eBPF is a networking technology used for packet processing at the kernel level. It provides a way to modify the behaviour of packets as they are received and transmitted. eBPF programs are compiled into native machine code by the kernel at runtime, giving them highly efficient execution without the need for an interpreter.

eBPF programs can intercept and inspect network packets in order to modify the packet and can attach to network hooks such as sockets to allow for packet processing. XDP is a specialized hook that allows eBPF programs to be attached to the receiving path of the network interface. This means XDP programs sit above the network stack and are able to process packets as they are received by the network interface, and before they are passed up the networking stack. eBPF programs are executed directly in the kernel making them ideal for low latency, high throughput packet processing programs such as load balancers.

eBPF and XDP are used in many networking applications, but in the case of load balancers they are used re-distribute incoming network traffic across multiple servers for efficient resource utilization and to provide high availability to the services that makes use of them. An explanation of how a simple load balancing system might be achieved follows below.

Firstly, all of the available servers and their associated information such as IP addresses and port number is stored in an array of structs or an eBPF map. The load balancing eBPF program would intercept incoming packets from a client at the XDP hook point on the network interface and then read the destination IP address and port in the packet’s headers. A server would then be selected using an algorithm such as round robin to select a destination to forward the packet to. When a server is selected by the algorithm the chosen server would be found in the eBPF map with its network details, the destination IP address and port in the packet’s headers would then be updated with the new server's IP address and port respectively. The packet would then be forwarded to the selected server using XDP\_TX. In certain circumstances, the program might also need to pass the packet to its original destination using XDP\_PASS, drop the packet entirely using XDP\_DROP, or use XDP\_ABORTED in the case of an error.