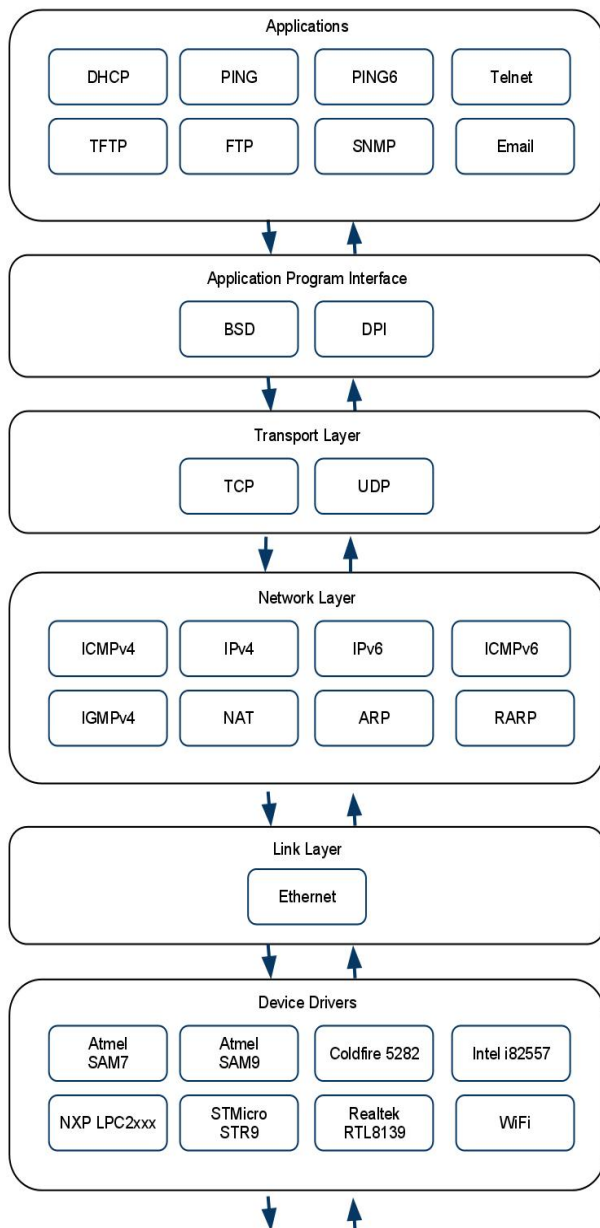# smxNS6 ™

IPv6 will usher in a new era of device connectivity, for purposes such as product registration, remote diagnosis, code updates, and data acquisition.

Dual IPv4 / IPv6 Stack Block Diagram

FEATURES

- Dual IPv4/IPv6 stack
- IPv6 Ready
- BSD Socket API for IPv4/6 hosts
- DPI Socket API for IPv4/6 hosts
- IPv6 Stateless Address Autoconfiguration (RFC 2462 compliant)
- Neighbor Discovery (RFC 2461 compliant)
- ICMPv6 (RFC 2463 compliant)
- IPv6 Resolver
- 67K ROM Footprint
- 16K RAM Footprint
- Transfers TCP information at 1887 Kbytes / sec (Atmel AT91SAM9260)
- Compatible with SMX RTOS and other Micro Digital products
- smxAware inspection of packet and network information

smxNS6 adds IPv6 capabilities to the smxNS TCP/IP stack, thus upgrading systems to the latest network standard and providing access to the new 128-bit network addresses. With IPv6's address space of more than 10^{38} addresses, there is room to deploy a huge number of devices with unique addresses without fear of running out of address space. In fact, manufacturers can assign unique IP addresses to all of their products pretty much forever. IPv6 devices can be uniquely accessed anytime, anywhere in the world as long as they are connected to an accessible network. Adoption of IPv6 will eliminate various IP address extension mechanisms such as NAT, in favor of simple, direct access to devices.

