Editors:
Prof. Nikos E. Mastorakis, Military Institutes of University Education (ASEkI), HNA, GREECE
Prof. Metin Demiralp, Istanbul Technical University, TURKEY
Prof. Valeri Mladenov, Technical University of Sofia, BULGARIA
Prof. Zoran Bojkovic, Technical University of Belgrade, SERBIA

Recent Advances in Computer Engineering
A Series of Reference Books and Textbooks

NEW ASPECTS OF APPLIED INFORMATICS AND COMMUNICATIONS

Rhodes, Greece, August 20-22, 2008

Proceedings of the 8th WSEAS International Conference on
APPLIED INFORMATICS and COMMUNICATIONS (AIC'08)

ISSN: 1790-5109
ISBN: 978-960-6766-94-7

Published by WSEAS Press
www.wseas.org

NEW ASPECTS of APPLIED INFORMATICS and COMMUNICATIONS

Proceedings of the 8th WSEAS International Conference on APPLIED INFORMATICS and COMMUNICATIONS (AIC'08)

Rhodes, Greece, August 20-22, 2008

Recent Advances in Computer Engineering
A Series of Reference Books and Textbooks

Published by WSEAS Press
www.wseas.org

ISSN: 1790-5109
ISBN: 978-960-6766-94-7

NEW ASPECTS of APPLIED INFORMATICS and COMMUNICATIONS

Proceedings of the 8th WSEAS International Conference on APPLIED INFORMATICS and COMMUNICATIONS (AIC'08)

Rhodes, Greece, August 20-22, 2008

Recent Advances in Computer Engineering
A Series of Reference Books and Textbooks

Published by WSEAS Press
www.wseas.org

All papers of the present volume were peer reviewed by two independent reviewers. Acceptance was granted when both reviewers' recommendations were positive.
See also: http://www.worldses.org/review/index.html

ISSN: 1790-5109
ISBN: 978-960-6766-94-7

World Scientific and Engineering Academy and Society

NEW ASPECTS of APPLIED INFORMATICS and COMMUNICATIONS

Proceedings of the 8th WSEAS International Conference on APPLIED INFORMATICS and COMMUNICATIONS (AIC'08)

Rhodes, Greece, August 20-22, 2008

Editors:

Prof. Nikos E. Mastorakis, Military Institutes of University Education (ASEkI), HNA, GREECE
Prof. Metin Demiralp, Istanbul Technical University, TURKEY
Prof. Valeri Mladenov, Technical University of Sofia, BULGARIA
Prof. Zoran Bojkovic, Technical University of Belgrade, SERBIA

Development of Click modules: DSRED and SDRED algorithms

DMITRY KULYABOV ANNA KOROLKOVA MIGRAN GEVORKYAN
Peoples‘ Friendship University of Russia
Telecommunication Systems Department
Miklukho–Maklaya str. 6, 117198 Moscow
RUSSIA
dharma@mx.pfu.edu.ru akorolkova@sci.pfu.edu.ru mngevorkyan@gmail.com

Abstract: The objective of this work is development of two modules. These modules implement two discipline of Active Queue Management – DSRED and SDRED.

Key–Words: Click, NS-2, nsclick, RED, DSRED, SDRED.

1 Introduction

This article is a part of research work in algorithms of RED family functioning.

However in NS-2 architecture has discovered a number of fundamental constraints. These constraints don't allow to perform experiment with desired accuracy. The cause of this problem lies in a bad realisation of ISO/OSI model because three levels of ISO/OSI model are implemented in NS-2. ISO/OSI is implemented in the NS-2 through primitive entities — agents (Agent) and applications (Application) and does not reflect the actual functioning of the real network protocols.

Study of Click showed that it can support almost all traffic control algorithms of modern QoS. Click also make possible to perform simulation together with nsclick. Nsclick package allows building a simulation model using NS-2 with real Click modules.

In this work two modules have been developed. They implement two discipline of Active Queue Management (AQM) – DSRED and SDRED.

2 Problem Formulation

The goal of research is to study the behaviour of the Active Queue Management (AQM) algorithms (RED, SDRED and DSRED). Widely used simulation tool NS-2 does not explore the real characteristics of these algorithms (due to incomplete implementation of the ISO/OSI stack).

3 Problem Solution

As a solution of this problem are encouraged to use package nsclick. The primary problem is to create modules for Click which implements these algorithms. This article proposes a solution of this problem.

3.1 NS-2

NS-2 is an object-oriented software, multiplatform and open source. Its kernel is written in C++. As an interpreter scripting language OTcl (Object oriented Tool Command Language) is used, with a fairly simple syntax that allows to connect blocks of different programming languages. NS-2 fully supports the hierarchy of class C++ (named in terms of NS-2 compiled hierarchy) and a similar hierarchy of OTcl classes (called interpretability hierarchy). This approach is a compromise between speed and simplicity of usage. On the one hand, we should use a system programming language for detailed protocols simulation which provides high speed and capable enough to manipulate large volumes of data, and on the other hand, users need powerfull high level language. The interface between objects C++ and OTcl serves TclCl (Classes Tcl).

