TÀI LIỆU THÍ NGHIỆM MÔN HỌC

KỸ THUẬT TRUYỀN SỐ LIỆU

(Phiên bản cập nhật ngày 3/12/2009.

Tài liệu phục vụ môn học. Lưu hành nội bộ)

Biên soạn: **PGS-TS Trần Xuân Nam**
Bộ môn Thông tin,
Khoa Vô tuyến Điện tử
Học viện Kỹ thuật Quân sự

HÀ NỘI 2007

MỤC LỤC

Bài 1

GIỚI THIỆU

1.1 Mục đích

Mục đích của tập bài thí nghiệm phân tích giao thức mạng này là giúp cho học viên nắm vững quá trình trao đổi dữ liệu diễn ra giữa các giao thức thuộc các lớp mạng tương ứng của bộ giao thức TCP/IP sử dụng trong Internet. Các bài thí nghiệm phân tích giao thức mạng sẽ giúp cho sinh viên trực tiếp thực hiện thiết lập cấu hình, thu kết dữ liệu và phân tích kết quả, quan sát chuỗi các bản tin trao đổi giữa hai thực thể (entities) giao thức, đào sâu vào chi tiết của hoạt động giao thức, và điều khiển các giao thức thực hiện một số hoạt động nhất định rồi quan sát các hoạt động đó và hiệu quả của chúng. Các nội dung này có thể được thực hiện theo hai phương pháp: mô phỏng hoặc phân tích môi trường mạng thực. Trong phạm vi bài thí nghiệm này chúng ta sẽ sử dụng phương pháp thứ hai nhờ sử dụng gói phần mềm phân tích giao thức mạng Wireshark. Đây là gói phần mềm mã mở được sử dụng phổ biến ở nhiều trường đại học và các viện nghiên cứu trên thế giới. [1]

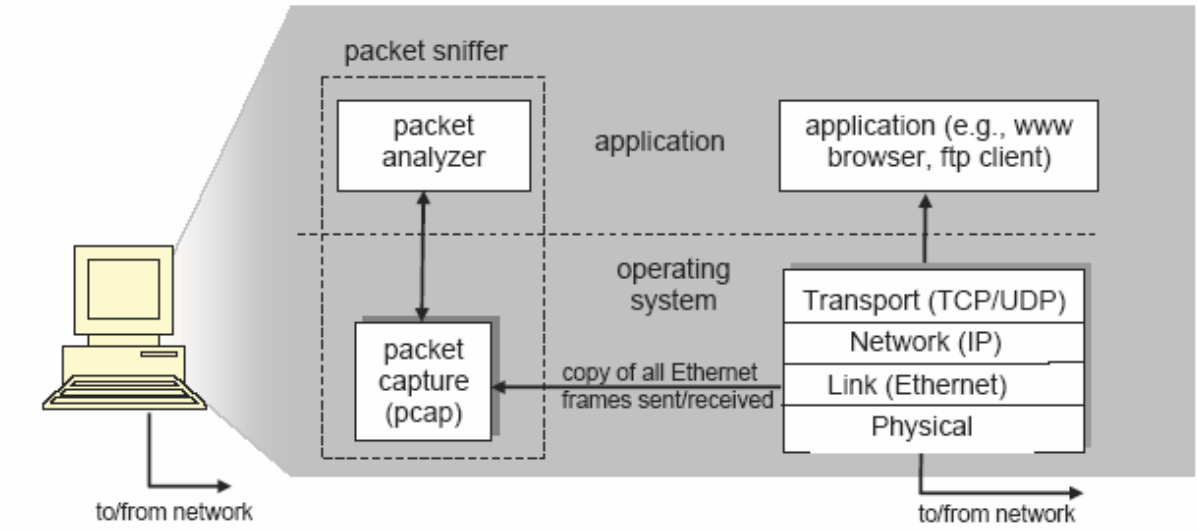
Học viên sẽ chạy một số ứng dụng mạng trong các tình huống khác nhau sử dụng máy tính ở trường hoặc ở nhà. Quan sát các giao thức mạng sử dụng máy tính của mình học viên có thể trực tiếp tương tác và trao đổi bản tin với các thực thể giao thức trên Internet. Vì vậy, học viên và máy tính sẽ đóng vai trò là một phần tích hợp của các bài thí nghiệm "thực" này. Thông qua bài thí nghiệm học viên sẽ nắm bắt được kiến thức nhờ quá trình "học đi đôi với hành".

Công cụ cơ bản để quan sát các bản tin trao đổi giữa các thực thể giao thức đang chấp hành được gọi là "packet sniffer". Một chương trình packet sniffer bắt bản tin đang được phát/thu từ/bởi máy tính của học viên; nó cũng cho phép lưu giữ và/hoặc hiển thị nội dung của các trường giao thức của các bản tin bắt được. Bản thân packet sniffer là một chương

[1] Nội dung các bài thí nghiệm trong tài liệu này được biên soạn lại từ tài liệu do J.F.Kurose and Keith W. Ross biên soạn. Để xem toàn bộ các bài thí nghiệm chi tiết bằng tiếng Anh, xin truy nhập địa chỉ sau đây: http://gaia.cs.umass.edu/ethereal-labs/

trình thụ động với ý nghĩa là nó chỉ quan sát các bản tin đang được phát và thu bởi các ứng dụng và giao thức đang chạy trên máy tính chứ không tự phát đi các gói tin. Một cách tương tự, các bản tin cũng không bao giờ được đánh địa chỉ đến packet sniffer một cách rõ ràng (trực tiếp). Thay bằng, một packet sniffer nhận một bản sao của các packet được phát/thu từ/bởi ứng dụng hay các giao chức chạy đang trên máy tính.

Hình 1.1 chỉ ra cấu trúc của một packet sniffer. Ở bên phải của Hình 1.1 là các giao thức (trong trường hợp này là các giao thức Internet) và các ứng dụng (ví dụ như trình duyệt web hay một ftp client) thường chạy trên máy tính. Packet sniffer được mô tả bên trong hình chữ nhật đứt nét là một phần chương trình được cài đặt vào máy tính, và gồm hai phần. Phần thư viện bắt gói tin (packet capture library) thu các bản sao của các frame của lớp liên kết (link layer) được phát đi hoặc thu từ máy tính. Theo lý thuyết ở bài giảng thì các bản tin trao đổi bởi các giao thức lớp phía trên như HTTP, FTP, TCP, UDP, DNS, hay IP đều được đóng gói vào các frame của lớp liên kết được phát đi qua môi trường vật lý như cáp trong mạng Ethernet chẳng hạn. Ở sơ đồ Hình 1.1, môi trường giả thiết là Ethernet, và vì vậy, các giao thức lớp trên được đóng gói vào trong một Ethernet frame. Việc bắt tất cả các frame của lớp liên kết cho phép thu được tất cả các bản tin phát/thu từ/bởi tất cả các giao thức và ứng dụng đang chạy trên máy tính.



Hình 1.1: Cấu trúc packet sniffer

Thành phần thứ hai của một packet sniffer là bộ phân tích gói tin (packet analyzer), cho phép hiển thị nội dung của tất cả các trường trong một bản tin giao thức. Để làm được điều này, packet analyzer cần phải "hiểu" cấu trúc của tất cả các bản tin trao đổi giữa các giao thức. Ví dụ, giả sử chúng ta quan tâm đến việc hiển thị các trường trong các bản tin trao đổi bởi giao

thức HTTP như ở Hình 1.1. Packet analyzer hiểu cấu trúc của định dạng Ethernet frame, và vì vậy có thể xác định được IP datagram bên trong Ethernet frame. Packet analyzer cũng hiểu định dạng của IP datagram, và có thể tách được TCP segment bên trong IP datagram. Tương tự, packet analyzer cũng biê· t cấu trúc của TCP segment và, vì vậy, cho phép tách được bản tin HTTP chứa trong TCP segment. Cuối cùng, packet analyzer hiểu giao thức HTTP và, vì vậy, biết được, byte đầu tiên trong một bản tin HTTP có chứa các lệnh điều khiển như các tự "GET," "POST," hoặc "HEAD".

Trong phạm vi các bài thí nghiệm này, chúng ta sẽ sử dụng Wireshark packet sniffer để hiển thị nội dung của các bản tin đang phát/thu từ/bởi các giao thức ở các lớp khác nhau của chồng giao thức TCP/IP. Chương trình này hoạt động trên các máy tính có sử dụng Ethernet hay ADSL để kết nối tới Internet, cũng như các giao thức điểm-nối-điểm như PPP (Point-to-Point Protocol). Wireshark là tên gọi của chương trình Ethereal trước đó, bắt nguồn từ giao thức lớp liên kết dữ liệu Ethernet như đã học trong bài giảng.

1.2 Cài đặt Wireshark

Để chạy Wireshark, máy tính cần phải được cài đặt cả hai phần mềm packet sniffer Wireshark và thư viện bắt gói tin *libpcap*. Nếu phần mềm *libpcap* chưa được cài đặt vào trong hệ điều hành của máy, cần phải cài đặt *libpcap*. Để biết địa chỉ download, xem thêm tại địa chỉ http://www.wireshark.org/download.html.

- Download và cài gói phần mềm Wireshark: truy nhập đến địa chỉ http://www.wireshark.org, truy nhập vào mục **Download**, chọn một server ở gần để download Wireshark. Phiên bản hiện tại của Wireshark là **Wireshark 0.99.7**.
- Download và cài đặt *libpcap*[2]: với Windows, phần mềm *libpcap* thường được biết đến với tên gọi *WinPCap*. Để download *WinPCap* truy nhập vào địa chỉ http://www.winpcap.org/, truy nhập đến menu **Get WinPCap**, và download từ mục **Installer for Windows.** Phiên bản hiện tại của WinPCap là **WinPCap 4.0.2.**

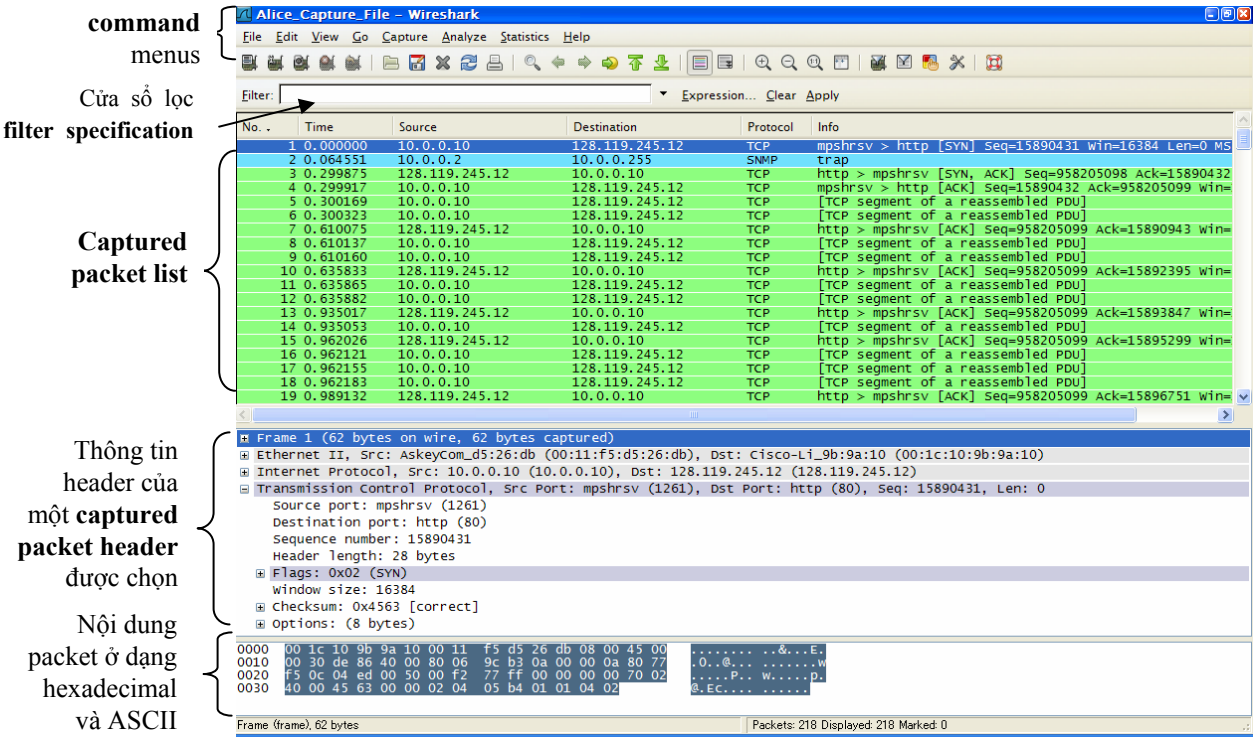
1.3 Khởi động Wireshark

Sau khi khởi động Wireshark, giao diện đồ họa người dùng của Wireshark sẽ được hiển thị như ở Hình 1.2. Ban đầu không có dữ liệu được hiển thị ở các cửa sổ.

Giao diện Wireshark có năm thành phần chính:

[2] Các phiên bản mới của Wireshark đã có thể bao gồm WinPCap nên cần kiểm tra lại trước khi cài đặt

- Menu câu lệnh (command menus) là các menu kéo xuống đặt ở phía trên đầu của cửa sổ. Hai menu đáng quan tâm nhất là menu *File* và *Capture*. Menu *File* cho phép lưu giữ dữ liệu gói tín bắt được và mở một tệp chứa dữ liệu gói bắt được, và thoát khỏi ứng dụng Wireshark. Menu Capture cho phép bắt đầu bắt gói tin.



Hình 1.2. Giao diện người dùng Wireshark

- Cửa sổ liệt kê gói tin (packet-listing window) hiển thị một dòng tóm tắt về mỗi gói tin bắt được, bao gồm cả số thứ tự gói do Wireshark gán, thời gian bắt được gói tin, địa chỉ nguồn và địa chỉ đích của gói tin, kiểu giao thức, và thông tin về giao thức chứa trong gói tin. Phần liệt kê gói tin có thể được xắp xếp phân loại theo bất kỳ loại nào nhờ bấm vào một tên cột. Trường kiểu giao thức (protocol) liệt kê giao thức mức cao nhất thực hiện phát hoặc thu gói tin này, tức là, giao thức nguồn hay đích của gói tin này.
- Cửa sổ chi tiết về packet header (packet-header details window) cung cấp chi tiết về gói tin được chọn (highlighted) ở trong cửa sổ liệt kê gói tin. (Để chọn một gói tin trong của sổ liệt kê gói tin, đặt con trỏ vào dòng tóm tắt về gói tin ở trong cửa sổ liệt kê gói tin và click bằng phím chuột trái). Các chi tiết này bao gồm thông tin về Ethernet frame và IP datagram chứa gói tin này. Lượng thông tin của Ethernet và lớp IP có thể được mở rộng hay thu hẹp lại bằng cách clicking vào mũi tên chỉ

sang phải hay xuống dưới về phía trái của dòng Ethernet frame hay IP datagram ở cửa sổ chi tiết về gói tin. Nếu các gói tin được mang bởi TCP hay UDP, chi tiết về TCP hay UDP sẽ được hiển thị. Cuối cùng, chi tiết về giao thức lớp cao nhất phát hay thu gói tin này cũng được cung cấp.

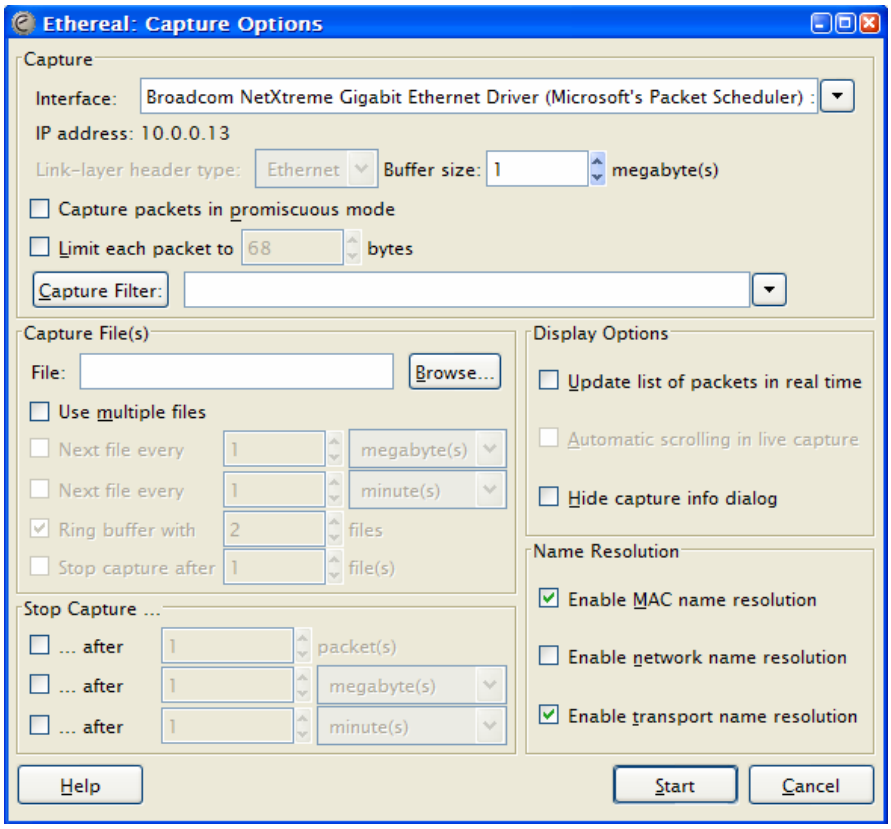
- Cửa sổ nội dung gói tin (packet-contents window) hiển thị toàn bộ nội dung của frame bắt được, cả ở dạng ASCII và cơ số 16 (hexadecimal).
- Trường lọc hiển thị gói (packet display filter field) ở phía trên của giao diện đồ họa người sử dụng Wireshark cho phép nhập tên hay các thông tin khác về giao thức để lọc thông tin hiển thị cửa sổ liệt kê gói tin (và vì vậy, đầu gói tin và cửa sổ nội dung gói tin). Ở ví dụ dưới đây chúng ta sử dụng trường lọc hiển thị gói để lọc các gói Ethernet ẩn, ngoại trừ các gói tương ứng với các bản tin HTTP.

1.4 Chạy thử Wireshark

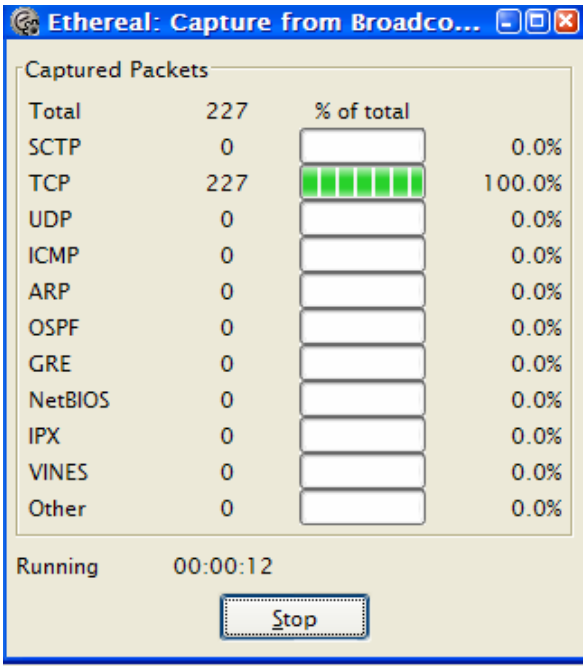
Để chạy thử Wireshark thực hiện các bước sau đây

1. **Bước 1:** Khởi động web browser (Ví dụ: Internet Explorer hay Firefox), nhập vào trang website lựa chọn.
2. **Bước 2:** Khởi động phần mềm Wireshark. Sẽ thấy có một cửa sổ tương tự ở Hình 1.2, ngoại trừ không có gói dữ liệu hiển thị ở các cửa sổ packet-listing, packet-header, hay packet-contents, do Wireshark chưa bắt đầu bắt gói.
3. **Bước 3:** Để bắt đầu "bắt" gói, chọn menu kéo xuống *Capture* và chọn *Start*. Thao tác này sẽ làm cho của sổ "*Wireshark: Capture Options*" hiển thị như ở Hình 1.3.
4. **Bước 4:** Sinh viên có thể sử dụng tất cả giá trị default trong cửa sổ values . Các giao diện mạng (tức là, các kết nối vật lý) mà máy tính có để nối đến mạng sẽ được hiển thị ở menu kéo xuống *Interface* ở phía trên của cửa sổ *Capture Options*. Trong trường hợp máy tính có nhiều giao diện mạng (ví dụ, nếu máy tính có cả kết nối mạng hữu tuyến Ethernet và kết nối vô tuyến), bạn sẽ cần chọn một giao tiếp sẽ sử dụng để thu và phát packets (thông thường là giao diện hữu tuyến Ethernet). Sau khi chọn xong giao diện mạng (hoặc sử dụng giao diện default của Wireshark), click OK. Chương trình bắt đầu bắt packet, tức là, tất cả các packet được phát/thu từ/bởi máy tính của bạn sẽ được chương trình Wireshark bắt.
5. **Bước 5:** Khi bắt đầu bắt packet, một cửa sổ thông tin vắn tắt về bắt packet sẽ xuất hiện như ở Hình 1.4. Cửa sổ này cho thông tin tóm tắt về số packets thuộc các kiểu

khác nhau đang bị bắt, và một phím *Stop* cho phép dừng bắt packet.



