Emergency Service in IP using Multi-Level AQM

Project Final Report
CSC 573 Internet Protocols

Date: Nov 30, 2014

Mitul Mahendrakumar Panchal (200047044)
Gowtham Ramesh(200062672)
Apoorv Joshi (200059694)

https://sites.google.com/a/ncsu.edu/ncsu-ip-project/home

Abstract

With attraction to internet of things, in near future all kind of traffic will flow through internet. Internet is designed to support all types of traffic. There will be a need to provide emergency service over IP. Due to convergence of transport service to internet, the emergency service may include a emergency call from IP Telephony, emergency call from wireless carriers, emergency data from wireless body area networks, emergency communication for public warning system like ETWS [10]. This project aims to implement the possible modification for emergency services support in forwarding engines. Emergency services can be supported at various layers in internet architecture like alternate routing, resource reservation or active queue management. The project focuses on active queue management strategy to integrate emergency services in internet architecture.

The objective of this project is to provide acceptable quality of service for emergency packets and best effort services to non-emergency packets using active queue management. Emergency packets are given preferential treatment by dropping non-emergency packets in favor of emergency packets in active queue management. We have implemented a new queuing discipline which uses Multiple Average Multiple Threshold RED algorithm and exempt emergency packets during probabilistic drop maintaining the different service types for emergency services. We also propose a method for managing contention between emergency traffic of coupled queues after crossing max threshold for preferential dropping within emergency service types. We developed a kernel module to add support for new queuing discipline “ered” in linux networking stack and modified tc of iproute2 package to configure linux network interface with “ered” qdisc.

Table of Contents

1. Introduction

1.1. Motivation

The forwarding engine is the core to ubiquitous services provided by the internet. The design philosophy of Arpanet [7] provided the initial goals of internet and the order of importance. Survivability and support for multiple types of service were listed as important goals of early internet. The survivability goal in internet is implemented by multiple routes to the destination. This goal can be further helpful to sustain communication in emergency situations like natural disasters when there is increased use of network by all type of services and emergency services may be required to given preference without guarantee of any better QoS performance like jitter and delay in comparison with non-emergency traffic.

The another motivation for this project is Linux networking stack which provides open source and efficient way to implement and use various traffic control mechanisms using queuing discipline.

1.2. Objective

The objective of this project is to avail a new queuing discipline to the traffic control of linux networking module that provides preferential treatment to emergency users using multiple average multiple threshold RED implementation with emergency traffic support. The design of the queuing discipline is based on [1] with practical modifications as described in system level and detail level design in the report. We refer to new queuing discipline as ERED in this project.

1.3. Scope

The project focus is on the queuing discipline enhancement to support emergency users without losing the traffic classification. Though the authentication and admission control mechanism for emergency users and classification of service types using Diff Serv code points are important issues these are assumed to be handled by the edge router and not included in scope of this project.

2. Background

This section provides the background of the technology and concept used toimplement the new queuing discipline and integrate it with the Linux traffic control.

2.1. Queuing Mechanism

The existing qdisc in current distribution of Linux can be used for providing differential treatment to emergency packets by assigning a separate DSCP code point. But that would lose the differentiation of type of service for emergency users. The requirement of emergency user is no different than that of non-emergency user in terms of QoS performance like jitter and delay [11]. The important requirement for emergency user is availability and dependability which can be provided by preferential treatment in active queue management. Hence we use the DSCP code point only for classification of service types like video, audio, background etc and not to identify or provide differential treatment to emergency user. The DSCP code points for traffic segregation can be selected by network administrator and same can be used to to color packets of different service types within a forwarding class.

Why not RED?

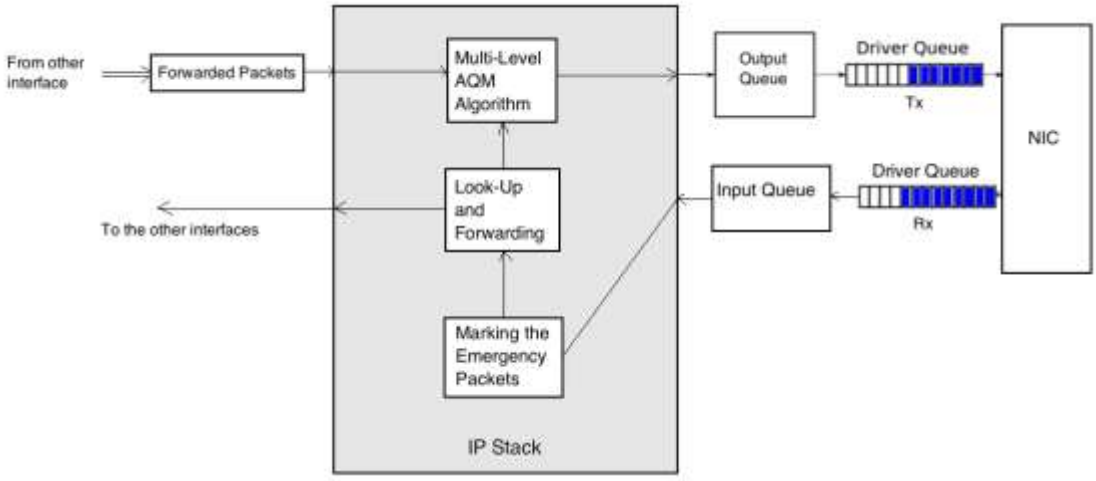
- RED is classless queuing discipline and hence it does not offer differentiated services.
- Once the queue average is greater than the min_th, it starts to probabilistically drop the packets without preference to particular type of service.
- RED does not exempt any packet from dropping after min threshold.

Why not WRED or RIO?

- WRED or RIO can be used for providing differentiated service to separate sub class of traffic like AF11, AF12, AF13. However gives the same drop precedence to non-emergency users.
- Still RIO or WRED does not exempt a packet from dropping. They are probabilistically dropped with different precedence within the sub classes.
- It does not treat emergency traffic differently for probabilistic dropping.

2.2. Linux Networking Stack

Packet flow in the Linux networking stack is shown in figure below



2.3. Traffic Control Utility (tc)

A traffic control application (tc) of iproute2 package [8] provides a mechanism to communicate with the linux kernel. The iproute2 package implements various utilities to configure the linux networking stack with the available tools for traffic handling. The kernel sub modules cannot be manipulated from user mode and hence the netlink socket is required to communicate with the linux kernel.

The tc program performs following task

1. Parse user input
   - Validate the input parameters to configure the traffic control module like add/change/remove, qdisc type, min threshold, max threshold, queue limit etc.
2. Send TCA message using netlink socket
   - Uses RTM_NEWQDISC, RTM_DELQDISC, RTM_GETQDISC message type of rtnetlink socket to add/modify, delete or show queuing discipline in linux kernel.

tc can be used to configure ingress queue and egress queue on linux networking interface.

3. System Design

Environment:

- NETLABS pods with 6 linux machines for end to end Demo.
- Ubuntu 14.04 (Linux 3.13) for kernel module development, testing and demo.
- Github for source code management. (Due to availability of netlabs pod, we maintain a local branch on netlabs pod and sync with repository at required checkpoints)
- gdb and printk for debugging.
- iperf application
- iproute2 package [8] for tc. (dependency on following packages of compilation)
    - libdb5.3-dev
    - flex
    - bison

The project contains 3 stages of implementation

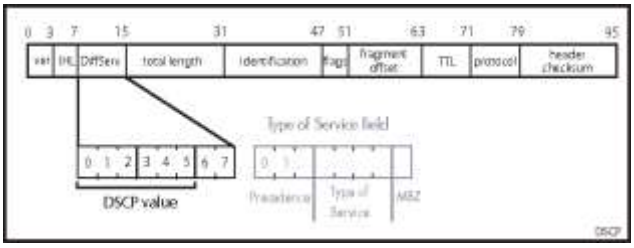
- Generate the traffic by user which contains multiple AF class packets with emergency marking on them
- Adding a new queuing discipline to TC (ERED) which identifies the emergency packets when it is being forwarded and preferentially prevents them from drop and queue to the egress queue.
- Collection of statistics for performance evaluation of ERED.