Deploying smxNS6 now is the best way to future-proof network-connected products that are currently in

development. smxNS6 will run well in an IPv4 environment where there is just IPv4 traffic, and it will have the ability to handle IPv6 traffic when IPv6 systems are connected. Alternatively, new products can use smxNS now and be upgraded to smxNS6 when it becomes necessary. There are some extensions to the current smxNS API (see below) and other differences to take into account, when transitioning from IPv4 to IPv6, therefore some rework of application code should be expected in order to upgrade.

General

smxNS6 uses the "dual stack" approach for adding IPv6 support. A dual stack implementation contains modules to handle both IPv4 and IPv6 packets that pass through the network layer of the stack, as shown in the diagram.

For incoming traffic, a two-byte field in the Ethernet header known as the "Ethertype" field may specify either IPv4 (code 0x0800) or IPv6 (code 0x86dd). Depending on the value in this field, the incoming packet processing will direct the packet to either the IPv4 module or the IPv6 module.

For outbound traffic, when a network application sets up a connection to another host, the type of the host can be detected based on the IP address that is specified for the connection. If the remote host is specified via a host name, then a preference for IPv4, IPv6, or either can be specified using the "hints" parameter in the call that opens the connection.

For network servers, the application typically performs a "passive" open, accepting remote connections from all hosts, so long as they are attempting to connect to the "port" assigned to the network service. For example, a Telnet server listens on port 23, and accepts connections from all hosts trying to reach that port. The server application can specify whether it will service IPv4, IPv6, or either type of host when performing the call to open the connection.

smxNS6 can be configured at compile time to support any combination of IPv4 and IPv6 traffic using two simple macros. If SNS_PROTO_IPV4 and SNS_PROTO_IPV6 are both set to 1, then the system compiles as a dual-stack implementation. Turning either of the macros off will drop support for that protocol. For example, if SNS_PROTO_IPV6 is set to 0, incoming IPv6 packets will be ignored.

Existing users of the standard smxNS TCP/IP stack should find smxNS6 to be quite familiar, since it simply adds new modules for IPv6-specific functions along with updated project files that support building the dual-stack version.

Features

smxNS6 integrates with the standard smxNS TCP/IP stack by adding modules specific to IPv6, ICMPv6, Neighbor Discovery and other IPv6 features, and by adding small sections of IPv6-specific logic to existing modules such as Ethernet and TCP. A project file specific to a given toolchain pulls in the full set of modules and creates an IPv4/IPv6 dual stack that can handle both IPv4 and IPv6 traffic.

IPv6 increases the minimum link MTU from 576 bytes to 1280 bytes. The MTU is the Maximum Transmission Unit, which is the amount of information that can be put into a single packet. This increase means that information can be packaged in sizes that increase throughput and minimize overhead for transferring information.

Duplicate Address Detection is specified as part of IPv6 Stateless Address Autoconfiguration. This is an automatic feature that ensures that all network addresses are unique, whether they are allocated automatically or manually. The initial check takes about 1 second to complete, and the protection mechanism continues to check as the system is running. With Duplicate Address Detection, IPv6 stacks are capable of automatic configuration. This avoids the risk of time-consuming troubleshooting due to misconfigured interfaces.

IPv6 eliminates broadcast delivery of packets and instead uses multicast addresses specific to the functions that need to be supported. This means that the stack no longer needs to process every broadcast packet that is sent on the network, but only certain multicast frames. The end result is fewer interruptions of the processor and more cycles for other tasks.

smxNS6 includes diagnostic tools that allow you to inspect network objects such as connections, address information and buffer memory usage within your Integrated Development Environment debugger. smxNS6 can also maintain a log of events, ranging from the opening of a network interface to summarizing every packet that flows through the stack. These network specific features in smxAware make it simpler to develop and optimize network applications.

smxNS6 has passed the "IPv6 Ready Phase 1" verification test suite, which ensures that the implementation is correct. This test suite consists of more than 300 individual tests and is developed and maintained by the IPv6 Forum, an industry group with the goal of verifying correct implementations and promoting the adoption of IPv6.

smxNS6 supports a wide variety of Ethernet controllers, and new drivers are regularly being added. The drivers include support for many System-on-Chip devices, some with relatively modest on-board RAM, which is possible due to the small memory footprint of smxNS6. See the "Ether" column of the chart at www.smxrtos.com/processors.