Hình 1.3: Cửa sổ tùy chọn của Wireshark

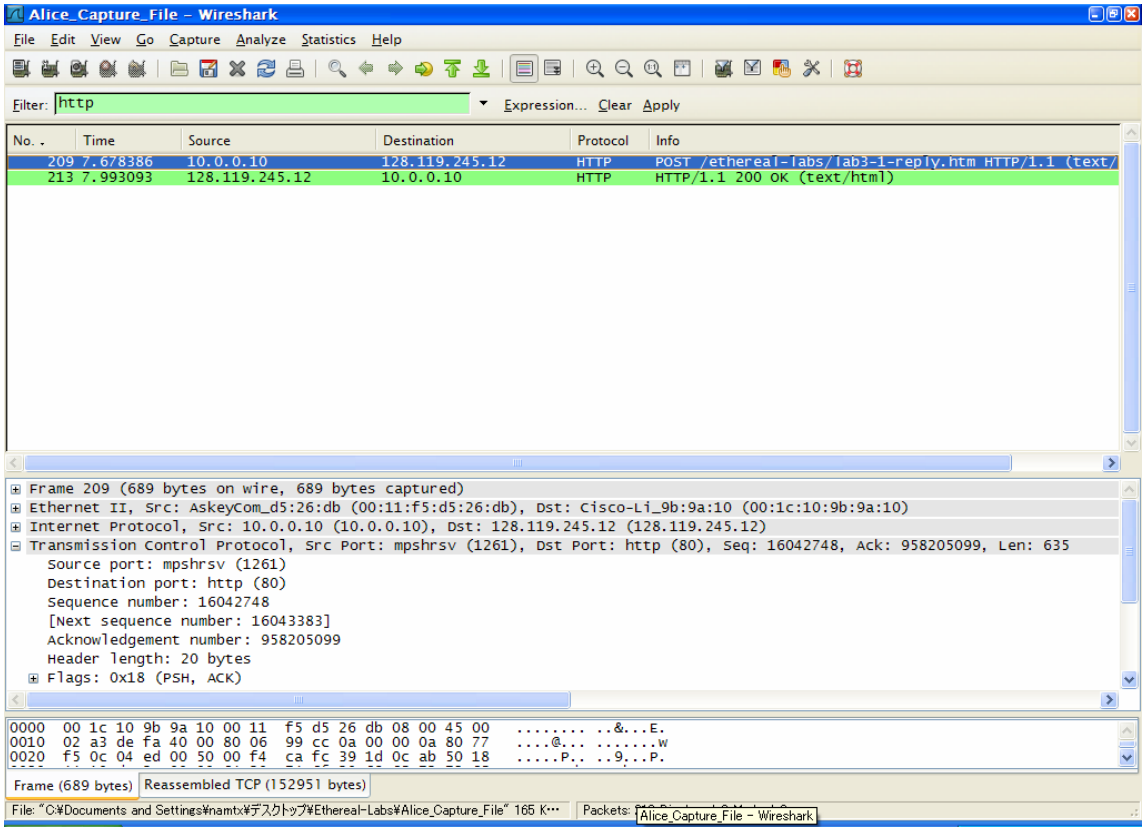


Hình 1.4: Cửa sổ captured packet của Wireshark

6. **Bước 6:** Trong khi Wireshark đang chạy, nhập vào một địa chỉ URL, ví dụ: http://www.lqdtu.edu.vn/index.htm để hiển thị nội dung trang ở web browser. Để hiển

thị nội dung trang web này, browser sẽ liên hệ với HTTP server tại http://www.lqdtu.edu.vn/index.htm và trao đổi các bản tin HTTP với server để download trang. Các Ethernet frames chứa các bản tin HTTP này sẽ bị Wireshark bắt để phân tích.

7. **Bước 7:** Sau khi browser hiển thị nội dung trang index.html, dừng quá trình bắt packet của Wireshark bằng cách chọn *Stop* ở cửa sổ *Wireshark Capture*, để hiển thị tất cả các packets bắt được từ khi bắt đầu bắt packet. Cửa sổ chính của Wireshark sẽ có dạng tương tự như cửa sổ ở trên Hình 1.2. Lúc này chúng ta có dữ liệu gói "thực" (live) chứa tất cả các bản tin trao đổi giữa máy tính và các thực thể khác của mạng. Bản tin HTTP trao đổi với server của www.lqdtu.edu.vn sẽ được hiển thị ở trong danh sách các gói bắt được. Tuy nhiên, cũng có nhiều loại gói khác cũng sẽ được hiển thị. Điều này có nghĩa là mặc dù bạn chỉ thực hiện thao thác download một trang web, nhưng đã có nhiều giao thức khác chạy ngầm trong máy tính của bạn
8. **Bước 8:** Nhập vào "http" (không có dấu ngoặc kép và ở dạng chữ in thường – ở Wireshark thì tất cả các tên protocol đều ở dạng chữ in thường) vào trong cửa sổ lọc hiển thị ở đầu cửa sổ Wireshark chính. Sau đó chọn *Apply*. Thao tác này sẽ lọc hiển thị riêng bản tin HTTP ở cửa sổ packet-listing.



Hình 1.5: Cửa sổ hiển thị thông tin của Wireshark sau bước 8

9. **Bước 9:** Chọn bản tin **http** đầu tiên trong cửa sổ packet-listing. Đó phải là bản tin HTTP GET được gửi đi từ máy của bạn tới HTTP server của trang www.lqdtu.edu.vn. Khi bạn chọn bản tin HTTP GET, thông tin đầu khung của Ethernet frame, IP datagram, TCP segment, và bản tin HTTP sẽ được hiển thị ở cửa sổ packet-header. Bằng cách click vào đầu mũi tên sang phải và xuống dưới ở phía bên trái của cửa sổ chi tiết về packet, có thể lọc bớt hiển thị thông tin của Ethernet frame, IP, và TCP. *Maximize* lượng thông tin hiển thị về giao thức HTTP. Wireshark của bạn sẽ trông gần giống như ở Hình 1.5.
10. **Bước 10:** Thoát Wireshark bằng cách vào *File* → *Quit*

Đến đây bạn đã hoàn thành xong bài tập đầu tiên.

1.5 Nội dung thí nghiệm cần báo cáo

Mục đích của bài thí nghiệm đầu tiên này là giới thiệu và giúp học viên làm quen với Wireshark. Dựa trên 10 bước thí nghiệm vừa thực hiện, trả lời các câu hỏi sau:

1. Liệt kê các giao thức xuất hiện trên cột giao thức ở cửa sổ packet-listing chưa được filter ở Bước 7.
2. Thời gian từ khi bản tin HTTP GET được gửi đi đến khi bản tin phúc đáp HTTP OK được nhận là bao lâu? (Theo mặc định, giá trị của cột *Time* ở cửa sổ packet-listing window là lượng thời gian tính theo giây từ khi Wireshark bắt đầu bắt. Để hiển thị trường *Time* ở dạng thời gian time-of-day, chọn menu kéo xuống *View,* sau đó chọn Time *Display Format*, sau đó chọn tiếp *Time-of-day*.)
3. Xác địa chỉ Internet của www.lqdtu.edu.vn? Xác định địa chỉ Internet của máy tính của bạn?
4. In hai bản tin **HTTP** hiển thị ở Bước 9 nói trên. Để in chọn *Print* từ menu câu lệnh *File*, và chọn "*Selected Packet Only"* và *"Displayed"* và click OK.

Bài 2

GIAO THỨC TCP

2.1 Mục đích

Mục đích của bài thí nghiệm này là giúp cho học viên quan sát và phân tích một quá trình trao đổi dữ liệu thực diễn ra giữa hai thực thể giao thức ở hai đầu kết nối, giúp học viên củng cố được kiến thức lý thuyết đã học trên lớp. Trong bài thí nghiệm này, học viên sẽ sử dụng gói phần mềm packet sniffer Wireshark để giám sát và nghiên cứu hoạt động thực của giao thức truyền dẫn tin cậy TCP (Transmission Control Protocol) cung cấp dịch vụ hướng kết nối (connection-oriented service) trên Internet.

Sau khi kết thúc bài thí nghiệm yêu cầu học viên nắm vững được quá trình hoạt động của giao thức truyền tải tin cậy TCP và biết cách sử dụng một công cụ packet sniffer để giám sát và phân tích quá trình trao đổi bản tin trên các giao thức.

2.2 Phương pháp

Để phân tích hoạt động của TCP chúng ta có thể sử dụng bất kỳ một ứng dụng yêu cầu dịch vụ truyền dẫn tin cậy như: HTTP, Telnet, FTP, hay SMTP để gọi giao thức TCP thực hiện kết nối, trao đổi dữ liệu, và ngắt kết nối. Trong bài thí nghiệm này để thuận tiện chúng ta sẽ sử dụng ứng dụng HTTP để POST một file từ local client là máy của học viên lên trên một remote server. Máy tính local client được cài đặt và kích hoạt Wireshark để bắt một trace của các packet từ khi thiết lập kết nối, trao đổi dữ liệu, cho đến khi ngắt kết nối.

2.3 Chuẩn bị bài thí nghiệm

Để chuẩn bị tiến hành thí nghiệm, cần chuẩn bị các yếu tố sau đây:

- Máy tính có kết nối đến Internet thông qua mạng LAN hay ADSL đóng vai trò local

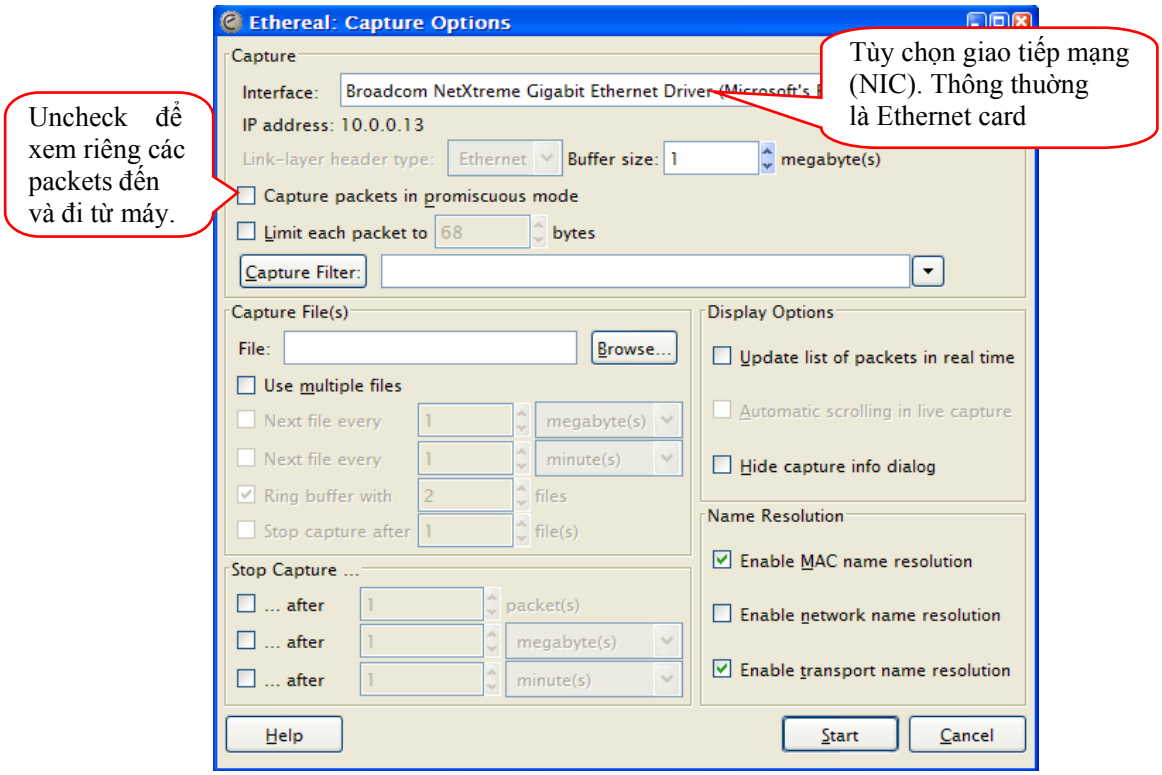
client và được cài đặt sẵn gói phần mềm packet sniffer Wireshark.

- Tạo một thư mục sử dụng cho bài thí nghiệm. Ví dụ: `D:/Wireshark-Labs`
- Một file text có kích thước tương đối lớn để upload lên remote server. Để thống nhất chúng ta chọn một file văn bản được thiết kế cho bài thí nghiệm có kích thước và nội dung phù hợp cho bài thí nghiệm Wireshark tại địa chỉ:

  `http://gaia.cs.umass.edu/ethereal-labs/alice.txt`

  Sau khi nhập vào địa chỉ trên vào thanh địa chỉ của web browser, sẽ có nội dung câu chuyện `ALICE'S ADVENTURES IN WONDERLAND` (*Alixơ ở xứ sở diệu kỳ*) ở dạng ASCII hiển thị trên web browser. Vào menu *File* → *Save page as*… của browser để lưu lại file câu chuyện với tên file `alice.text` vào thư mục thí nghiệm tại thư mục `D:/Wireshark-Labs`.
- Một account trong remote server cho phép post dữ liệu lên. Trong trường hợp không có account cho phép post file, chúng ta có thể sử dụng remote server hỗ trợ cho các bài thí nghiệm Wireshark đặt tại trường Đại học Massachuset tại địa chỉ:

  `http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html`



Hình 2.2 Cửa sổ tùy chọn Capture Options

2.4 Nội dung thí nghiệm

Để tiến hành bắt packet, tạo trace và phân tích hoạt động của TCP, thực hiện tuần tự các bước thí nghiệm sau đây:

2.4.1 Đặt Capture Options

1. **Bước 1:** Khởi động Wireshark bằng cách nhấp đúp vào biểu tượng của Wireshark trên desktop.
2. **Bước 2**: Trên thanh menu kéo xuống, chọn *Capture* → *Options....* Trên dòng menu tùy chọn *Interface*, chọn giao tiếp NIC kết nối tới Internet để bắt gói dữ liệu. Thông thường lựa chọn Ethernet nếu máy tính được nối tới mạng LAN hay modem ADSL. Nếu thực hiện kết nối vô tuyến đến Internet qua mạng WiFi, thì chọn wireless card. Tiếp theo uncheck *Capture packets in promiscous mode* để bắt riêng các gói đến và đi qua máy tính. Ví dụ mô tả lựa chọn các Capture Options được minh họa ở Hình 2.2.

2.4.2 Chuẩn bị capture các gói

3. **Bước 3**: Khởi động một trình duyệt web brower như Internet Explorer hoặc Firefox.
4. **Bước 4**: Khởi động bắt packet: bằng cách truy nhập vào menu *Capture* → *Start* để bắt đầu quá trình bắt gói.

2.4.3 Thiết lập kết nối TCP

5. **Bước 5**: Để thực hiện thiết lập một kết nối TCP, chúng ta sẽ thực hiện upload file `alice.text` đã lưu lại ở phần chuẩn bị lên server hỗ trợ bài thí nghiệm tại địa chỉ

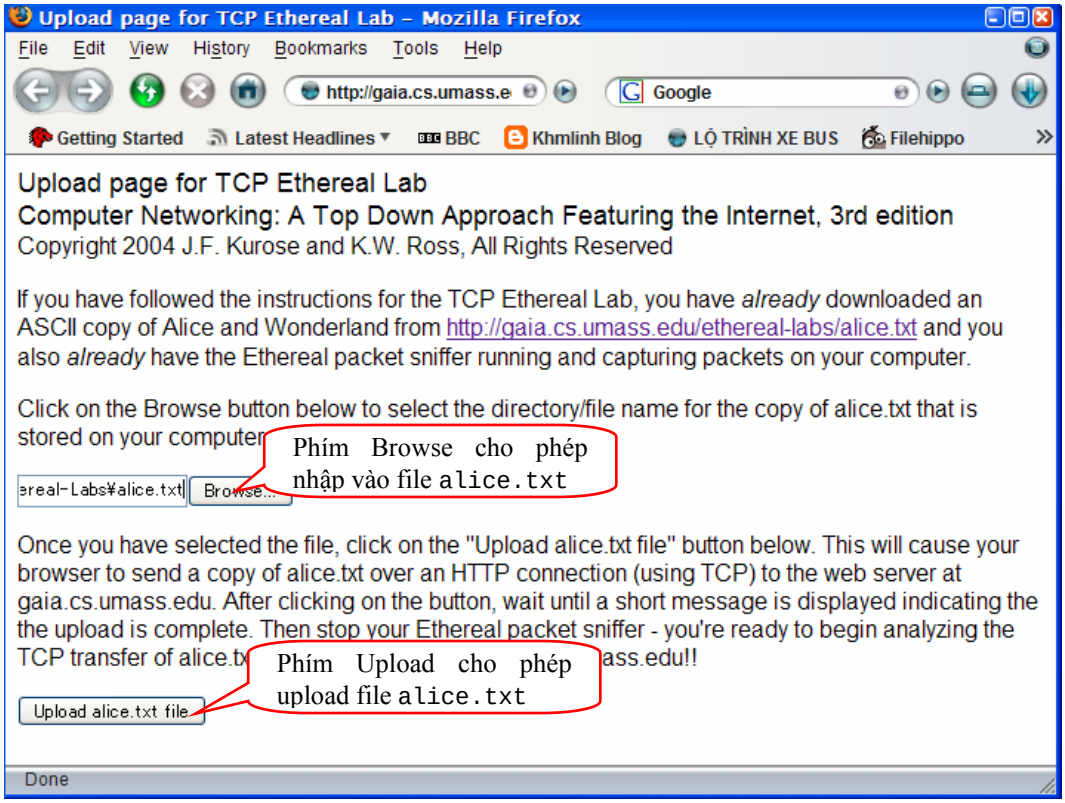
   `http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html`

   Thông qua việc upload file, giao thức HTTP sử dụng phương pháp POST để truyền tải file dữ liệu từ máy tính client của học viên lên remote server. Để upload file, nhập địa chỉ sau:

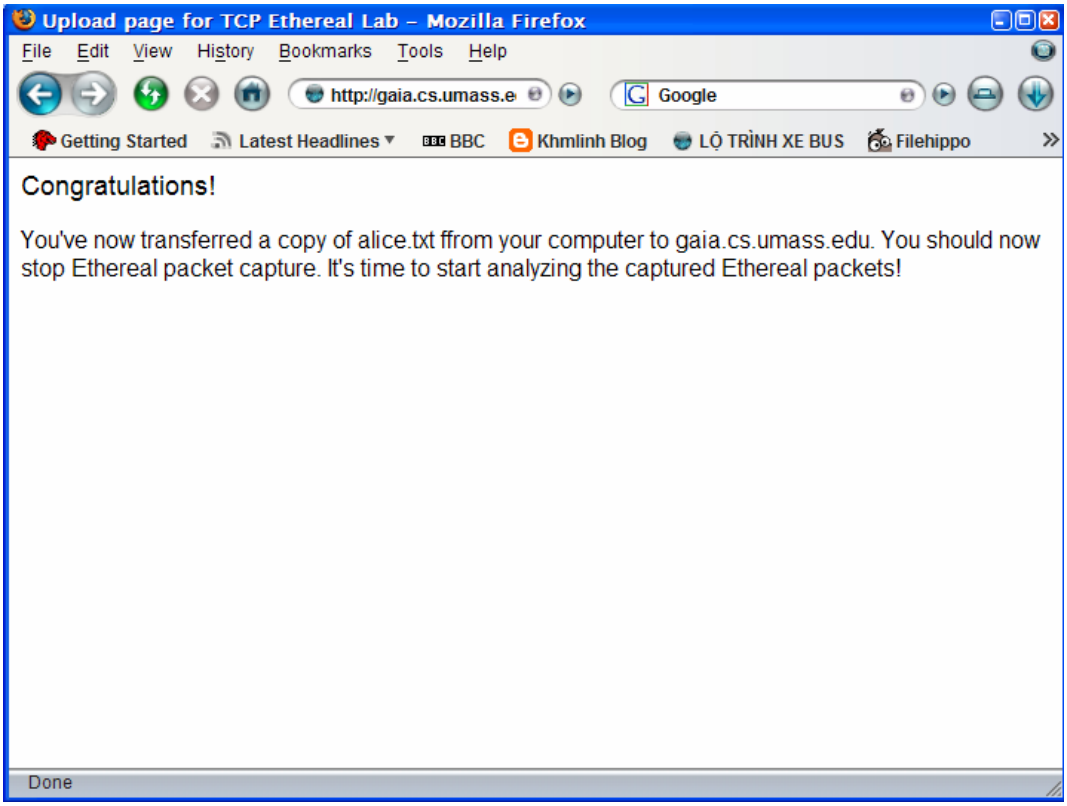
   `http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html`

   vào thanh địa chỉ của web-browser. Trên màn hình sẽ xuất hiện trang web có chứa phím *Browse* cho phép upload file như ở Hình 2.3.

