống lớp thấp hơn. Do đó, rất quan trọng trong việc thiết lập băng tần của đường truyền một cách chính xác ở lớp này. Để cho thuận tiện, thời gian truyền được mã hóa trong phần tiêu đề MAC, có thể sử dụng thông tin này ở MAC nhận để tính xem nó đợi bao nhiêu lâu để phát hiện một xung đột trên một gói dữ liệu, ví dụ thế.

Tiếp theo, gói dữ liệu được gửi đến giao diện truyền (`Phy_tx`) của lớp SatPhy. Đối tượng này vừa gửi gói đến kênh truyền được gắn vào. Chúng ta đã có giả thiết rằng tất cả các giao diện gắn vào kênh là một phần của danh sách lien kết cho kênh đó. Điều này tuy nhiên không đúng với giao diện truyền. Chỉ giao diện nhận được gắn vào một kênh bao gồm danh sách đường truyền này, từ đó chỉ có giao diện nhận nhận được bản sao của gói dữ liệu truyền. Việc sử dụng giao diện truyền và nhận riêng rẽ phản ánh trung thực các đường truyền vệ tinh song công được tạo ra bởi kênh RF ở các tần số khác nhau.

Gói dữ liệu đi được gửi tiếp đến một `SatChannel`, mà nó sẽ sao gói dữ liệu đến mỗi giao diện nhận (hoặc lớp SatPhy) trên mỗi kênh. `Phy_rx` gửi gói dữ liệu đến lớp MAC. Ở tại lớp MAC, gói dữ liệu được giữ trong suốt thời gian truyền của nó (và bất cứ sự phát hiện xung đột nào được tìm thấy sẽ được thi hành nếu lớp MAC, như là Aloha MAC, hỗ trợ nó).Nếu một gói dữ liệu đến lớp MAC an toàn, nó sẽ truyền tiếp đến một đối tượng ErrorModel, nếu nó tồn tại. Nếu không thì gói sẽ chuyển qua bất cứ đối tượng truy vết nhận nào đến đối tượng SatLL . Đối tượng SatLL truyền gói đến sau trễ xử lý (nhắc lại rằng theo mặc định, giá trị cho delay_ là bằng 0).

Gói dữ liệu đã nhận sẽ tới đối tượng cuối cùng mà đối tượng này thuộc về lớp Network-Interface. Đối tượng này thêm nhãn `iface_field` vào tiêu đề chung cùng với giá trị chỉ mục riêng của ngăn xếp mạng. Điều này được dùng để thực hiện bám vết xem ngăn xếp mạng nào mà một gói dữ liệu đã tới.

Gói dữ liệu này sau đó sẽ tới đầu vào của SatNode (thường là bộ phân loại địa chỉ). Các ngăn xếp mạng (network stacks) của vệ tinh loại lặp-địa tĩnh chỉ bao gồm một `Phy_tx` và một `Phy_rx` thuộc về lớp `Repeater-Phy`, và một SatLinkHead. Các gói dữ liệu nhận bởi `Phy_rx` được gửi tới `Phy_tx` với trễ bằng không. Vệ tinh loại lặp-địa tĩnh là một nút vệ tinh phát lại. Nó không có các bộ phận như bám vết, quản lý chuyển giao, định tuyến hoặc bất kỳ giao diện đường truyền nào khác ngoại trừ giao diện chuyển tiếp.

14.4. Commands at a glance

Cấu hình nút mạng này biểu thị các nút mạng mới đến sau được tạo sẽ là `<type>`, trong đó `<type>` có thể là một trong các loại sau: `geo, geo-repeater, polar, terminal`. Các trường yêu cầu khác cho các nút mạng vệ tinh (cho cài đặt đường truyền ban đầu và kênh truyền) như ở dưới:

```
$ns_ node-config -llType <type>
$ns_ node-config -ifqType <type>
$ns_ node-config -ifqLen <length>
$ns_ node-config -macType <type>
```

```
$ns_ node-config -channelType <type>
$ns_ node-config -downlinkBW <value>
```

