

```

    "Content-type: text/html\n" +
    "Connection: close\n" +
    "\n<HTML>\n" +
    "<HEAD>\n" +
    "<TITLE>Hello World</TITLE>\n" +
    "</HEAD>\n" +
    "<BODY> <a href = \"localhost\"> Hello World </a></BODY>\n" +
    "</HTML> \n\n";
}

class TestWebServer {
    public static void Main(String[] args) {
        SimpleWebServer webServer1 = new SimpleWebServer();
        webServer1.StartListening(8080);
    }
}

```

3.4. Giao thức FTP

3.4.1. Cơ bản về giao thức FTP

Giao thức FTP là một giao thức trao đổi file khá phổ biến

Khái niệm và hoạt động cơ bản của FTP:

Những người phát triển giao thức FTP cần phải cân bằng nhu cầu giữa việc phát triển một bộ giao thức vừa có nhiều chức năng và vừa đơn giản trong triển khai. Do đó, FTP không đơn giản như “đứa em trai” của nó – là giao thức TFTP. Hoạt động của giao thức này có thể chia ra thành nhiều thành phần nhỏ, hoạt động cùng nhau để thực hiện các công việc như khởi tạo kết nối, truyền thông tin điều khiển và truyền lệnh.

Nội dung mà bài viết này hướng tới là đưa tới người đọc những khái niệm quan trọng nhất đằng sau giao thức FTP, cũng như giải thích tổng quan về nguyên lý hoạt động của nó.

1 - Mô hình hoạt động của FTP, các thành phần trong giao thức, và các thuật ngữ cơ bản

Giao thức FTP được mô tả một cách đơn giản thông qua mô hình hoạt động của FTP. Mô hình này chỉ ra các nguyên tắc mà một thiết bị phải tuân theo khi tham gia vào quá trình trao đổi file, cũng như về hai kênh thông tin cần phải thiết lập giữa các thiết bị đó. Nó cũng mô tả các thành phần của FTP được dùng để quản lý các kênh này ở cả hai phía – truyền và nhận. Do đó, mô hình này tạo cho ta một khởi điểm lý tưởng để xem xét hoạt động của FTP ở mức khái quát.

Tiến trình Server-FTP và User-FTP

FTP là một giao thức dạng client/server truyền thông, tuy nhiên thuật ngữ client thông thường được thay thế bằng thuật ngữ user – người dùng – do thực tế là người sử dụng mới là đối tượng trực tiếp thao tác các lệnh FTP trên máy clients. Bộ phần mềm FTP được cài đặt trên một thiết bị được gọi là một tiến trình. Phần mềm FTP được cài đặt trên máy Server được gọi là tiến trình Server-FTP, và phần trên máy client được gọi là tiến trình User-FTP.

Kênh điều khiển và kênh dữ liệu trong FTP

Một khái niệm cốt yếu mà ta cần phải nắm về FTP là: mặc dù giao thức này sử dụng kết nối TCP, nhưng nó không chỉ dùng một kênh TCP như phần lớn các giao thức truyền thông khác. Mô hình FTP chia quá trình truyền thông giữa bộ phận Server với bộ phận client ra làm hai kênh logic:

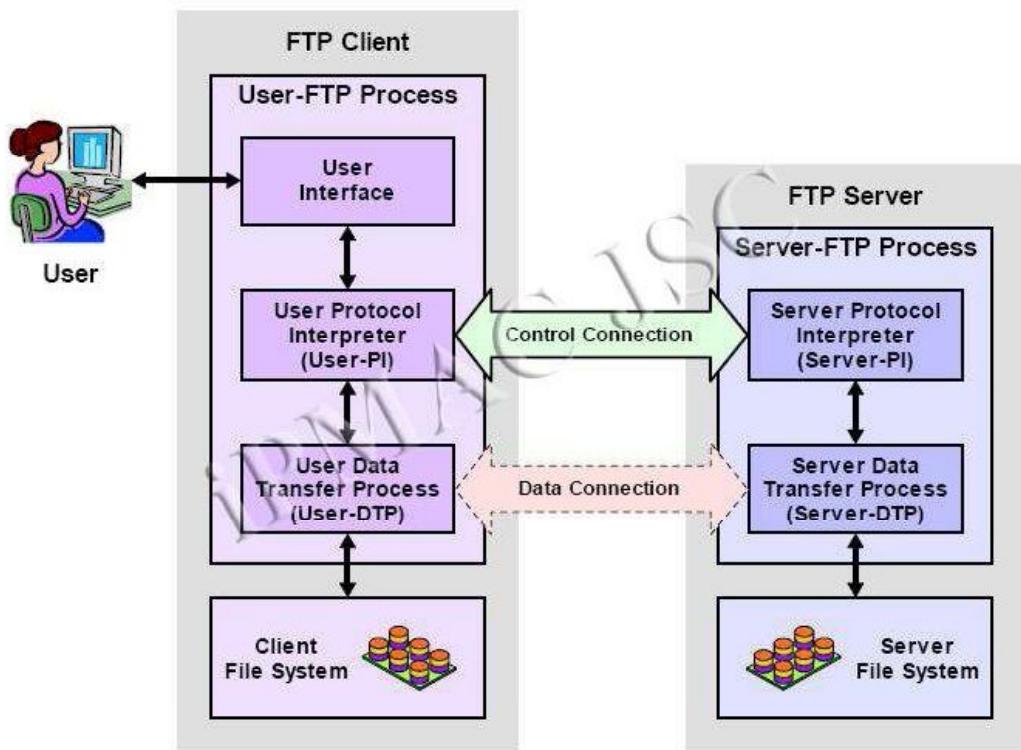
- Kênh điều khiển: đây là kênh logic TCP được dùng để khởi tạo một phiên kết nối FTP. Nó được duy trì xuyên suốt phiên kết nối FTP và được sử dụng chỉ để truyền các thông tin điều khiển, như các lệnh và các hồi đáp trong FTP. Nó không được dùng để truyền file

