Packet Tracer 5.0

Packet Tracer Messaging Protocol (PTMP)

Specifications Document

Modification History

Revision	Date	Originator	Comments
1	January 10, 2008	Michael Wang (miwang@cisco.com)	Creation for general public
2	April 8, 2008	Michael Wang (miwang@cisco.com)	Updated all sections from latest PT version
3	July 3, 2008	Michael Wang (miwang@cisco.com)	Added Keep-alive section

Table of Contents

1. Introduction

1.1. Goal

Packet Tracer Messaging Protocol (PTMP) is explained in this document. This document also explains the specifications of the PTMP that is added to Packet Tracer 5.0.

1.2. Audience

The scope of this document is intended for Packet Tracer 5.0 CEP developers. It is used to validate requirements and describes detail implementation issues.

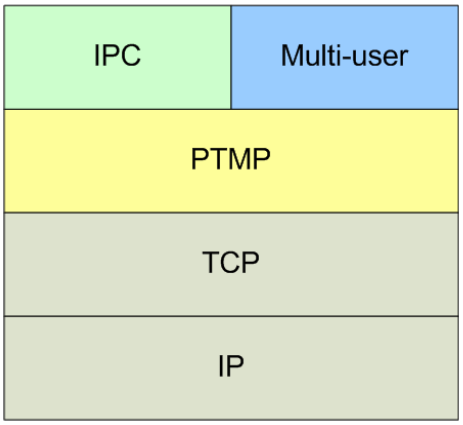
1.3. Abstract

This document describes the features of PTMP which supports the IPC and Multi-user features of the Packet Tracer 5.0 and details about the its architectural design .

Packet Tracer 5.0 has features which require network communication among different Packet Tracer instances and/or other applications. This communication must be uniform and transparent to the other components so that they can use this communication easily. PTMP implements this communication with the following aspects: connection negotiation, encoding, encryption, compression, and authentication.

2. Architecture

For optimal network utilization, PTMP is visualized as a protocol working over TCP/IP. This is designed as a general messaging protocol. An application utilizing PTMP, such as PT instances or other applications, will be referred to as a PTMP application. The messages sent between PTMP applications are not governed by PTMP. It is the responsibilities of the components utilizing PTMP to specify and follow their own messaging formats and behaviors. These messages can be network packets in the case of Multi-user or IPC calls in the case of IPC. In general, PTMP can be considered as a TCP extension that applications can use to communicate with PT instances.



The communication protocol decided is TCP. Other protocols such as UDP have been considered. But, TCP is the standard and most versatile across different platforms and networks.

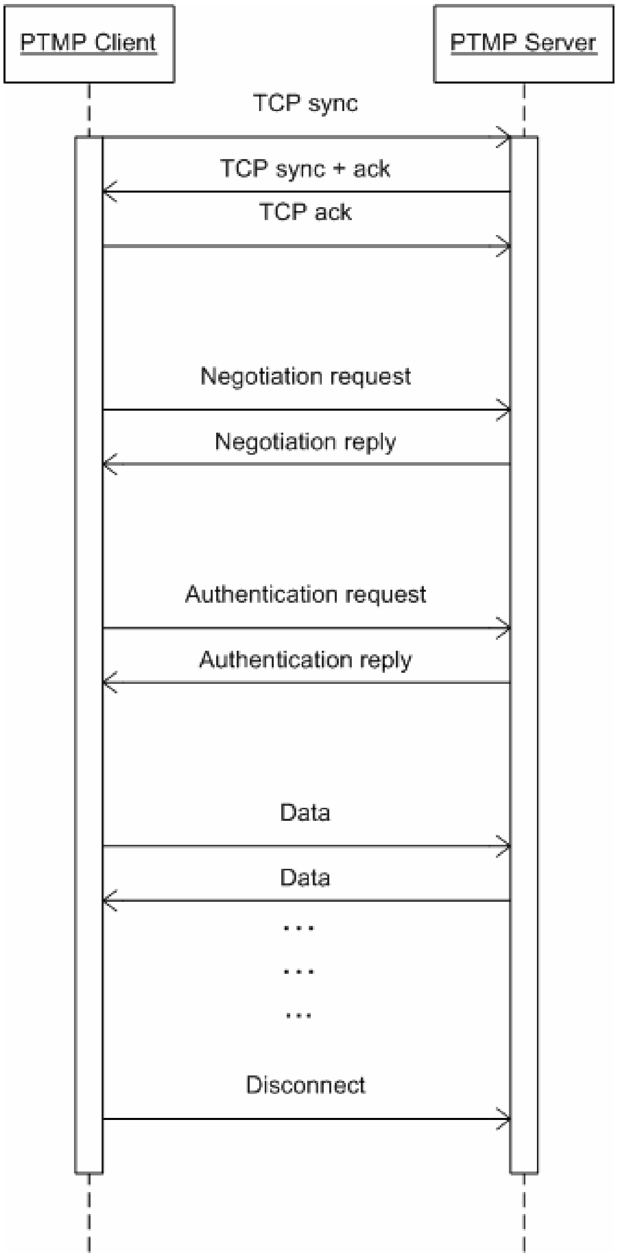
The communication model for PTMP applications is client-server, based on TCP. A PTMP application can start listening on a TCP port and allow other instances to connect to it. The default TCP port for Multi-user is 38000 and for IPC is 39000. These port numbers are currently unassigned in the IANA registered TCP numbers (http://www.iana.org/assignments/port-numbers). The component using PTMP can also change the port. Other PTMP applications can connect to one that is listening by providing the IP address and port number.

