



# **IPV6 NETWORKING PROTOCOL UPGRADE SERVICES**

## The Client Overview

Our client is a leading global provider of network traffic, application performance management and business continuity solutions for mission-critical application environments. Our client's dependable software, services and support utilize today's technologies to provide cost-effective, innovative and reliable solutions allowing their customers to gain competitive advantage and better utilize their existing infrastructure for enhanced return on investment. Hundreds of customers worldwide rely on our client's products to optimize the performance and availability of their business applications and information.

## Engagement situation & Challenges

In networking parlance, load balancing is the act of distributing workload evenly across two or more computers, network links, CPUs, or other resources, in order to obtain optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Using multiple components with load balancing, reliability can be increased through redundancy, as opposed to the use of single component. The load balancing function is best achieved through the use of software.

Our client's existing product line was capable of handling IPv4 (RFC 791) TCP/IP traffic. IPv4 uses 32-bit (four-byte) addresses, which limits the address space to 4,294,967,296 (2<sup>32</sup>) possible unique addresses. With increasing use of internet and mobile devices, the IPv4 addresses are getting exhausted. So a new Internet Protocol IPv6 (RFC 2460) is designed to succeed IPv4. IPv6 uses a 128-bit address. The new address length thus supports 2<sup>128</sup> addresses, several multiples more. This expansion provides flexibility in allocating addresses and routing traffic thereby eliminating the primary need for network address translation (NAT), which gained widespread deployment as an effort to alleviate IPv4 address exhaustion. There was a huge demand from client's customers for support of IPv6 protocol in their products.

Due to change in the address size, structure and protocol for IPv6, there was a need to port the existing product line to support IPv6 addresses, and do the network load balancing for the IPv6 traffic. This would include changes in the device driver, application software and GUI to support new and bigger IPv6 address.

Our client wanted a solution that would provide the same features for handling IPv6 protocol as it used to support for IPv4 protocol. The objective was to enable the customers use the product for their IPv6 enabled network in the same way as they used it for their IPv4 network.

Since the product includes driver for various OS platforms, there was a need to understand implementation of IPv6 by various OS platforms that the product is supported on. Since the existing

product was not well documented, there was a need to sort out the business functionality based on validations from the existing code.

In the simplest IPv4 version of product, a VIP (Virtual IP used to represent the cluster as a whole to the client) has a dedicated scheduler whose responsibility is to collect the requests from the clients and forward them to the respective servers configured to serve the requests. The VIP gets added on the loopback interface of all the servers which are configured to be part of the cluster for a VIP. This method is in place so that the server completing the request could directly respond back to the client. This improves the response time because the request does not have to go back to the scheduler and then to the client. But in the IPv6 version the addresses added on the loopback adapter is not recognized by the IPv6 stack, hence the IPv6 VIP address could not be added on loopback interface.

The main challenges at the driver level were:

- Making the existing data structures used in the kernel compatible with IPv6 addresses.
- Preventing duplicate address detection since the IPv6 VIP being added on the active interface, triggers duplicate address detection. Requirement for special handling of the IPv6 packets.
- Requirement that the Address Resolution Protocol (ARP) be implemented for IPv6 in a similar manner as it is implemented in case of IPv4, so that the clients in the new environment can detect the scheduler in a similar manner. Existing product for IPv4 uses ARP (Address Resolution Protocol) for neighbor detection to start the communication between scheduler and client..

The primary challenges at the user level were:

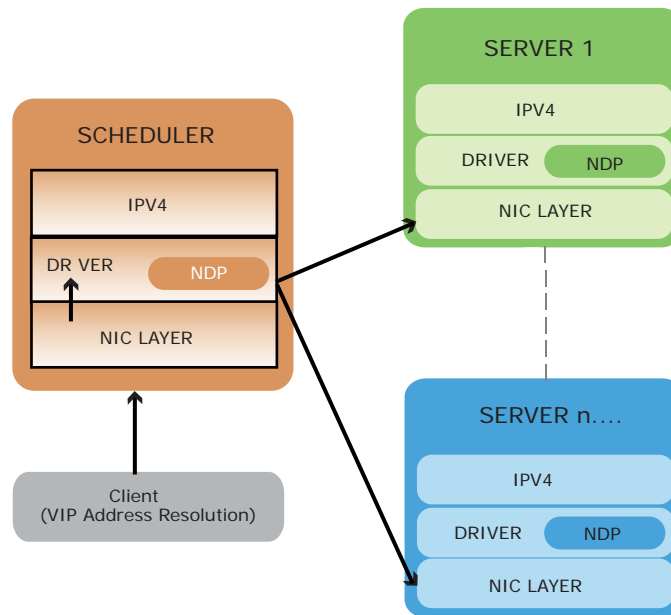
- Since the product is supported on multiple OS platforms, it was required to maintain the code base platform neutral as much as possible.
- Due to change in the IP address format, GUI has to be modified to display and handle the IPv6 address properly.
- For any performance related issue tracking, there was a need to emulate heavy IPv6 load generation.
- The product has many supporting utilities, which needed to interoperate in IPv6 mode.