- Kênh dữ liệu: Mỗi khi dữ liệu được truyền từ server tới client, một kênh kết nối TCP nhất định lại được khởi tạo giữa chúng. Dữ liệu được truyền đi qua kênh kết nối này – do đó nó được gọi là kênh dữ liệu. Khi file được truyền xong, kênh này được ngắt. Việc sử dụng các kênh riêng lẻ như vậy tạo ra sự linh hoạt trong việc truyền dữ liệu – mà ta sẽ thấy trong các phần tiếp theo. Tuy nhiên, nó cũng tạo cho FTP độ phức tạp nhất định.

Các tiến trình và thuật ngữ trong FTP

Do các chức năng điều khiển và dữ liệu sử dụng các kênh khác nhau, nên mô hình hoạt động của FTP cũng chia phần mềm trên mỗi thiết bị ra làm hai thành phần logic tương ứng với mỗi kênh. Thành phần Protocol Interpreter (PI) là thành phần quản lý kênh điều khiển, với chức năng phát và nhận lệnh. Thành phần Data Transfer Process (DTP) có chức năng gửi và nhận dữ liệu giữa phía client với server. Ngoài ra, cung cấp cho tiến trình bên phía người dùng còn có thêm thành phần thứ ba là giao diện người dùng FTP - thành phần này không có ở phía server.

Do đó, có hai tiến trình xảy ra ở phía server, và ba tiến trình ở phía client. Các tiến trình này được gắn với mô hình FTP để mô tả chi tiết hoạt động của giao thức FTP. Dưới đây là hình ảnh chiết các tiến trình vào trong mô hình FTP:



Các tiến trình phía server:

Các tiến trình phía server bao gồm hai giao thức:

- Server Protocol Interpreter (Server-PI): chịu trách nhiệm quản lý kênh điều khiển trên server. Nó lắng nghe yêu cầu kết nối hướng tới từ users trên cổng dành riêng. Khi kết nối đã được thiết lập, nó sẽ nhận lệnh từ phía User-PI, trả lời lại, và quản lý tiến trình truyền dữ liệu trên server.
- Server Data Transfer Process (Server-DTP): làm nhiệm vụ gửi hoặc nhận file từ bộ phận User-DTP. Server-DTP vừa làm nhiệm thiết lập kết nối kênh dữ liệu và lắng nghe một kết nối kênh dữ liệu từ user. Nó tương tác với server file trên hệ thống cục bộ để đọc và chép file.

Các tiến trình phía client:

- User Protocol Interpreter (User-PI): chịu trách nhiệm quản lý kênh điều khiển phía client. Nó khởi tạo phiên kết nối FTP bằng việc phát ra yêu cầu tới phía Server-PI. Khi kết nối đã được thiết lập, nó xử lý các lệnh nhận được trên giao diện người dùng, gửi chúng tới Server-PI, và nhận phản hồi trả lại. Nó cũng quản lý tiến trình User-DTP.
- User Data Transfer Process (User-DTP): là bộ phận DTP nằm ở phía người dùng, làm nhiệm vụ gửi hoặc nhận dữ liệu từ Server-DTP. User-DTP có thể thiết lập hoặc

lắng nghe yêu cầu kết nối kênh dữ liệu trên server. Nó tương tác với thiết bị lưu trữ file phía client.

- User Interface: cung cấp giao diện xử lý cho người dùng. Nó cho phép sử dụng các lệnh đơn giản hướng người dùng, và cho phép người điều khiển phiên FTP theo dõi được các thông tin và kết quả xảy ra trong tiến trình.

2 - Thiết lập kênh điều khiển và chứng thực người dùng trong FTP:

Mô hình hoạt động của FTP mô tả rõ các kênh dữ liệu và điều khiển được thiết lập giữa FTP client và FTP server. Trước khi kết nối được sử dụng để thực sự truyền file, kênh điều khiển cần phải được thiết lập. Một tiến trình chỉ định sau đó được dùng để tạo kết nối và tạo ra phiên FTP lâu bền giữa các thiết bị để truyền files.

Như trong các giao thức client/server khác, FTP server tuân theo một luật passive trong kênh điều khiển. Bộ phận Server Protocol Interpreter (Server-PI) sẽ lắng nghe cổng TCP dành riêng cho kết nối FTP là cổng 21. Phía User-PI sẽ tạo kết nối bằng việc mở một kết nối TCP từ thiết bị người dùng tới server trên cổng đó. Nó sử dụng một cổng bất kỳ làm cổng nguồn trong phiên kết nối TCP.