When connections are made, a negotiation step is used to determine how the two PTMP applications will communicate with each other. This includes negotiation on authentication, encoding, encryption, and compression methods. Then an authentication step is used to verify the client PTMP application.

Once a TCP connection is made between two PTMP applications and after the negotiation and authentication steps, there is no concept of server and client. Both applications have the same role and functionalities. Now, the two PTMP applications can send messages to each other using PTMP.

There can be more than one connection to different PTMP applications. For each connection, a dedicated TCP session is established.

When a PTMP application needs to disconnect, it sends a disconnect message for a graceful disconnection.



3. Modeling

3.1. Encoding

Before going into details about the protocol, we must address the encodings available in PTMP and the selected encoding during each connection. Because some of the PTMP applications that will be developed are in Flash 8, it is necessary to support text encoding. However, binary encoding allows for efficient conversion and shorter messages, thus it is also necessary to support binary encoding.

The encoding used for each connection is negotiated in the beginning of each connection.

PTMP also specifies the basic types natively supported in PTMP. These types can be automatically converted by PTMP. Custom types that are built on-top of these basic types are also possible but it is the responsibilities of the components using PTMP to specify and follow these custom types.

The encoding formats to be followed for different basic types are as follows.

Name	**Binary Encoding**	**Text Encoding (all terminated by \0)**
byte	An 8-bit signed value	A signed value between -128 to 127
bool	An 8-bit value -- true and false	"true" or "false"
short	A 16-bit signed number	A signed number between -2^15 to 2^15 – 1
int	A 32-bit signed number	A signed number between -2^31 to 2^31 – 1
long	A 64-bit signed number	A signed number between -2^63 to 2^63 – 1
float	A single precision 32-bit	A decimal number
double	A double precision 64-bit	A decimal number
string	Variable-length Unicode characters terminated by \0	Variable-length Unicode characters terminated by \0
QString	Variable-length Qt Unicode characters terminated by \0	Variable-length Qt Unicode characters terminated by \0
IP address	A 32-bit value	An IP address in the x.x.x.x format
IPv6 address	A 128-bit value	An IPv6 address in the x:x:x:x:x:x:x:x format
MAC address	A 48-bit value	A MAC address in the xxxx.xxxx.xxxx format
uuid	A 128-bit value	A UUID in the {HHHHHHHH-HHHH-HHHH-HHHH-HHHHHHHHHHHH} format

Maximum width is allocated for each data type to accommodate future requirements and different programming languages. All the above data types may not be required. But for completeness we can keep the encoding information. Unsigned types are not available because languages such as Java do not have native support for unsigned types.

Binary encoding specifies the length or terminating character for each type of values. Thus it is straightforward to separate the values for reading. However, values in text encoding do not specify a length per type. A terminating character (\0) is used to separate the values for reading.

3.2. General Message Format

Messages are sent between PTMP applications once the TCP connection is established. These messages need not be sent in one TCP segment. It can be of almost-infinite length (2^31 – 1 = maximum number for int) and be sent over in multiple TCP segments. However, they are processed only when the entire message is received. All of these messages must follow a common message format in order to differentiate the different types of messages as well as determining the end of the message. The commonly used Length-Type-Value (LTV) is used for all PTMP messages.

