VIVOTEK
www.vivotek.com
IP Surveillance

Application Notes of Transparent HTTP Tunnel

Version 1.0b

2008/01/21

www.vivotek.com

ATD Document Template

VIVOTEK may make changes to specifications and product descriptions at any time, without notice.

The following is trademarks of VIVOTEK INC., and may be used to identify VIVOTEK products only: VIVOTEK. Other product and company names contained herein may be trademarks of their respective owners.

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.

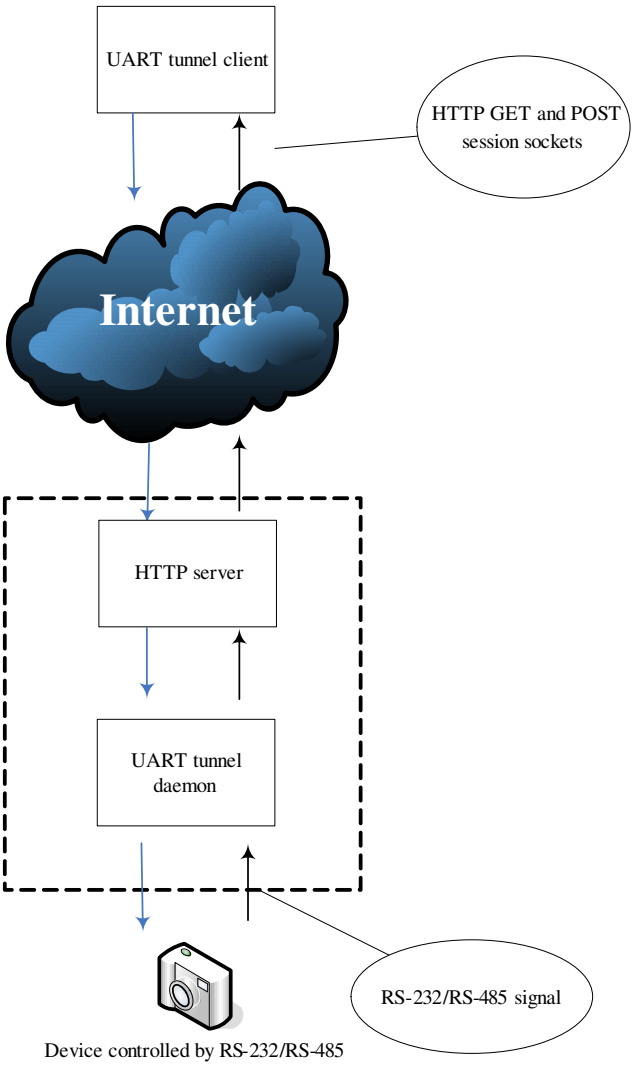
Table of Contents

Introduction..... 4
Feature 5
Mechanism of HTTP tunnel establishment 6
Command format 10
Appendix: 11
1. The flow to set up http tunnel 11

Introduction

This document describes the usage and mechanism of tunnel establishment of UART HTTP tunnel. Originally, client can control video server/network camera via 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.



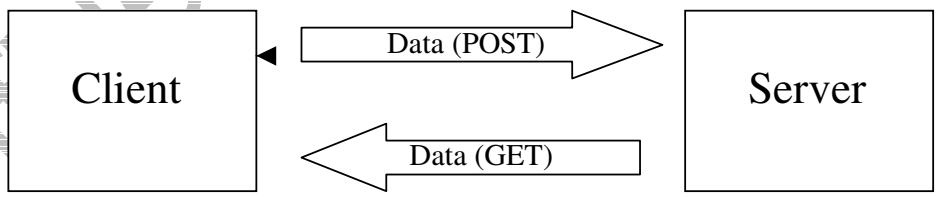
Feature

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 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
(2) Receive 200 OK from server
(3) Establish the other TCP socket to server (upload socket) and send POST HTTP message
(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 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-rtsp-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.

From client to server (in upload socket)

```
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
```

From server to client (in download socket)

```
HTTP tunnel accept=1
```

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. 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)

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. In this case, if the client wants to tear down the tunnel session, he should send the normal POST command by replacing the URI by "/cgi-bin/operator/uartchannel.cgi?**connect=close**". The HTTP tunnel will close the current session and wait for next connection.

After HTTP tunneled sockets are established successfully, client is ready to send control, event subscribe/unsubscribe messages and receive event notify via the tunneled sockets. All the following control or event messages in the upload/POST tunneled sockets should be base64 encoded to traverse HTTP proxy.