Khi TCP đã được cài đặt xong, kênh điều khiển giữa các thiết bị sẽ được thiết lập, cho phép các lệnh được truyền từ User-PI tới Server-PI, và Server-PI sẽ đáp trả kết quả là các mã thông báo. Bước đầu tiên sau khi kênh đã đi vào hoạt động là bước đăng nhập của người dùng (login sequence). Bước này có hai mục đích:

- Access Control - Điều khiển truy cập: quá trình chứng thực cho phép hạn chế truy cập tới server với những người dùng nhất định. Nó cũng cho phép server điều khiển loại truy cập như thế nào đối với từng người dùng.
- Resource Selection - Chon nguồn cung cấp: Bằng việc nhận dạng người dùng tạo kết nối, FTP server có thể đưa ra quyết định sẽ cung cấp những nguồn nào cho người dùng đã được nhận dạng đó.

Trình tự truy cập và chứng thực FTP

Quy luật chứng thực trong FTP khá đơn giản, chỉ là cung cấp username/password.

Trình tự của việc chứng thực như sau:

- 1 - người dùng gửi một username từ User-PI tới Server-PI bằng lệnh USER. Sau đó password của người dùng được gửi đi bằng lệnh PASS.
- 2 - Server kiểm tra tên người dùng và password trong database người dùng của nó. Nếu người dùng hợp lệ, server sẽ gửi trả một thông báo tới người dùng rằng phiên kết

nối đã đ.c mở. Nếu người dùng không hợp lệ, server yêu cầu người dùng thực hiện lại việc chứng thực. Sau một số lần chứng thực sai nhất định, server sẽ ngắt kết nối.

Giả sử quá trình chứng thực đã thành công, server sau đó sẽ thiết lập kết nối để cho phép từng loại truy cập đối với người dùng được cấp quyền. Một số người dùng chỉ có thể truy cập vào một số file nhất định, hoặc vào một số loại file nhất định. Một số server có thể cấp quyền cho một số người dùng đọc và viết lên server, trong khi chỉ cho phép đọc đối với những người dùng khác. Người quản trị mạng có thể nhờ đó mà đáp ứng đúng các nhu cầu truy cập FTP.

Một khi kết nối đã được thiết lập, server có thể thực hiện các lựa chọn tài nguyên dựa vào nhận diện người dùng. Ví dụ: trên một hệ thống nhiều người dùng, người quản trị có thể thiết lập FTP để khi có bất cứ người dùng nào kết nối tới, anh ta sẽ tự động được đưa tới "home directory" của chính anh ta. Lệnh tùy chọn ACCT (account) cũng cho phép người dùng chọn một tài khoản cá nhân nào đó nếu như anh ta có nhiều hơn một tài khoản.

Mở rộng về bảo mật FTP

Giống như phần lớn các giao thức cũ, phương pháp đăng nhập đơn giản của FTP là một sự kế thừa từ những giao thức ở thời kỳ đầu của Internet. Ngày nay, nó không còn bảo đảm tính an toàn cần thiết trên môi trường Internet toàn cầu vì username và password được gửi qua kênh kết nối điều khiển dưới dạng clear text. Điều này làm cho các thông tin đăng nhập có thể bị nghe lén. Chuẩn RFC 2228 về các phần mở rộng cho bảo mật FTP đã định ra thêm nhiều tùy chọn chứng thực và mã hóa phức tạp cho những ai muốn tăng thêm mức độ an toàn vào trong phần mềm FTP của họ.

3 - Quản lý kênh dữ liệu FTP, kết nối kênh dữ liệu dạng chủ động (mặc định) và bị động cùng với việc sử dụng cổng

Kênh điều khiển được tạo ra giữa Server-PI và User-PI sử dụng quá trình thiết lập kết nối và chứng thực được duy trì trong suốt phiên kết nối FTP. Các lệnh và các hồi đáp được trao đổi giữa bộ phận PI (Protocol Interpreter) qua kênh điều khiển, nhưng dữ liệu thì không.

Mỗi khi cần phải truyền dữ liệu giữa server và client, một kênh dữ liệu cần phải được tạo ra. Kênh dữ liệu kết nối bộ phận User-DTP với Server-DTP. Kết nối này cần thiết

cho cả hoạt động chuyển file trực tiếp (gửi hoặc nhận một file) cũng như đối với việc truyền dữ liệu ngầm, như là yêu cầu một danh sách file trong thư mục nào đó trên server.