   Sau khi chọn được file cần upload `alice.txt` ở trong thư mục `D:/Wireshark-Labs`, bấm phím *Upload* để POST file lên server. Sau khi file đã được upload lên server, sẽ có thông báo file đã được upload thành công, yêu cầu bạn **Stop** capture để bắt đầu phân tích dữ liệu như ở Hình 2.4.

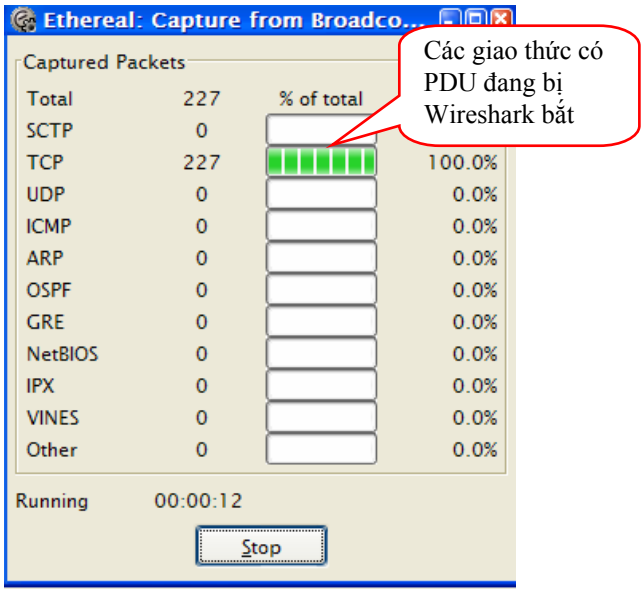


Hình 2.3 Trang web cho phép upload file dữ liệu thí nghiệm



Hình 2.4 Màn hình thông báo kết quả upload file `alice.txt`

Quan sát trên cửa sổ Capture Window của Wireshark sẽ thấy có thông tin về các gói đang được bắt như ở Hình 2.5. Tuy nhiên, lúc này chưa nên bấm phím *Stop* vội mà nên chuyển đến bước ***Ngắt kết nối*** ở mục sau để Wireshark bắt thêm các gói trao đổi trong quá trình ngắt kết nối.



Hình 2.5 Cửa sổ Capture Window hiển thị các packet thuộc các giao thức khác nhau đang bị bắt

2.4.4 Ngắt kết nối

6. **Bước 6:** Để ngắt kết nối TCP vừa thiết lập trong quá trình upload file, thực hiện đóng trình duyệt web brower (như Internet Explorer hoặc Firefox) bằng cách vào menu *File→ Close*.

2.4.5 Kết thúc capture

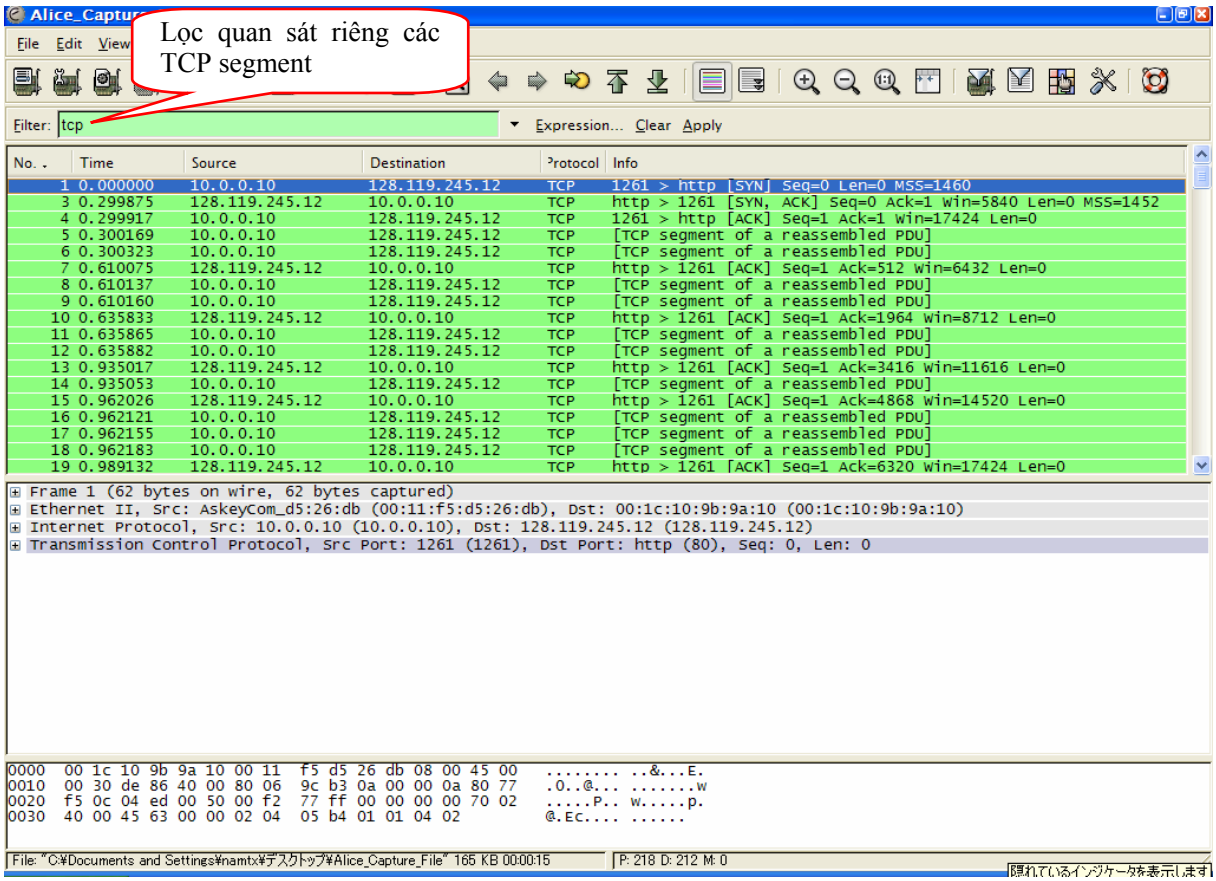
7. **Bước 7:** Để kết thúc bắt gói tin, bấm vào phím **Stop** trên Capture Window của Wireshark. Sau khi cửa sổ Capture đóng trên màn hình của Wireshark sẽ xuất hiện các thông tin về quá trình trao đổi dữ liệu của giao thức TCP như ở Hình 2.6. Các kết quả capture này có thể lưu trữ lại được để phục vụ cho việc phân tích sau này. Để lưu trữ kết quả capture, vào menu *File→Save as* của Wireshark, chọn thư mục thí nghiệm `D:/Wireshark-Labs` và đặt tên file mong muốn (ví dụ `Alice_Capture_file`) rồi bấm *Save*. Chú ý là sau khi đã lưu trữ file dữ liệu capture được, bạn vẫn có thể sử dụng cửa sổ của file đã lưu trữ để thực hiện phân tích giao thức ở bước 8.

   Chú ý: trong trường hợp máy tính nối đến mạng nội bộ Intranet của Học viện không

truy nhập được đến mạng Internet, học viên có thể thiết lập một kết nối TCP thông qua giao thức HTTP bằng cách thực hiện đọc trang chủ của Học viện tại địa chỉ

`http://www.mta.edu.vn`

Các thao tác thực hiện tiếp theo tương tự như trường hợp upload file `alice.txt` thực hiện ở trên. Giao thức HTTP sẽ thực hiện gọi giao thức TCP và yêu cầu thiết lập một kết nối TCP từ client là máy tính của học viên đến web-server lưu trữ trang chủ của website Học viện. Trường hợp không thấy có các gói hiển thị trên cửa sổ Capture cần bấm nút **Refresh** hoặc **Reload** trên web browser của bạn để cưỡng bức quá trình trao đổi dữ liệu



Hình 2.6: Cửa sổ chứa thông tin về quá trình trao đổi dữ liệu của giao thức TCP

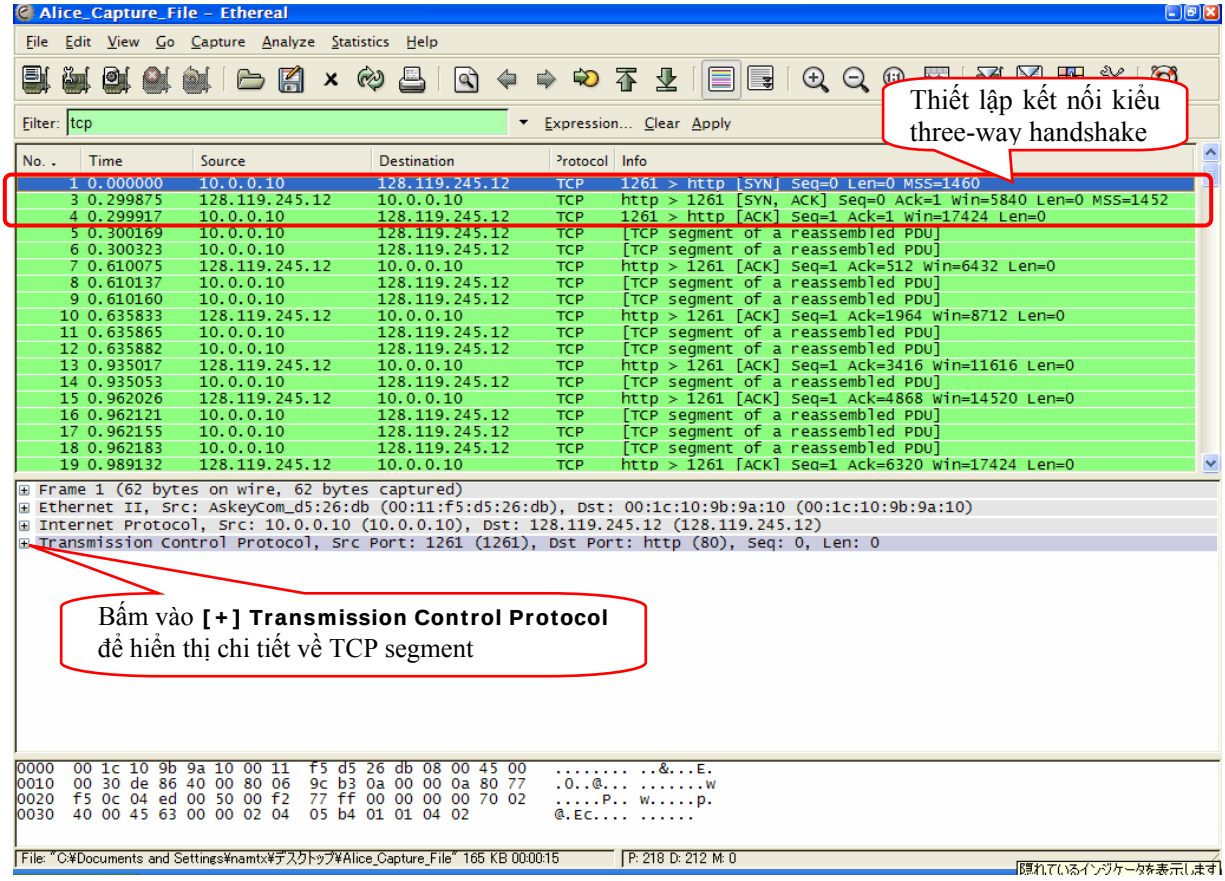
2.4.6 Phân tích quá trình trao đổi dữ liệu

Để tiến hành phân tích quá trình trao đổi dữ liệu, sử dụng cửa sổ Wireshark chứa các thông tin capture được như ở Hình 2.6. Trường hợp sử dụng lại dữ liệu đã lưu trữ cần load lại file đã lưu trữ bằng cách vào menu *File→Open* của Wireshark và chọn file đã lưu trữ. Để quan sát quá trình trao đổi dữ liệu của giao thức TCP, trên cửa sổ lọc quan sát của Wireshark nhập vào tcp (tất cả chữ thường) và bấm Enter để lọc quan sát riêng các TCP segments. Minh họa quá

trình này được chỉ ra ở Hình 2.6.

a/ Quá trình thiết lập kết nối kiểu three-way handshake

8. **Bước 8:** Quá trình thiết lập kết nối kiểu three-way handshake: trên cửa sổ Wireshark, dựa trên các TCP segment [SYN] chúng ta có thể theo dõi quá trình thiết lập kết nối diễn ra giữa máy tính local client của học viên và remote webserver theo hai hướng Client→ Server và Server→Client như mô tả trên Hình 2.7.

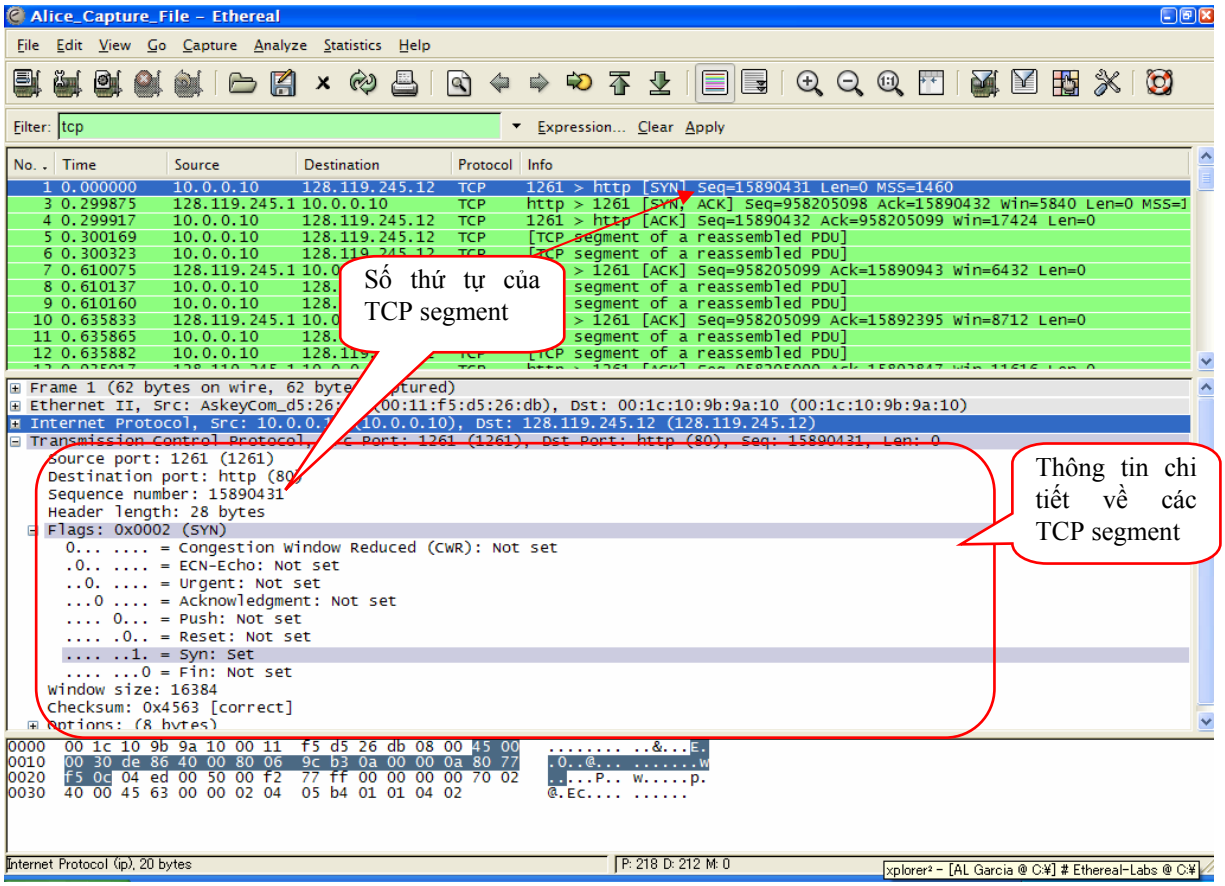


Hình 2.7 Quá trình thiết lập kết nối TCP

Để quan sát chi tiết hơn các thông về các TCP segment, bấm vào dấu [+] Transmission Control Protocol như minh họa trên Hình 2.7. Các thông tin chi tiết như Source port, Destination port, Sequence number, Header length, Flags, Window size, Checksum sẽ được hiển thị. Để hiển thị thêm các thông tin chi tiết về các bit trên các trường đầu khung tương ứng với segment điều khiển, bấm vào dấu [+] Flags. Lúc này chúng ta đã có đầy đủ các thông tin chi tiết về các TCP segment. Minh họa về các thông tin chi tiết này được chỉ ra ở Hình 2.8.

b/ Trao đổi dữ liệu các TCP segment

9. **Bước 9:** Lựa chọn số thứ tự ban đầu cho hướng thu và hướng phát: một đặc điểm của TCP là cho phép thỏa thuận số thứ tự của segment trong quá trình thiết lập kết nối nhằm hạn chế thu nhầm các gói đến bị trễ từ kết nối trước đó. Trong trường hợp minh họa ở Hình 2.8, số thứ tự (sequence number) thỏa thuận ban đầu `Seq=15890431.` Trường hợp số thứ tự thỏa thuận ban đầu hiển thị `Seq=0` biểu thị số thứ tự tương đối của các khung Wireshark bắt được. Để chuyển hiển thị về số thứ tự thực (tuyệt đối), vào *Edit→Preference→Protocol→TCP*, uncheck `Relative sequence number and window scaling.`



