IPv6 support

Availability

IPv6 support is available in the following platforms:

- Windows families, e.g.: Windows 7/8/10, Windows XP, Windows Server 2003, and Windows Vista, using Microsoft Platform SDK for Windows Server 2003 SP1
- Linux/Unix/MacOS X with GNU development systems, IPv6 features are detected by the `./configure` script.
- iOS
- Symbian

Note about Windows 2000:

- Windows 2000 does not have IPv6 installed, but it may be added by installing ?Microsoft IPv6 Technology Preview for Windows 2000 (IPv6Kit). This software was designed to work for Win2K SP1 and SP2, however there is an information on how to install IPv6Kit on Win2K SP4, please see ?IPv6 FAQ.
- Even with IPv6Kit installed, pjsip still would **NOT** run normally because the Win2K's IPHLPAPI.DLL does not support enumerating IPv6 interfaces. Because of this, pjsip would always return the loopback interface ("::1") as the host's IPv6 address, unless a specific IPv6 address is specified when creating the transports.

IPv6 is **NOT** supported by Platform SDK that comes with Microsoft Visual Studio 6. To use Visual Studio 6, you must install and download the newer Platform SDK above. Please see ?PJSIP Getting Started Guide on how to configure VS6 with newer Platform SDK.

IPv6 Support in pjlib

The work for adding IPv6 support in pjlib is documented by ticket #415.

pjlib supports IPv6, but for now this has to be enabled in `pj/config_site.h`:

```
#define PJ_HAS_IPV6 1
```

Socket Addresses:

- An IPv4 socket address is represented by `pj_sockaddr_in` structure, while an IPv6 socket address is represented by `pj_sockaddr_in6` structure. The `pj_sockaddr` is a union which may contain IPv4 or IPv6 socket address, depending on the address family field.
- Various new socket address API's have been added which can work on both IPv4 and IPv6 address, such as:
  - pj_inet_pton()
  - pj_inet_ntop()
  - pj_inet_ntop2()
  - pj_sockaddr_init()
  - pj_sockaddr_get_addr()
  - pj_sockaddr_has_addr()

- pj_sockaddr_get_addr_len()
- pj_sockaddr_set_str_addr()
- pj_sockaddr_get_port()
- pj_sockaddr_set_port()
- pj_sockaddr_get_len()

Socket and IOQueue API:

- The `pj_sockaddr` structure is a union which may contain IPv4 or IPv6 socket address. Application may pass this structure to various pjlib socket functions which take `pj_sockaddr_t` as an argument, such as `pj_sock_bind()`, `pj_sock_sendto()`, `pj_sock_recvfrom()`, `pj_sock_accept()`, etc.
- The *ioqueue* also supports IPv6 sockets.

Address Resolution API:

- A new API `pj_getaddrinfo()` has been added to resolve both IPv4 and IPv6 addresses, in addition to existing `pj_gethostbyname()` API which resolves IPv4 address.
- The `pj_gethostip()` and `pj_getdefaultipinterface()` API has been modified to take an additional address family argument.

IP Helper API:

- The `pj_enum_ip_interface()` API has been modified to take an additional address family argument.

IPv6 Support in pjsip

The work for adding IPv6 support in pjlib is documented by ticket #421.

IPv6 SIP Transport:

- The SIP UDP transport now supports IPv6 sockets, and new API's have been added to facilitate IPv6 transport creation (`pjsip_udp_transport_start6()` and `pjsip_udp_transport_attach2()`). Note that if application wants to support both IPv4 and IPv6 UDP transports, two separate UDP transport instances must be created, one for each address family.
- The SIP TCP and TLS transports also support IPv6 sockets since version 2.1 (ticket #1585).

IPv6 Address Representation:

- IPv6 address may appear in two types of places in the SIP message: in a host part of a header field (such as host part of an URI, or host part in a Via header), and as a parameter value (such as the value of *received* and *maddr* parameter).
- Although in the SIP ABNF grammar an IPv6 may or may not be enclosed in square brackets ([and] characters), in pjsip all IPv6 addresses will be represented **without** the square brackets, for consistency. This means pjsip will remove the square brackets, if they are present, during parsing process, and will enclose the address with square brackets as necessary when pjsip prints the Ipv6 address in a packet for transmission. When application inspects a message component that contains IPv6 address, it will always find it without the enclosing brackets.

IPv6 Support in pjlib-util (DNS SRV and AAAA resolution)

The work for adding IPv6 support in pjlib-util is documented by ticket #419 and continued in ticket #1927.