(chú ý rằng Geo SatNodeType loại "lặp-địa tĩnh" chỉ yêu cầu định rõ loại kênh, còn tất cả các lựa chọn khác không được quan tâm đến. Hãy tham khảo ví dụ trong *tcl/ex/sat-repeater.tcl.)* `$ns_ satnode-polar <alt> <inc> <lon> <alpha> <plane> <linkargs> <chan>`

Đây là một phương pháp "bao bọc – wrapper method" dung trong mô phỏng để tạo ra một nút mạng vệ tinh cực. Hai đường truyền lên và xuống, được tạo ra với hai kênh truyền, kênh lên và kênh xuống. <alt> là vĩ độ của vệ tinh cực, còn <inc> là góc nghiêng di chuyển theo đường xích đạo w.r.t, <lon> là kinh độ của điểm lên "ascending node", <alpha> đưa ra vị trí ban đầu của vệ tinh dọc theo quỹ đạo này, <plane> định nghĩa mặt phẳng quỹ đạo vệ tinh cực này. <linkargs> là danh sách các lựa chọn biến đường truyền định nghĩa giao diện mạng (giống như LL< Qtype,Qlim,Phy, MAC,v.v)

`$ns_ satnode-geo <lon> <linkargs> <chan>`

Đây là phương pháp "bao bọc" tạo ra nút mạng của bộ lặp vệ tinh địa tĩnh GEO, đầu tiên tạo ra một nút mạng vệ tinh cùng với hai giao diện đường truyền (tuyến lên và tuyến xuống) và hai kênh vệ tinh (kênh lên và kênh xuống). <chan> Định nghĩa loại kênh

`$ns_ satnode-geo-repeater <lon> <chan>`

Đây là phương pháp "bao bọc" tạo ra nút mạng của bộ lặp vệ tinh địa tĩnh GEO, đầu tiên tạo ra một nút mạng vệ tinh cùng với hai giao diện đường truyền (tuyến lên và tuyến xuống) và hai kênh vệ tinh (kênh lên và kênh xuống).

`$ns_ satnode-terminal <lat> <lon>`

Còn đây là phương pháp bao bọc tạo nút mạng của thiết bị đầu cuối. <lat> và <lon> định nghĩa kinh độ và vĩ độ tương ứng của thiết bị đầu cuối.

`$ns_ satnode <type> <args>`

Đây là một phương pháp nguyên thủy để tạo ra loại satnodes <type> có thể là cực, địa tĩnh, hoặc thiết bị đầu cuối. Đó là:

`$satnode add-interface <type> <ll> <qtype> <qlim> <mac_bw> <phy>`

Đây là một phương pháp nội bộ của Node/SatNode cài đặt cho lớp liên kết dữ liệu, lớp MAC, hàng đợi giao diện và cấu trúc lớp vật lý cho các nút mạng vệ tinh.

`$satnode add-isl <ltype> <node1> <node2> <bw> <qtype> <qlim>`

Phương pháp này tạo ra ISL (đường truyền vệ tinh nội bộ) giữa hai nút mạng. Loại đường truyền (inter, intra, hoặc crossseam), dải thông của đường truyền, loại hàng đợi và giới hạn hàng đợi đều được định rõ.

`$satnode add-gsl <ltype> <opt_ll> <opt_ifq> <opt_qlim> <opt_mac> <opt_bw> <opt_phy> <opt_inlink> <opt_outlink>` Phương pháp này tạo một GSL (đường truyền

từ mặt đất tới vệ tinh). Đầu tiên, một stack mạng được tạo ra định nghĩa bởi LL, IfQ, Qlim, MAC, BW, và lớp PHY. Tiếp theo nút mạng được gắn vào kênh truyền lên và kênh truyền xuống.