Each of them is explained below

3.1. Traffic Generator and Packet Identification

Traffic Generator:

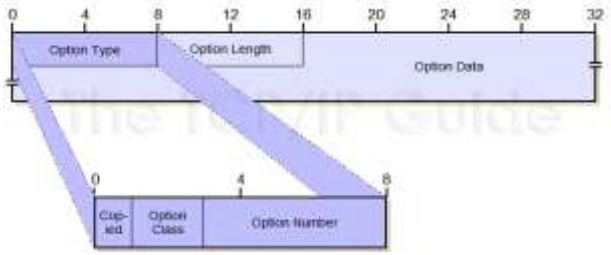
An application program is made using socket APIs in C to generate traffic for testing and demonstration. To provide differentiated services and have separate subclasses, the application will use DSCP code points. The location of the DSCP code point in IP header is given below



The TOS field will be set with values of AF11, AF12 and AF13 to differentiate three different traffic types.

New Options Emergency:

Emergency packets are marked with the help of options field in IP header.Option field in IP header structure is shown below.

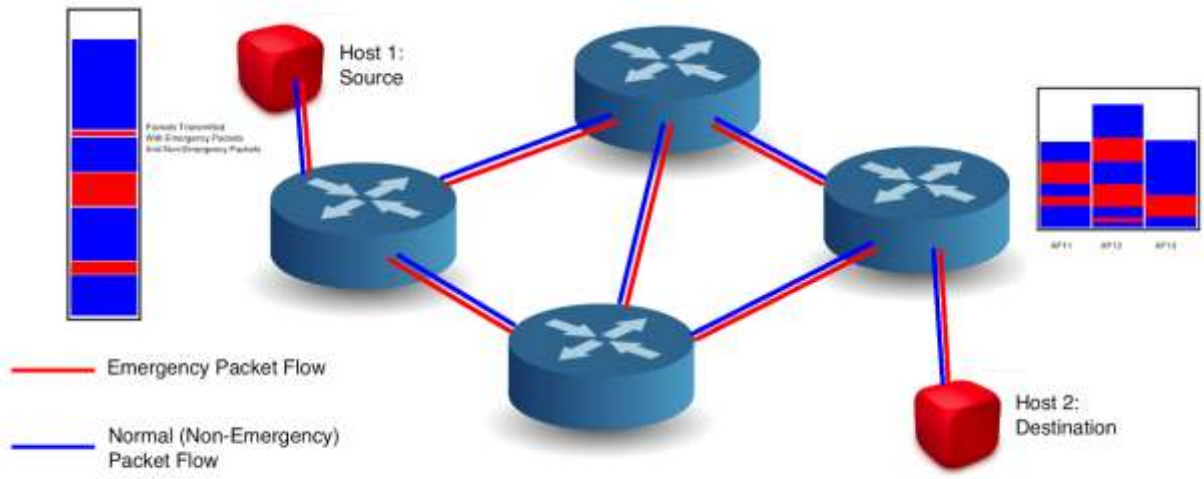


IP Options Field

- Ideally, the option field can be set to 150 to indicate it as an emergency packet.
  Copy = 1
  Option class = 0
  Option Number = 22

- But for ease of use and demonstration purposes, we will set the option field with value - 1, which can be used to identify emergency user by ered.
- As of now, Linux kernel does not parse the option field if the value is 150. So 1 is used to mark the packets as emergency for demonstration purposes.

Hence, the Option field is set to 1 to indicate the packet as emergency packet and the TOS field is set to either of AF11/AF12/AF13 to set the class of the packet.



Traffic flow containing Emergency traffic

As shown in the diagram above, the packets from the source are pumped in the network from the source. The source sends emergency as well as non-emergency packets, but all have a DSCP marking which represents its AF. At the router, the sub-queues are created by module, according to the DSCP markings for AF11, AF12, and AF13.

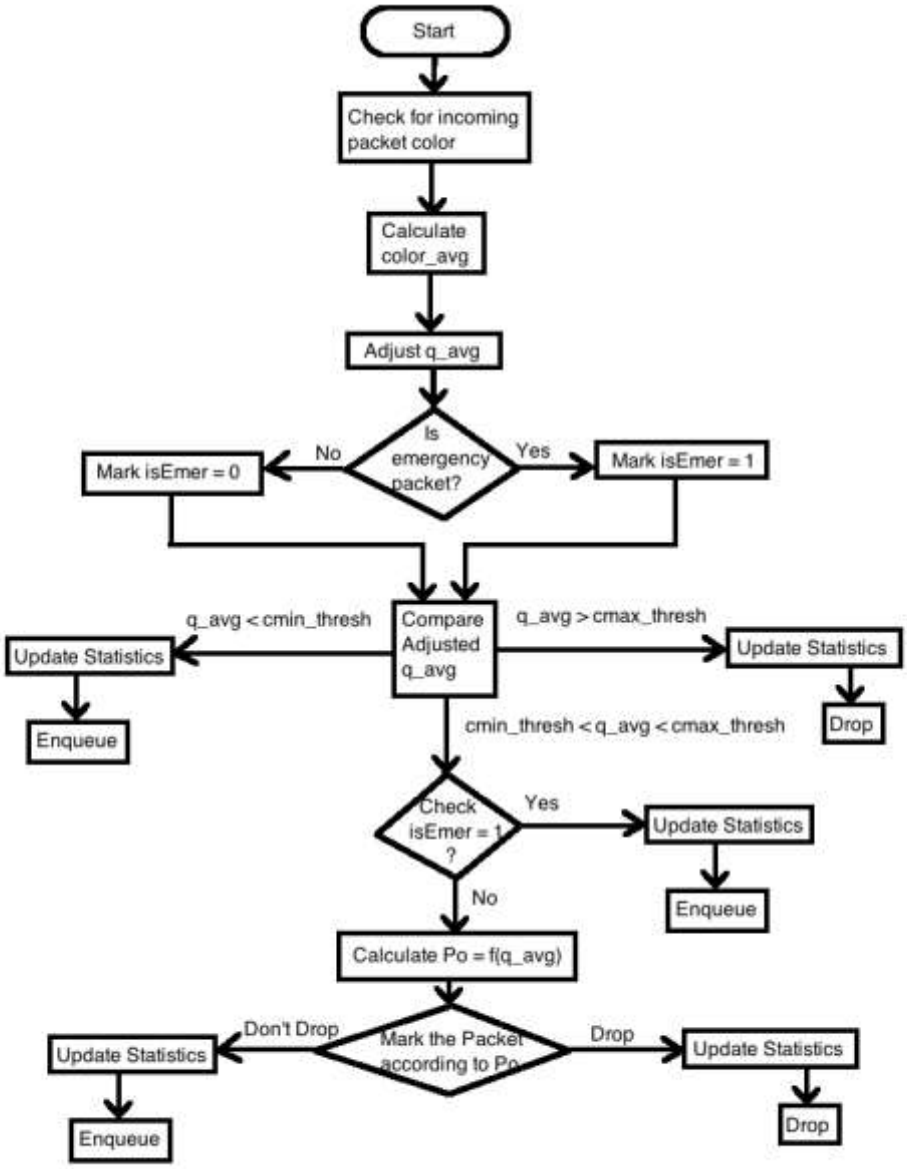
3.2. Design of Emergency RED Queuing mechanism

The new queuing discipline ERED shall support emergency users and provide preferential treatment by avoiding probabilistic drop. The following steps describe the behavior of system with ERED queuing discipline.

1. The router receives mix of emergency and non-emergency packets at the ingress queue. These packets are colored at the ingress queue based on DSCP marking. For example AF11 as red (color_id =0), AF12 as yellow (color_id = 1), AF13 as blue (color_id = 2).
2. The incoming packets will be forwarded to the outgoing interface and queue to egress queue according to ERED queue management policies. The egress queue would be configured with the new queuing discipline ERED implemented as kernel module sch_ered.ko

3.2.1. ERED queuing policy (algorithm)

The ERED queuing flowchart is shown below:



- When a packet is to be enqueued, check for the incoming packet color
- Calculate the corresponding color average **color_avg.**
    - For example, If the packet to be enqueued is red color packet, calculate the **red_avg.**
- Adjust the queue average (**q_avg**) according to color priority.
    - For example, adjust the red_average by adding it with yellow and blue colored averages (which are of lesser priorities)
- If the packet is an emergency_packet, (as identified in section 3.1), mark **isEmer**flag as 1 else mark **isEmer** flag as 0. isEmer can be flag in ered data structure.
- Compare the adjusted **q_avg**with the corresponding color threshold value.
    - If the **q_avg**is less than the corresponding minimum threshold min_th, then enqueue the packet
    - Else If the **q_avg**is greater than the corresponding maximum threshold max_th, then drop the packet

- Else If the **q_avg**is between min_th and max_th, then
    - If isEmer flag is 1, then update the statistics and enqueue the packet and exit.
    - Else IfisEmer flag is 0, then it means that it is one of the non-emergency colored packets.
        - Now calculate the probability of dropping using corresponding color_average.
        - If the probability function marks to drop the packet, then update statistics and drop it.
        - Else update statistics and enqueue the packet.