Network scripts can be fully written on OTcl, including parameters lines and hubs, for example, delays, queues, etc. In case we need to implement a specific function, such as a discipline service which is not implemented in NS-2 at the level of the kernel, we should use code in C++.

In NS-2 only 3-level model ISO / OSI are — physical, transport and applied. On physical layer only unidirectional (simplex) and bidirectional (duplex) lines between nodes can be simulated. Protocols of transport level are implemented with the assistance of OTcl-Agent (for example, Agent / TCP or Agent / UDP). Application layer protocoles are implemented with the assistance of OTcl-Application (for example, Application / Traffic / FTP). But, as noted above, the implementation of NS-2 applications protocol does not reflect their real features.

3.2 AQM algorithms

Random Early Detection (RED), an active queue management scheme, has been proposed by the Internet Engineering Task Force (IETF) to improve the throughput of TCPAP based networks in the next generation routers. RED has a number of problems such as low throughput, large delay/jitter; and inducing instability in networks. Previous enhancements to RED attempted to improve the performance of RED by modifying the parameters of RED. In this article we study two different algorithms which improve RED‘s weak sides. Double Slope Random Early Detection (DSRED) Scheme and State Dependent RED.

SDRED scheme enhance both the queue utilization and delay/jitter performances for real-time traffic. DSRED scheme improves the gateway performance in terms of normalized throughput, queuing delay, queue size, and packet drop.

3.2.1 RED

In this part we will remind the main RED algorithm principles [1]. The RED calculates the average queue size using a lowpass filter with an exponential weighted moving average. The average queue size is compared to two thresholds: a minimum and a maximum threshold. When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are, in fact, dropped or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold. When the average queue size is between the minimum and maximum thresholds, each arriving packet is marked with probability p_a, where p_a , is a function of the average queue size avg. Each time a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connections share of the bandwidth at the gateway.

The detailed algorithm for the RED [1] is given in below.

```
▷ Initialization:
$avg \leftarrow 0$
$count \leftarrow -1$
for each packet arrival
    do calculate the new average queue size $avg$:
       if the queue is nonempty
         then
              $avg \leftarrow (1 - w_q)\ avg + w_q$
         else
              $m - f(time - q_{time})$
              $avg \leftarrow (1 - w_q)^m\ avg$
       if $min_{th} \leq avg < max_{th}$
         then
              increment $count$
              ▷ calculate probability $p_a$:
              $p_b \leftarrow max_p \frac{(avg - min_{th})}{(max_{th} - min_{th})}$
              $p_a \leftarrow p_b/(1 - count\, p_b)$
              ▷ with probability $p_a$ :
              MARK THE ARRIVING PACKET
              $count \leftarrow 0$
       elseif $max_{th} < avg$
         then
              MARK THE ARRIVING PACKET
              $count \leftarrow 0$
         else $count \leftarrow -1$
when queue becomes empty
$q_{time} \leftarrow time$
```

```
▷ Saved Variables:
$avg$: average queue size
$q_{time}$: start of the queue idle time
$count$: packets since last marked packet
▷ Fixed parameters:
$w_q$: queue weight
$min_{th}$: minimum threshold for queue
$max_{th}$: maximum threshold for queue
$max_p$: maximum value for $p_b$
▷ Other:
$p_a$: current packet-marking probability
$q$ : current queue size
$time$ : current time
$f(t)$: a linear function of the time $t$
```

3.2.2 DSRED

The idea of Double Slope Random Early Detection (DSRED) [2] is that the gateway buffer segment between K_I and K_h is divided into two sub-segments separated by K_m. The overall drop function from K_i to K_h are described by two linear segments with slope α and β respectively. The slopes for these two linear segments are complementary and are adjusted by the mode selector γ. Here, the $K_m = 0.5(K_l - K_h)$, which can be configured by gateway administrator. The drop function, $p_d(avg)$, of DSRED can be expressed as [2]:

$$p_b(avg) = \begin{cases} 0, & avg < K_l, \\ \alpha(avg - K_l), & K_l \leq avg < K_m, \\ 1 - \gamma + \beta(avg - K_m), & K_m \leq avg < K_h, \\ 1, & K_h \leq avg \leq N, \end{cases} \tag{1}$$

where

$$\alpha = \frac{2(1-\gamma)}{(K_h - K_l}, \quad \beta = \frac{2\gamma}{K_h - K_l}, \tag{2}$$
$$avg = (1-w)avg + wq.$$