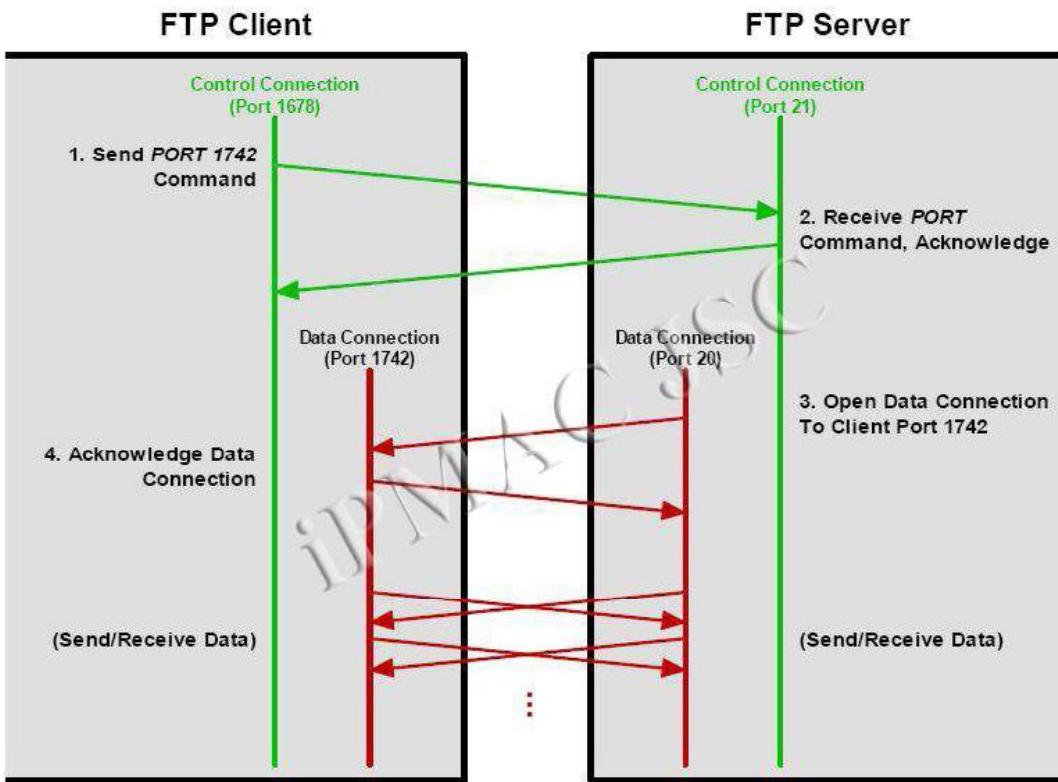
Chuẩn FTP chỉ định hai phương thức khác nhau để tạo ra kênh dữ liệu. Khác biệt chính của hai phương thức đó là ở mặt thiết bị: phía client hay phía server là phía đã đưa ra yêu cầu khởi tạo kết nối. Điều này nghe qua có vẻ khá đơn giản, nhưng kỳ thực nó lại khá quan trọng.

Kết nối kênh dữ liệu dạng chủ động

Phương thức đầu tiên đôi khi còn được gọi là kết nối kênh dữ liệu dạng thông thường (vì nó là phương pháp mặc định) và đôi khi được gọi là kết nối dạng chủ động (để đối chiếu với dạng kết nối bị động mà ta sẽ xét ở phần sau). Trong dạng kết nối này, phía Server-DTP khởi tạo kênh dữ liệu bằng việc mở một cổng TCP cho phía User-DTP. Phía server sử dụng cổng được dành riêng, là cổng 20 cho kênh dữ liệu. Trên máy client, một giá trị cổng được chọn theo mặc định chính là cổng được sử dụng đối với kênh điều khiển, tuy nhiên phía client sẽ luôn chọn hai cổng riêng biệt cho hai kênh này.

Giả sử phía User-PI thiết lập một kết nối điều khiển từ cổng bất kỳ của nó là 1678 tới cổng điều khiển trên server là cổng 21. Khi đó, để tạo một kênh dữ liệu cho việc truyền dữ liệu, phía Server-PI sẽ báo cho phía Server-DTP khởi tạo một kênh kết nối TCP từ cổng 20 tới cổng 1678 của phía client. Sau khi phía client chấp nhận kênh được khởi tạo, dữ liệu sẽ được truyền đi.

Thực tế, việc sử dụng cùng một cổng cho cả kênh dữ liệu và kênh điều khiển không phải là một ý hay, nó làm cho hoạt động của FTP trở nên phức tạp. Do đó, phía client nên chỉ định sử dụng một cổng khác bằng việc sử dụng lệnh PORT trước khi truyền dữ liệu. Ví dụ: giả sử phía client chỉ định cổng 1742 với lệnh PORT. Phía Server-DTP sau đó sẽ tạo ra một kết nối từ cổng 20 của nó tới cổng 1742 phía client thay vì cổng 1678 như mặc định. Quá trình này được mô tả trong hình dưới đây.



Thông thường, đối với kênh dữ liệu FTP, phía server sẽ khởi tạo việc truyền dữ liệu bằng cách mở kết nối dữ liệu tới client.