The below diagram represents how the packets are enqueued in the egress queue

- AF11 packets are marked as red color packets
- AF12 packets are marked as yellow color packets and
- AF13 packets are marked as blue color packets.

Emergency packets are depicted as orange color packets. They are still either of AF11, AF12 or AF13 packets

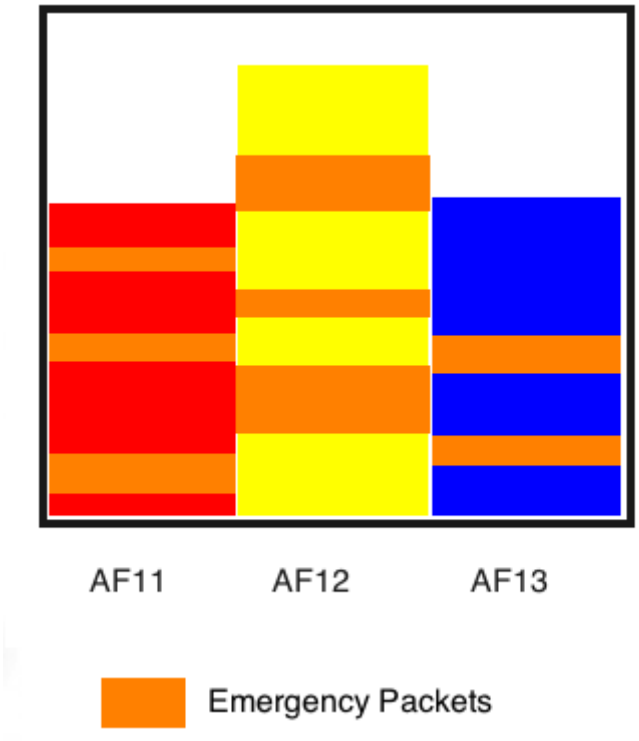


Figure 1: ered egress queue

ERED Multiple average multiple threshold calculation

Average calculation says how many packets of particular color are present in the queue on an average.

- Network administrator, while configuring the ERED qdisc, will set different thresholds for each of the colored packets using new tc application.
- ERED queueing discipline shall calculate an average for each color.

For example, when a red colored packet arrives,

red_average = red_average + (red_qLen)*W

Where W is decides the weightage of previous average samples

Probability Calculation

The drop probability is calculated based on the adjusted average.

P = max_P * (q_avg - min_th)/(max_th - min_th)

wheremin_th and max_th are the corresponding color thresholds

- If the packet is an emergency packet, then we do not calculate the probability using the above formula, but directly mark the packet as **DONT_MARK.**
- If the particular color packet has crossed the threshold value of that color, then the packets are dropped irrespective of emergency or non-emergency.

4. Detail Level Design & Implementation

4.1. Traffic Generator Application module

“etraffic” application is developed to generate emergency traffic. C socket programming function ‘setsockopt’ for updating the value of the options field for representing the emergency packets.

```
setsockopt(udp_socket, IPPROTO_IP, IP_OPTIONS, &opt_val, sizeof(opt_val));
```

For setting the DSCP marking, we use the same function i.e. setsockopt but with different parameters. For setting the DSCP markings, the opt_name is given as IP_TOS and required DSCP value obtained as command line argument.

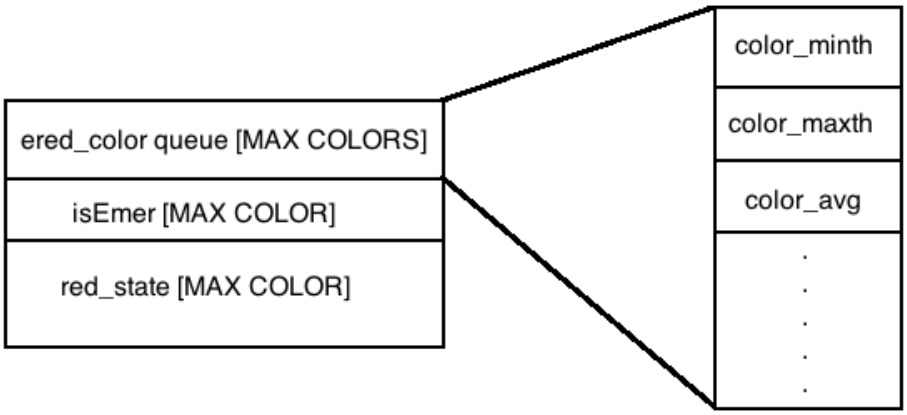
4.2. Emergency RED Implementation

4.2.1. Queuing Discipline Kernel Module Design

ered kernel module implements a new egress queuing discipline “ered” to support emergency users. The ered module is based on [12]. This module implements following APIs.

1. **Module Initialization** (**ered_module_init()**)
   - This API is called when the ered module is inserted using insmod ered.ko. This api registers the new qdisc id “ered” in the kernel data structure which contains the pointers to enqueue and dequeue APIs.
2. **Queue initialization** (**ered_init()**)
   - This API is called when network administrator configures the eredqdisc on the egress port (outgoing interface) using new tc which supports ered parsing.
   - This function instantiates the ered data structure.
3. **ERED enqueue API** (**ered_enqueue()**)
   - This API is registered as a function pointer during qdisc registration by module init function (ered_module_init).
   - When the packet is to be enqueued to the egress queue this function will be called if eredqdisc is successfully configured.
   - This API implements the ered queuing algorithm as described in Section 3.2.
4. **ERED dequeue API** (**ered_dequeue()**)
   - This API is called when the packet is scheduled out of the egress queue.
   - This API is used to update the statistics and current queue length required for average calculations.
5. **Emergency Toggle (toggleEmer() and setEmer())**
   - This API is called when cat /proc/emergencyOnOff is invoked.
6. **Print Stats API(showEmerStats())**
   - showEmerStats API is called when cat /proc/estats is invoked. It prints the various statistics as described in Section 8.2.1.

The important fields of ered data structure are shown in figure below



Code snippet:

```
struct ered_color_queue {
        u32             limit;          /* max color queue length       */
        u32             colorId;        /* color id                     */
        u32             colorBytes;     /* number of bytes of this color*/
        u32             colorPkts;      /* packets of this color   */
        u32             colorOccupancy;         /* current queue occupancy      */
        u8              prio;           /* the precedence of this color */
        /* may not be required */
        struct red_parms parms;
        struct red_vars  vars;
        struct red_stats stats;
};

struct ered_sched {
        struct ered_color_queue *colorq[MAX_COLORS]; /* red yellow blue queue */
        unsigned long   flags;
        u32             red_flags;
        u32             colors;
        u32             def; /* should be 0 */
        struct red_vars   wred_set;
        u8              isEmergency[MAX_COLORS]; /* Indicates emeregency packet */

};
```

4.2.2. Traffic Control Module Design

The iproute2 package [8] is reused and modified to enable new qdisc id “ered” in the tc parser. The recompiled tc application shall be used to configure eredqdisc in egress interface as follows:

tcqdisc add dev<eth> root ered ...

The each color (red, yellow and blue) parameters like min, max threshold etc shall be provided as arguments.

4.2.3. Statistics Module Design

To analyze the performance of ERED we use a structure in the ered that will maintain the statistics of the emergency packets. The structure includes following variables to hold statistics.

- Number of incoming packets
- Number of incoming emergency packets
- Number of normal packets that were dropped
- Number of emergency packets that were dropped

```
Struct emer_stats {
        u32etotal;
        u32edropped;
        u32emarked;
        u32emin;
        u32totalIncoming;
        u32totalOutgoing;
        … // All other statistical data collected
};
```

4.3. Ingress Packet marking module

The ingress queue shall be configured to filter the incoming packets according to the dscp marking into corresponding AF class and subclass.