The detailed algorithm for the DSRED [2] is given in below.

```
 1  **for** each package arrival
 2  Calculate average queue length $avg$
 3  **if** $avg < K_l$
 4     **then**
 5          no drop
 6  **elseif** $K_l \leq avg \leq K_m$
 7     **then**
 8          Calculate drop probability based on slope $\alpha$
 9          Drop package
10  **elseif** $K_m \leq avg < K_h$
11     **then**
12          Calculate drop probability based on slope $\beta$
13          Drop packet
```

where

K_I — threshold for average queue lengthto start packet dropping at the buffer;

K_h — threshold for average queue length to start packet dropping at the buffer with a probability of 1;

K_m — threshold for average queue length to change the drop function slope;

α — drop function slope for the first linear segment between K_l and K_m;

β — drop function slope for second linear segment between K_m and K_h;

γ — mode selector for adjusting drop function slopes;

q — instantaneous gateway queue length in packet;

avg — average queue length;

w — weight parameter to calculate avg.

3.2.3 SDRED

The RED controls queue size, avoids congestion and solves the TCP global synchronization problem. However, if congestion continues to some degree, the packet discarding ratios increase and queue utilization decreases. These problems are caused by the intrinsic characteristics of the RED that use xed parameter values and have a tendency to keep the average queue size. Keeping the average queue size that is the core of the RED controlling congestion brings a late reaction.

SDRED [3] induces a faster reaction and an improved queue utilization by maintaining the average queue size (this is the major characteristic of the RED) and by modifying the RED operation when congestion begins. By revising drop-tail queue operations after the queue size exceeds the maximum threshold, the SDRED can especially reduce jitter which is a primary factor for real-time services.

Among four parameters of the RED [1], the maximum threshold and the queue weight are changeable values for the average queue size in the SDRED. In general, in cases where the queue size increases even after the queue reaches the maximum threshold, the average queue size also increases and exceeds the maximum threshold. Also, when the average queue size stays at the maximum threshold as a result of the continuity of the congestion, all packets are discarded. Different from the previous result, the SDRED is designed to increase the maximum threshold up to 90% of the total queue size. Moreover, a decreasing drop probability caused by the increased maximum threshold makes the average queue size reflect on the current queue size more by increasing the queue weight. That is, after the initial maximum threshold, the larger the average queue size is, the more the current queue size is reflected.

When a packet arrives, the SDRED calculates the average queue size like the RED [1]. However, the SDRED increases or decreases the threshold based on the current queue level at which the average queue size reaches. The characteristic of the SDRED, at this moment, is that the queue weight also increases or decreases depending on the maximum threshold, and that the average queue size is recalculated based on the updated parameters.

The detailed algorithm for the SDRED [3] is given in below.

```
▷ Package arrive
Calculate $avg_q$
if $min_{th} > avg_q$
    then
        Package pass
elseif $init\_max_{th} > avg_q$
    then
        Calculate $max_{th} = init\_max_{th}$
        Calculate $w_q = k^0 \, init\_w_q$
        Calculate $P_b$
        Discard according to $P_b$
elseif $0.7Q > avg_q$
    then
        Calculate $max_{th} = init\_max_{th} + 0.1Q$
        Calculate $w_q = k^1 \, init\_w_q$
        Calculate $P_b$
        Discard according to $P_b$
elseif $0.8Q > avg_q$
    then
        Calculate $max_{th} = init\_max_{th} + 0.2Q$
        Calculate $w_q = k^2 \, init\_w_q$
        Calculate $P_b$
        Discard according to $P_b$
elseif $0.9Q > avg_q$
    then
        Calculate $max_{th} = init\_max_{th} + 0.3Q$
        Calculate $w_q = k^3 \, init\_w_q$
        Calculate $P_b$
        Discard according to $P_b$
    else
    then
        Discard
```

The value k is a parameter used as a weighting factor to the w_q value change according to the increase of the queue occupancy ratio. If k is set to 1, it means that the SDRED operates just like the original RED. As the queue size increases, it reects more on the current queue size by multiplying the w_q with the constants k_0 , k_1 , k_2 and k_3.

3.3 Modules for Click

Click is a new software architecture for building flexible and configurable routers. There is kernel module for GNU/Linux and user-level driver for other Unix-like systems [4, 5].

A Click router is assembled from packet processing modules called elements with united configuration system. Individual elements implement simple router functions like packet classification, queuing, scheduling, and interfacing with network devices.

Click configurations are modular and easy to extend. A standards-compliant Click IP router has sixteen elements on its forwarding path.

Every action performed by a Click router's software is encapsulated in an element. The user determines what a Click router does by choosing the elements to be used and the connections among them. Inside a running router, each element is a C++ object that may maintain private state. Connections are represented as pointers to element objects, and passing a packet along a connection is implemented as a single virtual function call. A router configuration is a directed graph with elements at the vertices; packets ow along the edges of the graph. Several features make individual elements more powerful and complex configurations easier to write, including pull connections, which model packet ow driven by transmitting hardware devices, and ow-based router context, which helps an element locate other interesting elements.