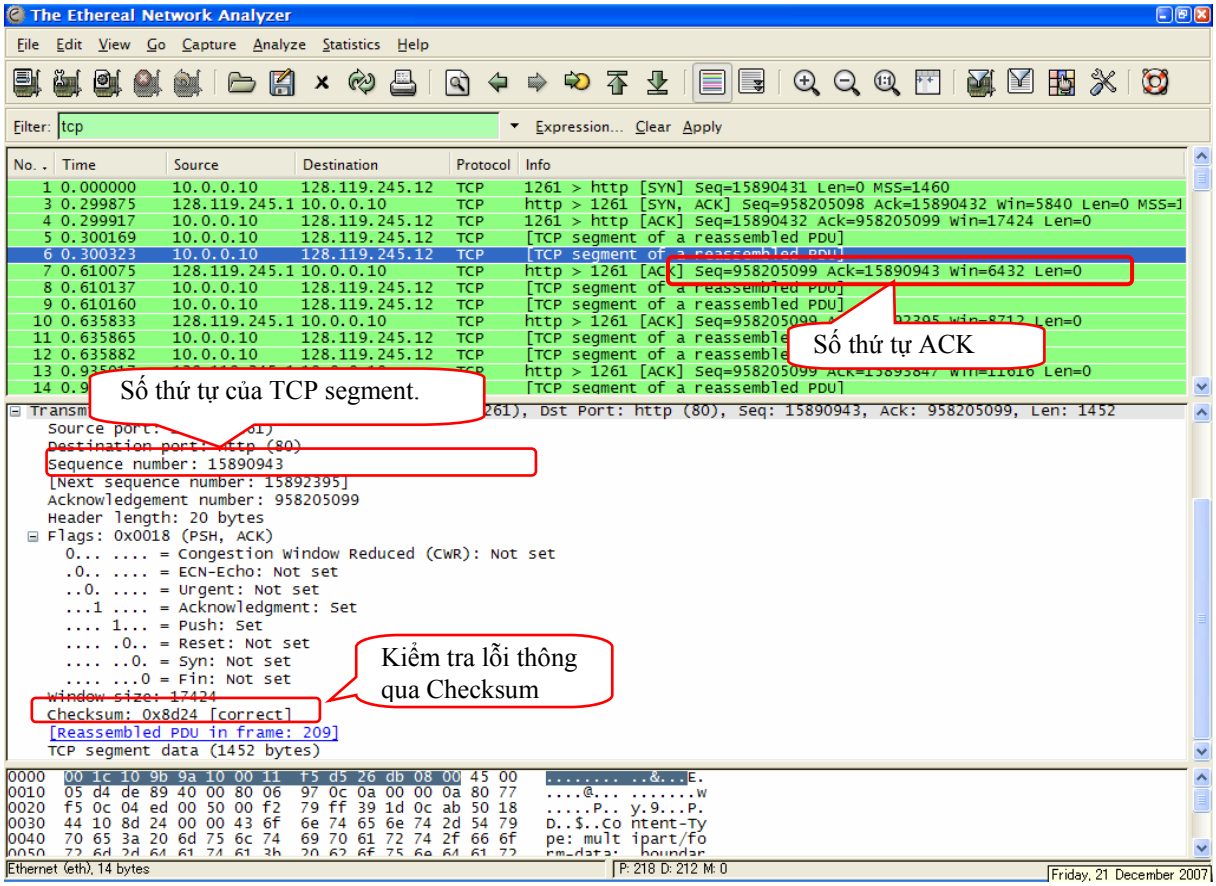
Hình 2.8 Thông tin chi tiết về TCP segment

10. **Bước 10:** Phúc đáp cho segment đã thu được: để đảm bảo cung cấp dịch vụ truyền dẫn tin cậy, TCP thực hiện phúc đáp các gói thu được không có lỗi bằng cách gửi các ACK trở lại cho máy phát. Để biết một TCP segment có bị lỗi hay không, máy thu thực hiện kiểm tra checksum của TCP header. Trường hợp checksum thỏa mãn điều kiện kiểm tra, sẽ có chỉ thị checksum correct như ở Hình 2.9.

c/ Quá trình điều khiển lỗi

Quá trình điều khiển lỗi diễn ra theo các bước sau đây. Ở bước thứ nhất, checksum được kiểm tra xem có thỏa mãn điều kiện kiểm tra hay không. Nếu thỏa mãn, sẽ có thông báo checksum correct. Trường hợp checksum chính xác, số thứ tự của khung sẽ được kiểm tra ở bước thứ hai để xem có segment bị mất hay không? Trường hợp nếu có segment bị mất hoặc checksum không chính xác (Hình 2.9) máy thu sẽ yêu cầu máy phát phát lại segment đó theo phương thức Selective Repeat ARQ.

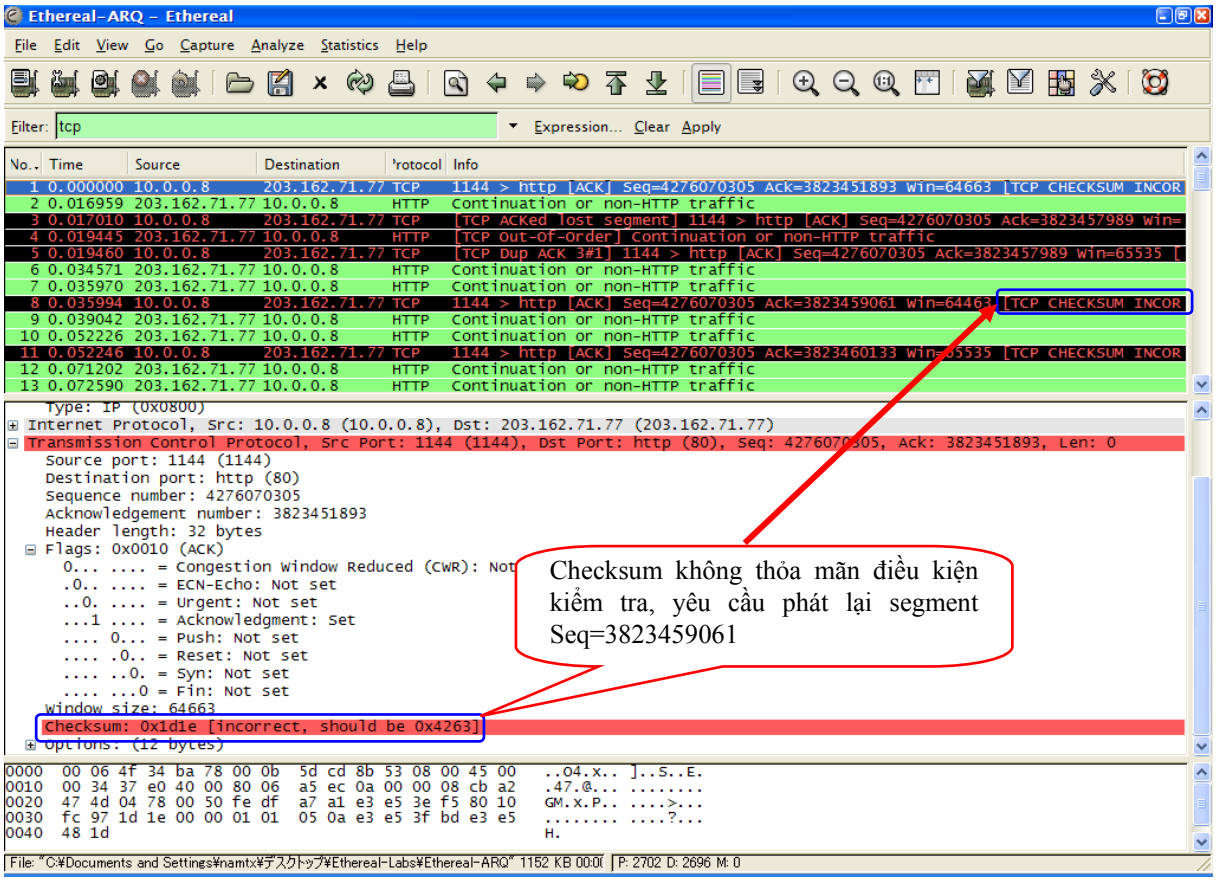
11. **Bước 11:** Kiểm tra checksum: sau khi nhận được một TCP segment từ lớp IP chuyển lên, giao thức TCP phía thu sẽ kiểm tra tính chính xác của checksum. Trường hợp checksum bị sai tương ứng với segment bị lỗi, TCP phía thu sẽ gửi một ACK yêu cầu phát lại segment bị lỗi như ở Hình 2.9. Trên cửa sổ packet list và packet detail của Wireshark sẽ thấy có thông báo tương ứng là Checksum incorrect và [TCP CHECKSUM INCORRECT].



Hình 2.9 Quá trình trao đổi TCP segment

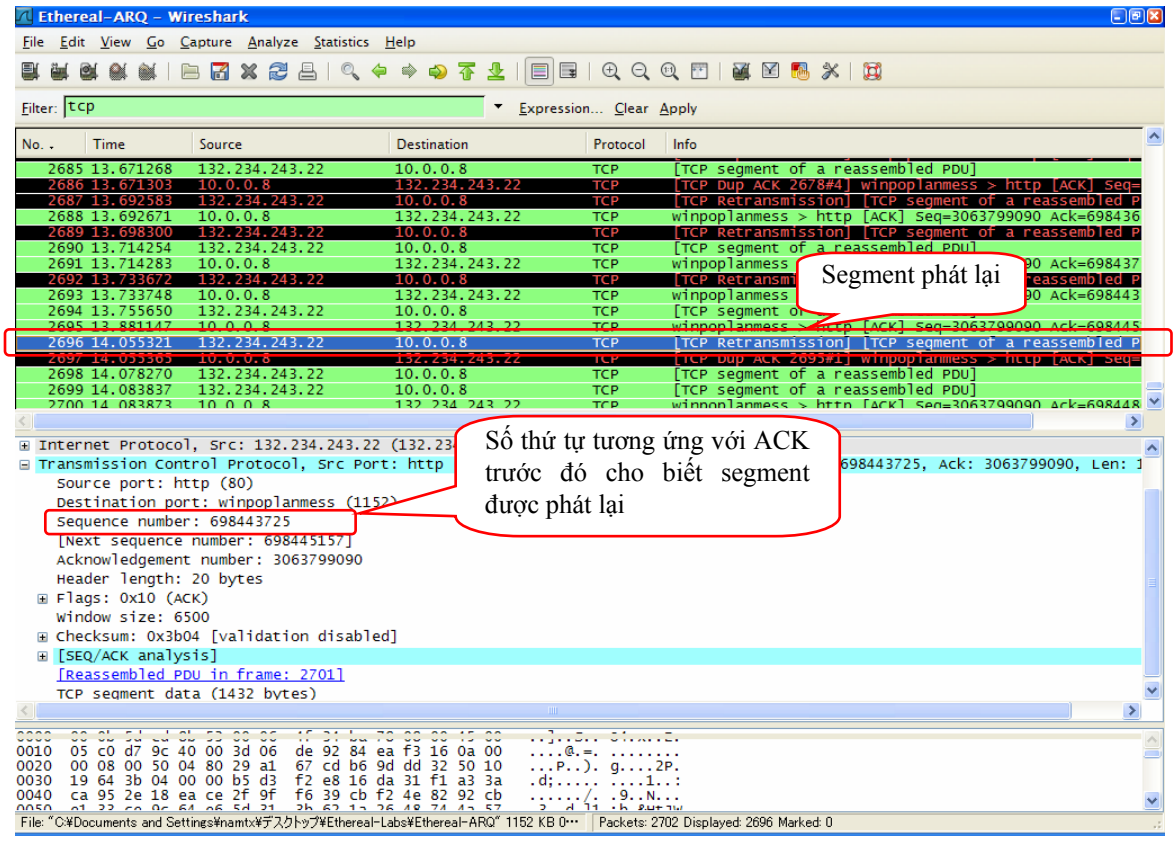
12. **Bước 12:** Tự động phát lại một segment có lỗi hoặc bị mất sử dụng Selective Repeat ARQ: khi có một segment bị mất hoặc không thỏa mãn điều kiện checksum, nhận được

ACK yêu cầu phát lại segment lỗi, TCP phát sẽ phát lại segment lỗi như ở Hình 2.10. Để xác định chi tiết về segment được phát lại click chuột vào dòng có thông tin có chứa segment đó và quan sát ở cửa sổ packet details.

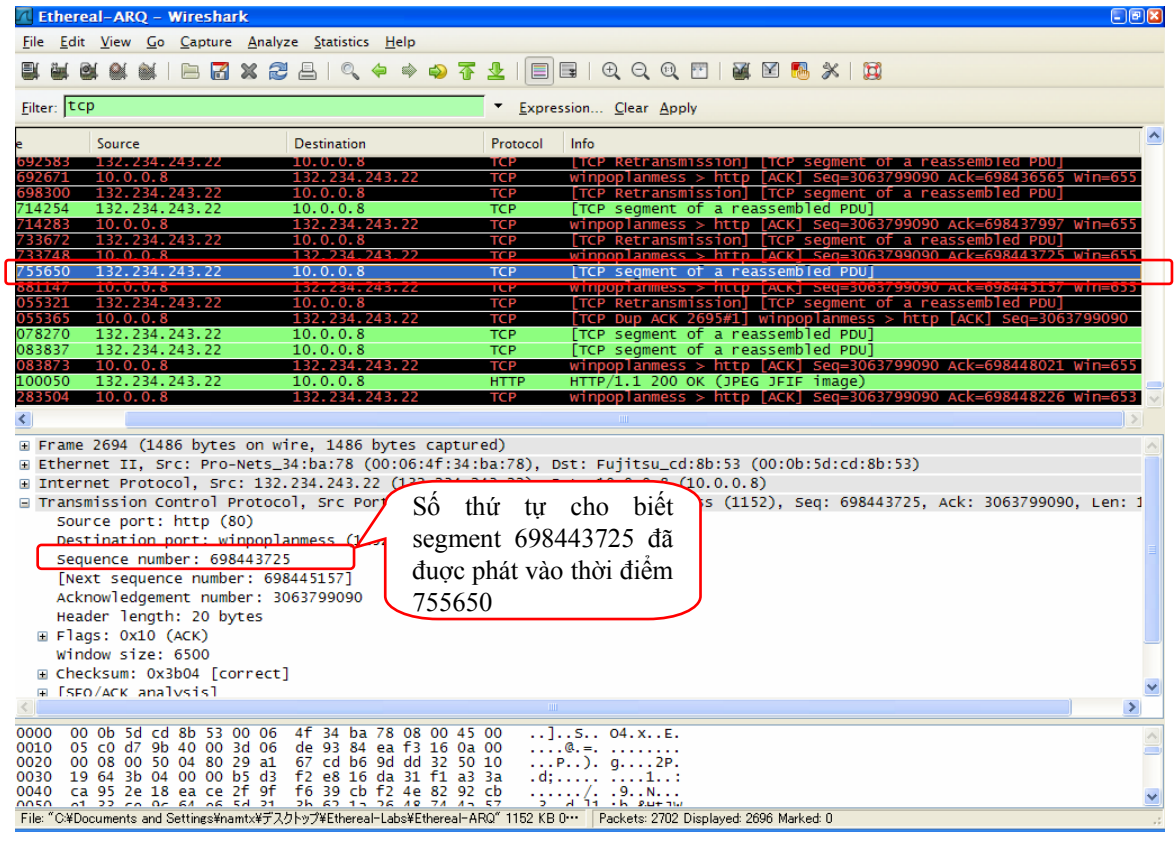


Hình 2.10 Kiểm tra checksum của TCP segment

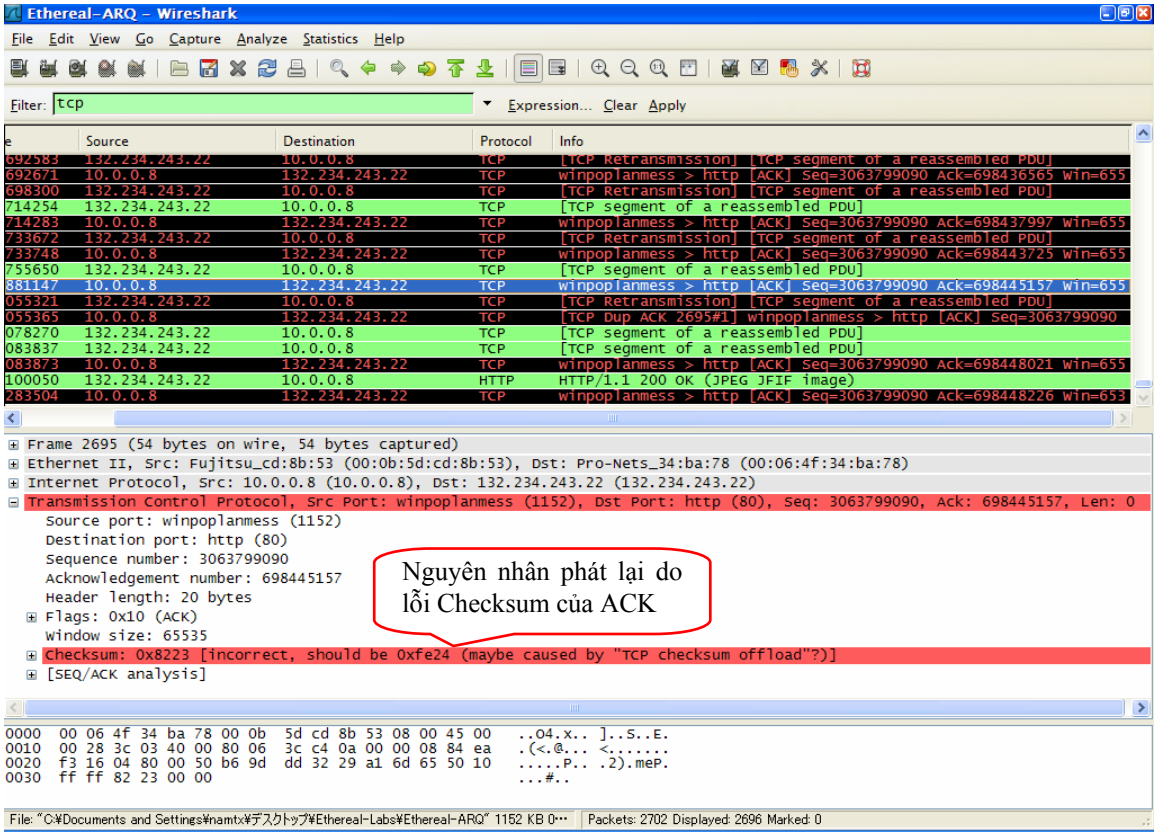
Lấy ví dụ cho sự kiện 055321 ở trường hợp Hình 2.10 chúng ta thấy ở cột packet info có thông tin [TCP Retransmission] cho biết đây là segment được phát lại. Để biết thêm nguyên nhân phát lại chúng ta click con trỏ vào dòng chứa thông tin về segment này (xem Hình 2.11). Ở cửa sổ packet details chúng ta thấy segment phát lại có số thứ tự Sequence_number 698443725. Click con trỏ vào dòng chứa segment đã phát trước đó và quan sát ở cửa sổ packet details chúng ta thấy segment 698443725 đã được phát vào thời điểm 755650 như ở Hình 2.12. Để tìm ra nguyên nhân gây nên phát lại, click vào sự kiện 881147 ở dưới tương ứng với [ACK] cho segment 698443725, và xem xét thông tin ở cửa sổ packet details, chúng ta thấy nguyên nhân gây phát lại là do ACK cho segment 698443725 bị lỗi checksum nên bị coi là có lỗi (Hình 2.13). Vì vậy, máy phát TCP đã tự động phát lại segment 698443725.