4.3.1. Ingress packet marking using netfilter

The ingress queue can be created on the router using tc.

tcqdisc add dev<eth> handle ffff: ingress

(<eth> shall be the ingress interface)

Following filters mark tc_index field in sk_buf of incoming packet based on the DSCP marking. The tc_index field will be used by the enqueue function to decide the color and apply appropriate action according to ERED queuing policy.

```
tc filter add dev<eth> parent ffff: protocol ipprio 1 u32
        matchiptos 0x28 flowid: 1
tc filter add dev<eth> parent ffff: protocol ipprio 2 u32
        matchiptos 0x30 flowid: 2
tc filter add dev<eth> parent ffff: protocol ipprio 3 u32
        matchiptos 0x38 flowid: 3
```

5. Project Progress

Sr. No.	Component	Estimated Date of Completion	Progress of the Component
1.	Planning and Proposal	28th September 2014	100%
2.	Ramp up	30th September 2014	100%
2.1	Linux kernel Code familiarity/Build environment	30th September 2014	100%
2.2	AQM algorithm understanding	30th September 2014	100%
3.	Development	30th October 2014	100 %
3.1	User application for emergency packet generation (etraffic)	3th October 2014	100%
3.2	Emergency packet detection in the Rx path	8th October 2014	100%
3.3	ERED implementation	30th October 2014	100%
	Interim Report	30th October 2014	100%
3.4	Counter pegging and logging	1st November 2014	100%
4.	Unit Testing	14th November 2014	100 %
4.1	Debugging	10th November 2014	100%
4.2	Test stub coding/module testing	14th November 2014	100%
5	System Validation	28th November 2014	100%
5.1	Traffic Generator (iperf) and congestion simulation	18th November 2014	100%
5.2	Test Execution and Validation	25th November 2014	100%
5.3	Demo Setup	28th November 2014	100%
	Final Report	30th November 2014	100%

6. Self-Study

We have performed following self-study for the project execution

- Linux networking stack reverse engineering to understand the flow of function calls in packet processing path of linux.
- Understanding the netlink socket interface to communicate with kernel. Also understanding the tc application which parse the configuration inputs to tc and sends the netlink message to add/change qdisc.
- We made a test kernel module which uses printk while insert and removal to obtain the traces in dmesg. We plan to use the dmesg traces for debugging and unit testing the ered apis.

We have developed etraffic application and performed following experiments

1. **Packet stream generation**

   This self-study exercise was to verify the DSCP marking and packet stream generation feasibility on Ubuntu hosts.

   - Connected 2 host machines using cross cable in netlabs pod 2.
   - Assign IP address 192.168.1.10 to host 1
   - Assign IP address 192.168.1.1 to host 2.
   - Run etraffic application with following parameters

   ./etraffic 192.168.1.1 10000 192.168.10.1 5000 1 0x28 10

The above command runs etraffic application and sends 10 emergency packets with DSCP marking AF11 and options field set to identify emergency packet.

./etraffic 192.168.1.1 10000 192.168.10.1 5001 1 0x30 10
./etraffic 192.168.1.1 10000 192.168.10.1 5002 1 0x38 10

At host 2 capture wireshark and verify the logs.

2. **Configure ingress queue on router machine as follows.**

   This is required to mark the tc_index field of skbuff which can be used by the ered_enqueue() function to identify the color of the packet when the incoming packet will be forwarded to egress port.

   tcqdisc add dev eth0 handle ffff: ingress

   Apply netfilter to update tc_index field with corresponding color id of incoming packet sk_buff.

   tc filter add dev<eth> parent ffff: protocol ipprio 1 u32
   matchiptos 0x28 flowid: 1
   tc filter add dev<eth> parent ffff: protocol ipprio 2 u32
   matchiptos 0x30 flowid: 2
   tc filter add dev<eth> parent ffff: protocol ipprio 3 u32

matchiptos 0x38 flowid: 3

- Now run etraffic application to with DSCP marking AF11 (TOS 0x28) to verify it updates the ingress queue statistics accordingly.

tc –s show dev eth0

3. **Data on RED egress queue**

We plan to take following statistics to demonstrate the dropping of emergency packets in absence of ered.

Change the egress qdisc to RED which is already provided in linux distribution.

tcqdisc add dev eth1 root red …

Run UDP iperf stream at R1 rate from host 1 to host 2 via router machine. We calculate approximate arrival rate using M/M/1 queue model and adjust it with experimentation such that average number of packets in the queue is greater than RED min threshold.

$L = \lambda / (\mu - \lambda) >$ min threshold

μ is the service rate.

λ is the arrival rate.

Collect the egress queue statistics using tc

tc –s show dev eth1

Also collect the wireshark logs at host2 and filter the packets using options field to obtain the number of emergency packets. Compare the number of emergency packets received at host2 with number of packets sent by host1.

4. **Data on ERED egress queue**

After the unit testing phase of ERED, we plan to collect similar data with eredqdisc in egress of router machine.

- Insert the ered kernel module

  Insmodsch_ered.ko

- Run the new tc application to configure ered on egress port.

tcqdisc add dev eth1 root ered setup

Verify the egress qdisc configuration

tcqdisc show dev eth1

Run the udp stream with rate R1, R2, R3 to stabilize color average above min threshold.
Run etraffic application to send emergency packet of particular AF

Collect wireshark at host2 and egress statistics of router machine.

7. Unit Test Plan

Table 1: Unit Test

T ID	Test Name	Scenario	Expected Behavior/Verification	Conclusion
T1	Host Verification	Run etraffic application to send packet with emergency marking. Capture Wireshark/tcpdump at outgoing interface.	The packet dump shall show IP header size of 24 bytes with NOP and EOL options in options field.	The host is generating emergency packets with emergency marking.
TEST CASE PASSED: verified using wireshark logs Output: Wireshark log shows the emergency packets which can be identified by IP header length greater than 24 or looking at options field in IP header.				
T2	Emergency Packet Classification	Run test application in test bed setup as show above. Enable emergency packet counter using cat /proc/emergencyOnOff	cat /proc/estats Verify counter is incremented.	Router is classifying emergency packets
TEST CASE PASSED: verified using cat /proc/estats output. Etotal is incremented for corresponding color. Output: Emergency Red statistics color:0, total = 19454, min = 17154, pdontdrop=0, pdrop = 0, fdrop = 0 qlen=1042 qavg=0 qmin=19454|etotal = 2300, emin = 2300, emark = 0,epdontdrop=0,epdrop=0,efdrop = 0 color:2, total = 19452, min = 17152, pdontdrop=0, pdrop = 0, fdrop = 0 qlen=1042 qavg=0 qmin=19452|etotal = 2300, emin = 2300, emark = 0,epdontdrop=0,epdrop=0,efdrop = 0 color:3, total = 19452, min = 17152, pdontdrop=0, pdrop = 0, fdrop = 0 qlen=1042 qavg=0 qmin=19452|etotal = 2300, emin = 2300, emark = 0,epdontdrop=0,epdrop=0,efdrop = 0				
T3	Threshold Setting	Run the new tc command with new qdisc “ered” and setup ered with 3 sub class.	The thresholds for each of the subclass should be set.	newqdisc is enabled.
Test Case 3 : PASSED Insert module ered.ko using insmodered.ko Run config_ered.sh script to setup eredqdisc using modified tc utility Verify the qdisc configuration using dmesg logs Output : dmesg log : module inserted successfully [29776.258821] color:0,limit=60000, min=5000, max=40000 [29776.258825] color:2,limit=60000, min=5000, max=40000 [29776.258829] color:3,limit=60000, min=5000, max=40000				
T4	ERED Algorithm	Run the application on the test bed by sending normal	The packets shall be dropped according to	ERED algorithm

		and emergency traffic.	ERED algorithm	verification
Test Case 4: PASSED Results in section 8.1 Test Results and Analysis				
T5	Statistics	Run the application on the test bed to send traffic.	The statistics of dropped packets, per queue buffer occupancy etc shall be updated	Provides queue statistics for performance analysis.