Tài liệu tham khảo

[1] The VINT Project. *The ns Manual(formerly ns Notes and Documentation)*, April 15, 2006

[2] Eitan Altman, Tanima Jiménez. *NS Simulator for beginners* , Lecture Note, 2003 - 2004

[3] Mrc Greis. *Tutorial for the Network Simulator "ns"*,
URL: www.isi.edu/nsnam/ns/tutorial/

[4] Gaeil Ahn and Woojik Chun. *Design and Implementation of MPLS Network Simulator*

[5] TCL web page. *http://otcl-tclcl.sourceforge.net/tclcl/*

[6] Christian Huitema. *Routing in the Internet, Prentice Hall*, April 1995. 2nd Edition 1999

[7] A.Ballardie *Core Based Trees (CBT) Multicast Routing Architecture, Internet RFC 2201 (Experimenatal),* September1997

[8] D. Thaler, D.Estrin, D.Meyer(Editor). *Border Gateway Multicast Protocol (BGMP): Protocol Spection,Internet Draft,* November 2000

[9] S.Deering, D.Estrin, D. Farinacci, V.Jacobson, Ching-Gung Liu, and L Wei. *An architecture for wise-area multicast routing.Technical Report USC-SC-94-565,* Computer Science Department, University of Southern Clifornica, Los Angeles, CA 90089, 1984

[10] Floyd, S., and V. Jacobson. *Random Early Detection gateways for Congestion Avoidance* V.1 N.4, August 1993, p. 397-413. Available at http://www.icir.org/floyd/papers/red/red/html

[11] A. Ballardie. *Core Based Trees (CBT) Multicast Routing Architecture* Internet RFC 2201 (Experimental), September 1997

[12] S. Floyd, R. Gummadi and S. Shenker. *Adaptive Red: an algorithm for increasing the robustness of RED's active queue management.* Submitted

[13] E. Altman and T. Jiménez. *Simulation analysis of RED whit short lived TCP connections.* submitted

[14] P. Pieda, J. Ethridge, M.Baines and F. Shallwani. *A network simulator differentiated serices implementation.* Open IP, Nortel Networks, July 26, 2000.

[15] W. Noureddine and F. Tobagi. *Improving the Performance of Interactive TCP Applications using Service Differentiation.* Proceeding of IEEE infocom, New-York, USA, June 2002

[16] *D. D Clark and W. Fang. Explicit Allocation of Best-E ort Packet Delivery Service.* IEEE/ACM Trans on Networking, 6(4), 362-372, August 1998

[17] W, Gang, N. Seddigh and B. Nandy . *A Time Sliding Window Three Colour Marker(TSWTCM), RFC 2589.*

[18] J. Heinanen andd R. Guerin. *A tow rate three color marker.* RFC 2698 Sept 1999 June 2000

1. Giới thiệu nhóm Advanced Networking Group

Chào các bạn

Ban Quản Trị vntelecom.org đã xúc tiến thành lập một working group với tên gọi là Advanced Networking Group(AdNet). Xin mời các bạn đọc giới thiệu về AdNet và mời cùng tham gia vào AdNet.

Để tham gia vào AdNet, mời bạn gửi email đăng ký tới: adnet@vntelecom.org với nội dung bao gồm:

1. Họ tên:
2. Giới tính:
3. Trình độ và chuyên ngành:
4. Nơi công tác (ví dụ như tên công ty) hoặc khu vực bạn ở (ví dụ như Hà Nội)

Group page http://groups.google.com/group/adnet

Địa chỉ google group: adnet@googlegroups.com

1.1. Thông tin sơ bộ về AdNet

1. Birthday: Ngày 4 tháng 11 Năm 2008
2. Lĩnh vực
    - Mạng máy tính và mạng truyền thông (Computer and communications networks)
    - Tập trung vào các chuyên mục thuộc link layer, network layer
    - Quan tâm tới các chuyên mục thuộc transport layer, application layer
3. Hoạt động

- Trao đổi các vấn đề nghiên cứu qua mailing list, qua diễn đàn trên vntelecom
- Tổ chức các seminar, các buổi gặp gỡ trao đổi kiến thức
- Thực hiện các dự án chuyên môn

4. Quy định

    - Các thành viên tham gia mailling list với tư cách cá nhân cùng với việc cung cấp các thông tin tối thiểu về bản thân như tên, giới tính, trình độ, nơi công tác tới các thành viên khác