Hình 2.11 Thông tin về segment bị phát lại



Hình 2.12 Xác định segment phát đi bị lỗi



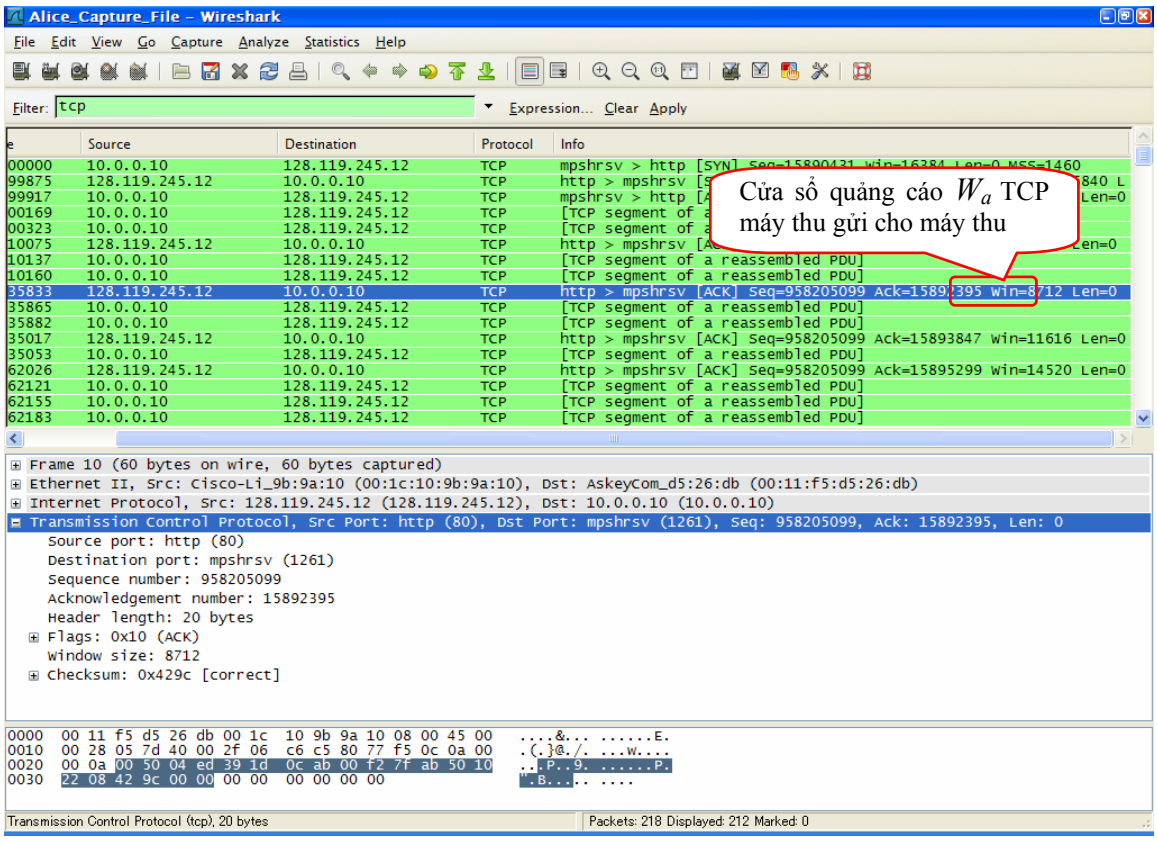
Hình 2.13 Lỗi checksum của ACK gây phát lại

Với cách phân tích tương tự chúng ta có thể tìm ra các nguyên nhân phát lại do mất gói khi thấy có segment thu được kèm theo thông báo [TCP Retransmission] và các sự kiện trước đó với thông báo [Previous segment lost].

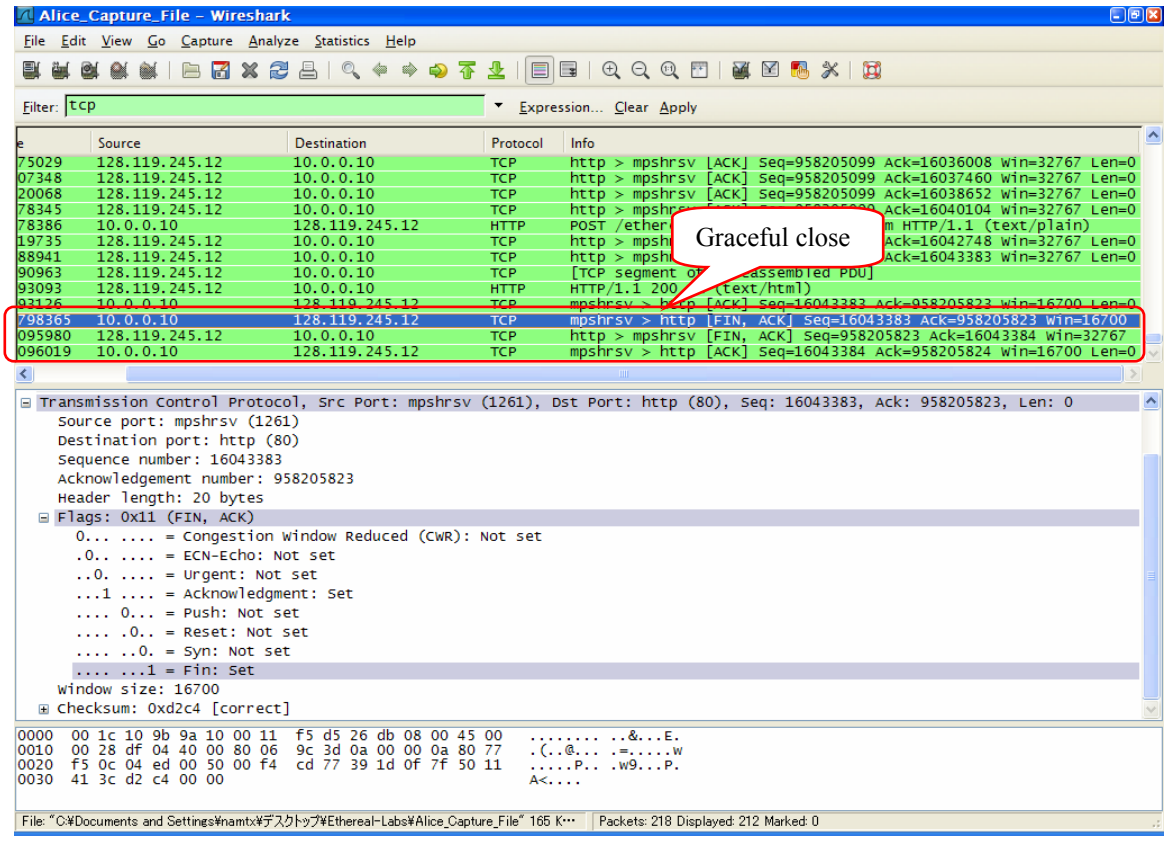
c/ Điều khiển luồng

Để thực hiện điều khiển luồng, giao thức TCP ở máy thu điều khiển giao thức TCP ở máy phát sao cho máy phát không phát được quá số lượng segment cho phép tránh làm tràn bộ nhớ đệm máy thu, gây nên mất gói không cần thiết. Để làm được việc này máy thu sử dụng một cửa sổ quảng cáo (advertise window) W_a dựa trên tốc độ xử lý và bộ nhớ đệm máy thu. Cửa sổ quảng cáo W_a được tính bởi công thức: $W_a = W_R - (R_{new} - R_{last})$. Máy phát có trách nhiệm duy trì điều kiện: $(S_{recent} - S_{last}) \leq W_a$. Thông qua quá trình điều khiển này mà tốc độ dữ liệu truyền từ máy phát sang máy thu có thể điều khiển được.

13. **Bước 13:** Cửa sổ quảng cáo W_a do máy thu cung cấp: quan sát trên các dòng chứa thông tin của các ACK gửi từ máy thu đến máy phát, chúng ta có thể xác định được cửa sổ quảng cáo W_a. Trên ví dụ ở Hình 2.14, cửa sổ quảng cáo W_a=8712 bytes, ở lần ACK trước đó W_a=6432 bytes, và ở lần ACK tiếp theo là 11616.



Hình 2.14: Cửa sổ quảng cáo máy thu TCP gửi đến cho máy phát



Hình 2.15: Ngắt kết nối kiểu graceful close của TCP

2.4.7 Ngắt kết nối graceful close

TCP thực hiện ngắt kết nối kiểu graceful close, tức là ngắt kết nối riêng cho mỗi hướng.

14. **Bước 14:** Ngắt kết nối kiểu graceful close: quá trình ngắt kết nối diễn ra cho cả hai hướng từ Client→Server và từ Server→Client như trên Hình 2.15. Quá trình diễn ra theo 3 bước do yêu cầu ngắt kết nối từ Server→Client thực hiện cõng AKC theo phương pháp piggyback.

2.5 Nội dung thí nghiệm cần báo cáo

- Hoàn thành 14 bước thí nghiệm nói trên, ghi lại màn hình và kết quả phân tích cho từng bước
- Trả lời các câu hỏi sau đây:
  1. Địa chỉ IP và số TCP port sử dụng bởi máy tính client (nguồn) đang truyền tải file tới gaia.cs.umass.edu? Để trả lời câu hỏi này, thì cách tốt nhất là chọn một bản tin HTTP và xem xét chi tiết TCP packet được sử dụng để mang bản tin HTTP này, sử dụng "details of the selected packet header window".
  2. Địa chỉ IP của gaia.cs.umass.edu? Server đang gửi và nhận các TCP segment bằng cổng nào cho kết nối này?
  3. Số thứ tự của TCP SYN segment được sử dụng để khởi tạo kết nối TCP giữa máy tính client và gaia.cs.umass.edu? Phần nào trong segment xác định segment là một SYN segment?
  4. Số thứ tự của SYNACK segment gửi bởi gaia.cs.umass.edu đến máy tính khi phúc đáp lại SYN segment? Giá trị của trường ACKnowledgement ở SYNACK segment? gaia.cs.umass.edu xác định giá trị đó như thế nào? Phần nào xác định segment là một SYNACK segment?
  5. Số thứ tự của TCP segment chứa câu lệnh HTTP POST? Chú ý rằng để tìm được lệnh POST, bạn cần tìm hiểu kỹ trường nội dung packet ở phía dưới của cửa sổ Wireshark window, tìm một segment với "POST" trong trường DATA của nó.
  6. Coi TCP segment chứa HTTP POST như là segment đầu tiên ở kết nối TCP. Xác định số thứ tự của 6 segments đầu tiên ở kết nối TCP (bao gồm cả segment chứa HTTP POST)? Mỗi segment được gửi tại những thời điểm nào? ACK được nhận khi

nào cho mỗi segment?

7. Xác định độ dài của từng segment trong số 6 TCP segment đầu tiên?

8. Xác định kích thước buffer quảng cáo W_a ở máy thu cho 6 TCP segment đầu tiên?

9. Có segment phát lại nào trong trace file không? Làm cách nào để kiểm tra trace để trả lời câu hỏi này?

10. Máy thu thường sử dụng bao nhiêu dữ liệu trong một ACK? Bạn có thể xác định các trường hợp máy thu xác nhận từng segment thu được.

11. Vào *Statistics→TCP Stream Graph→Throughput Graph*, vẽ đồ thị throughput và phân tích kết quả

Bài 3

GIAO THỨC IP

3.1 Mục đích

Trong bài thí nghiệm này học viên sẽ nghiên cứu giao thức lớp mạng IP (Internet Protocol) sử dụng trong Internet. Học viên sẽ tiến hành quá trình bắt các IP datagram trao đổi giữa máy tính client của học viên và một máy tính khác trên Internet. Sau khi bắt được một trace (vết) các IP datagrams, học viên sẽ tiến hành phân tích các trường dữ liệu trong IP datagram, và nghiên cứu chi tiết thao tác phân đoạn trong giao thức IP. Kết thúc bài thí nghiệm học viên sẽ nắm chắc hoạt động của giao thức IP, củng cố thêm kiến thức đã học trên lớp.

3.2 Phương pháp

Để phân tích hoạt động của giao thức IP, chúng ta sẽ sử dụng chương trình `traceroute` để phát đi các packet [đối với Unix và Linux là các UDP datagram với các port đích trong khoảng 33434 to 33534, còn với Windows là các ICMP (Internet Control Message Protocol) echo request] đến một địa chỉ đích định trước trên mạng IP và nhận lại các packet phản hồi từ các router, và nhờ đó biết được route đi từ host đích đến host nguồn. Để theo dõi quá trình trao đổi thông tin giữa trạm nguồn, các router và địa chỉ đích, chúng ta sử dụng chương trình network protocol analyzer Wireshark để bắt và lưu lại một trace của các IP datagram và packet đã phát và thu.

3.3 Bắt các gói nhờ chương trình `traceroute`

Để tạo một trace của các IP datagrams cho thí nghiệm, chúng ta sẽ sử dụng chương trình `traceroute` để gửi đi các packet có kích thước khác nhau tới một địa chỉ *X* nào đó trên Internet. Nguyên tắc làm việc của chương trình `traceroute`, dựa trên chương trình `ping`, thực hiện trước tiên gửi đi một hoặc nhiều packet được đóng gói vào trong IP datagram với trường TTL (Time to Live) ở trong header được đặt bằng 1; sau đó gửi tiếp một loạt các

packet về phía cùng địa chỉ với giá trị TTL bằng 2; rồi bằng 3 và tương tự. Chú ý rằng mỗi khi nhận được một IP datagram, mỗi router sẽ giảm giá trị trường TTL ở IP datagram thu được đi 1. Khi giá trị của TTL bằng 0, router sẽ nhận biết đây là một IP datagram lỗi do thời gian tồn tại đã hết và sẽ tự động gửi trả lại host nguồn thông báo lỗi bằng một bản tin ICMP (type 11 – Time-to-live exceeded). Kết quả là một IP datagram với TTL bằng 1 do chương trình `traceroute` gửi sẽ làm cho router cách host nguồn một hop tự động gửi một bản tin "ICMP Time-to-live exceeded" ngược trở về phía host nguồn; IP datagram được gửi với TTL bằng 2 sẽ làm cho router cách 2 hop gửi một bản tin ICMP ngược về phía nguồn; và tương tự. Dựa vào địa chỉ nguồn chứa trong các bản tin ICMP vượt quá TTL, chúng ta có thể xác định được địa chỉ của router gửi bản tin ICMP. Và như vậy, bằng cách này, host chạy chương trình `traceroute` có thể xác định được các routers đặt giữa nó và đích *X*.

Để sử dụng `traceroute` trong Windows, chúng ta có thể sử dụng chương trình `tracert` có sẵn từ command window. Tuy nhiên, do chương trình `tracert` trong Windows không cho phép thay đổi kích thước của bản in yêu cầu tiếng vọng ICMP (ping) do chương trình tracert gửi, nên không thể sử dụng nó để nghiên cứu quá trình phân đoạn trong giao thức IP. Vì vậy, trong bài thí nghiệm này chúng ta sẽ sử dụng một chương trình shareware hỗ trợ `traceroute` là PingPlotter.

3.4 Chuẩn bị bài thí nghiệm

Để phục vụ cho bài thí nghiệm, học viên cần chuẩn bị các bước sau đây:

- Chương trình `tracert` được sử dụng phổ biến cho windows là PingPlotter. Để có PingPlotter, có thể download từ địa chỉ http://www.pingplotter.com/download.html. Ở địa chỉ của trang PingPlotter, có 3 chương trình là *Freeware*, *Standard* và *Pro*. Phiên bản *Freeware* hiện tại (version 3.20.1) không cho phép thay đổi kích thước của gói, trong khi phiên bản *Pro* lại không cho chạy thử, nên chỉ có bản *Standard* có thể sử dụng được ở dạng shareware với 30 ngày dùng thử, đủ cho thời gian làm thí nghiệm của học viên. Sau khi đã download được chương trình PingPlotter phiên bản *Standard,* học viên cần cài đặt vào trong máy tính để phục vụ cho thí nghiệm sau này.
- Máy tính có nối tới mạng Internet (qua mạng LAN hay ADSL).
- Chương trình network protocol anlyzer Wireshark hay phiên bản cũ của nó là Ethereal, đã được cài đặt vào trong máy tính.

3.5 Nội dung bài thí nghiệm

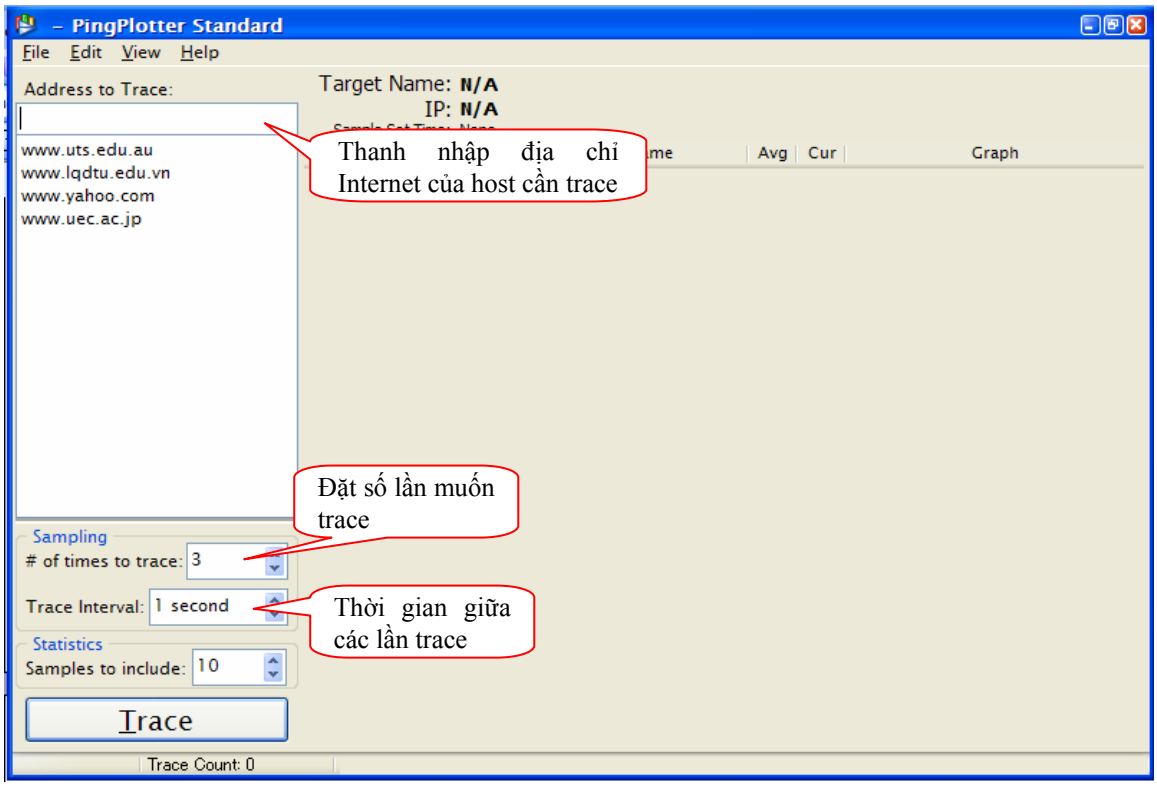
Nội dung bài thí nghiệm gồm các bước chính sau đây:

3.5.1 Thiết lập tham số trace

1. **Bước 1**: Khởi động Wireshark và bắt đầu bắt gói *(Capture→Start),* sau đó bấm *OK* trên màn hình Packet Capture Options của cửa sổ. Xem hướng dẫn chi tiết ở Bài 2.

2. **Bước 2:** Chọn một địa chỉ Internet để chương trình `tracert` (PingPlotter) tạo ra một trace đến host có địa chỉ đó. Ví dụ: địa chỉ www.uts.edu.au của trường đại học University of Technology Sydney (UTS).

3. **Bước 3:** Khởi động PingPlotter. Sau khi khởi động xong trên màn hình sẽ xuất hiện cửa sổ chương trình như trên Hình 3.1.