Test Case 5 PASSED

Verified using output of cat /proc/estats command. The output shows the packet counters for emergency and non-emergency traffic. Please refer the Result analysis section for description of counters.

Output:

#cat /proc/estats

Emergency Red statistics

color:0, total = 117411, min = 0, pdontdrop=43484, pdrop = 71615, fdrop = 1472 qlen=39596 qavg=37355 qmin=0|etotal = 2300, emin = 0, emark = 1302,epdontdrop=840,epdrop=1302,efdrop = 158

color:2, total = 117408, min = 0, pdontdrop=8532, pdrop = 14209, fdrop = 94507 qlen=10420 qavg=42195 qmin=0|etotal = 2300, emin = 0, emark = 289,epdontdrop=160,epdrop=289,efdrop = 1851

color:3, total = 117406, min = 0, pdontdrop=1867, pdrop = 2290, fdrop = 113238 qlen=3126 qavg=42829 qmin=0|etotal = 2300, emin = 0, emark = 23,epdontdrop=11,epdrop=23,efdrop = 2266

8. Test Results and Analysis

8.1. Test Execution

The unit test plan has been executed and results are appended in Section 7 Table 1. All planned test cases are executed and successful.
This section describes the test results and analysis of the ERED queuing algorithm (expansion of test case ID T4).

Objective: To verify and analyze ERED queuing algorithm
Testbed configuration:
H1, H2 and H3 host machines with 1Gbps NIC connected to R1 with cross cable.
H4 is the receiver machine with 10Mbps NIC connected to R1 with cross cable.
All hosts are equipped with etraffic application package, wireshark, tcpdump and iperf 2.0.5 package.
R1 router machine with 4 interfaces:
Eth0 : egress port with 10Mbps NIC
Eth1, Eth2, Eth3: ingress port with 1Gbps NIC
Test scenario:
Scenario ID 1: 3 Mbps nonEmer1 + 3 Mbps nonEmer2 + 3 Mbps nonEmer3 + etraffic
Scenario ID 2: 3.2 Mbps nonEmer1 + 3.2 Mbps nonEmer2 + 3.2 Mbps nonEmer3 + etraffic
Scenario ID 3: 5 Mbps nonEmer1 + 5 Mbps nonEmer2 + 5 Mbps nonEmer3 + etraffic
Scenario ID 4: 10 Mbps nonEmer1 + 10 Mbps nonEmer2 + 10 Mbps nonEmer3 + etraffic
Scenario ID 5: 20 Mbps nonEmer1 + 20 Mbps nonEmer2 + 20 Mbps nonEmer3 + etraffic
Scenario ID 6: 30 Mbps nonEmer1 + 30 Mbps nonEmer2 + 30 Mbps nonEmer3 + etraffic

In each scenario enable/disable emergency support on R1 using "**cat /proc/emergencyOnOff**" and collect statistics using "**cat /proc/estats**"
nonEmer1: The non-emergency data generated using iperf with DSCP 0x28 (AF11) for colorid=0 (Red) from H1
nonEmer2: The non-emergency data generated using iperf with DSCP 0x30 (AF12) for colorid=2 (Yellow) from H2
nonEmer3: The non-emergency data generated using iperf with DSCP 0x38 (AF13) for colorid=3 (Blue) from H3
etraffic: Emergency traffic generated using etraffic application (./send_emergency.sh 10) from each host using designated DSCP marking.
Queue parameters and configuration on R1:

Tc configuration commands on R1 (file: **config_ered.sh**)

```
sysctl -w net.ipv4.ip_forward=1
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc add dev eth0 root ered setup DPs 4 default 1
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc change dev eth0 root ered DP 0 limit 60K min 5K max 40K avpkt 1000 burst 17 prio 1
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc change dev eth0 root ered DP 2 limit 60K min 5K max 40K avpkt 1000 burst 17 prio 1
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc change dev eth0 root ered DP 3 limit 60K min 5K max 40K avpkt 1000 burst 17 prio 1
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc add dev eth1 handle ffff: ingress
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc add dev eth2 handle ffff: ingress
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tcqdisc add dev eth3 handle ffff: ingress
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth1 parent ffff: protocol ipprio 1 u32 match iptos 0x28 0xfffflowid :0
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth1 parent ffff: protocol ipprio 1 u32 match iptos 0x30 0xfffflowid :2
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth1 parent ffff: protocol ipprio 1 u32 match iptos 0x38 0xfffflowid :3
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth2 parent ffff: protocol ipprio 1 u32 match iptos 0x28 0xff flowid :0
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth2 parent ffff: protocol ipprio 1 u32 match iptos 0x30 0xff flowid :2
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth2 parent ffff: protocol ipprio 1 u32 match iptos 0x38 0xff flowid :3
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth3 parent ffff: protocol ipprio 1 u32 match iptos 0x28 0xff flowid :0
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth3 parent ffff: protocol ipprio 1 u32 match iptos 0x30 0xff flowid :2
/home/csc573/ered_08Nov/iproute2-3.16.0/tc/tc filter add dev eth3 parent ffff: protocol ipprio 1 u32 match iptos 0x38 0xff flowid :3
```

8.2. Test Results

The following table shows the consolidated results of the ERED queuing discipline with emergency support ON and emergency support OFF. The Table 2 shows results for scenario 1, 3 and 5 which shows the operation below min threshold, between thresholds and near max threshold. Please refer section 8.2.1 for statistics interpretation.

Table 2: Consolidated Test Results

Scenario ID	INPUT RATE	EMERGENCY ON				EMERGENCY OFF			COMMON STATISTICS		
1	**Q_AVG <min_th** 3MBPS + 3MBPS + 3MBPS + 50 EMERGENCY PACKETS PER SECOND THROUGH EACH HOST	No Packets dropped				No Packets Dropped			455K TOTAL PACKETS		
3	**Min_th<Q_AVG <max_th** 5MBPS + 5MBPS + 5MBPS + 50 EMERGENCY PACKETS PER SECOND THROUGH EACH HOST	**Color ID**	**E_Mark**	**EP_Drop**	**EF_Drop**	**E_Mark**	**EP_Drop**	**EF_Drop**	**Q_AVG**	**F_Drop**	**P_Drop**
		0: AF11	700	0	0	600	600	0	11K	0	23K
		2: AF12	1250	0	0	1200	1200	0	20K	0	45K
		3: AF13	1600	0	0	1800	1800	0	28K	0	62K
5	**Q_AVG >max_th** 20MBPS + 20MBPS + 20MBPS + 50 EMERGENCY PACKETS PER SECOND THROUGH EACH HOST	**Color ID**	**E_Mark**	**EP_Drop**	**EF_Drop**	**E_Mark**	**EP_Drop**	**EF_Drop**	**Q_AVG**	**F_Drop**	**P_Drop**
		0: AF11	1800	0	90	2000	2K	90	38K	4K	277K
		2: AF12	400	0	2300	500	500	2300	41K	368K	52K
		3: AF13	8	0	3000	60	60	3000	42K	450K	60

The Table 3 shows actual test data for all the scenarios with emergency OFF in ered queuing discipline.The Table 4 shows actual test data for all the scenarios with emergency ON. Following data is further used for graphical analysis.
From the below tables, it is very clear that, whenever ERED is enabled, and when queue average is in the probabilistic region, then "pdrop" is always 0. This indicates that emergency packets are not dropped probabilistically. While ERED is disabled, the "pdrop" column highlights the number of emergency packets dropped.