The fields are as follow:

- Length (int): number of bytes or characters in the message excluding the length field but including the type and value fields; this field is never encrypted or compressed
- Type (int): specifies the type of message
- Value: variable length

The following are different values in the Type field:

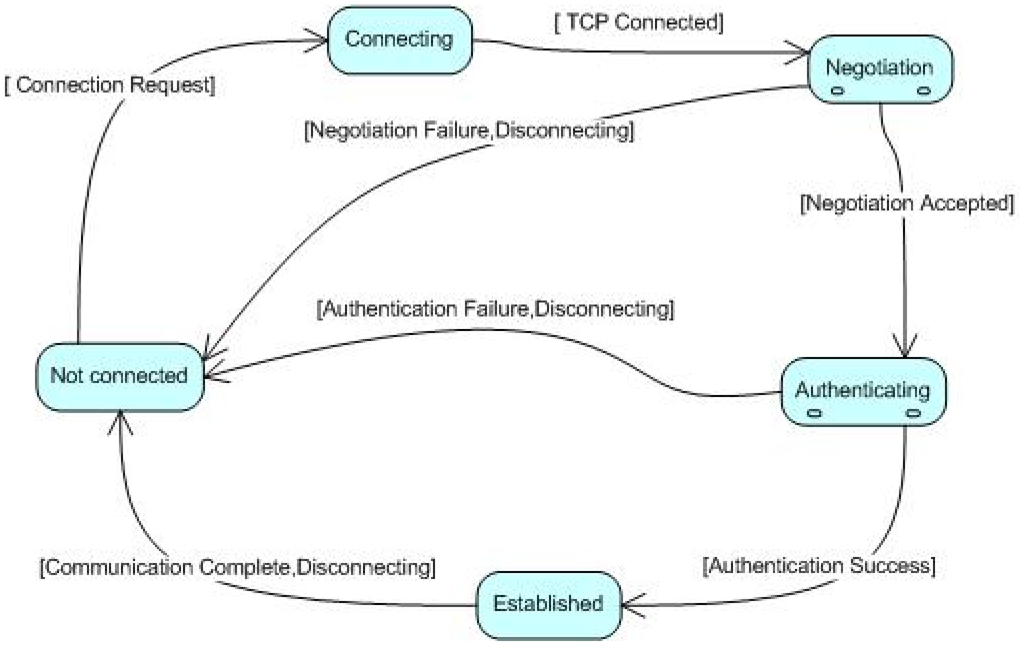
Value	**Message**
0	Negotiation request
1	Negotiation response
2	Authentication request
3	Authentication challenge
4	Authentication response
5	Authentication status
6	Keep-alive
7	Disconnect
>= 100, < 200	IPC messages
>= 200, < 300	Multi-user messages

3.3. State Diagram

PTMP has the following states:

- Not connected: created but not initiated TCP connect, or disconnected
- Connecting: TCP is connecting
- Negotiating: exchanging connection information
- Authenticating: exchanging username and password
- Established: authenticated and fully established