API Extensions

smxNS6 adds the following functions to the socket and DPI (proprietary) APIs used in standard smxNS so that client and server applications can access IPv6 networks.

getaddrinfo(hostname, servname, hints, res) obtains address information based on host and port information. This function returns the value 0 for success, otherwise it provides an error code explaining why the function did not succeed. getaddrinfo() can be used with both IPv4 and IPv6 hosts, and can be applied where gethostbyname() or gethostbyaddr() would ordinarily be used. The hostname parameter can be either a host name that is translated to a numeric IP address using a DNS server, or it can be an IP address. The servname parameter specifies the port number for the connections. The hints parameter can be used to specify the preferred socket type or protocol. The res parameter points to the address data structures that are returned by this function.

freeaddrinfo(res) returns memory allocated for the getaddrinfo() function back to the memory pool. The res function points to the address data structure. There is no return value.

gai_strerror(errcode) converts an error code returned by getaddrinfo() into a string and returns a pointer to that string.

inet_ntop(af, src, dst, cnt) converts the given address data structure into a string. This function works with both IPv4 and IPv6 addresses. The af parameter specifies the address family (IPv4 or IPv6) for the conversion. The src parameter points to the address data structure to be converted. The dst parameter points to a buffer that can receive the converted address. The cnt parameter specifies the size of the receiving buffer. The function returns a pointer to the converted string, or it returns NULL if there was a problem in making the conversion.

inet_pton(af, src, dst) converts a string into an address data structure. This function works with both IPv4 and IPv6 addresses. The af parameter specifies the address family for the conversion. The src parameter points to the address as a character string. The dst parameter points to an address data structure that will receive the conversion. The function will return 1 for success, otherwise it will return a code that specifies why the conversion could not be completed.

Nopen(to, protoc, local_port, remote_port, flags) from the smxNS DPI API is extended by accepting an IPv6

string in the "to" field, and by accepting strings such as "TCP/IPv6" in the protoc field. The definitions for the other parameters remain the same as they are in the IPv4 version of smxNS.

SOCKET_FAMILY(conno) macro returns the address type, either IPv4 or IPv6, for a given connection.

SOCKET_HASMYADDR6(conno) macro allows an application to determine if the site local address for a given connection has been established yet. The macro will evaluate as 0 if the address has not yet been established, otherwise the macro will return non-zero once the address has been established.

SOCKET_MYADDR6(conno) macro returns the site local IPv6 address for the given connection.

SOCKET_IPADDR6(conno) macro returns the remote host's IPv6 address for a given connection.

SOCKET_OWNIPADDR6(conno) macro returns the local host's link local IPv6 address for a given connection.

Memory Usage

The smxNS IPv6 stack makes efficient use of ROM and RAM. Following are the memory requirements for a typical basic configuration that supports both IPv4 and IPv6. The IPv6 features in smxNS6 add about 20K ROM to the memory requirements as compared to the IPv4 version of the stack.

Setting	Name	Value
Network Interfaces	NNETS	1
Connections	NCONNS	4
Router table entries	NCONFIGS	8
Frame buffers	NBUFFS	5

Component	ARM Thumb			ARM			ColdFire		
	RAM	**Stack**	**ROM**	**RAM**	**Stack**	**ROM**	**RAM**	**Stack**	**ROM**
Core library	15.6	2.0	46.0	15.6	2.0	66.6	23.3	2.4	87.9

Notes
- See the smxNS datasheet for other component (add-on) sizes, which are nearly the same for IPv6.
- These memory requirements are typical for a system that services one active TCP session at a time.
- For each additional active session, smxNS should be configured with NCONNS increased by 1, NCONFIGS by 1, and NBUFFS by 5. So each additional active session (client or server) adds about 8KB to the RAM requirement.
- About 250 bytes more stack are needed for the main NetTask (than for IPv4) and for each application task that establishes a connection to an IPv6 host.