Application Notes of Transparent HTTP Tunnel

Version 1.0e

2009/08/17

www.vivotek.com

Storage Manager

Revision History

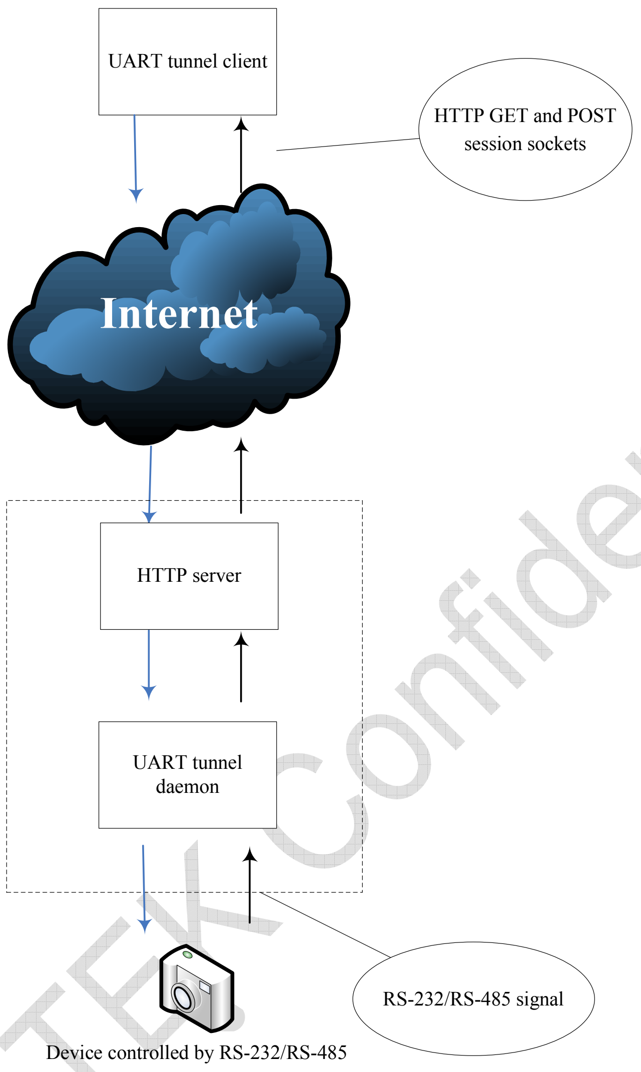
Version	Issue date	Editor	Comment
1.0a	2007/08/16	David Liu	Make first draft.
1.0b	2008/01/21	David Liu	Revise the URI of UART tunnel.
1.0c	2008/07/02	David Liu	1. Add the description. 2. Revise the query string of URI.
1.0d	2009/08/13	David Liu	Modify the part of tunnel establishment.
1.0e	2009/08/17	David Liu	Revise the description of tunnel establishment.

Table of Contents

Introduction

This document describes the usage and mechanism of tunnel establishment of UART HTTP tunnel. UART HTTP tunnel is used to forward the commands transparently from the tunnel client side to the server side. As for the responses from UART device, the HTTP tunnel server will forward them to the client side. With tunnel mechanism, the client is able to send custom commands to the device even the originally server doesn't support the protocol. Originally, client can control video server/network camera via certain CGI commands. This method is not very real time because the overhead of TCP connection establishment of each CGI command is considerable. On the other hand, HTTP tunneled connection can be consistent so that TCP connection won't be established each time a message would be delivered. Also, HTTP connection can be the most traversable to firewall and compliant to HTTP authentication if security issue is concerned. Thus, the video server/network camera can be managed more efficiently and systematically.

The figure below shows the application scenario of UART tunnel.