The Simplified State diagram of the PTMP is given below



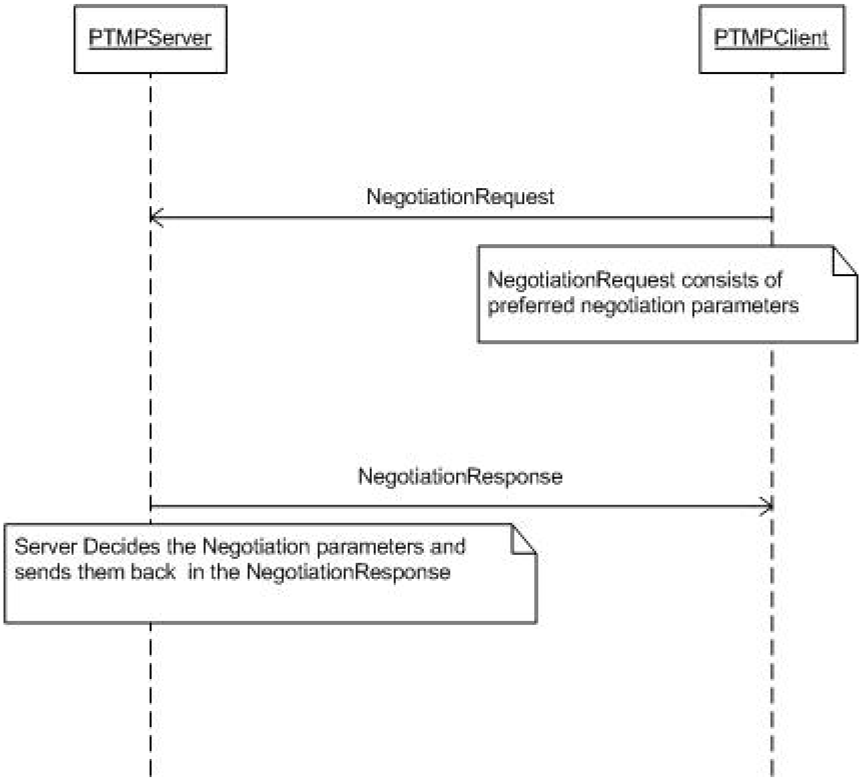
3.4. Connection Negotiation

Once a TCP connection is made, the first step of the two PTMP applications is connection negotiation. This is to decide on a set of common properties to be used on both sides of the connection.

PTMP applications exchange information during negotiation using the following message format:

- PTMP identifier (string): a constant identifier, “PTMP”
- PTMP version number (int): 1
- PTMP application ID (uuid): UUID of the application sending this negotiation message
- Encoding (int): 1 = text, 2 = binary
- Encryption (int): 1 = none, 2 = XOR
- Compression (int): 1 = no, 2 = zlib default
- Authentication (int): 1 = clear text, 2 = simple, 4 = MD5
- Timestamp (string): local time when connection is initiated in the format YYYYMMDDHHMMSS
- Keep-alive (int): keep-alive period in seconds
- Reserved (string): currently unused

The client first sends a negotiation request message to the server specifying the desired connection properties. The server replies with the decided connection properties in a negotiation response message. The entire negotiation process is in text encoding.

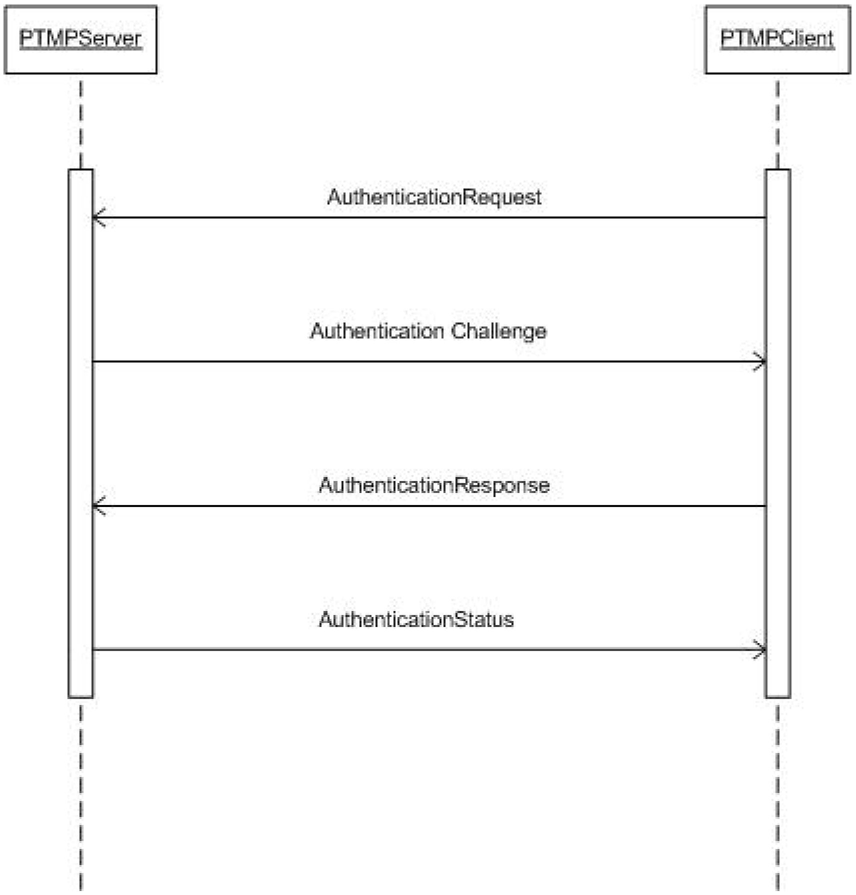

PTMPServer
PTMPClient
NegotiationRequest
NegotiationRequest consists of preferred negotiation parameters
NegotiationResponse
Server Decides the Negotiation parameters and sends them back in the NegotiationResponse

3.5. Authentication

The PTMP follows CRAM (Challenge Response Authentication Mechanism), a challenge response mechanism for authentication.

