COMP3301/COMP7308 2020 Assignment 1 - Generic Network Virtualization Encapsulation (Geneve) using Event Driven Programming

- Due: 31st of August 2020, 4pm
- $Revision: 226 $

1 Geneve: Generic Network Virtualization Encapsulation

This assignment asks you to write a simplified implementation of Geneve: Generic Network Virtualization Encapsulation called `gnveu`. The implementation will support multiplexing multiple Ethernet tunnels over a single UDP socket.

The purpose this assignment is to expose you to event driven programming, working against existing APIs and libraries, and the use of the OpenBSD code style and tool chain.

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code will be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: https://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism

2 Specifications

`gnveu` is used to tunnel multiple network connections over a single UDP connection. It supports tunnelling of Ethernet (layer 2) packets inside the Geneve protocol.

Each network tunnel is connected to the kernel via the `tap(4)` device driver. Packets read from the kernel are encapsulated in Geneve before being sent on the UDP socket. Geneve packets received from the socket are decapsulated and written to the `tap(4)` device driver. The Geneve protocol will also be used to filter Ethernet traffic.

2.1 Code Style

Your code is to be written according to OpenBSD's style guide, as per the `style(9)` man page.

2.2 Compilation

Your code is to be built on an `amd64` OpenBSD 6.5 system. It must compile as a result of running `make(8)` in the root directory of your submitted assignment. Compilation must produce a binary called `gnveu` in either the same directory, or in the `obj` directory if it exists.

Compilation of `gnveu` must succeed with `-Wall` passed as a flag to the compiler.

2.3 Invocation

When run with no arguments, or extra or unknown options, `gnveu` should print a usage message to `stderr`:

```
usage: gnveu [-46d] [-l address] [-p port] -t 120
            -e /dev/tapX@vni
            server [port]
```

and terminate with a non-zero exit code. Note VNI stands for Virtual Network Identifier.

`gnveu` takes the following command line arguments:

-4 Force `gnveu` to use IPv4 addresses only.

-6 Force `gnveu` to use IPv6 addresses only.

-d Do not daemonise. If this option is specified, `gnveu` will run in the foreground and log to `stderr`. By default `gnveu` will daemonise.

-t idle_timeout (value specified in seconds) Close and exit after idle_timeout duration is exceeded. If no traffic is received for a duration that exceeds the timeout, then the system is considered to be idle. The duration must be a postive value specified in seconds. If no idle_timeout is given (or is invalid), then an error message must be displayed. If a negative value is given, then no idle_timeout should occur.

-l address (Optional and does not need to be used) Bind to the specified local address. By default `gnveu` does not bind to a local address.

-p port (Optional and does not need to be used) Use the specified source port. By default `gnveu` should use the destination port as the source port.

-e /dev/tapX@vni Tunnel enter/exit point for Ethernet (level 2) traffic for the specified tunnel device. The @vni must be specified. Note VNI stands for Virtual Network Identifier per the draft-ietf-nvo3-geneve-16 specification. The VNI is used to filter the Ethernet traffic (depending on the payload type). When either of the following VNI is specified, only ethernet packets containing the corresponding payload type must be encapsulated and tunnelled. Other types of ethernet traffic should be filtered out and not tunnelled. Hint: Version field in IPv4 and IPv6 packets.

VNI	Traffic
4096	IPv4 only (filter out other types)
8192	IPv6 only (filter out other types)
Any not listed above	Any Ethernet

server The address or host name of the remote tunnel endpoint.

port Use the specified port on the server as the remote tunnel endpoint. By default `gnveu` should use port 6081 as per the draft-ietf-nvo3-geneve-16 specification.

At least one tunnel must be configured.

If `gnveu` is unable to bind to the specified local address and port, connect to the remote server and port, or open the specified tunnel interfaces, it should generate an appropriate error and terminate with a non-zero exit status.

2.4 Functionality

- `gnveu` should use a single UDP socket to connect to the address specified by the command line arguments for exchange of Geneve UDP packets

  - it must support both IPv4 and IPv6 sockets
  - addresses may be numeric IP addresses or host names URL
  - ports may be numeric or a name listed in `/etc/services`
- Repeated specification of command line options may overwrite previously set values
- `gnveu` is only required to implemented a simplified version of draft-ietf-nvo3-geneve-16:
  - it only has to create a single UDP connection to the server
  - it can expect to receive replies to the local port it transmits from
  - encoding entropy by using multiple source ports is not required
  - it will act as a client talking to a server
- It only needs to support Geneve encapsulation of Ethernet (Geneve type: 0x6558) packets.
- Tunnels are uniquely identified by their Virtual Network Identifier (VNI)
  - The VNI is used to specify the type of traffic to encapsulate.