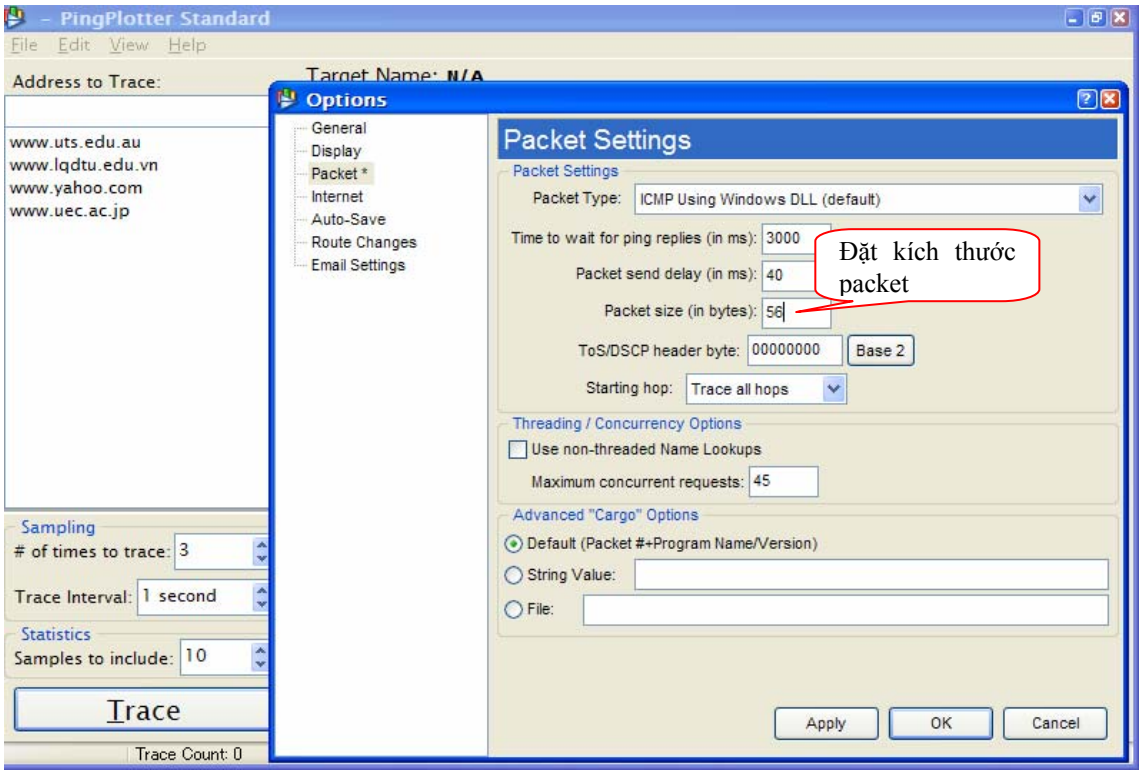


Hình 3.1 Cửa sổ chương trình PingPlotter

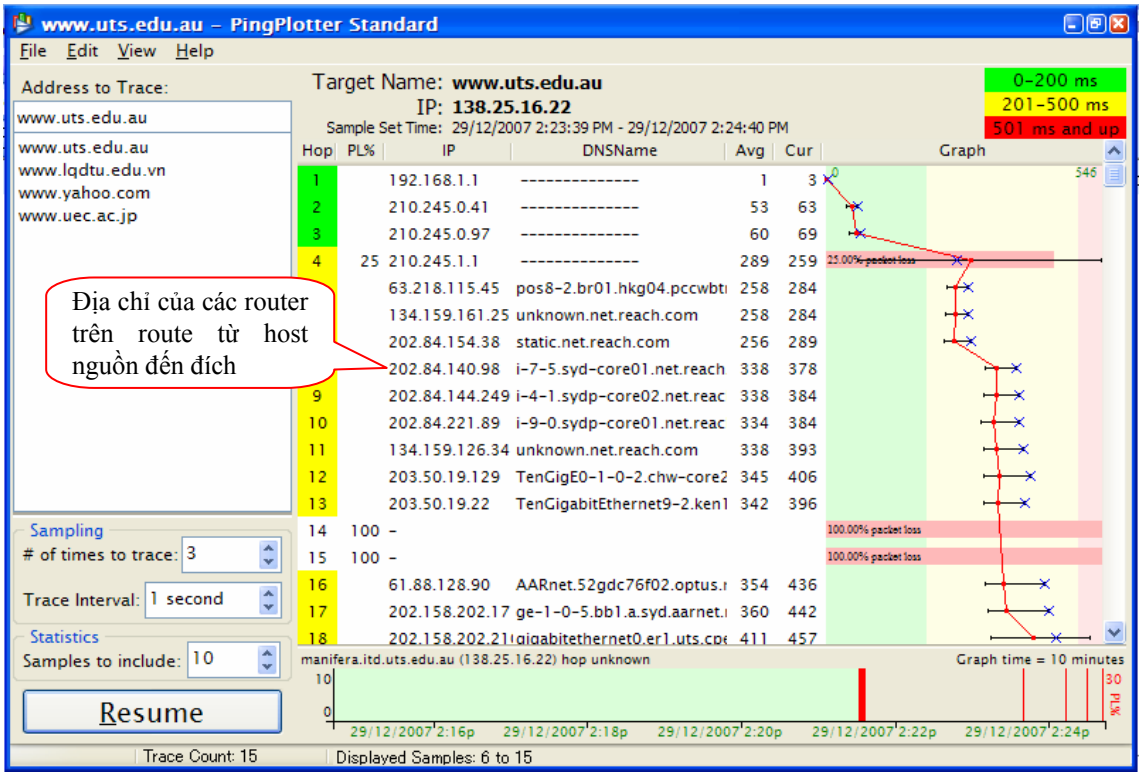
Một số điểm đáng chú ý trên cửa sổ là thanh nhập địa chỉ host cần trace *Address to Trace*, phần cài đặt *Sampling* với các trường thiết lập số lần *trace # of times to trace,* và khoảng thời gian giữa các lần trace *Trace Interval*. Trong phạm vi bài thí nghiệm này chúng ta đặt 3 cho *# of times to trace.*

3.5.2 Bắt đầu trace

4. **Bước 4:** trên thanh menu của PingPlotter, chọn menu *Edit→Advanced Options→Packet Options* và nhập vào giá trị 56 ở trường *Packet Size* và bấm OK như mô tả trên Hình 3.2. Sau đó bấm phím *Trace*. Thao tác này cho phép chúng ta gửi đi các ICMP echo request packet có kích thước bằng 56bytes về host đích là www.uts.edu.au và nhận về các bản tin ICMP Time-to-live exceeded. Bạn sẽ thấy có cửa sổ PingPlotter tương tự ở Hình 3.1 dưới đây.



Hình 3.2 Đặt kích thước packet



Hình 3.3 Cửa sổ trace của PingPlotter

Tiếp theo, gửi một tập hợp các packet có độ dài lớn hơn bằng cách chọn *Edit→Advanced Options→Packet Options* và nhập vào giá trị 2000 (bytes) ở trường *Packet Size* và bấm OK. Sau đó bấm phím Resume. Cuối cùng, gửi một tập các packet với độ dài lớn hơn bằng cách chọn *Edit→ Advanced Options → Packet Options* và nhập vào giá trị 3500 (bytes) ở trường *Packet Size* và sau đó bấm OK. Sau đó bấm phím Resume.

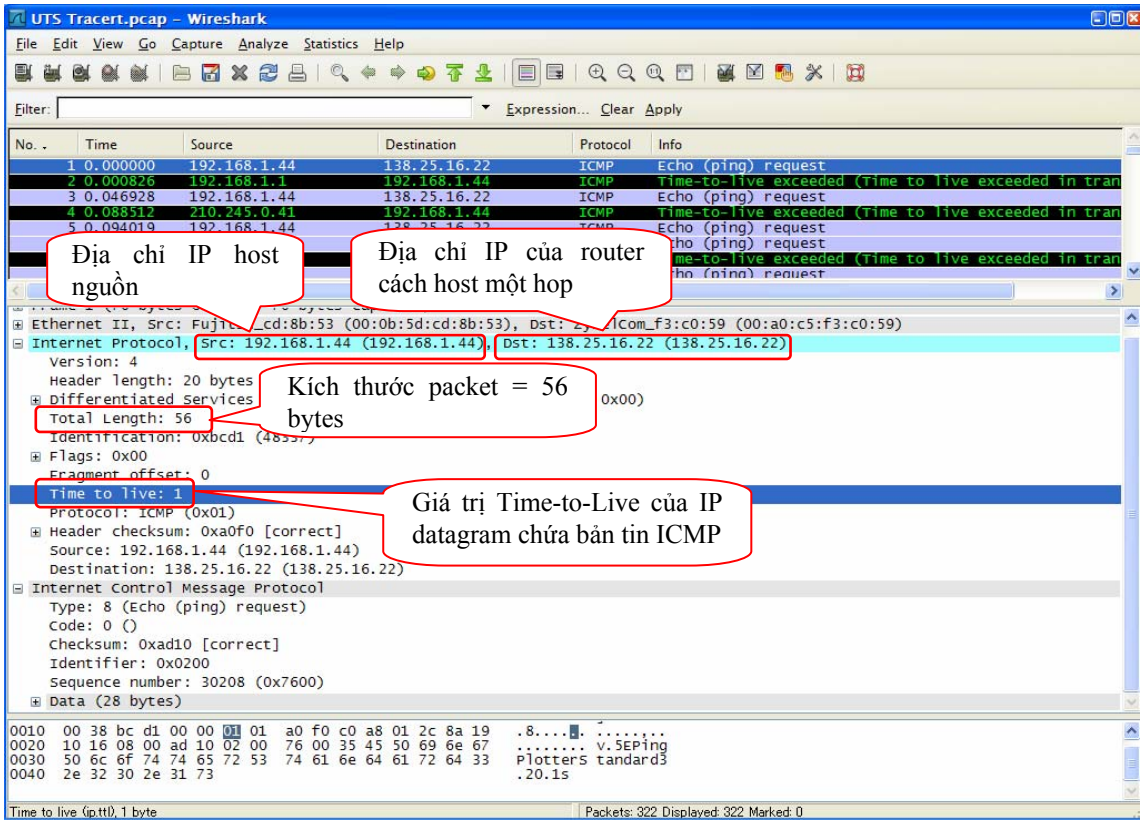
3.5.3 Kết thúc trace

5. **Bước 5:** Để kết thúc trace, dừng chương trình Wireshark bằng cách bấm Stop ở cửa sổ Capture của Wireshark.

3.5.4 Phân tích trace bắt được

Ở trong trace do Wireshark thu được, bạn sẽ thấy một loạt các packet "ICMP Echo Request" gửi bởi host nguồn là máy tính của bạn và các packet chứa bản tin "ICMP Time-to-Live exceeded" do các router trên tuyến gửi ngược trở lại về máy tính của bạn.

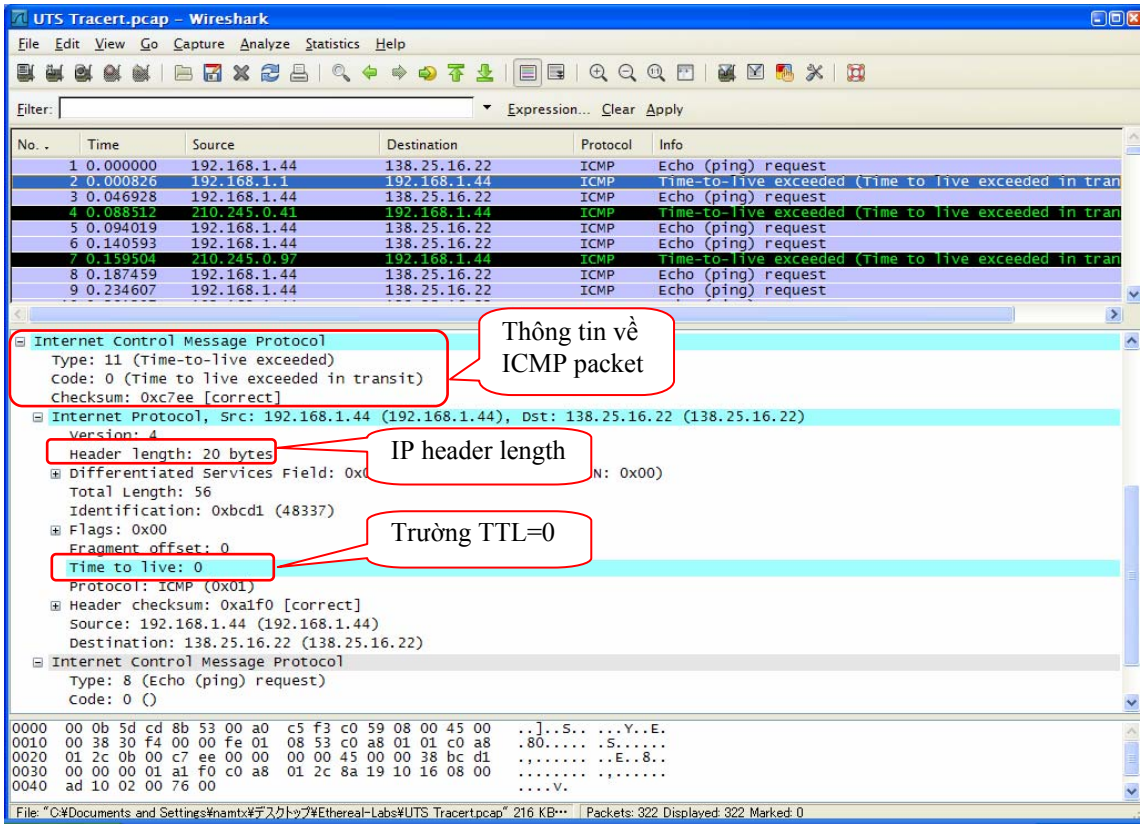
6. **Bước 6:** Bản tin "ICMP Echo Request". Trên cửa sổ phân tích của Wireshark chọn bản tin bắt được đầu tiên như ở Hình 3.4



Hình 3.4 Cửa sổ Wireshark hiển thị thông tin của packet đầu tiên

Trong phần hiển thị thông tin về header của các packet, chọn [+] Internet Protocol để xem các thông tin về header của IP datagram chứa ICMP echo (ping) request packet. Dựa trên thông tin hiển thị trên cửa sổ (xem Hình 3.4) chúng ta có thể xác định được địa chỉ IP của host nguồn là 192.168.1.44, địa chỉ host đích là 138.25.16.22, và giá trị trường TLL của IP datagram bằng 1. Điều này cho chúng ta biết host www.uts.edu.au có địa chỉ IP là 138.25.16.22.

7. **Bước 7:** Bản tin "ICMP Time-to-Live exceeded". Trên cửa sổ phân tích của Wireshark, chọn bản tin bắt được thứ hai. Như thấy trên cửa sổ đây là packet chứa bản tin ICMP Time-to-Live exceeded.



Hình 3.5 Cửa sổ Wireshark hiển thị thông tin của packet ICMP Time-to-Live exceeded

Ở phần hiển thị thông tin về packet chúng ta có thể thấy địa chỉ nguồn (địa chỉ của router gửi ICMP packet) là 192.168.1.1 và địa chỉ đích [địa chỉ của host gửi ICMP echo (ping) request] là 192.168.1.44. Điều này có nghĩa là host (router) 192.168.1.1 là router đầu tiên trên tuyến đến host www.uts.edu.au mà chúng ta đang trace. Khi thu được IP datagram chứa ICMP echo (ping) request packet, router 192.168.1.1 giảm giá trị TTL đi 1. Do TTL=0, nên router 192.168.1.1 loại bỏ IP datagram đó và tự động gửi về host nguồn chứa bản tin "ICMP Time-to-Live exceeded".

Trên cửa sổ thông tin chi tiết của Wireshark chọn [+] Internet Control Message Protocol, rồi chọn tiếp [+] Internet Protocol để hiển thị các thông tin về ICMP packet và header của IP datagram như ở Hình 3.5. Trên Hình 3.5, ở phần hiển thị thông tin header của packet, [-] Internet Control Message Protocol → [-] Internet Protocol, chúng ta có thể thấy địa chỉ nguồn là 192.168.1.44 và địa chỉ đích là 138.25.16.22. Hai địa chỉ nguồn và đích này là hai địa chỉ nguồn và đích chứa trong IP datagram của packet" ICMP echo (ping) request" trước đó.

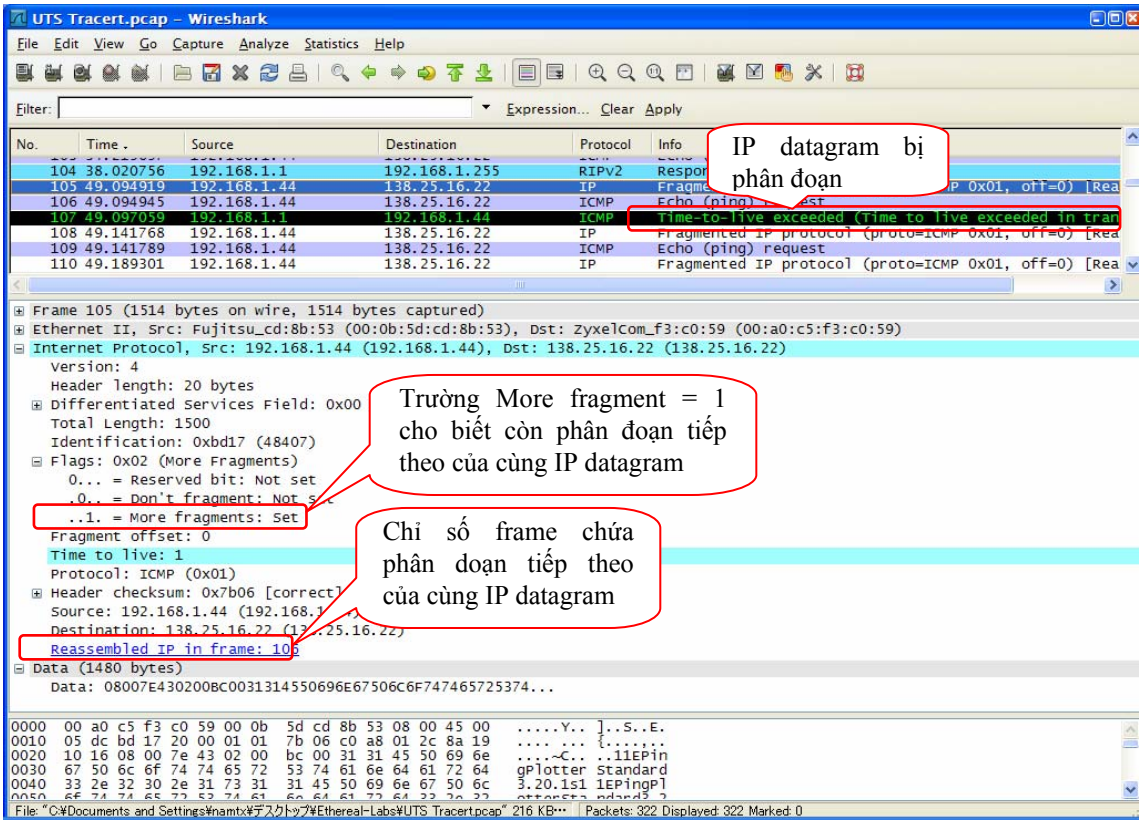
8. **Bước 8:** Kích thước IP header. Trên cửa sổ chi tiết về header của packet, ở phần [-]

Internet Protocol (xem Hình 3.5) chúng ta thấy có thông tin Header length: 20 bytes, cho biết độ dài header của IP datagram. Từ kích thước của IP datagram và IP header chúng ta có thể biết được độ dài của phần payload trong IP datagram.

