COMP3301/COMP7308 Assignment 3

- School of Information Technology and Electrical Engineering
- The University of Queensland
- Semester Two, 2015
- Due: 8pm Saturday 31 October 2015
- Revision: $Revision: 100 $

1 Ethernet over UDP tunnelling

This assignment asks you to implement a network driver in the OpenBSD kernel.

The majority of network drivers in OpenBSD are for the various hardware devices in supported systems, but a handful implement virtual interfaces that encapsulate or abstract a network on top of another network or transport. For example, vxlan(4) is an industry standard protocol for encapsulating ethernet packets inside a UDP packet, gif(4) implements IPv4 or IPv6 encapsulation over IPv4 or IPv6, and pppoe(4) provides PPP over Ethernet encapsulation.

The protocol specified in this assignment supports tunnelling Ethernet packets over UDP frames.

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 may 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: http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism

2 Specification

You will add a new pseudo-device called eou(4) to the OpenBSD kernel that implements the client side of the protocol described below.

The Ethernet over UDP protocol allows encapsulated Ethernet packets to be exchanged between a client and server. The client periodically requests a connection or data link with a server, which replies with a message acknowledging the link is active. Once a link has been established, data packets containing the encapsulated ethernet frame can be exchanged in either direction between the client and server.

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 modifications and additions to the OpenBSD system must be integrated into the existing build infrastructure. The kernel will be rebuilt with your supplied code by following the process in the release(8) man page.

2.3 Protocol

EoU is a client/server protocol for enabling the exchange of Ethernet frames over UDP. The client initiates a connection between itself and the server by sending a PING message. The server replies with a PONG message if it can validate the clients PING message.

Once the connection has been established, the server and client are free to exchange DATA messages that encapsulate Ethernet frames.

The client must periodically send PING messages to the server to maintain the connection. If the client fails to send a PING message within an certain interval, the server can consider the client as disconnected and no longer send DATA messages to it. If the server does not reply with PONG messages to the clients PING messages, it may consider the server as disconnected and will no longer send DATA messages to it.

The client should send PING packets every 30 seconds. If the server does not receive a PING packet for 100 seconds, it may consider the client disconnected.

The EoU protocol defines a 6 byte header inside a UDP packet as defined below:

```
struct eou_header {
        uint32_t                eou_network;
        uint16_t                eou_type;
} __packed;

#define EOU_T_DATA        0x0000
#define EOU_T_PING        0x8000
#define EOU_T_PONG        0x8001
```

The EoU network field refers to a unique network id that scopes the connection and therefore the encapsulated Ethernet frames. It is possible to tunnel many separate Ethernet networks between the same client and server.

The type field indicates the type of message and payload for the current packet.

EoU DATA packets contain the header followed immediately by the encapsulated Ethernet frame.

EoU PING and PONG messages use the payload defined below:

```
struct eou_pingpong {
        struct eou_header        hdr;
        uint16_t                _pad;
        uint64_t                utime;
        uint8_t                        random[32];
        uint8_t                        mac[8];
} __packed;
```

The _pad field exists simply to align the subsequent fields to an 8 byte boundary, allowing loads and stores of fields as words on strict alignment architectures.

The utime field stores the senders current time in seconds past the UNIX epoch.

The random field is simply 32 random bytes generated by the sender for every packet.

EoU PING and PONG messages contain a MAC (message authentication codes). The MAC is to be generated using the SipHash-2-4 algorithm. The SipHash key is a secret shared by the client and server, but does not appear in the wire protocol. The MAC is generated by feeding the algorithm with the EoU network id, the utime, and random fields.

The peers validate each others messages by comparing their clocks to the time present in the PING and PONG messages, and validating the MAC. If the clock differs by more than 30 seconds each way, the packet is rejected. If the recomputed MAC does not match the one in the packet, it is rejected.

The EoU DATA packets are unauthenticated.

All EoU word fields are to be encoded in network byte order for transmission and reception.

Each EoU network id requires individual negotiation, ie, a client and server pair may tunnel multiple Ethernet networks by using unique EoU network ids, but each of those networks must be negotiated with separate PING and PONG messages.

2.4 COMP3301 Client Implementation

The OpenBSD kernel will be modified to add an eou(4) network pseudo-device.

It will only be required to interoperate with a single existing server, and should not require userland modifications, therefore several paramaters available in the protocol can be hard coded into the driver.