Table 3: Test data for MAMT with EmergencyOFF

Without ERED																	
colorId	Total	etotal	min	emin	pdontdrop	epdontdrop	pdrop	emark	epdrop	fdrop	efdrop	qlen	Actual Qlen	qavg	drop ratio	edrop ratio	Data rate
0	19454	2300	17154	2300	0	0	0	0	0	0	0	1042	3126	0	0	0	3
2	19452	2300	17152	2300	0	0	0	0	0	0	0	1042	3126	0	0	0	3
3	19452	2300	17152	2300	0	0	0	0	0	0	0	1042	3126	0	0	0	3
0	21369	2350	17209	1727	1762	611	60	12	12	0	0	4168	10420	2933	0.00280781	0.00510638	3.2
2	21369	2350	10102	0	8508	1989	770	361	361	0	0	3126	10420	5087	0.03603351	0.15361702	3.2
3	21368	2350	798	0	16661	2147	1762	203	203	0	0	3126	10420	7153	0.08245975	0.08638298	3.2
0	31934	2400	0	0	23503	1888	6543	512	512	0	0	12504	28003	13881	0.20489134	0.21333333	5
2	31882	2350	0	0	17508	1362	13012	988	988	0	0	6252	28003	21733	0.40812998	0.42042553	5
3	31932	2400	0	0	13925	1015	16992	1385	1385	0	0	9247	28003	30985	0.53213078	0.57708333	5
0	59741	2300	0	0	31380	1236	27125	1064	1064	0	0	20840	42722	24752	0.45404329	0.4626087	10
2	59739	2300	0	0	17704	633	35007	1294	1294	6395	373	13546	42722	37332	0.69304809	0.72478261	10
3	59738	2300	0	0	4579	148	8271	294	294	46740	1858	8336	42722	40763	0.92087114	0.93565217	10
0	117411	2300	0	0	43484	840	71615	1302	1302	1472	158	39596	53142	37355	0.62248852	0.63478261	20
2	117408	2300	0	0	8532	160	14209	289	289	94507	1851	10420	53142	42195	0.92596757	0.93043478	20
3	117406	2300	0	0	1867	11	2290	23	23	113238	2266	3126	53142	42829	0.98400422	0.99521739	20
0	173502	2300	0	0	49046	526	96546	1019	1019	27384	755	40433	48769	40310	0.71428571	0.77130435	30
2	173502	2300	0	0	3343	48	4110	82	82	165935	2120	6252	48769	46049	0.98007516	0.9573913	30
3	173502	2300	0	0	1000	15	600	28	28	171868	2257	2084	48769	47980	0.99404041	0.99347826	30

Table 4: Test data for MAMT with EmergencyON

With ERED																	
colorId	Total	etotal	min	emin	pdontdrop	epdontdrop	pdrop	emark	epdrop	fdrop	efdrop	qlen	Actual Qlen	qavg	drop ratio	edrop ratio	Data rate
0	19526	2300	17226	2300	0	0	0	0	0	0	0	1042	3126	0	0	0	3
2	19525	2300	17225	2300	0	0	0	0	0	0	0	1042	3126	0	0	0	3
3	19525	2300	17225	2300	0	0	0	0	0	0	0	1042	3126	0	0	0	3
0	20870	2300	17228	2102	1321	194	21	4	0	0	0	4168	10420	2936	0.001006229	0	3.2
2	20868	2300	9733	0	8412	2004	423	296	0	0	0	3126	10420	5135	0.02027027	0	3.2
3	20869	2300	1139	0	15841	2072	1589	228	0	0	0	3126	10420	7288	0.076141646	0	3.2
0	30910	2300	0	0	22642	1801	5968	499	0	0	0	13546	28134	14123	0.193076674	0	5
2	30908	2300	0	0	16908	1160	11700	1140	0	0	0	8336	28134	23629	0.378542772	0	5
3	30908	2300	0	0	13362	942	15200	1358	0	46	0	6252	28134	30599	0.493270351	0	5
0	60634	2300	0	0	31697	1255	26637	1045	0	0	0	29176	58352	24677	0.439307979	0	10
2	60631	2300	0	0	17503	546	32693	1102	0	8787	652	20840	58352	41092	0.684138477	0.283478261	10
3	60681	2350	0	0	5034	178	8740	323	0	46406	1849	8336	58352	44185	0.908785287	0.786808511	10
0	118825	2300	0	0	43688	777	70629	1316	0	2415	207	31260	44806	33133	0.614719125	0.09	20
2	118822	2300	0	0	8655	113	14683	226	0	95145	1961	10420	44806	41285	0.924306947	0.852608696	20
3	118821	2300	0	0	2043	14	2557	23	0	114184	2263	3126	44806	43090	0.982494677	0.983913043	20
0	175572	2300	0	0	49046	431	95002	915	0	30178	954	40638	43764	39799	0.712983847	0.414782609	30
2	175566	2300	0	0	3670	14	4866	30	0	166986	2256	2084	43764	41474	0.978845562	0.980869565	30
3	175565	2300	0	0	1177	0	992	0	0	173396	2300	1042	43764	42496	0.99329593	1	30

8.2.1. Abbreviation for Test data

colored: Red, Yellow and Blue color packets
Total: Total number of non-emergency packets
Etotal: Total number of emergency packets
Min: Number of non-emergency packets below min threshold
Emin: Number of emergency packets below min threshold
Pdontdrop: Lucky non-emergency packets between min and max threshold not probabilistically dropped
Epdontdrop: Lucky emergency packets between min and max threshold not probabilistically dropped
Pdrop: Unlucky non-emergency packets between min and max threshold not probabilistically dropped
Emark: Emergency packets between min and max threshold that could be saved by ERED

Epdrop: Probabilistically dropped emergency packets between min and max
Fdrop: Force dropped non-emergency packets beyond max threshold
Efdrop: Force dropped emergency packets beyond max threshold
qlen: Number of bytes of queue filled by colored packet
Actual Qlenq: Total number of bytes filled
qavg: Effective queue average based on MAMT
drop ratio: ratio of non-emergency packets dropped to the total ingress packets
edrop ratio: ratio of emergency packets dropped to the total ingress packets
Data rate: Offered load in Mbps

8.3. Graphical analysis

Graphs have been plotted from the data extracted from our ERED algorithm in Figure 2 and Figure 3.
The three lines in each plot represent each colored packet's drop rate.
Data retrieved is from the Table 3 and 4

Graph Notation:

Red stripped line – Red color packets (AF11)
Black stripped line – Yellow color packets (AF12)
Blue stripped line – Blue color packets (AF13)

- **Dropping function of non-emergency traffic without ERED :**
  - This graph shows the non-emergency packet drop ratio when emergency OFF in ERED.
  - We can see that the drop ratio increases as the queue length / average increases and the drop ratio is dependent on the color of the packet. RED color packets have less drop ratio compared to yellow and blue.
- **Dropping function of Emergency traffic without ERED :**
  - This graph shows the emergency packet drop ratio when emergency OFF in ERED
  - We see that, the emergency packets are dropped at the same rate as non-emergency packets as seen in graph 1.
  - Probabilistic dropping of emergency packets could not be avoided if ERED is disabled which is highlighted in the plot.
- **Dropping function of non-emergency traffic with ERED :**
  - This graph is for non-emergency packet drop ratio when emergency ON in ERED.
  - With ERED enabled the drop ratio for non-emergency traffic is still a function of packet color.
- **Dropping function of Emergency traffic with ERED :**
  - This graph is plotted for emergency packets drop ratio when emergency ON in ERED.
  - We infer from the plot that, until the queue length/queue average reaches max threshold, emergency packets **drop ratio is almost 0**
  - All emergency packets are saved from being dropped probabilistically.
  - Once the queue average reaches max threshold, then the packets starts dropping according to their actual packet color's drop precedence
  - We see three different drop ratios according to their packet color after max threshold.
  - Each color packet starts dropping at different queue average.

All graphs have samples populated in the probability region [min_th < q_avg < max_th]