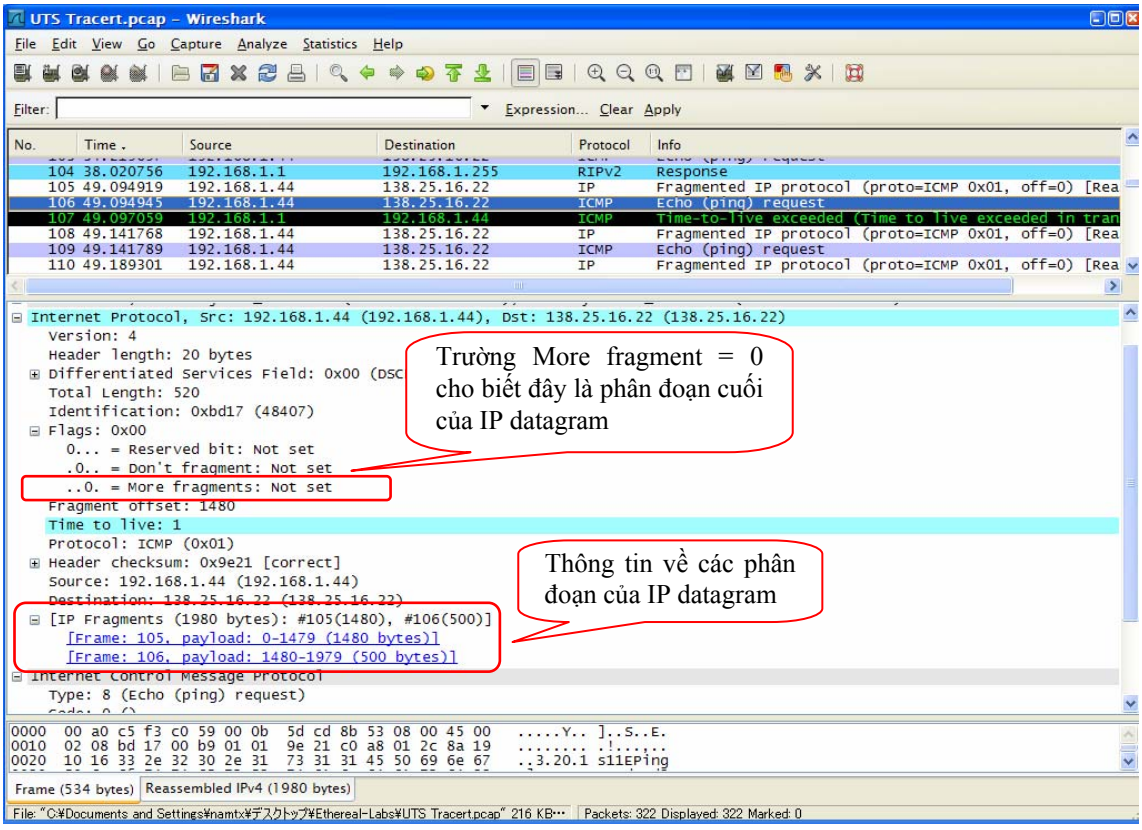
9. **Bước 9:** Phân đoạn IP (IP fragment). Theo lý thuyết chúng ta biết mạng Ethernet là lớp cung cấp dịch vụ cho lớp mạng IP chỉ cho phép truyền đi các frame với độ dài tối đa 1500 bytes. Vì vậy, các IP datagram có độ dài lớn hơn 1500 bytes sẽ bị phân đoạn và truyền trên các Ethernet frame khác nhau. Trong trường hợp bài thí nghiệm đang tiến hành, khi gửi đi các packet có kích thước 2000 bytes và 3500 bytes, Ethernet sẽ phân đoạn các packet đó thành các fragment và gửi đi trên các Ethernet frame liên tiếp nhau. Xét ví dụ trường hợp packet có kích thước 2000bytes. Do kích thước packet (chính là kích thước IP datagram) đặt là 2000 bytes nên trừ đi 20 bytes header, phần payload còn chứa 1980 bytes. Phần dữ liệu này được chia thành 2 hai đoạn: một có kích thước 1480 bytes để đóng khung vừa đủ vào 1500 bytes của Ethernet frame, và một đoạn thứ hai chứa phần payload còn lại, tức là, 500bytes.

Trên cửa sổ Captured packet list của Wireshark, chọn dòng sự kiện có chứa Fragmented IP protocol…như ở Hình 3.6(a). Trên cửa sổ chứa thông tin header chi tiết của packet, ở phần phía dưới của [-] Internet Prototocol có thông tin Reassembled IP in frame:106 cho biết packet hiện tại là một phân đoạn (fragement) của một IP datagram và phân đoạn còn lại của IP datagram được chứa trong frame số 106 tương ứng với dòng sự kiện thứ 106.

Chọn tiếp dòng tiếp theo (106 trong ví dụ này) sẽ thấy cửa sổ tương tự Hình 3.6(b). Dòng này chứa mô tả nội dung Echo (ping) request cho biết toàn bộ nội dung của IP datagram gửi trong phân đoạn trước và phân đoạn này chứa bản tin Echo (ping) request. Ở phần thông tin chi tiết của packet header chúng ta thấy các thông tin:



(a) Đoạn thứ nhất của một IP datagram



Hình 3.6 Phân đoạn IP. (b) Đoạn thứ hai của một IP datagram

[-] [IP fragments (1980) bytes: #105(1480), #106(500)]

[Frame: 105, payload: 0-1479 (1480 bytes)]

[Frame: 105, payload: 1480-1979 (500 bytes)]

Thông tin này, giống như đã phân tích ở trên, có nghĩa là tổng số payload trong IP datagram là 1980bytes, được phân thành 2 đoạn. Đoạn thứ nhất chứa 1480bytes, bao gồm các byte từ số 0 đến 1479 trong IP datagram gốc, được truyền đi bởi frame số 105. Đoạn thứ hai chứa 500 bytes còn lại, từ byte số 1480 đến byte 1979, và được truyền đi bởi frame số 106.

10. **Bước 10:** Bản tin "ICMP Echo Request". Trên cửa sổ phân tích của Wireshark chọn bản tin bắt được đầu tiên như ở Hình 3.4

3.6 Nội dung kết quả cần nộp

- Lặp lại 10 bước thí nghiệm ở trên, phân tích phân đoạn IP cho trường hợp gửi packet với kích thước 3500. In kết quả màn hình và gắn vào báo cáo.
- Thực hiện các nội dung phân tích tiếp theo và trả lời các câu hỏi sau đây:

  Sắp xếp các gói traced được theo địa chỉ IP nguồn bằng cách bấm vào *Source* column header; sẽ có một mũi tên chỉ xuống dưới (v) bên cạnh từ *Source*. Nếu mũi tên quay lên trên, click lại vào *Source* column header. Chọn bản tin ICMP Echo Request đầu tiên do máy tính gửi, và mở rộng phần Internet Protocol ở cửa sổ chi tiết của packet header. Ở phần cửa sổ liệt kê các packet capture được, sẽ thấy tất cả các bản tin ICMP kế tiếp phía dưới bản tin đầu ICMP tiên. Sử dụng mũi tên xuống dưới để chuyển qua các bản tin ICMP gửi bởi máy tính của bạn.

  1. Xác định các trường trong IP datagram thay đổi giữa các datagram trong loạt các bản tin ICMP gửi từ máy tính của bạn?
  2. Các trường nào không thay đổi? Các trường nào phải cố định? Các trường nào cần thay đổi? Giải thích tại sao?
  3. Xác định giá trị ở trường Identification và trường TTL?
  4. Các giá trị này có cố định với tất cả các phúc đáp ICMP TTL-exceeded gửi tới máy tính của bạn từ router gần nhất? Tại sao?

Tài liệu tham khảo

1. A. Leon-Garcia and I. Widjaja, *Communication Networks: Fundamental Concepts and Key Architectures*, McGraw-Hill, 1999.
2. F. Kurose and K.W. Ross, *Ethereal Labs*, www-net.cs.umass.edu/**ethereal-labs**/
3. www.wireshark.org

Phụ lục 1

The Transmission Control Protocol[3]

Abstract

It is important to understand TCP if one is to understand the historic, current and future architecture of the Internet protocols. Most applications on the Internet make use of TCP, relying upon its mechanisms that ensure safe delivery of data across an unreliable IP layer below. In this paper we explore the fundamental concepts behind TCP and how it is used to transport data between two endpoints.

A1.1. Introduction

The Transmission Control Protocol (TCP) standard is defined in the Request For Comment (RFC) standards document number 793 [10] by the Internet Engineering Task Force (IETF). The original specification written in 1981 was based on earlier research and experimentation in the original ARPANET. The design of TCP was heavily influenced by what has come to be known as the "end-to-end argument" [3].

As it applies to the Internet, the end-to-end argument says that by putting excessive intelligence in physical and link layers to handle error control, encryption or flow control you unnecessarily complicate the system. This is because these functions will usually need to be done at the endpoints anyway, so why duplicate the effort along the way? The result of an end-to-end network then, is to provide minimal functionality on a hop-by-hop basis and maximal control between end-to-end communicating systems.

The end-to-end argument helped determine how two characteristics of TCP operate; performance and error handling. TCP performance is often dependent on a subset of algorithms and techniques such as flow control and congestion control. Flow control determines the rate at which data is transmitted between a sender and receiver. Congestion

[3] Nguồn: http://condor.depaul.edu/~jkristof/technotes/tcp.html

control defines the methods for implicitly interpreting signals from the network in order for a sender to adjust its rate of transmission.

The term congestion control is a bit of a misnomer. Congestion avoidance would be a better term since TCP cannot control congestion per se. Ultimately intermediate devices, such as IP routers would only be able to control congestion.

Congestion control is currently a large area of research and concern in the network community. A companion study on congestion control examines the current state of activity in that area [9].

Timeouts and retransmissions handle error control in TCP. Although delay could be substantial, particularly if you were to implement real-time applications, the use of both techniques offer error detection and error correction thereby guaranteeing that data will eventually be sent successfully.

The nature of TCP and the underlying packet switched network provide formidable challenges for managers, designers and researchers of networks. Once regulated to low speed data communication applications, the Internet and in part TCP are being used to support very high speed communications of voice, video and data. It is unlikely that the Internet protocols will remain static as the applications change and expand. Understanding the current state of affairs will assist us in understanding protocol changes made to support future applications.

a/ Transmission Control Protocol

TCP is often described as a byte stream, connection-oriented, reliable delivery transport layer protocol. In turn, we will discuss the meaning for each of these descriptive terms.

b/ Byte Stream Delivery

TCP interfaces between the application layer above and the network layer below. When an application sends data to TCP, it does so in 8-bit byte streams. It is then up to the sending TCP to segment or delineate the byte stream in order to transmit data in manageable pieces to the receiver[1]. It is this lack of 'record boundaries" which give it the name "byte stream delivery service".

c/ Connection-Oriented

Before two communicating TCPs can exchange data, they must first agree upon the willingness to communicate. Analogous to a telephone call, a connection must first be made before two parties exchange information.

d/ Reliability

A number of mechanisms help provide the reliability TCP guarantees. Each of these is described briefly below.

Checksums. All TCP segments carry a checksum, which is used by the receiver to detect errors with either the TCP header or data.

Duplicate data detection. It is possible for packets to be duplicated in packet switched network; therefore TCP keeps track of bytes received in order to discard duplicate copies of data that has already been received.[2]

Retransmissions. In order to guarantee delivery of data, TCP must implement retransmission schemes for data that may be lost or damaged. The use of positive acknowledgements by the receiver to the sender confirms successful reception of data. The lack of positive acknowledgements, coupled with a timeout period (see timers below) calls for a retransmission.

Sequencing. In packet switched networks, it is possible for packets to be delivered out of order. It is TCP's job to properly sequence segments it receives so it can deliver the byte stream data to an application in order.

Timers. TCP maintains various static and dynamic timers on data sent. The sending TCP waits for the receiver to reply with an acknowledgement within a bounded length of time. If the timer expires before receiving an acknowledgement, the sender can retransmit the segment.

e/ TCP Header Format

Remember that the combination of TCP header and TCP in one packet is called a TCP segment. Figure 1 depicts the format of all valid TCP segments. The size of the header without options is 20 bytes. We will briefly define each field of the TCP header below.

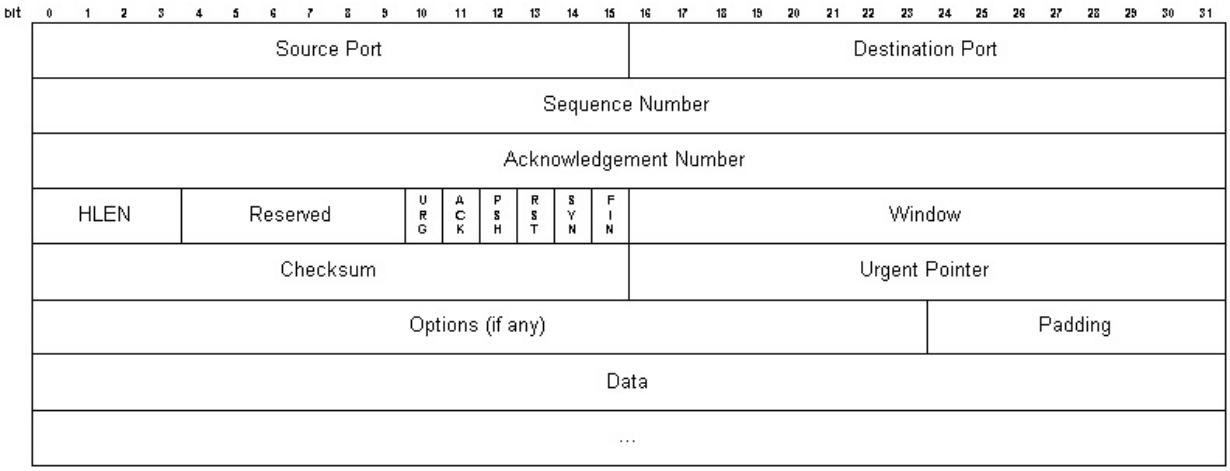


Figure 1 - TCP Header Format

f/ Source Port

A 16-bit number identifying the application the TCP segment originated from within the sending host. The port numbers are divided into three ranges, well-known ports (0 through 1023), registered ports (1024 through 49151) and private ports (49152 through 65535). Port assignments are used by TCP as an interface to the application layer. For example, the TELNET server is always assigned to the well-known port 23 by default on TCP hosts. A complete pair of IP addresses (source and destination) plus a complete pair of TCP ports (source and destination) define a single TCP connection that is globally unique. See [5] for further details.

g/ Destination Port

A 16-bit number identifying the application the TCP segment is destined for on a receiving host. Destination ports use the same port number assignments as those set aside for source ports [5].

h/ Sequence Number

A 32-bit number identifying the current position of the first data byte in the segment within the entire byte stream for the TCP connection. After reaching 2^{32} -1, this number will wrap around to 0.

i/ Acknowledgement Number

A 32-bit number identifying the next data byte the sender expects from the receiver. Therefore, the number will be one greater than the most recently received data byte. This field is only used when the ACK control bit is turned on (see below).

k/ Header Length

A 4-bit field that specifies the total TCP header length in 32-bit words (or in multiples of 4 bytes if you prefer). Without options, a TCP header is always 20 bytes in length. The largest a TCP header may be is 60 bytes. This field is required because the size of the options field(s) cannot be determined in advance. Note that this field is called "data offset" in the official TCP standard, but header length is more commonly used.

l/ Reserved

A 6-bit field currently unused and reserved for future use.

m/ Control Bits

Urgent Pointer (URG). If this bit field is set, the receiving TCP should interpret the urgent pointer field (see below).

Acknowledgement (ACK). If this bit field is set, the acknowledgement field described earlier is valid.

Push Function (PSH). If this bit field is set, the receiver should deliver this segment to the receiving application as soon as possible. An example of its use may be to send a Control-BREAK request to an application, which can jump ahead of queued data.

Reset the Connection (RST). If this bit is present, it signals the receiver that the sender is aborting the connection and all queued data and allocated buffers for the connection can be freely relinquished.

Synchronize (SYN). When present, this bit field signifies that sender is attempting to "synchronize" sequence numbers. This bit is used during the initial stages of connection establishment between a sender and receiver.

No More Data from Sender (FIN). If set, this bit field tells the receiver that the sender has reached the end of its byte stream for the current TCP connection.

n/ Window

A 16-bit integer used by TCP for flow control in the form of a data transmission window size. This number tells the sender how much data the receiver is willing to accept. The maximum value for this field would limit the window size to 65,535 bytes, however a "window scale" option can be used to make use of even larger windows.

o/ Checksum

A TCP sender computes a value based on the contents of the TCP header and data fields. This 16-bit value will be compared with the value the receiver generates using the same computation. If the values match, the receiver can be very confident that the segment arrived intact.

p/ Urgent Pointer

In certain circumstances, it may be necessary for a TCP sender to notify the receiver of urgent data that should be processed by the receiving application as soon as possible. This 16-bit field tells the receiver when the last byte of urgent data in the segment ends.

q/ Options

In order to provide additional functionality, several optional parameters may be used between a TCP sender and receiver. Depending on the option(s) used, the length of this field will vary in size, but it cannot be larger than 40 bytes due to the size of the header length field (4 bits). The most common option is the maximum segment size (MSS) option. A TCP receiver tells the TCP sender the maximum segment size it is willing to accept through the use of this option. Other options are often used for various flow control and congestion control techniques.

r/ Padding

Because options may vary in size, it may be necessary to "pad" the TCP header with zeroes so that the segment ends on a 32-bit word boundary as defined by the standard [10].

s/ Data

Although not used in some circumstances (e.g. acknowledgement segments with no data in the reverse direction), this variable length field carries the application data from TCP sender to receiver. This field coupled with the TCP header fields constitutes a TCP segment.

A1.2. Connection Establishment and Termination

TCP provides a connection-oriented service over packet switched networks. Connection-oriented implies that there is a virtual connection between two endpoints.[3] There are three phases in any virtual connection. These are the connection establishment, data transfer and connection termination phases.

A1.2.1 Three-Way Handshake

In order for two hosts to communicate using TCP they must first establish a connection by exchanging messages in what is known as the three-way handshake. The diagram below depicts the process of the three-way handshake.

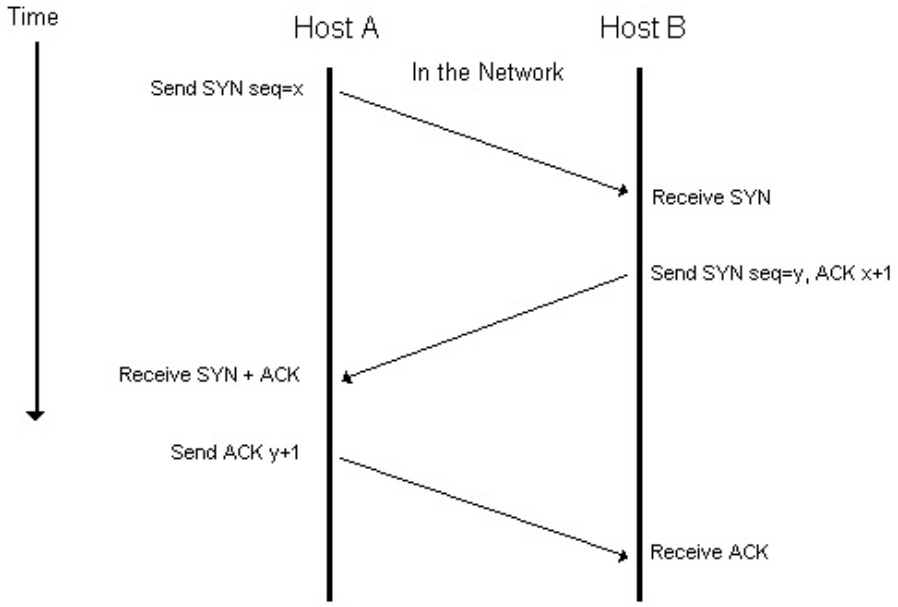


Figure 2 - TCP Connection Establishment

From figure 2, it can be seen that there are three TCP segments exchanged between two hosts, Host A and Host B. Reading down the diagram depicts events in time.

To start, Host A initiates the connection by sending a TCP segment with the SYN control bit set and an initial sequence number (ISN) we represent as the variable x in the sequence number field.

At some moment later in time, Host B receives this SYN segment, processes it and responds with a TCP segment of its own. The response from Host B contains the SYN control bit set and its own ISN represented as variable y. Host B also sets the ACK control bit to indicate the next expected byte from Host A should contain data starting with sequence number x+1.

When Host A receives Host B's ISN and ACK, it finishes the connection establishment phase by sending a final acknowledgement segment to Host B. In this case, Host A sets the ACK control bit and indicates the next expected byte from Host B by placing acknowledgement number y+1 in the acknowledgement field.