2.4.1 Functional Requirements

- The OpenBSD driver name should be eou(4)
- eou(4) should integrate into the kernel build as a pseudo-device
- eou(4) must be a clonable interface, ie, `ifconfig eou0 create` should cause an instance of eou(4) to be created and attached
- eou(4) must appear as an Ethernet network driver
- The eou(4) ioctl routine must handle the `SIOCSLIFPHYADDR`, `SIOCGLIFPHYADDR`, and `SIOCDIFPHYADDR` commands used by the `ifconfig(8)` tunnel command to specify the server and client address
- If the `SIOCSLIFPHYADDR` ioctl command does not specify a port on the server, eou(4) must assume port 3301
- The eou(4) ioctl routine must must handle the `SIOCSVNETID`, and `SIOCGVNETID` commands used by the `ifconfig(8)` vnetid command to specify the EoU network id
- Multiple instances of eou(4) may be configured between the local system and a server
- eou(4) must only send PING messages when the interface is configured up
- eou(4) must send PING messages every 30 seconds
- eou(4) must report the status of the server negotiation as interface link state, ie, until the server replies with valid PONG messages the link must appear down, after it must appear up
- eou(4) link must appear down if the server does not send a PONG packet for 100 seconds-
- eou(4) traffic for different networks between the same client and server (identified by a client IP address, server IP address, and server port) must use a single UDP connection

2.4.2 COMP3301 EoU Server Parameters

- Server addresses: 130.102.96.36:3301 and 130.102.96.36:1033
- Network IDs: 0 and 2310
- Pre-shared SipHash key: `63 6f 6d 70 33 33 30 31 63 6f 6d 70 37 33 30 38`

The server implementation may be made available for testing purposes.

2.5 Recommendations

2.5.1 Functionality

- eou(4) should use the `socreate(9)` API

2.5.2 APIs

The APIs may be useful in the implementation of the required functionality.

- `rwlock(9)` - interface to read/write locks
- `malloc(9)` - kernel memory allocator
- `timeout_add(9)` - execute a function after a specified period of time
- `task_add(9)` - task queues
- `mbuf(9)` - kernel memory management for networking protocols
- `socreate(3)` - kernel socket interface
- `queue(3)` - implementations of singly-linked lists, doubly-linked lists, simple queues, and tail queues
- `tree(3)` - implementations of splay and red-black trees

2.6 Constraints

2.6.1 Uniprocessor kernels

The kernel zones implementation is not expected to be multi-processor safe and will only be tested on a uni-processor system running a GENERIC (not GENERIC.MP) kernel.

3 Submission

Submission must be made electronically by pushing changes to a git repository on `source.eait.uq.edu.au`. In order to mark your assignment the markers will clone the "master" branch from your repository. Code checked in to any other branch of your repository will not be marked.

Your repository is named "comp3301-s1234567-a3" or "comp7308-s1234567-a3" depending on which course you are enrolled in. s1234567 is replaced with your UQ username.

The repository has been pre-populated with a checkout of the OPENBSD_5_7 tag from the OpenBSD CVS repository.

The repository may be cloned via the following command:

```
git clone s1234567@source.eait.uq.edu.au:comp3301-s1234567-a3 src
```

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

The due date for this assignment is 8pm on Saturday the 31st of October, 2015. Note that no submissions can be made more than 120 hours past the deadline under any circumstances. The time at which changes are pushed to the repository is considered the time of submission, regardless of when the commits within the repository were made.

4 Marking Scheme

WARNING If your code doesn't compile and run, you won't get credit for it.

1. For this assignment, break the functionality into stages or pieces that you can work on independently.
2. Get one piece fully functional and thoroughly tested.
3. Go on to the next piece.
4. If you've tried to implement everything but haven't completed any pieces, you won't get credit for anything.

Individual functionality will be checked, tested, and marked.

For each piece of functionality, percentages of marks will be allocated as:

- 0 - 20% Function not implemented, or no working code
- 20 - 40% Some simple but incorrect functionality
- 40 - 60% Significant problems, such as frequent crashes or infinite loops
- 60 - 80% Moderate problems, fails relatively often
- 80 - 100% Few or no problems

Total marks out of 100:

- Coding style, readability, organization [10]
- Kernel configures and builds with eou(4) [5]
- Kernel boots and runs [5]
- eou(4) `create` and `destroy` work [10]
- eou(4) `tunnel` and `vnetid` configuration works [10]
- eou(4) enforces unique tunnel and vnetid configurations [10]
- eou(4) negotiates link [20]
- eou(4) reports link state correctly [10]
- Ability to communicate over eou(4) [20]

5 Revisions

Changes to the assignment specification can be reviewed at https://source.eait.uq.edu.au/viewvc/comp3301-pracs/2015/assignment3.md?view=log.