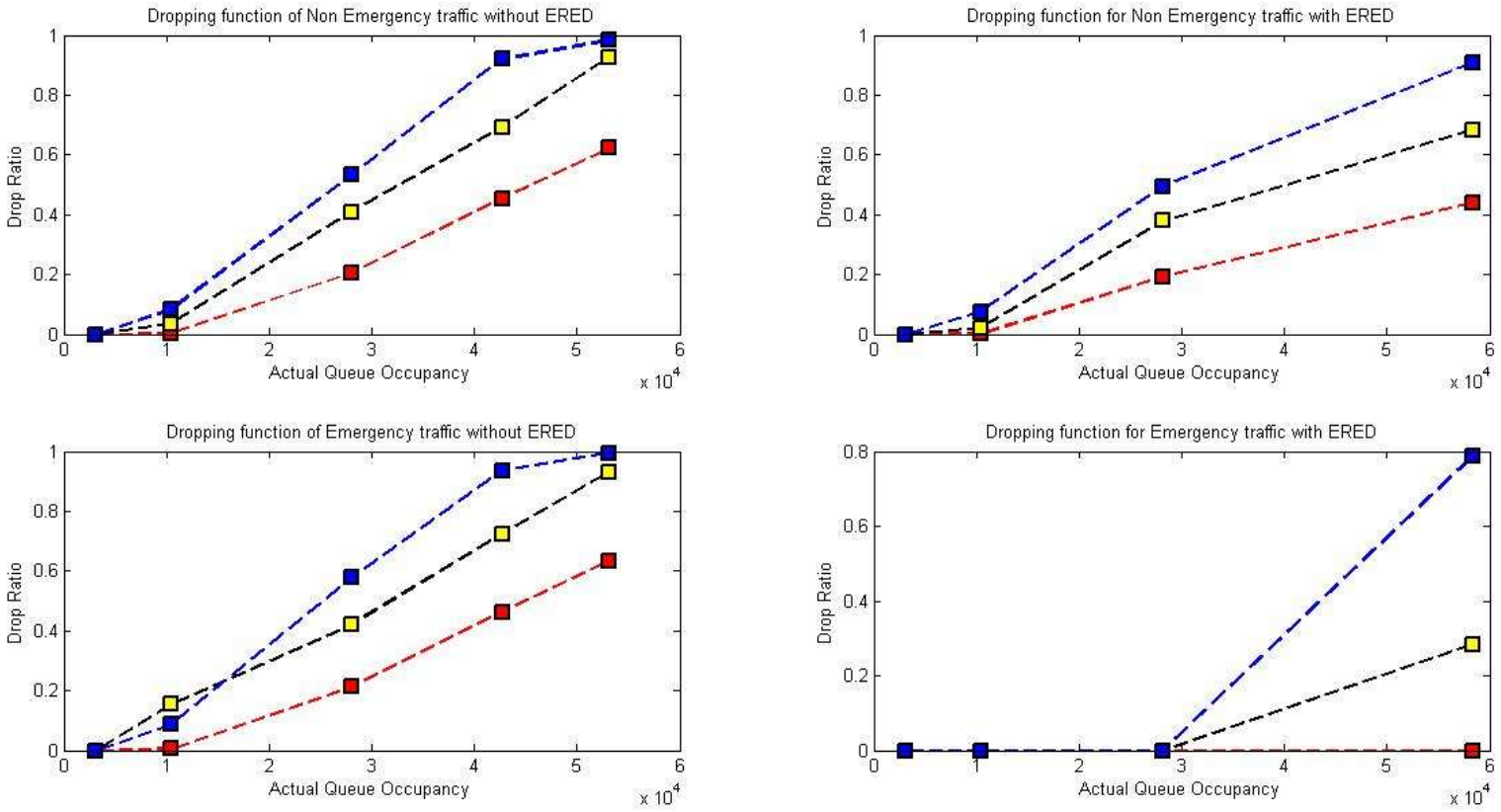


Figure 2: Plot of drop ratio vs actual queue length

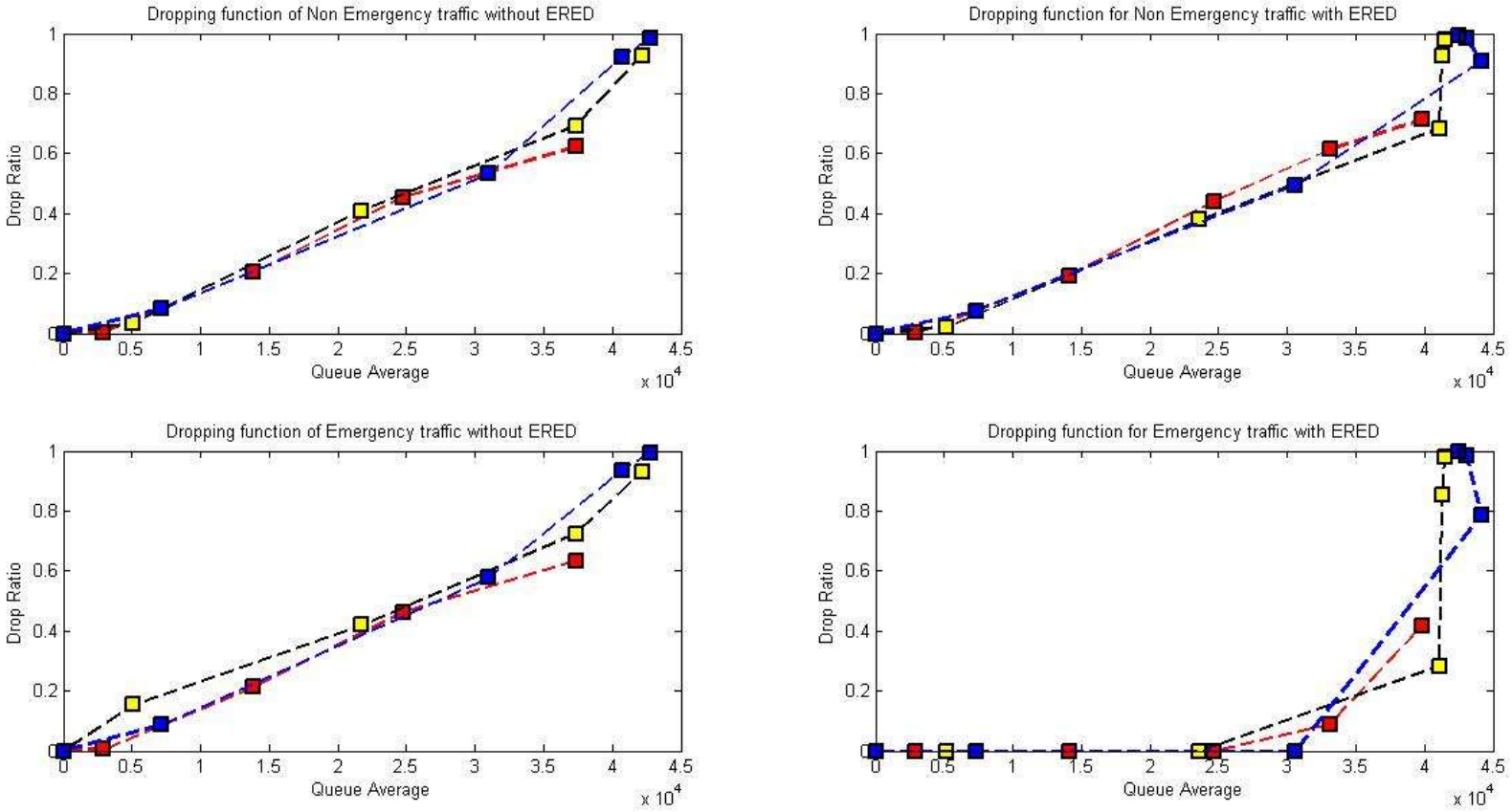


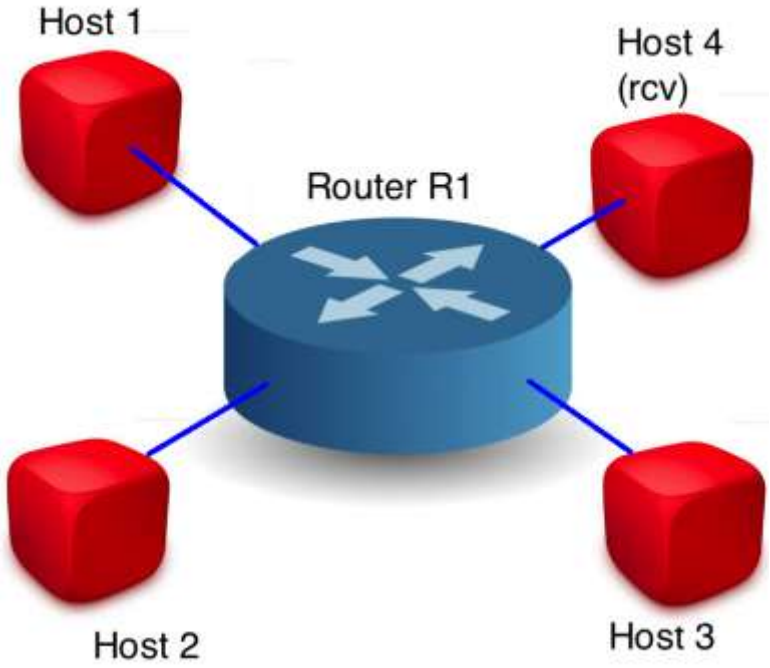
Figure 3: Plot of drop ratio vs queue average

9. Demo

The Demo shall be provided in Netlabs pod 2.