UART tunnel client
HTTP GET and POST session sockets
Internet
HTTP server
UART tunnel daemon
RS-232/RS-485 signal
Device controlled by RS-232/RS-485

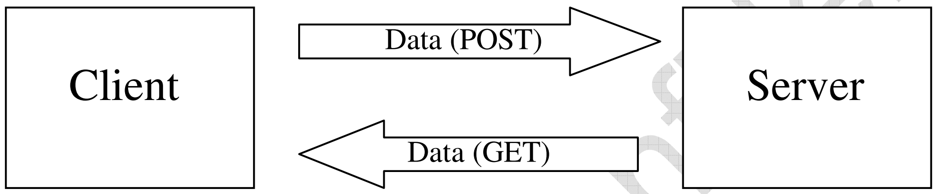
Feature

UART tunnel daemon has the following features.

- Support one tunnel client at one time
- Use a FIFO queue to store incoming UART commands
- Able to send response from UART device to the tunnel client
- The target UART device can be updated without establishing another HTTP tunnel

Mechanism of HTTP tunnel establishment

The HTTP tunneled connections use the capability of HTTP's GET and POST methods to carry an indefinite amount of data in their reply and message body respectively. Generally, the client makes an HTTP GET request to the server to open the one way connection of server-to-client. Client can use this connection to receive data from server. Then the client makes a HTTP POST request to the server to open the one way connection of client-to-server. Client can use this connection to send data to server. Server will bind these 2 paired connections according to the x-sessioncookie filed in HTTP header to form a virtual full-duplex connection makes it possible to send and receive data from one client.



To work with HTTP tunneled connection, client must

(1) Establish one TCP socket to server (download socket) and send GET HTTP message.
Please note that the sessioncookie in the HTTP header should be unique and the length of cookie should be less than or equal to 32 bytes.
The target channel (UART device) number must be specified in the query string. In the example below, the channel number is "0".

(2) Receive 200 OK from server

(3) Establish the other TCP socket to server (upload socket) and send POST HTTP message with the same x-sessioncookie.

(4) Receive tunnel status string in download socket ("http tunnel accept=1") to confirm one pair of tunneled sockets are ready

(5) Client is ready to send data in upload socket and receive reply in download socket

For example:

From client to server (in download socket)

```
GET /cgi-bin/operator/uartchannel.cgi?channel=0 HTTP /1.0
```

```
User-Agent: TunnelClient
x-sessioncookie: 5AasdGTHfgjwsqDdSF33
Accept: application/x-vvtk-tunnelled
Pragma: no-cache
Cache-Control: no-cache
Connection: Keep-Alive
```

To make HTTP tunnel works optimally:

- Be made using HTTP version 1.0
- Include in the header an x-sessioncookie directive whose value is a globally unique identifier (GUID). The GUID makes it possible for the server to unambiguously bind the two connections.
- In POST requests, the application/x-vvtk-tunneled MIME type for both the Content-Type and Accept directives must be specified; this MIME type reflects the data type that is expected and delivered by the client and server.

From server to client (in download socket)

```
HTTP/1.1 200 OK
Content-Type: application/x-vvtk-tunnelled
Date: Sun, 9 Jan 1972 00:00:00 GMT
Cache-Control: no-store
Pragma: no-cache
x-server-ip-address: 168.95.2.32
Server: Boa/0.94.14rc21
Accept-Ranges: bytes
Connection: close
```

When the server receives an HTTP GET request from a client, it must respond with a reply whose header specifies the application/x-vvtk-tunneled MIME type for both the Content-Type and Accept directives.

Server reply headers may optionally include the Cache-Control: no-store and Pragma: no-cache directives to prevent HTTP proxy servers from caching the transaction. It is recommended that implementations honor these headers if they are present.

The Date directive specifies an arbitrary time in the past. This keeps proxy servers from caching the transaction.

Server clusters are often allocated connections by a round-robin DNS or other load-balancing algorithm. To insure that client requests are directed to the same server among potentially several servers in a server farm, the server may optionally include the x-server-ip-address directive followed by an IP address in dotted decimal format in the header of its reply to a client's initial GET request.

When this directive is present, the client must direct its POST request to the specified IP address regardless of the IP address returned by a DNS lookup.

The sample client POST request includes three optional header directives that are present to control the behavior of HTTP proxy servers:

- The Pragma: no-cache directive tells many HTTP 1.0 proxy servers not to cache the transaction.
- The Cache-Control: no-cache directive tells many HTTP 1.1 proxy servers not to cache the transaction.
- The Expires directive specifies an arbitrary time in the past. This directive is intended to prevent proxy servers from caching the transaction.