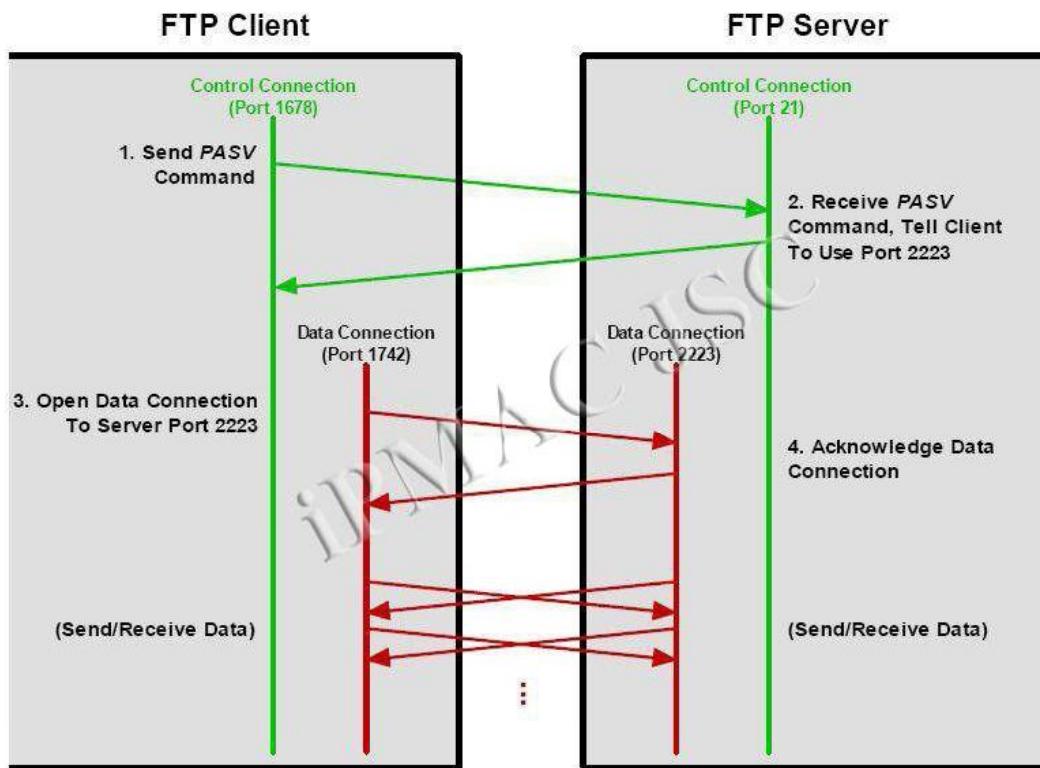
Trong trường hợp trên, phía client trước tiên sẽ đưa ra lệnh PORT để yêu cầu server sử dụng cổng 1742. Sau đó, server sẽ mở kết nối kênh dữ liệu từ cổng 20 mặc định của nó tới cổng 1742 phía client. Dữ liệu sau đó sẽ được truyền giữa các thiết bị qua các cổng này.

Kết nối kênh dữ liệu dạng bị động

Phương pháp kế tiếp được gọi là kết nối dữ liệu dạng bị động. Phía client sẽ nhận server là phía bị động, làm nhiệm vụ chấp nhận một yêu cầu kết nối kênh dữ liệu được khởi tạo từ phía client. Server trả lời lại phía client với địa chỉ IP cũng như địa chỉ cổng mà nó sẽ sử dụng. Phía Server-DTP sau đó sẽ lắng nghe một kết nối TCP từ phía User-DTP trên cổng này.

Mặc định, phía client sử dụng cùng một cổng đối với cả hai kênh điều khiển và dữ liệu như trong trường hợp kết nối chủ động ở trên. Tuy nhiên, ở đây, một lần nữa phía client có thể chọn sử dụng một giá trị cổng khác cho kênh dữ liệu. Ta sẽ xét lại ví dụ ở trên một lần nữa, với cổng điều khiển phía client là 1678 tới cổng 21 phía server.

Nhưng lần này truyền dữ liệu theo phương thức kết nối bị động, như mô tả trong hình dưới đây:



Phía client sẽ sử dụng lệnh PASV để yêu cầu server rằng nó muốn dùng phương thức điều khiển dữ liệu bị động. Phía Server-PI sẽ trả lời lại phía client với một giá trị cổng mà client sẽ sử dụng, từ cổng 2223 trên nó. Sau đó phía Server PI sẽ hướng cho phía Server-DTP lắng nghe trên cổng 2223. Phía User-PI cũng sẽ hướng cho phía User-DTP tạo một phiên kết nối từ cổng 1742 phía client tới cổng 2223 phía server. Sau khi Server chấp nhận kết nối này, dữ liệu bắt đầu được truyền đi.

Các vấn đề về tính hiệu quả và tính bảo mật trong việc chọn một phương thức kết nối

Vấn đề phía nào là phía khởi tạo kết nối kênh dữ liệu đưa ra một câu hỏi: sự khác nhau giữa hai phương thức là gì? Điều này cũng giống như việc hỏi ai đã thực hiện một cuộc điện thoại nội bộ. Câu trả lời là sự bảo mật. Việc FTP sử dụng nhiều hơn một kết nối TCP có thể giải quyết các vấn đề về phần mềm cũng như về phần cứng mà người dùng cần phải có để đảm bảo sự an toàn cho hệ thống của họ.