DNS AAAA resolution will be performed for each DNS SRV record when flag `PJ_DNS_SRV_RESOLVE_AAAA` or `PJ_DNS_SRV_RESOLVE_AAAA_ONLY` is set in `option` param when invoking `pj_dns_srv_resolve()`. Also flag `PJ_DNS_SRV_FALLBACK_AAAA` will allow resolver to fallback to DNS AAAA resolution when the SRV resolution fails.

IPv6 Support in pjmedia (SDP, media transport)

The work for adding IPv6 support in pjmedia is documented by ticket #420.

The SDP representation has been updated to support IPv6 address, and the UDP media transport now also supports IPv6 sockets. There have been some *incompatible* changes introduced in pjmedia:

- the `rtp_addr_name` and `rtcp_addr_name` fields in `pjmedia_sock_info` structure (which describes the media transport address information) has been changed to use `pj_sockaddr` union rather than `pj_sockaddr_in` structure which is specific to IPv4,
- the `rem_addr` and `rem_rtcp` fields in `pjmedia_stream_info` also have been changed to use `pj_sockaddr` union.

IPv6 Support in pjnath (ICE)

The work for adding IPv6 support in pjnath is documented by ticket #422.

STUN and TURN transports support IPv6 already, and now ICE stream transport support IPv6 too. Also, ICE stream transport has been updated to be able to have multiple STUN and TURN transports and each STUN/TURN transport may use either IPv4 and IPv6.

Modifications in `pj_ice_strans_cfg`:

- Deprecated `af` field, if it is set, the value will be ignored, address family setting is now specified via STUN/TURN transport setting, i.e: `stun_tp.af` and `turn_tp.af`.
- Deprecated `stun` and `turn` fields, but for backward compatibility, those fields will still be used only if `stun_tp_cnt` and/or `turn_tp_cnt` is set to zero.
- Added `stun_tp` and `turn_tp` as replacement of `stun` and `turn` respectively, and they are array so application can have multiple STUN/TURN transports.
- Added function `pj_ice_strans_stun_cfg_default()` and `pj_ice_strans_stun_cfg_default()` to initialize `stun_tp` and `turn_tp` respectively with default values.
- Added compile-time settings `PJ_ICE_MAX_STUN` and `PJ_ICE_MAX_TURN` to specify maximum number of STUN/TURN transports in each ICE component.

Enabling IPv6 support in application using PJSUA

Application needs to configure SIP transport and SIP account with IPv6 support.

Creating SIP transport

Here is sample code for IPv6 SIP transport initializations.

```
pjsua_transport_config tp_cfg;
pjsip_transport_type_e tp_type;
pjsua_transport_id     tp_id = -1;

pjsua_transport_config_default(&tp_cfg);
tp_cfg.port = 5060;

/* TCP */
tp_type = PJSIP_TRANSPORT_TCP6;
status = pjsua_transport_create(tp_type, &tp_cfg, &tp_id);
if (status != PJ_SUCCESS)
    ...

/* UDP */
tp_type = PJSIP_TRANSPORT_UDP6;
status = pjsua_transport_create(tp_type, &tp_cfg, &tp_id);
if (status != PJ_SUCCESS)
    ...

/* TLS */
tp_type = PJSIP_TRANSPORT_TLS6;
tp_cfg.port = 5061;
tp_cfg.tls_setting.ca_list_file = pj_str("<path to CA file>");
tp_cfg.tls_setting.cert_file    = ...;
tp_cfg.tls_setting.privkey_file = ...;
tp_cfg.tls_setting.password     = ...;
status = pjsua_transport_create(tp_type, &tp_cfg, &tp_id);
if (status != PJ_SUCCESS)
    ...
```

Adding SIP Account

As described in #1926, it is recommended to explicitly bind an IPv6 account to an IPv6 SIP transport, i.e: via `pjsua_acc_config.transport_id` or `pjsua_acc_set_transport()`. Also, IPv6 usage must be explicitly set for media transport, i.e: via `pjsua_acc_config.ipv6_media_use`.