2. Dự án NetSim của Advanced Networking Group

2.1. Dự án tài liệu mô phỏng bằng tiếng Việt (NetSim-Viet)

"Network simulation - NetSim" – "Mô phỏng mạng" là phần không thể thiếu được trong đào tào và nghiên cứu. Hiện nay ở Việt Nam, số lượng sinh viên làm đồ án tốt nghiệp cần đến NetSim ngày càng nhiều. Có rất nhiều chương trình mô phỏng (simulation tool) khác nhau và thông dụng như NS2 (free), OPNET (cracked) ... nhưng tài liệu hướng dẫn và các thông tin về chúng (phần mở rộng, các module mới) bằng tiếng Việt vẫn chưa nhiều và rải rác. Tài liệu này sẽ rất hữu ích cho sinh viên đại học và cao học, đạc biệt những bạn vẫn còn hạn chế khi sử dụng tiếng Anh.

AdNet đề xuất thực hiện dự án "Biên soạn tài liệu mô phỏng mạng bằng tiếng Việt". Nội dung dự định của tài liệu:

1. Giới thiệu chung về lý thuyết mô phỏng mạng
    - Lý thuyết chung về Discrete event simulation
    - Nguyên tắc thống kê và đánh giá kết quả
2. Giới thiệu chung về các tool phổ biến
    - NS2
    - OPNET
    - OMNET++
3. Hướng dẫn cài đặt, giới thiệu và đánh giá các module mới của NS2
4. Hướng dẫn cài đặt, giới thiệu và đánh giá các module mới của OPNET
5. Hướng dẫn cài đặt , giới thiệu và đánh giá các module mới của OMNET++
6. Tool khác ....

2.2. Cách thức biên soạn

Bước 1 - Xác định nội dung của tài liệu.

Bước 2 - Tìm kiếm các tài liệu tiếng Việt đã có sẵn và liên hệ với các tác giả để mời đứng tên cùng.

Bước 3 - Biên dịch các nội dung chưa có phần tiếng Việt.

Bước 4 - Hiệu đính và đưa lên vntelecom.

Bước 5 - Cập nhật và sửa chữa tài liệu.

2.3. Tổ chức NetSim theo các nhóm

Nhóm không có Trưởng hay Phó mà chỉ có Báo Cáo Viên (BCV) số 1 hay số 2 gọi là BCV1 và BCV2. - Nhiệm vụ của BCV là : Duy trì sự liên lạc giữa các thành viên của nhóm Báo cáo kết quả hoạt động tới NetSim - Mỗi thành viên có thể tham gia 1 hoặc vài nhóm.

Các nhóm dự kiến

2.3.1 Nhân Sự (Human Resource - HR)

Nhiệm vụ của thành viên:

- Cập nhật tình hình thành viên gia nhập vào NetSim hay rút lui khỏi NetSim
- Quản lý mailing list
- Giám sát nhân lực đang hoạt động ở các nhóm khác: bổ sung, hỗ trợ v...v
- Liên hệ với các cá nhân ngoài NetSim để mời hợp tác, đồng tác giả v...v

2.3.2 Quản lý nội dung NetSim (Content Management - CM)

Nhiệm vụ của thành viên:

- Biên soạn nội dung tổng thể của dự án NetSim
- Phân chia các nhiệm vụ cần thực hiện và xác định thời hạn (deadline) cho mỗi nhiệm vụ
- Giám sát tiến độ thực hiện và điều chỉnh nội dung nếu cần

2.3.3. Thực hành mô phỏng mạng (Simulation Practice - SP)

Nhiệm vụ của thành viên:

- Cài đặt simulation tool, học và làm việc với tool
- Kiểm tra đánh giá các tính năng của simulation tool

Yêu cầu đối với thành viên

- Muốn học và sử dụng thành thạo simulation tool nào đó
- Có máy tính riêng (tất nhiên rồi)
- Trình độ tiếng Anh: căn bản

2.3.4 Biên dịch và biên soạn tài liệu (Translation & Edition-TE)

Nhiệm vụ của thành viên

- Tham gia dịch (translation) và hiệu đính tài liệu Anh sang Việt theo nội dung nhóm CM đề ra.
- Tham gia soạn (edition) tài liệu tiếng Việt có sẵn để ghép vào NetSim

Yêu cầu đối với thành viên:

- Trình độ tiếng Anh: căn bản
- Trình độ tiếng Việt: TỐT