Khi xem xét việc gì sẽ xảy ra trong trường hợp kênh dữ liệu chủ động như trong ví dụ phía trên:

Đối với phía client, có một kênh kết nối điều khiển được thiết lập từ cổng 1678 client tới cổng 21 server. Nhưng kênh dữ liệu lại được khởi tạo từ phía server. Do đó, client sẽ nhận được một yêu cầu kết nối tới cổng 1678 (hoặc cổng nào khác). Một số client sẽ nghi ngờ về việc nhận được những kết nối tới như vậy, vì trong tình huống thường, client mới là phía khởi tạo kết nối chứ không phải đáp trả kết nối. Do các kênh kết nối TCP hướng tới có thể mang theo những mối đe dọa nhất định, một số client có thể sẽ ngăn chặn các luồng kết nối hướng tới bằng việc sử dụng tường lửa.

Tại sao người ta lại không làm cho phía client luôn chấp nhận kết nối từ một chỉ số port được dùng trong kênh điều khiển? Vấn đề ở đây là vì client thường dùng các cổng khác nhau cho mỗi phiên kết nối bằng việc sử dụng câu lệnh PORT. Và tại sao điều này lại được thực hiện? Vì theo luật TCP: sau khi một kết nối được đóng lại, có một khoảng thời gian trống trước khi cổng đó có thể được sử dụng lại – điều này để ngừa tình trạng các phiên kết nối liên tiếp bị lẫn với nhau. Điều này sẽ tạo ra độ trễ khi gửi nhiều file – do đó phía client thường dùng các giá trị cổng khác nhau cho mỗi kết nối. Điều này rất hiệu quả nhưng cũng dẫn tới việc firewall của client sẽ hỏi có chấp nhận phiên kết nối tới với nhiều giá trị cổng không ổn định hay không.

Việc dùng kết nối kiểu kênh gián tiếp sẽ giảm thiểu vấn đề này một cách hiệu quả. Phần lớn các tường lửa có nhiều vấn đề liên quan tới kết nối hướng về với các giá trị cổng bất kỳ, hơn là gặp vấn đề với các kết nối hướng đi. Ta có thể xem chi tiết hơn về vấn đề này trong chuẩn RFC 1579. Chuẩn này khuyến nghị rằng phía client nên sử dụng kết nối kiểu bị động làm dạng mặc định thay vì sử dụng kiểu kết nối dạng chủ động cùng với lệnh PORT, để ngăn chặn tình trạng block theo cổng. Tất nhiên, phương thức kết nối kiểu bị động không hoàn toàn giải quyết được vấn đề, chúng chỉ đẩy vấn đề về phía server mà thôi. Phía server, giờ đây phải đổi mặt với việc có nhiều kênh kết nối hướng về trên hàng loạt các cổng khác nhau. Tuy nhiên việc xử lý các vấn đề bảo mật trên một nhóm nhỏ server vẫn dễ hơn nhiều so với việc phải đổi mặt với một lượng lớn các vấn đề từ nhiều client. FTP server phải được cấu hình chấp nhận phương thức truyền bị động từ client, do đó cách thông thường để thiết lập trên server là thiết lập chấp nhận một số cổng kết nối hướng về trên server trong khi vẫn khóa các yêu cầu kết nối hướng về trên các cổng khác.

4 - Các phương thức truyền dữ liệu trong FTP

Khi kênh dữ liệu đã được thiết lập xong giữa Server-DTP với User-DTP, dữ liệu sẽ được truyền trực tiếp từ phía client tới phía server, hoặc ngược lại, dựa theo các lệnh

được sử dụng. Do thông tin điều khiển được gửi đi trên kênh điều khiển, nên toàn bộ kênh dữ liệu có thể được sử dụng để truyền dữ liệu. (Tất nhiên, hai kênh logic này được kết hợp với nhau ở lớp dưới cùng với tất cả các kết nối TCP/UDP khác giữa hai thiết bị, do đó điều này không hẳn đã cải thiện tốc độ truyền dữ liệu so với khi truyền trên chỉ một kênh – nó chỉ làm cho hai việc truyền dữ liệu và điều khiển trở nên độc lập với nhau mà thôi)

FTP có ba phương thức truyền dữ liệu, nếu lên cách mà dữ liệu được truyền từ một thiết bị tới thiết bị khác trên một kênh dữ liệu đã được khởi tạo, đó là: stream mode, block mode, và compressed mode

Stream mode

Trong phương thức này, dữ liệu được truyền đi dưới dạng các byte không cấu trúc liên tiếp. Thiết bị gửi chỉ đơn thuần đầy luồng dữ liệu qua kết nối TCP tới phía nhận. Không có một trường tiêu đề nhất định được sử dụng trong phương thức này làm cho nó khá khác so với nhiều giao thức gửi dữ liệu rời rạc khác. Phương thức này chủ yếu dựa vào tính tin cậy trong truyền dữ liệu của TCP. Do nó không có cấu trúc dạng header, nên việc báo hiệu kết thúc file sẽ đơn giản được thực hiện việc thiết bị gửi ngắt kênh kết nối dữ liệu khi đã truyền xong.