Here is sample code for setting up account using IPv6 SIP server.

```
#define SIP_USER        "user"
#define SIP_SERVER      "example.com"
//#define SIP_SERVER_IPv6 "[1234::5678]"
#define SIP_PASSWD      "pwd"

pjsua_acc_config acc_cfg;
pjsua_acc_config_default(&acc_cfg);

acc_cfg.id         = pj_str("sip:" SIP_USER "@" SIP_SERVER);
acc_cfg.reg_uri    = pj_str("sip:" SIP_SERVER);
acc_cfg.cred_count = 1;
acc_cfg.cred_info[0].realm     = pj_str("*");
acc_cfg.cred_info[0].scheme    = pj_str("digest");
acc_cfg.cred_info[0].username  = pj_str(SIP_USER);
acc_cfg.cred_info[0].data_type = PJSIP_CRED_DATA_PLAIN_PASSWD;
acc_cfg.cred_info[0].data      = pj_str(SIP_PASSWD);
```

```
/* Bind the account to IPv6 transport */
acc_cfg.transport_id = udp6_tp_id; // udp6_tp_id is an UDP IPv6 transport ID, e.g: outputed by
                                   // pjsua_transport_create(PJSIP_TRANSPORT_UDP6, ..., &udp6_tp_id)

/* Enable IPv6 in media transport */
acc_cfg.ipv6_media_use = PJSUA_IPV6_ENABLED;

/* Finally */
status = pjsua_acc_add(&acc_cfg, PJ_TRUE, NULL);
if (status != PJ_SUCCESS)
    ...
```

NAT64

In its doc, Apple suggests/requires that applications are capable of ?supporting IPv6 DNS64/NAT64 Networks. A common misconception in the SIP world is that by using NAT64, IPv4 and IPv6 interoperability can be automatically achieved (i.e. SIP registration, calls, and media flow will work seamlessly and smoothly between any two endpoints regardless of their address families (IPv4/IPv6)). As the doc says: `This (DNS64/NAT64) is an IPv6-only network that continues to provide access to IPv4 content through translation`, so a client behind a NAT64 network can reach an IPv4 endpoint, but not necessarily the other way around.

In more detail, an IPv6-only SIP client behind a NAT64 can communicate with IPv6 (or dual stack) server or clients just fine, but will experience problems with IPv4-only server or clients, because there are IPv6 address literals in the SIP/SDP fields (Via, Contact, SDP), which the IPv4 instance cannot understand.

According to ?RFC 6157, IPv6 Transition in the Session Initiation Protocol (SIP):

- Section 3.1: `In order to support both IPv4-only and IPv6-only user agents, it is RECOMMENDED that domains deploy dual-stack outbound proxy servers or, alternatively, deploy both IPv4-only and IPv6-only outbound proxies.`
- Section 4: `An IPv6 node SHOULD also be able to send and receive media using IPv4 addresses, but if it cannot, it SHOULD support Session Traversal Utilities for NAT (STUN) relay usage [8].`
- Section 4.2: `When following the ICE procedures, in addition to local addresses, user agents may need to obtain addresses from relays; for example, an IPv6 user agent would obtain an IPv4 address from a relay.`
- Section 4.2: `Implementations are encouraged to use ICE; however, the normative strength of the text above is left at a SHOULD since in some managed networks (such as a closed enterprise network) it is possible for the administrator to have control over the IP version utilized in all nodes and thus deploy an IPv6-only network, for example. The use of ICE can be avoided for signaling messages that stay within such managed networks.` (our note:? which means when network is not standardized to one IP version, the use of ICE is a "must").

Therefore, to support IPv6-IPv4 interoperability in NAT64 environment:

1. Our RECOMMENDATION is that when the client is put with an IPv6-only connectivity, the SIP server must also support IPv6 connectivity. For the media, user needs a "dual stack" TURN (a TURN server which supports IPv6 connectivity and able to provide an IPv4 relay address upon request). Then all the application needs to do is enable ICE and use TURN (support for dual stack TURN is only available in PJSIP 2.6 or later).
2. If 1) is not possible (no IPv6 server or not desirable to use TURN), we will need to replace all IPv6 occurences with IPv4 in the SIP messages and SDP. This feature is available in release 2.7.
    1. You need a STUN server which resides in an IPv4 network. Then set your `pjsua_config` to try IPv6 resolution of the STUN servers.
    2. Create UDP6 transport to get an IPv4-mapped address from the above STUN server.
    3. Bind account to a specific IPv6 transport, and enable IPv6 in media transport.
    4. Set account's NAT64 option to PJSUA_NAT64_ENABLED.

```
cfg->stun_try_ipv6 = PJ_TRUE;

tp_type = PJSIP_TRANSPORT_UDP6;
status = pjsua_transport_create(tp_type, &tp_cfg, &udp6_tp_id);

acc_cfg.transport_id = udp6_tp_id; // or tcp6_tp_id or tls6_tp_id
acc_cfg.ipv6_media_use = PJSUA_IPV6_ENABLED;

acc_cfg.nat64_opt = PJSUA_NAT64_ENABLED;
```