## Xoriant's contributions in engagement

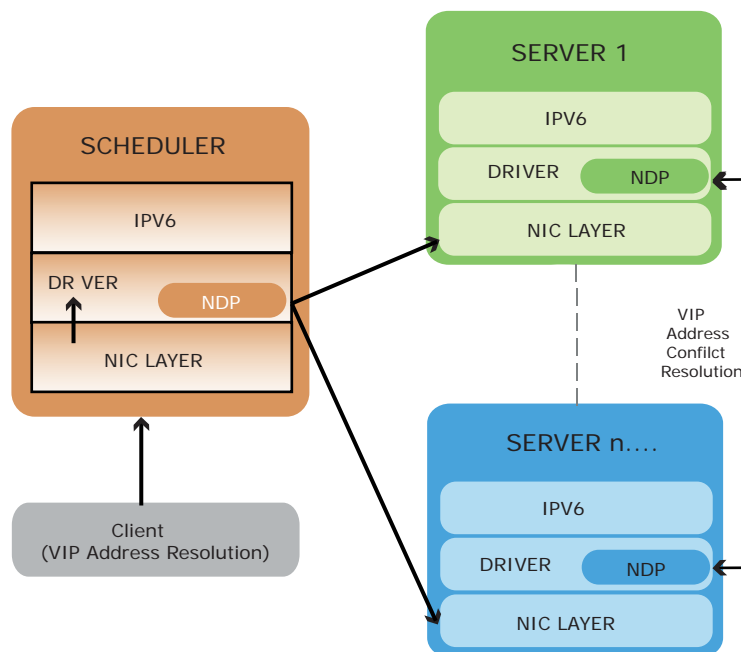
Our client was looking for a partner who has expertise in the areas TCP/IP stack, network management, kernel driver development, kernel level and network level debugging, web technologies, user

interfaces and shell scripts. Our teams worked with our client to gain understanding of the product with a view to re-architecting it to support IPV6 protocol. Since the product included driver for various OS platforms, there was a need to understand implementation of IPV6 by various OS platforms that the product is supported on. Since the existing product is not well documented, there was a need to go through the existing code to understand the implementation of various features by the product.

### IPv4 Stack



### IPv6 Stack



### Some key aspects of the design and implementation phase were:

- Modification of the kernel data structures to enable it to store longer IPv6 addresses.
- NDP (Neighbor Discovery Protocol) similar to ARP (Address Resolution Protocol) in IPv4 was implemented in the driver. This prevented duplicate address detection for IPv6 VIP on all the servers, by allowing only the scheduler to respond to the clients even though the same IPv6 VIP is present on the physical NICs on all the servers in the site.
- Since the IPv6 packets have different structure and field lengths, the packet handler was modified to parse the IPv6 packet properly. Also due to difference in the packet structure a new checksum calculation method was implemented for IPv6 to calculate checksum for TCP packet.
- When the first client packet, which is forwarded by the scheduler, reaches the server, the IPv6 stack on the server needs to know if the client is active. Special handling of the packet sent by the IPv6 stack to find if the client is active was implemented.
- Coding was done in a manner that same code base could be used for different OS platforms by using platform dependent macros and functions.
- Appropriate GUI changes were made to support longer address format. IPv6 address parsing was handled in configuration files.
- Special handling of the IPv6 addresses representation in GUI was implemented where the same IPv6 address can be represented in compact or expanded formats.
- IPv6 load generator was developed to test the product for the performance depending on the intensity of the load and number of simultaneous requests.
- Intercommunication of the processes for various purposes like heartbeat and syncing of configuration was completely implemented in IPv6.
- All the user level utilities used with the product were completely ported over to work for IPv6.
- All the above implementation was done to execute the product on 32 bit as well as 64 bit systems.

### Xoriant's contributions in testing were:

- A setup of IPv6 enabled cluster was done to test the product on a virtualized environment. Various IPv6 enabled applications were deployed for doing the load balancing test for them.
- A cluster consisting of different guest operating systems was specially developed to test the product for load balancing across different mix of platforms.
- A load generating utility was developed to generate IPv6 traffic for testing the product with

varying intensity of load for long duration, to test the product for performance, stability and scalability.

- Extensive test coverage of compounding scenarios was achieved through use of automated scripts and manual testing for different combination of loads and different mix of OS platforms.

#### Tools & Technologies

- **OS: Windows, Linux, Solaris**
- **Languages: C, C++, Java**
- **Tools: Visual Studio 2005, GCC, GDB, Windows Driver Kit**

#### Benefits

- By establishing a multi-faceted team in India, our client was able to augment the product development teams, thus increasing the engineering bandwidth in a cost-effective manner.
- The increased engineering bandwidth helped our client to simultaneously enhance the existing product functionality and re-architect the product to support IPv6 protocol, thus achieving time-to-market and cost goals.
- Using Xoriant's expertise in networking protocols, the first deliverable of the product was completed in less than 4 months.
- Since the product functionality and user experience was kept same as the version that supported IPv4 protocol, it was easy for our client's customers to upgrade their setup to support the IPv6 protocol.

#### Client Testimonial

Xoriant demonstrated a very flexible approach to help us enhance our existing product functionality and re-architect the product to support IPv6 protocol. Xoriant was able to put together a good team; consistently track and monitor the progress. My experience with Xoriant is that they have a team that is technically very competent especially in the areas of TCP/IP Stack, network management and also professional in its development approach.

-Director Engineering