After TCP connection establishment and connection negotiation, authentication process starts. It is assumed that the both the client and server applications utilizing PTMP have knowledge of the same credentials (username and password, or id and key). This is a pre requisite for successful authentication.

The client first sends an authentication request message. The server challenges it by sending an authentication challenge message. The challenge is a randomly generated string consisting of 32 characters. Depending on the negotiated authentication method, the client may need to calculate the “digest” by applying a hashing algorithm. If the negotiated authentication method is clear text, the password is to be sent back in clear text. If the negotiated authentication method is simple authentication, it uses the simple authentication method described in Section 3.5.5 to encrypt the password. If the negotiated authentication method is MD5, it uses MD5 along with the challenge text to produce a digest of the password. The digest is sent back in an authentication response message. The server verifies the digest using the same authentication method. If the digest matches, the server sends back an authentication status message to the client and ending the authentication process. If the digest does not match, then the server sends back a disconnect message.



Upon disconnection of the TCP session, authentication is to be re-initiated.

3.5.1. Authentication Request Message Format

Fields of Authentication Request Message are as follows:

- Username (string): username or id of the client

3.5.2. Authentication Challenge Message Format

Fields of Authentication Challenge Message are as follows:

- Challenge text (string)

3.5.3 Authentication Response Message Format

Fields of Authentication Response Message are as follows:

- Username (string): username or id of the client

- Digest text (string): digest of the password generated using the negotiated authentication method
- Custom (string): reserved, currently unused

3.5.4 Authentication Status Message Format

Fields of Authentication Status Message are as follows:

- Status (bool): true = successful, false = failed

3.5.5 Simple Authentication Method

The simple authentication method uses a simple hashing function to generate a digest from the given password.

```
function simple_hash(string password)
{
        string hash;
        for (int i=0; i<password.length; i++)
                hash[i] = 158 – password[i];
        return hash;
}
```

3.6. Encryption

Confidentiality is the sole purpose of encryption in PTMP. Every message after the negotiation process is to be encrypted if it is negotiated to do so. The encryption method is simple XOR the data with a symmetric key (ie, both the server and client use the same key to encrypt and decrypt).

The encryption key is derived from the UUIDs and the timestamps of the server and client in this sequence:

1. Server's UUID in {HHHHHHHH-HHHH-HHHH-HHHH-HHHHHHHHHHHH} format
2. Client's UUID in {HHHHHHHH-HHHH-HHHH-HHHH-HHHHHHHHHHHH} format
3. "PTMP"
4. Server's timestamp in YYYYMMDDHHMMSS format
5. Client's timestamp in YYYYMMDDHHMMSS format

This encryption key is to be used to XOR with the data messages in the following way:

```
function encrypt(array data, array key)
{
        for (int i=0; i<data.length; i++)
```

```
                data[i] = data[i] ^ key[i % key.length];
}
```

The encrypt function can also be used as the decrypt function to

3.7. Compression

The compression method is zlib's compression. See http://www.zlib.net. All messages after the negotiation process are to be compressed if it is negotiated to do so.

3.8. Order of Operations

When sending messages, PTMP follows this order of operations:

1. Message with type and value fields are passed into PTMP for sending.
2. If state is authenticating or established:
    a. Compress message if compression is negotiated
    b. Encrypt message if encryption is negotiated
3. Prepend size of the message after compression and encryption.

When receiving messages, PTMP follows this order of operations:

1. Read the size of the message.
2. If state is authenticating or established:
    a. Decrypt message if encryption is negotiated
    b. Uncompress message if compression is negotiated
3. Send message to the component utilizing PTMP

3.9. Keep-alives

The Keep-alive mechanism in PTMP is used to detect uninformed disconnections from peers. The keep-alive period is negotiated during the negotiation phase. The client sends the desired keep-alive period and the server will take the same number. The period is in seconds, which is the number of seconds keep-alive messages are being sent apart. If the negotiated keep-alive period is zero, then keep-alives are not sent. The keep-alive message is merely an empty PTMP message with Keep-alive as the type. If keep-alive messages are not received in three times of the keep-alive period, then PTMP would consider the peer disconnected and would notify the PTMP application.