Server will check the correct URI and matching x-sessioncookie in the GET and POST message to bind tunneled sockets. Therefore there must be a header of x-sessioncookie with the same string in both GET and POST message for server to check.

There are two methods to generate proper POST message for tunnel establishment.

The first method is:

```
POST /cgi-bin/operator/uartchannel.cgi HTTP/1.1
Expires: Sun, 9 Jan 1972 00:00:00 GMT
x-sessioncookie: 5AasdGTHfgjwsqDdSF33
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 7
User-Agent: TunnelClient
Connection: Keep-Alive
VerifyPassProxy: yes
```

(1) Add "VerifyPassProxy: yes" in the HTTP header.

(2) Specify the Content-Length in the HTTP header to 7 and actually fill in the content of the POST body with 8 bytes. By this way the tunnel server can detect whether the HTTP proxy do check the content length and forward 7 bytes only and decide this session pair is suitable for consistent tunnel or not.

The second method is, in the POST message, header of Content-Length is used to keep the POST connection alive by marking large amount of data to upload (32767 bytes).

From client to server (in upload socket, the second method)

```
POST /cgi-bin/operator/uartchannel.cgi HTTP/1.1
Expires: Sun, 9 Jan 1972 00:00:00 GMT
x-sessioncookie: 5AasdGTHfgjwsqDdSF33
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 32767
User-Agent: TunnelClient
Connection: Keep-Alive
```

After the tunnel is successfully set up, the server will send message through the GET (download) socket.

From server to client (in download socket)

```
http tunnel accept=1
```

Some HTTP proxies might cache POST message. Since client send a POST request with Content-length 32767, sometimes HTTP proxy will not forward any data until certain amount of data is coming. To detect this behavior of HTTP proxy, the tunnel status string is used to confirm the success of tunnel socket binding. If tunnel status string is “http tunnel accept=0”, this means server fail to binding the GET message due to time out of POST message. In this case, client should teardown the paired sockets and re-establishes the paired sockets by “normal POST”

In HTTP tunnel mode, client won’t disconnect the tunneled sockets actively because the advantage of consistent connection with server will make information exchange more efficiently. However, in the above scenario, client will need to establish a new TCP socket each time to send a POST message with upload data as message body and disconnect TCP connection after finishing transmission. In the mean time, client still keeps GET (download socket) alive. This way can avoid HTTP proxy server to cache the upload message from client. Of course each POST message comes with the same x-sessioncookie header. This is different from tunneled socket which got 2 consistent connections for upload and download data. In this scenario, only download connection is consistent, the other one will be re-established each time a new data is ready to send (upload) from client. Once the client wants to tear up the tunnel session, he should send the normal POST command by replacing the URI by “/cgi-bin/operator/uartchannel.cgi?connect=close”. The UART tunnel will remove this session and wait for next connection.

After HTTP tunneled sockets are established successfully, the client is ready to send data to the UART device through the POST socket and retrieve the response from the GET socket. Please note that all the following data in the upload/POST tunneled sockets should be base64 encoded to traverse