The client is able to change the target UART device by issuing one POST request which replaces the original URI by "/cgi-bin/operator/uartchannel.cgi?**channel=N**", where **N** is the target UART channel number. The server will update the target UART channel respectively. Please note that the x-session cookie should be the same as the one of current session.

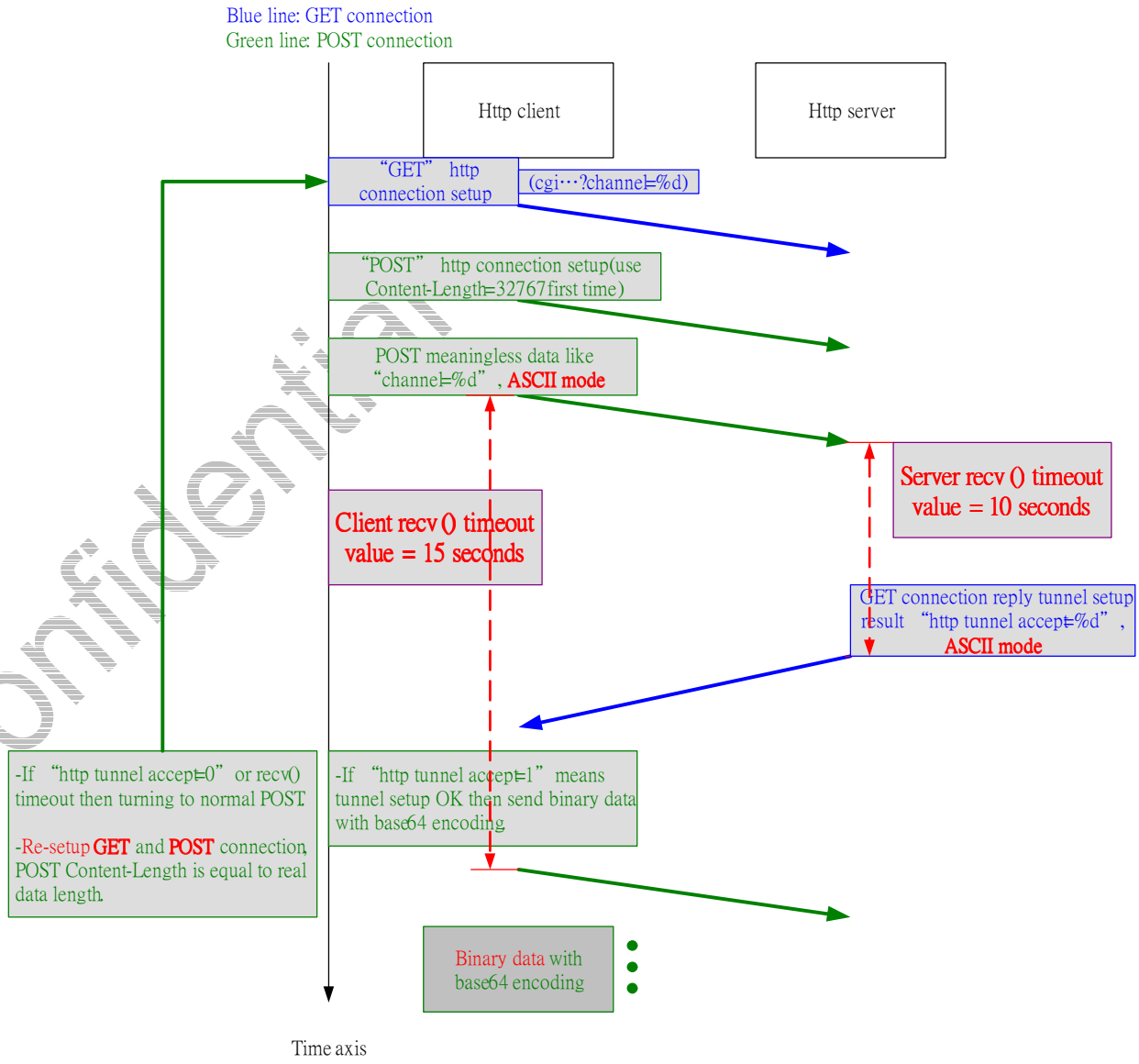
For detail about tunnel establishment, please refer to the **appendix** of this document.

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



Title 2

<End of document>