qbkit network driver

We require some drivers in order to provide connectivity between our qbkit development boards (running Linux) and the host computer (running Linux/OSX/Windows).

This should not depend on any specific USB device drivers on the host, beyond the usual USB controllers / hubs.

Initially, the driver should just provide networking (similar to g_ether), however we plan to extend the functionality in future by addition of extra endpoints.

Stack

In order to provide an IP network connection between the qbkit and the host, and in such a way that the host-side driver can be used to give internet access to the qbkit, we have the following design:

1. qbkit OS: IP networking
2. qbkit SW: TUN interface
3. qbkit SW: USB gadget driver (libusbg? FunctionFS+ConfigFS?)
4. (USB connection from qbkit to host)
5. host SW: USB device driver (libusb)
6. host SW: NAT
7. host OS: IP networking
8. host OS: internet connection
9. (Internet connection from host to the world)

Details

- A "TUN" interface is created on the qbkit, and is configured as the default route.
- The software which provides the TUN interface then forwards packets in both directions between the TUN and USB gadget interfaces.
- The USB gadget interface is ideally provided via existing userspace interfaces rather than with a new kernel driver, for example: FunctionFS.
- The qbkit is connected to the host via USB. The qbkit shall use custom vendor/product IDs.

- A custom USB driver on the host, using libUSB and compatible with OSX+Linux+Win7+Win8+Win10, provides a software interface from the host to the qbkit.
- The USB driver also implements NAT such that it allows the qbkit to make outgoing TCP connections to the internet, without requiring the user to do any specific network configuration on the host (such as "internet connection sharing").
- The USB driver therefore accesses the internet as any normal program would, by opening TCP sockets.

Language/quality

The qbkit-side software should be written in C, and should compile successfully with the following GCC flags:

```
-Wall -Wextra -Werror -std=gnu11
```

The host-side software may be written in any one of: C, C++, C#, Java.

C/C++ code should follow the Linux Kernel coding style with the following exceptions:

- Hard-wrapping is not mandatory and is also not forbidden.
- Descriptive variable/parameter names are preferred unless the purpose of the variable is really obvious.
- Prefer to declare variables in the scope and lines where they are used, rather than at the start of the function.

An example program for creating and using a TUN device is available on my Github @ github.com/battlesnake/tun-example.