9.1. Configuration

- Three hosts H1, H2, H3 - generates emergency and non- emergency traffic
- Router R1 - Enabled with ERED queueing discipline
- H4 - Receiving host



Demo configuration

9.2. Demo scenarios

1. etraffic application demo
2. Black box ERED demo - E2E packet capture verifiability
3. White box ERED demo - Statistics of ERED at the router

- **Non-emergency traffic generation :**

- Run iperf UDP stream with rate R1 from host H1 marked as Assured forwarding class AF11 (0x28).
- Run iperf UDP stream with rate R2 from host H2 marked as Assured forwarding class AF12 (0x30).
- Run iperf UDP stream with rate R3 from host H3 marked as Assured forwarding class AF13 (0x38).
- Select the iperf traffic rate such that the buffer occupancy of ered queue on outgoing interface stabilize between min threshold and max threshold of corresponding AF. The theoretical rates can be calculated considering constant rate arrival process.

- **Emergency Traffic generation :**

- Run etraffic application from host H1 to send 100 emergency packets (~1500 bytes) marked with AF11.
- Run etraffic application from host H2 to send 100 emergency packets marked with AF12
- Run etraffic application from host H3 to send 100 emergency packets marked with AF13

1. etraffic application demo:

- Enable ERED at R1
- Check if ERED is enabled using the command **tcqdisc show**
- Once ERED is enabled at R1, perform the following tasks

ID	Scenario	Expected Behavior	Conclusion
	Generate Emergency traffic from H1, H2, H3 destined to H4 through R1 using the etraffic application		
D1	- Capture the traffic using wireshark at H1, H2, H3, H4 and R1	All packets has Option field present	Emergency packets are generated successfully using ETraffic application
D2	- Capture the traffic using wireshark at H1, H2, H3, H4 and R1	All packets has TOS field marked with value AF11 or AF12 or AF13	All packets generated are classified as AF11/AF12/AF13

2. Black Box Demo

- Enable ERED at R1.
- Transmit emergency and non-emergency traffic as defined above

ID	Scenario	Expected Behavior	Conclusion
	Enable ERED using **tcqdisc add** command and transmit packets from H1, H2, H3 as defined above.		
D1	At H4, check for the sequence of packets received from H1, H2 and H3	All emergency packets are received at H4 while some non-emergency packets are not received.	ERED has successfully exempted emergency packets from getting dropped
	Disable ERED queuing discipline and transmit emergency packets from H1, H2, H3 as defined above		
D2	At H4, check for the sequence of packets received from H1, H2 and	Some emergency packets and some non-emergency are	Emergency packets are dropped with the same

	H3	not received at H4	probability as non-emergency packets

3. White Box Demo

- Enable ERED at R1.
- Transmit emergency and non-emergency traffic as defined above

ID	Scenario	Expected Behavior	Conclusion
D1	Once the transmission is complete, dump the statistics of ERED at R1	Number of incoming Emergency packets and number of enqueued emergency packets are same	Emergency packets are not dropped by ERED
D2	Once the transmission is complete, dump the statistics of ERED at R1	Number of incoming non-emergency packets are more than the number of enqueued non-emergency packets	Non-emergency packets have been probabilistically dropped
	Disable ERED		
D3	After disabling ERED, transmit the packets as defined above and dump the statistics	Emergency and non-Emergency packets drop counter show similar number of packet drops	Emergency packets are dropped with same probability as non-emergency packets (Emergency packets are not preferentially treated)

10. Conclusion and Future Work

Our analysis of ERED shows that emergency packets can be treated preferentially for queue contention in MAMT active queue management. The emergency packets are not probabilistically dropped. However the total drop ratio (emergency + non-emergency) is preserved and is a function of packet color. The current implementation does not provide efficient handling beyond max threshold. We plan to improve emergency packet handling beyond max threshold by efficiently utilizing the remaining buffer space.

11. References

[1] Manali Joshi, Ajay Mansata, SalilTalauliker, Cory Beard, Design and analysis of multi-level active queue management mechanisms for emergency traffic, Computer Communications, Volume 28, Issue 2, 10 February 2005, Pages 162-173, ISSN 0140-3664, http://dx.doi.org/10.1016/j.comcom.2004.07.007.

[2] "A Linux Implementation of a Differentiated Services Router." N.p., n.d. Web. <http://cds.unibe.ch/research/pub_files/BSESJS00.pdf>.

[3] “Linux Networking stack” Web. http://linux-ip.net/articles/Traffic-Control-HOWTO/

[4] "TC Man Page." LARTC - Linux Advanced Routing And Traffic Control. N.p., n.d. Web. <http://lartc.org/manpages/tc.txt>.

[5] IETF. "DSCP Configuration." RFC 4594. N.p., n.d. Web. <http://tools.ietf.org/html/rfc4594>

[6] Floyd, Sally; Jacobson, Van (August 1993). "Random Early Detection (RED) gateways for Congestion Avoidance".IEEE/ACM Transactions on Networking 1 (4): 397–413. doi:10.1109/90.251892. Retrieved 2008-03-16.

[7] [Clark88] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. Proc. of ACM SIGCOMM, August 1988, and Computer Communication Review 18(4).

[8] Alexey Kuznetsov. Linux Foundation, 19 Nov. 2009. Web. <http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2>.

[9] “setsockopt Man page”. http://linux.die.net/man/2/setsockopt

[10] 3gpp 23.828. Earthquake and Tsunami Warning System (ETWS); Requirements and solutions; Solution placeholder

[11] Carlberg, K., Brown, I., and C. Beard, "Framework for Supporting Emergency Telecommunications Service (ETS) in IP Telephony", RFC 4190, November 2005.

[12] J Hadi Salim. "Generic Random Early Detection." Linux Kernel Organization, Inc, 14 Mar. 2002. Web. <https://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/net/sched/sch_gred.c>.

12. Appendix

12.1. Project Package Description

The file final.tgz is submitted along with FinalReport.pdf.

Directory Structure:

final.tgz ---------|------etraffic.tgz
|
|------ ered_tc.tgz
|
|------FinalReport.pdf
|
|------testdata
|
|------Readme

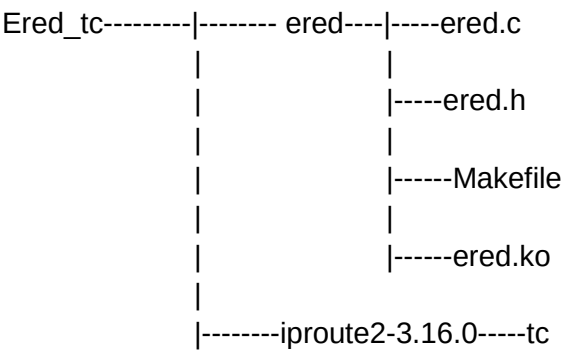
etraffic.tgz contains the code for etraffic module. etraffic module is implemented as C code for emergency traffic client (etraffic.o) and emergency traffic server (emergency_server.o).
The package also includes scripts to generate traffic by invoking etraffic.o and emergency_server.o

Extracted etraffic.tgz file will give following directory structure:

etraffic-----------|-----etraffic.c
|
|-----emergency_server.c
|
|-----Makefile
|
|-----nonEmerGen.sh
|
|-----nonEmerServ.sh
|
|-----send_emergency.sh
|
|-----etraffic.o
|
|----emergency_server.o

ered_tc.tgz contains the code for ered queuing discipline as kernel module and the iproute2 open source package with our modifications for tc to support ered in tc command utility.

Ered_tc directory structure:



ered.c contains the implementation of all the queue related functions. It also contains the statistics module functions and data structure.
ered.c code is extension of sch_red.c and adaptation of sch_gred.c
ered.h contains the implementation of emergency packet detection module and various APIs used for ered.
iproute2-3.16.0 contains the open source distribution of iproute2 package [8] with our modification for enabling parsing of ered in tc command utility.
ered.ko is generated after building source code. This kernel module enables the ered queuing discipline in linux networking stack.

Platform: Ubuntu 14.04 (Linux 3.13)

Software packages: Required for build environment

libdb5.3-dev
flex
bison