Trong số ba phương thức, stream mode là phương thức được sử dụng nhiều nhất trong triển khai FTP thực tế. Có một số lý do giải thích điều đó. Trước hết, nó là phương thức mặc định và đơn giản nhất, do đó việc triển khai nó là dễ dàng nhất. Thứ hai, nó là phương pháp phổ biến nhất, vì nó xử lý với các file đều đơn thuần như là xử lý dòng byte, mà không để ý tới nội dung của các file. Thứ ba, nó là phương thức hiệu quả nhất vì nó không tồn một lượng byte “overload” để thông báo header.

Block mode

Đây là phương thức truyền dữ liệu mang tính quy chuẩn hơn, với việc dữ liệu được chia thành nhiều khối nhỏ và được đóng gói thành các FTP blocks. Mỗi block này có một trường header 3 byte báo hiệu độ dài, và chứa thông tin về các khối dữ liệu đang được gửi. Một thuật toán đặc biệt được sử dụng để kiểm tra các dữ liệu đã được truyền đi và để phát hiện, khởi tạo lại đối với một phiên truyền dữ liệu đã bị ngắt.

Compressed mode

Đây là một phương thức truyền sử dụng một kỹ thuật nén khá đơn giản, là “run-length encoding” – có tác dụng phát hiện và xử lý các đoạn lặp trong dữ liệu được truyền đi

để giảm chiều dài của toàn bộ thông điệp. Thông tin khi đã được nén, sẽ được xử lý như trong block mode, với trường header. Trong thực tế, việc nén dữ liệu thường được sử dụng ở những chỗ khác, làm cho phương thức truyền kiểu compressed mode trở nên không cần thiết nữa. Ví dụ: nếu bạn đang truyền đi một file qua internet với modem tương tự, modem của bạn thông thường sẽ thực hiện việc nén ở lớp 1; các file lớn trên FTP server cũng thường được nén sẵn với một số định dạng như ZIP, làm cho việc nén tiếp tục khi truyền dữ liệu trở nên không cần thiết.

3.4.2. Cài đặt FTP Client/Server

Trên cơ sở giao thức FTP chúng ta thực hiện cài đặt FTP Client/Server để minh họa cho giao thức

Chương trình Simple FTP Server:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
class Program {
    static void Main(string[] args) {
        string rootDir = "C:/MyFTP";
        IPPEndPoint iep = new IPPEndPoint(IPAddress.Parse("127.0.0.1"), 2121);
        TcpListener server = new TcpListener(iep);
        server.Start();
        TcpClient client = server.AcceptTcpClient();
        StreamReader sr = new StreamReader(client.GetStream());
        StreamWriter sw = new StreamWriter(client.GetStream());
        sw.WriteLine("220 Chao mung ket noi toi MyFTP");
        sw.Flush();
        while (true) {
            string request = sr.ReadLine();
            string command="";
            if(request.Length!=0)      command = request.Substring(0, 4);
            switch (command.ToUpper().Trim()) {
                case "USER": {
                    sw.WriteLine("331. Nhap pass vao");
                    sw.Flush();
                    //sw.Close();
                    Console.WriteLine(request);
                    break;
                }
                case "PASS": {
                    sw.WriteLine("230. Dang nhap thanh cong");
                    sw.Flush();
                    Console.WriteLine(request);
                    break;
                }
            }
        }
    }
}
```