2.5 Required Dependencies

`gnveu` must only use the following libraries and the APIs they provide to implement the above functionality.

2.5.1 `libc`

`libc` refers to the ISO or POSIX standard C library provided by UNIX and UNIX like operating systems. A `libc` contains the APIs required by `gnveu` for such tasks as resolving host and service names to IP addresses and ports, creating network sockets and connections, and opening and reading files.

2.5.2 `libevent`

According to http://libevent.org/, `libevent`:

> ... provides a mechanism to execute a callback function when a specific event occurs on a file descriptor or after a timeout has been reached. Furthermore, libevent also support callbacks due to signals or regular timeouts.

OpenBSD ships with `libevent` 1.4 with local patches. It is this version of `libevent` in the base system what you are required to write `gnveu` against.

2.5.3 `tap(4)`

According to the `tap(4)` manual page:

> The tap driver provides an Ethernet interface pseudo-device. Packets sent to this interface can be read by a userland process and processed as desired. Packets written by the userland process are injected back into the kernel networking subsystem.

2.6 Logging API

`gnveu` may use the log.c and log.h file supplied by the course, rather than implementing this functionality itself.

The API and documentation is provided at https://github.com/dgwynne/lerr. If used, the code should copied into the submitted assignment.

2.7 Restrictions

- `gnveu` must operate as a single process/thread, therefore it may not fork (after daemonising) or create threads.

2.8 Recommendations

The focus of this assignment is event driven programming, not on creating new implementations of common functionality. It is strongly recommended that the following APIs are used as part of your program:

- `getopt(3)` - get option character from command line argument list
- `err(3)`, `warn(3)`, etc - formatted error messages
- `getaddrinfo(3)` - host and service name to socket address structure
- `gai_strerror(3)` - get error message string from EAI_xxx error code
- `strtonum(3)` - reliably convert string value to an integer
- `daemon(3)` - run in the background
- `ioctl(2)` - control device, used to configure non-blocking file descriptor operation

3 Submission

Submission must be made electronically by committing to your Subversion repository on `source.eait.uq.edu.au`. In order to mark your assignment the markers will check out `a1` from your repository. Code checked in to any other part of your repository will not be marked.

As per the `source.eait.uq.edu.au` usage guidelines, you should only commit source code and Makefiles

4 Testing

4.1 Ethernet

A server is running on the IPv4 address of `fungus.labs.eait.uq.edu.au` on port 3301 that implements a Geneve UDP server. This server implements a virtual Ethernet switch, allowing Geneve UDP tunnels to communicate, but has no support for IPv4 and IPv6 tunnels. The server has the following VNIs and IPv4 subnets configured:

- VNI 0: 100.64.1.1/24

IPv4 addresses can be requested via DHCP on all four networks. The server should support communication between tunnels connected to the same networks.

4.2 Packet Inspection

Several tools are available that are capture and disect network packets for inspection. These may be used to verify that the packets on the wire conform to the expected format. `tcpdump(8)` is available in the OpenBSD base system, and supports disection of Geneve inside UDP.

4.3 `tcpdump` Update Required

You must update `tcpdump` by doing the following:

1) Make sure your code is backed up in your SVN folder before continuing.
2) Login into your openBSD VM as a normal user.
3) `cd /usr`
4) `cvs -Qd anoncvs@anoncvs.au.openbsd.org:/cvs co -P src/sys`
5) `cvs -Qd anoncvs@anoncvs.au.openbsd.org:/cvs co -P src/lib`
6) `cvs -Qd anoncvs@anoncvs.au.openbsd.org:/cvs co -P src/sbin`
7) `cvs -Qd anoncvs@anoncvs.au.openbsd.org:/cvs co -P src/usr.sbin`
8) `cd src/usr.sbin/tcpdump`
9) `make obj`
10) `make`
11) `sudo make install`
12) `sudo reboot`

13) To check that tcpdump is installed - run “man tcpdump” and you should see under the “-T” option, “geneve”

4.4 `tcpdump` Command Example.

An example of it’s usage:

```
# tcpdump -v -e -i em0 -Tgeneve host fungus.labs.eait.uq.edu.au and port 3301
tcpdump: listening on em0, link-type EN10MB
10:57:34.583486 bc:2c:55:36:7f:fb 74:88:2a:d4:b2:00 ip 88: dlgbox.eait.uq.edu.au
.3301 > fungus.labs.eait.uq.edu.au.3301: [udp sum ok] gre [] 6558 fe:e1:ba:dd:8c
:f9 fe:e1:ba:da:ab:9c ip 42: 100.64.0.38 > 100.64.0.1: icmp: echo request (id:00
00 seq:0) [icmp cksum ok] (ttl 64, id 1, len 28) (ttl 64, id 1, len 74)
```