**Question 2**

Libuv is a multi-platform C library that provides the underlying runtime for Node.js. It provides an event-driven, non-blocking I/O model that allows for high-performance, scalable network applications to be developed in JavaScript.

Libuv works by creating a thread pool to handle blocking I/O operations such as file I/O or DNS resolution. When a blocking operation is requested, libuv assigns it to a thread in the pool, allowing the main event loop to continue processing other events. Once the blocking operation is complete, libuv sends a notification back to the event loop so that the appropriate callback can be executed.

Epoll is a scalable I/O event notification system that is used by Libuv to interact with the kernel. When a file descriptor is registered with epoll, it can be monitored for events such as incoming data or the establishment of a connection. This enables libuv to manage many I/O operations more efficiently by only processing those that have events ready, rather than constantly polling for activity.

**Question 3**

To send and receive packets to/from the E1000 device the user can create a socket using the socket() system call, then they can use the send()/sendto() or bind() and recv()/recvfrom() system calls respectively. The E1000 driver handles the details of sending/receiving the data from the network, in addition this driver provides functionality for operations other than sending/receiving packets such as initializing the device or configuring its settings.

**Question 4**

eBPF (extended Berkeley Packet Filter) is a virtual machine that can be used to execute custom programs at various points in the networking stack. These programs are executed in a secure sandboxed environment, ensuring that they cannot harm the system or other applications.

XDP (eXpress Data Path) is a framework that allows packet processing to be performed directly on the network interface card (NIC). By moving packet processing from the kernel space to the driver space, XDP enables fast packet processing. It works by intercepting packets as they enter the kernel space and applying a set of rules to determine how to process them.

Both technologies allow for fast and efficient packet processing in Linux.

This is how one could build a simple load balancer using both eBPF and XDP.

1. Create an eBPF program in which the load balancing logic is implemented. This program will run on the network interface card and inspect incoming packets to determine which backend server they should be forwarded to. The program can perform load balancing using a variety of techniques, such as round-robin or weighted random selection.
2. Use the bpf() system call to load the eBPF program into the kernel. This will connect the program to the NIC's XDP hook and cause it to be executed for each incoming packet.
3. Configure the NIC to use the XDP hook by setting the appropriate XDP mode. This can be done using the ethtool command or other system utilities.
4. Configure the backend servers to accept load-balanced traffic. This can be accomplished through a variety of means, including the configuration of virtual IP addresses, network routing rules, and the use of a load balancer such as HAProxy.