```

case "MKD": {
    string folderName = request.Substring(4, request.Length - 4);
    folderName = rootDir + "/" + folderName.Trim();
    try {
        Directory.CreateDirectory(folderName);
        sw.WriteLine("150 Tao thu muc thanh cong");
        sw.Flush();
    } catch(IOException){
        sw.WriteLine("550 Tao thu muc co loi");
        sw.Flush();
    }
    break;
}
case "RETR": {
    string fileName = request.Substring(4, request.Length - 4);
    fileName = rootDir + "/" + fileName.Trim();
    try {
        if (File.Exists(fileName)) {
            //Gui noi dung file ve cho client xu ly
            sw.WriteLine("150 Truyen File thanh cong");
            sw.Flush();
            FileStream fs = new FileStream(fileName, FileMode.Open);
            long totalLength = fs.Length;
            byte[] data = new byte[totalLength];
            fs.Read(data, 0, data.Length);
            sw.Write(totalLength);
            char[] kt = Encoding.ASCII.GetChars(data);
            sw.Write(kt, 0, data.Length);
            sw.Flush();
            fs.Close();
        } else {
            sw.WriteLine("550 File khong ton tai tren server");
            sw.Flush();
        }
    } catch (IOException) {
        sw.WriteLine("550 Khong truyen duoc file");
        sw.Flush();
    }
    break;
}
case "STOR": {
    string fileName =
request.Substring(request.LastIndexOf("/"), request.Length - request.LastIndexOf("/"));
    fileName = rootDir + "/" + fileName.Trim();
    try {
        FileStream fs = new FileStream(fileName, FileMode.CreateNew);
        long totalLength = sr.Read();
        byte[] data = new byte[totalLength];
        char[] kt = Encoding.ASCII.GetChars(data);

```

Chương trình Simple FTP Client:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace FtpClient {
    class Program {
        static void Main(string[] args) {
            IPPEndPoint iep = new IPPEndPoint(IPAddress.Parse("127.0.0.1"), 2121);
            TcpClient client = new TcpClient();
            client.Connect(iep);
            StreamReader sr = new StreamReader(client.GetStream());
            StreamWriter sw = new StreamWriter(client.GetStream());
            Console.WriteLine(sr.ReadLine());
            string input;
            string command = "";
            Console.WriteLine("Dang nhap bang USER Ten user, PASS Ten password");
            Console.WriteLine("Tao thu muc bang MKD ten thu muc can tao");
            Console.WriteLine("Upload bang cach STOR tenfile");
            Console.WriteLine("Download bang cach RETR tenfile");
            while (true) {
                input = Console.ReadLine();
                if (input == "MKD") {
                    string[] arr = input.Split(' ');
                    string name = arr[1];
                    client.GetStream().Write("MKD " + name + "\r\n", 0, 10);
                }
                else if (input == "STOR") {
                    string[] arr = input.Split(' ');
                    string file = arr[1];
                    string content = Console.ReadLine();
                    client.GetStream().Write("STOR " + file + "\r\n", 0, 10);
                    client.GetStream().Write(content, 0, content.Length);
                }
                else if (input == "RETR") {
                    string[] arr = input.Split(' ');
                    string file = arr[1];
                    client.GetStream().Write("RETR " + file + "\r\n", 0, 10);
                    byte[] buffer = new byte[1024];
                    int length;
                    while ((length = client.GetStream().Read(buffer, 0, 1024)) > 0) {
                        Console.WriteLine(Encoding.UTF8.GetString(buffer));
                    }
                }
                else if (input == "QUIT") {
                    client.GetStream().Write("QUIT\r\n", 0, 10);
                    break;
                }
                else {
                    Console.WriteLine("Command not found");
                }
            }
        }
    }
}
```

```
command = input.Substring(0, 4).Trim().ToUpper();
switch (command) {
    case "STOR": {
        //Doc file gui cho server
        sw.WriteLine(input);
        sw.Flush();
        FileInfo fl=null;
        try {
            fl = new FileInfo(input.Substring(4, input.Length - 4).Trim());
        } catch (IOException) {
            Console.WriteLine("File khong ton tai");
        }
        long totalLength = fl.Length;
        FileStream fs = fl.OpenRead();
        sw.Write(totalLength);
        byte[] data = new byte[totalLength];
        int bytes = fs.Read(data, 0, data.Length);
        char[] kt = Encoding.ASCII.GetChars(data);
        sw.Write(kt, 0, data.Length);
        sw.Flush();
        fs.Close();
        Console.WriteLine(sr.ReadLine());
        break;
    }
    case "RETR": {
        sw.WriteLine(input);
        sw.Flush();
        string s = sr.ReadLine();
        Console.WriteLine(s);
        if (s.Substring(0, 3).Equals("150")) {
            Console.Write("Nhap vao noi luu tep:");
            string filename = Console.ReadLine();
            FileStream fs = new FileStream(filename, FileMode.CreateNew);
            //Doc tep ve;
            long totalLength = sr.Read();
            byte[] data = new byte[totalLength];
            char[] kt= new char[data.Length] ;
            int sobyte = sr.Read(kt, 0, data.Length);
            data=Encoding.ASCII.GetBytes(kt);
            fs.Write(data, 0, data.Length);
            fs.Close();
        }
        break;
    }
    default: {
        sw.WriteLine(input);
        sw.Flush();
        Console.WriteLine(sr.ReadLine());
        break;
    }
}
```

```

        }
    }
    if (input.ToUpper().Equals("QUIT")) break;
}
sr.Close();
sw.Close();
client.Close();
}
}
}
}

```

3.5. DNS (Domain Name Server)

3.5.1. Vấn đề phân giải tên miền

Domain Name System:

- Là hệ cơ sở dữ liệu phân tán hoạt động có thính bậc bởi các name servers
- Là giao thức tầng ứng dụng : host, routers yêu cầu tới name servers để xác định tên miền (ánh xạ địa chỉ <->tên miền)
 - Note : là một chức năng của Internet, hoạt động như là giao thức tầng ứng dụng
 - Rất phức tạp.

Q: Ánh xạ giữa địa chỉ IP và tên?

- Tại sao không tập trung sự kiểm soát của DNS ?
 - Điểm hỏng duy nhất - nếu name-server “chết” thì cả mạng Internet sẽ “chết” theo.
 - Tốn đường truyền.
 - Cơ sở dữ liệu tập trung sẽ “xa” với đa số vùng
 - Bảo trì phức tạp.
 - Phải chia để trị !
 - Không có server nào có thể lưu toàn bộ được tên miền và địa chỉ IP tương ứng
- local name servers:
 - Mỗi ISP,công ty có local (default) name server
 - Câu hỏi truy vấn của host về DNS sẽ được chuyển tới local name server
- Chức năng của name server:
 - Đối với host: lưu địa chỉ IP và tên miền tương ứng của host
 - Có thể tìm tên miền ứng với địa chỉ IP và ngược lại
- Được yêu cầu bởi các local name server không thể xác định được tên.
- root name server:
 - Được yêu cầu nếu có authoritative name server không xác định.
 - Nhận và xử lý mapping
 - Trả về mapping cho local name server