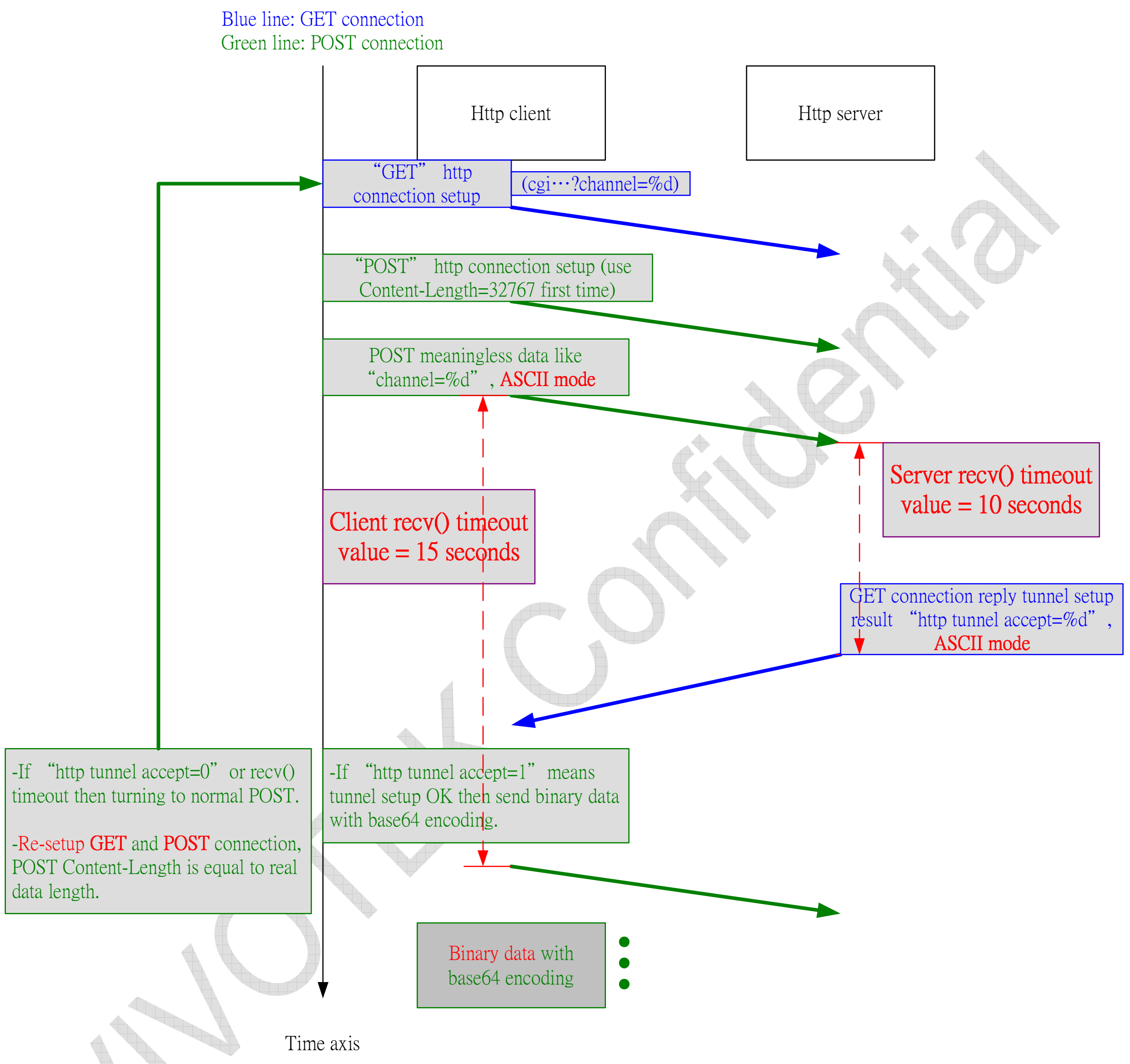
HTTP proxy.

Command format

The tunnel client should send binary UART commands encoded in **base64 format** through POST socket. The UART tunnel will decode the message and **transparently** forward these commands. If there are responses received from UART device, the UART tunnel will also forward them to the tunnel client. The response messages are **not** base64 encoded.

Appendix:

1. The flow to set up http tunnel


Blue line: GET connection
Green line: POST connection
Http client
Http server
"GET" http connection setup
(cgi…?channel=%d)
"POST" http connection setup (use Content-Length=32767 first time)
POST meaningless data like "channel=%d" , ASCII mode
Server recv() timeout value = 10 seconds
Client recv() timeout value = 15 seconds
GET connection reply tunnel setup result "http tunnel accept=%d" , ASCII mode
-If "http tunnel accept=0" or recv() timeout then turning to normal POST.
-Re-setup GET and POST connection, POST Content-Length is equal to real data length.
-If "http tunnel accept=1" means tunnel setup OK then send binary data with base64 encoding.
Binary data with base64 encoding
Time axis