In addition to the information shown in the diagram above, an exchange of source and destination ports to use for this connection are also included in each senders' segments.[4]

A1.2.2 Data Transfer

Once ISNs have been exchanged, communicating applications can transmit data between each other. Most of the discussion surrounding data transfer requires us to look at flow control and congestion control techniques which we discuss later in this document and refer to other texts [9]. A few key ideas will be briefly made here, while leaving the technical details aside.

A simple TCP implementation will place segments into the network for a receiver as long as there is data to send and as long as the sender does not exceed the window advertised by the receiver. As the receiver accepts and processes TCP segments, it sends back positive acknowledgements, indicating where in the byte stream it is. These acknowledgements also contain the "window" which determines how many bytes the receiver is currently willing to accept. If data is duplicated or lost, a "hole" may exist in the byte stream. A receiver will continue to acknowledge the most current contiguous place in the byte stream it has accepted.

If there is no data to send, the sending TCP will simply sit idly by waiting for the application to put data into the byte stream or to receive data from the other end of the connection.

If data queued by the sender reaches a point where data sent will exceed the receiver's advertised window size, the sender must halt transmission and wait for further acknowledgements and an advertised window size that is greater than zero before resuming.

Timers are used to avoid deadlock and unresponsive connections. Delayed transmissions are used to make more efficient use of network bandwidth by sending larger "chunks" of data at once rather than in smaller individual pieces.[5]

A1.2.3 Connection Termination

In order for a connection to be released, four segments are required to completely close a connection. Four segments are necessary due to the fact that TCP is a full-duplex protocol, meaning that each end must shut down independently.[6] The connection termination phase is shown in figure 3 below.

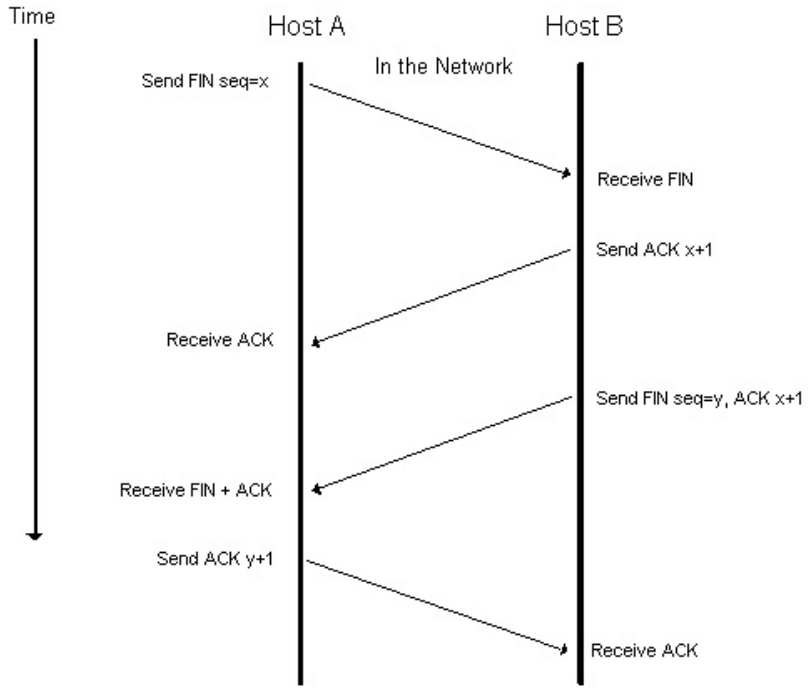


Figure 3 - TCP Connection Termination

Notice that instead of SYN control bit fields, the connection termination phase uses the FIN control bit fields to signal the close of a connection.

To terminate the connection in our example, the application running on Host A signals TCP to close the connection. This generates the first FIN segment from Host A to Host B. When Host B receives the initial FIN segment, it immediately acknowledges the segment and notifies its destination application of the termination request. Once the application on Host B also decides to shut down the connection, it then sends its own FIN segment, which Host A will process and respond with an acknowledgement.

A1.3. Sliding Window and Flow Control

Flow control is a technique whose primary purpose is to properly match the transmission rate of sender to that of the receiver and the network. It is important for the transmission to be at a high enough rate to ensure good performance, but also to protect against overwhelming the network or receiving host.

In [8], we note that flow control is not the same as congestion control. Congestion control is primarily concerned with a sustained overload of network intermediate devices such as IP routers.

TCP uses the window field, briefly described previously, as the primary means for flow control. During the data transfer phase, the window field is used to adjust the rate of flow of the byte stream between communicating TCPs.

Figure 4 below illustrates the concept of the sliding window.

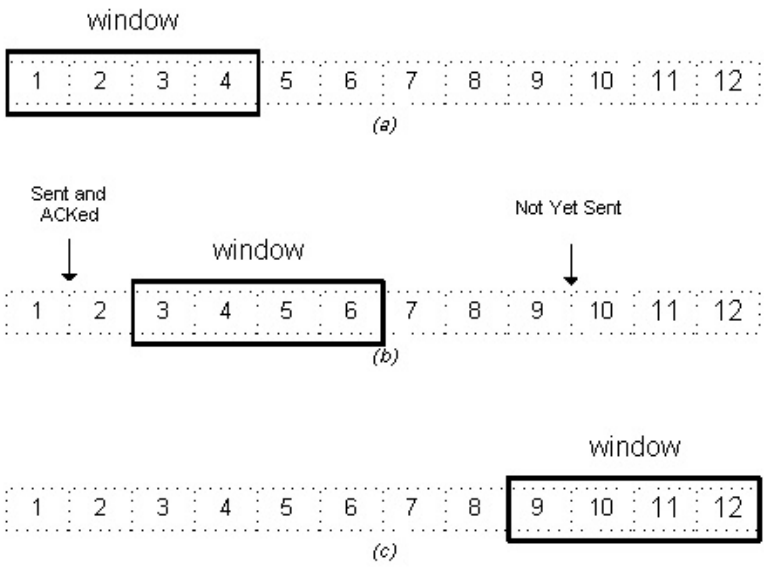


Figure 4 - Sliding Window

In this simple example, there is a 4-byte sliding window. Moving from left to right, the window "slides" as bytes in the stream are sent and acknowledged.[7] The size of the window and how fast to increase or decrease the window size is an area of great research. We again refer to other documents for further detail [9].

A1.4. Congestion Control

TCP congestion control and Internet traffic management issues in general is an active area of research and experimentation. This final section is a very brief summary of the standard congestion control algorithms widely used in TCP implementations today. These algorithms are defined in [6] and [7]. Their use with TCP was standardized in [1].

A1.4.1 Slow Start

Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate, otherwise known as sender-based flow control. This is accomplished through the return rate of acknowledgements from the receiver. In other words,

the rate of acknowledgements returned by the receiver determine the rate at which the sender can transmit data.

When a TCP connection first begins, the Slow Start algorithm initializes a congestion window to one segment, which is the maximum segment size (MSS) initialized by the receiver during the connection establishment phase. When acknowledgements are returned by the receiver, the congestion window increases by one segment for each acknowledgement returned. Thus, the sender can transmit the minimum of the congestion window and the advertised window of the receiver, which is simply called the transmission window.

Slow Start is actually not very slow when the network is not congested and network response time is good. For example, the first successful transmission and acknowledgement of a TCP segment increases the window to two segments. After successful transmission of these two segments and acknowledgements completes, the window is increased to four segments. Then eight segments, then sixteen segments and so on, doubling from there on out up to the maximum window size advertised by the receiver or until congestion finally does occur.

A1.4.2 Congestion Avoidance

During the initial data transfer phase of a TCP connection the Slow Start algorithm is used. However, there may be a point during Slow Start that the network is forced to drop one or more packets due to overload or congestion. If this happens, Congestion Avoidance is used to slow the transmission rate. However, Slow Start is used in conjunction with Congestion Avoidance as the means to get the data transfer going again so it doesn't slow down and stay slow.

In the Congestion Avoidance algorithm a retransmission timer expiring or the reception of duplicate ACKs can implicitly signal the sender that a network congestion situation is occurring. The sender immediately sets its transmission window to one half of the current window size (the minimum of the congestion window and the receiver's advertised window size), but to at least two segments. If congestion was indicated by a timeout, the congestion window is reset to one segment, which automatically puts the sender into Slow Start mode. If congestion was indicated by duplicate ACKs, the Fast Retransmit and Fast Recovery algorithms are invoked (see below).

As data is received during Congestion Avoidance, the congestion window is increased. However, Slow Start is only used up to the halfway point where congestion originally occurred. This halfway point was recorded earlier as the new transmission window. After this halfway point, the congestion window is increased by one segment for all segments in the transmission window that are acknowledged. This mechanism will force the sender to more

slowly grow its transmission rate, as it will approach the point where congestion had previously been detected.

A1.4.3 Fast Retransmit

When a duplicate ACK is received, the sender does not know if it is because a TCP segment was lost or simply that a segment was delayed and received out of order at the receiver. If the receiver can re-order segments, it should not be long before the receiver sends the latest expected acknowledgement. Typically no more than one or two duplicate ACKs should be received when simple out of order conditions exist. If however more than two duplicate ACKs are received by the sender, it is a strong indication that at least one segment has been lost. The TCP sender will assume enough time has lapsed for all segments to be properly re-ordered by the fact that the receiver had enough time to send three duplicate ACKs.

When three or more duplicate ACKs are received, the sender does not even wait for a retransmission timer to expire before retransmitting the segment (as indicated by the position of the duplicate ACK in the byte stream). This process is called the Fast Retransmit algorithm and was first defined in [7]. Immediately following Fast Retransmit is the Fast Recovery algorithm.

A1.4.4 Fast Recovery

Since the Fast Retransmit algorithm is used when duplicate ACKs are being received, the TCP sender has implicit knowledge that there is data still flowing to the receiver. Why? The reason is because duplicate ACKs can only be generated when a segment is received. This is a strong indication that serious network congestion may not exist and that the lost segment was a rare event. So instead of reducing the flow of data abruptly by going all the way into Slow Start, the sender only enters Congestion Avoidance mode.

Rather than start at a window of one segment as in Slow Start mode, the sender resumes transmission with a larger window, incrementing as if in Congestion Avoidance mode. This allows for higher throughput under the condition of only moderate congestion [23].

A1.5. Conclusions

TCP is a fairly complex protocol that handles the brunt of functionality in a packet switched network such as the Internet. Supporting the reliable delivery of data on a packet switched network is not a trivial task. This document only scratches the surface of the TCP internals, but hopefully provided the reader with an appreciation and starting point for further interest in TCP. Even after almost 20 years of standardization, the amount of work that goes into supporting and designing reliable packet switched networks has not slowed. It is an area of

great activity and there are many problems to be solved. As the Internet continues to grow, our reliance on TCP will become increasingly important. It is therefore imperative for network engineers, designers and researchers to be as well versed in the technology as possible.

---

1. The word "segment" is the term used to describe TCP's data unit size transmitted to a receiver. TCP determines the appropriate use of this segment size rather than leaving it up to higher layer protocols and applications.
2. Duplicate packets are typically caused by retransmissions, where the first packet may have been delayed and the second sent due to the lack of an acknowledgement. The receiver may then receive two identical packets.
3. As opposed to a connectionless-oriented protocol such as that used by the user datagram protocol (UDP).
4. There are additional details of the connection establishment, data transfer and termination phases that are beyond the scope of this document. For curious readers, I recommend consulting a more complete reference such as [4], [11] and of course the official standard RFC 793 [10].
5. It was discovered early on that some implementations of TCP performed poorly due to this scenario. It has been termed the silly window syndrome and documented in [2].
6. Although it is possible, it is not very common for TCP to be operating in the "half-close state". See [11] for further details.
7. We assume in this example that bytes are immediately acknowledged so that the window can move forward. In practice the sender's window shrinks and grows dynamically as acknowledgements arrive in time.

Abbreviations

ACK Acknowledgement

bit binary digit

IETF Internet Engineering Task Force

IP Internet Protocol

ISN Initial Sequence Number

RFC Request For Comments

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

UDP User Datagram Protocol

References

[1] Robert Braden. Requirements for Internet Hosts - Communication Layers, October 1989, RFC 1122.

[2] David D. Clark. Window Acknowledgement and Strategy in TCP, July 1982, RFC 813.

[3] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. In Proceedings SIGCOMM '88, Computer Communications Review Vol. 18, No. 4, August 1988, pp. 106-114).

[4] Douglas E. Comer. Internetworking with TCP/IP, Volume I: Principles, Protocols and Architecture. Prentice Hall, ISBN: 0-13-216987-8. March 24, 1995.

[5] Internet Assigned Numbers Authority. Port Number Assignment, February 2000.

[6] Van Jacobson. Congestion Avoidance and Control. Computer Communications Review, Volume 18 number 4, pp. 314-329, August 1988.

[7] Van Jacobson. Modified TCP Congestion Control Avoidance Algorithm. end-2-end-interest mailing list, April 30, 1990.

[8] S. Keshav. An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network. Addison Wesley, ISBN: 0-201-63442-2. July, 1997.

[9] John Kristoff. TCP Congestion Control, March 2000.

[10] Jon Postel. Transmission Control Protocol, September 1981, RFC 793.

[11] W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison Wesley, ISBN: 0-201-63346-9. January 1994.

Phụ lục 2

IP Fragment[4]

A2.1 Introduction

The purpose of this document is to present how IP Fragmentation and Path Maximum Transmission Unit Discovery (PMTUD) work and to discuss some scenarios involving the behavior of PMTUD when combined with different combinations of IP tunnels. The current widespread use of IP tunnels in the Internet has brought the problems involving IP Fragmentation and PMTUD to the forefront.

A2.2 IP Fragmentation and Reassembly

The IP protocol was designed for use on a wide variety of transmission links. Although the maximum length of an IP datagram is 64K, most transmission links enforce a smaller maximum packet length limit, called a MTU. The value of the MTU depends on the type of the transmission link. The design of IP accommodates MTU differences by allowing routers to fragment IP datagrams as necessary. The receiving station is responsible for reassembling the fragments back into the original full size IP datagram.

IP fragmentation involves breaking a datagram into a number of pieces that can be reassembled later. The IP source, destination, identification, total length, and fragment offset fields, along with the "more fragments" and "don't fragment" flags in the IP header, are used for IP fragmentation and reassembly. For more information about the mechanics of IP fragmentation and reassembly, please see RFC 791.

The image below depicts the layout of an IP header.

[4] Nguồn: http://www.cisco.com/warp/public/105/pmtud_ipfrag.html

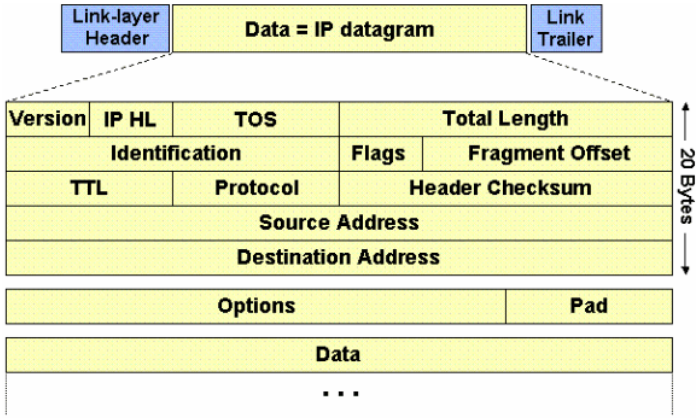


Figure 1. IP datagram header

The identification is 16 bits and is a value assigned by the sender of an IP datagram to aid in reassembling the fragments of a datagram.

The fragment offset is 13 bits and indicates where a fragment belongs in the original IP datagram. This value is a multiple of eight bytes.

In the flags field of the IP header, there are three bits for control flags. It is important to note that the "don't fragment" (DF) bit plays a central role in PMTUD because it determines whether or not a packet is allowed to be fragmented.

Bit 0 is reserved, and is always set to 0. Bit 1 is the DF bit (0 = "may fragment," 1 = "don't fragment"). Bit 2 is the MF bit (0 = "last fragment," 1 = "more fragments").

Value	Bit 0 Reserved	Bit 1 DF	Bit 2 MF
0	0	May	Last
1	0	Do not	More

The graphic below shows an example of fragmentation. If you add up all the lengths of the IP fragments, the value exceeds the original IP datagram length by 60. The reason that the overall length is increased by 60 is because three additional IP headers were created, one for each fragment after the first fragment.

The first fragment has an offset of 0, the length of this fragment is 1500; this includes 20 bytes for the slightly modified original IP header.

The second fragment has an offset of 185 (185 x 8 = 1480), which means that the data portion of this fragment starts 1480 bytes into the original IP datagram. The length of this fragment is 1500; this includes the additional IP header created for this fragment.

The third fragment has an offset of 370 (370 x 8 = 2960), which means that the data portion of this fragment starts 2960 bytes into the original IP datagram. The length of this fragment is 1500; this includes the additional IP header created for this fragment.

The fourth fragment has an offset of 555 (555 x 8 = 4440), which means that the data portion of this fragment starts 4440 bytes into the original IP datagram. The length of this fragment is 700 bytes; this includes the additional IP header created for this fragment.

It is only when the last fragment is received that the size of the original IP datagram can be determined.

The fragment offset in the last fragment (555) gives a data offset of 4440 bytes into the original IP datagram. If you then add the data bytes from the last fragment (680 = 700 - 20), that gives you 5120 bytes, which is the data portion of the original IP datagram. Then, adding 20 bytes for an IP header equals the size of the original IP datagram (4440 + 680 + 20 = 5140).

Original IP Datagram

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0	345	5140	0	0	0

IP Fragments (Ethernet)

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0-0	345	1500	0	1	0
0-1	345	1500	0	1	185
0-2	345	1500	0	1	370
0-3	345	700	0	0	555

A2.3 Issues with IP Fragmentation

There are several issues that make IP fragmentation undesirable. There is a small increase in CPU and memory overhead to fragment an IP datagram. This holds true for the sender as well as for a router in the path between a sender and a receiver. Creating fragments simply involves creating fragment headers and copying the original datagram into the fragments. This can be done fairly efficiently because all the information needed to create the fragments is immediately available.

Fragmentation causes more overhead for the receiver when reassembling the fragments because the receiver must allocate memory for the arriving fragments and coalesce them back into one datagram after all of the fragments are received. Reassembly on a host is not

considered a problem because the host has the time and memory resources to devote to this task.

But, reassembly is very inefficient on a router whose primary job is to forward packets as quickly as possible. A router is not designed to hold on to packets for any length of time. Also a router doing reassembly chooses the largest buffer available (18K) with which to work because it has no way of knowing the size of the original IP packet until the last fragment is received.

Another fragmentation issue involves handling dropped fragments. If one fragment of an IP datagram is dropped, then the entire original IP datagram must be resent, and it will also be fragmented. You see an example of this with Network File System (NFS). NFS, by default, has a read and write block size of 8192, so a NFS IP/UDP datagram will be approximately 8500 bytes (including NFS, UDP, and IP headers). A sending station connected to an Ethernet (MTU 1500) will have to fragment the 8500 byte datagram into six pieces; five 1500 byte fragments and one 1100 byte fragment. If any of the six fragments is dropped because of a congested link, the complete original datagram will have to be retransmitted, which means that six more fragments will have to be created. If this link drops one in six packets, then the odds are low that any NFS data can be transferred over this link, since at least one IP fragment would be dropped from each NFS 8500 byte original IP datagram.

Firewalls that filter or manipulate packets based on Layer 4 (L4) through Layer 7 (L7) information in the packet may have trouble processing IP fragments correctly. If the IP fragments are out of order, a firewall may block the non-initial fragments because they do not carry the information that would match the packet filter. This would mean that the original IP datagram could not be reassembled by the receiving host. If the firewall is configured to allow non-initial fragments with insufficient information to properly match the filter, then a non-initial fragment attack through the firewall could occur. Also, some network devices (such as Content Switch Engines) direct packets based on L4 through L7 information, and if a packet spans multiple fragments, then the device may have trouble enforcing its policies.