There are two ways to add an element class to Click [6]:

- in the main Click collection
- in a package

Package has several advantages – for example, it will keep your code separate from the main Click code. However, if you just want to compile a single new element, it will be easier to add it to the main Click collection.

First we should write element class . Each element class should be written as two C++ source files, `FILE.cc` and `FILE.hh`.

Click programmers spend most of their time writing elements, which are subclasses of class `Element` [6]. Element provides functionality of its own, particularly the `input()` and `output()` methods and associated `Element::Port` objects. More important, however, is the set of functions that derived classes override to define element behavior.

Important, that `.cc` file exports the element class with `EXPORT_ELEMENT`. For example, the nullelement.cc file ends with: `EXPORT_ELEMENT(NullElement)`

`EXPORT_ELEMENT` takes a single argument, the name of the C++ class corresponding to ours element. It can be multiple `EXPORT_ELEMENT` lines if source file declars multiple element classes. If element is meant only for the user-level driver, add this line near `ELEMENT_REQUIRES(userlevel)` Or, if it is meant for the Linux kernel module: `ELEMENT_REQUIRES(linuxmodule)`

Second, we should put element in an `elements/` directory, `elements/local` it is recommended, which is designed for locally-created elements. We need to provide `--enable-local` argument to `configure`. Third, run 'make elemlist, which will' check the source files in the `elements/` subdirectories for `EXPORT_ELEMENT` directives, and compiles a list of elements that Click should compile. After running `make elemlist`, check the `userlevel/elements.conf` and `linuxmodule/elements.conf` files to see if `.cc` file made it into this list.

3.4 DSRED and SDRED modules based

DSRED and SDRED modules based on a standard RED module, which is included in the Click distribution. The syntax of utilization of these elements remained unchanged. Specifying the queue is optional.

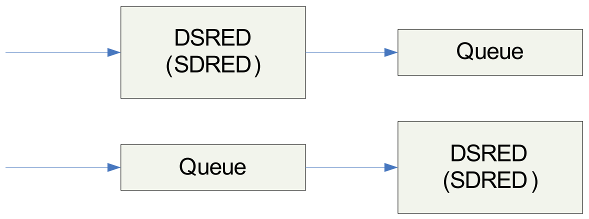

Figure 1: One queue DSRED-SDRED

Method `should_drop()` implements main AQM algorithm (RED, DSRED or SDRED). So the main difference between RED, DSRED and SDRED is in that function. Also DSRED uses other variables $K_l, K_m, K_h, \gamma, \alpha, \beta$.

To make DSRED or SDRED serve multiplie queue, we should use scheduler.

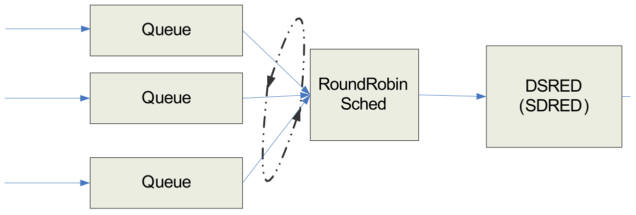

Figure 2: Multiple queues DSRED-SDRED

Listing 1 is a realisation of drop algorithm which is a part of DSRED for Click. And it is the main difference between RED and DSRED algorithms.

```
// Listing 1.
bool
DSRED::should_drop()
{
  int s = queue_size();
  if (s)
  {
    _size.update(s);
    _last_jiffies = 0;
  }
  else
  {
#if CLICK_HZ < 50
    int j = click_jiffies();
#else
    int j = click_jiffies()/(CLICK_HZ/50);
#endif
    _size.update_n(0, _last_jiffies ? j
       - _last_jiffies : 1);
    _last_jiffies = j;
  }
  unsigned avg = _size.unscaled_average();
  if (avg <= _Kh)
  {
    return false;
  }
  else if (avg > _Kh )
  {
    return true;
  }

  int p_b;
  if ((avg >=_Kl) && (avg < _Km))
  {
     p_b = (_alfa*
       (_size.scaled_average()-_Kl));
     _random_value = random();
  }
  else if ((avg >=_Km) && (avg < _Kh))
  {
     p_b = (1 - _gamma +
        _beta*(_size.scaled_average()-_Km));
     _random_value = random();
  }
  if (p_b>0 && 50/p_b>_random_value/p_b)
  {
     return true;
  }
}
```

Listing 2 is a realisation of drop algorithm which is a part of SDRED for Click. And it is the main difference between RED and SDRED algorithms.

```
// Listing 2.
unsigned avg = _size.unscaled_average();
if (_min_thresh > avg) {
  _count = -1;
#if SDRED_DEBUG
  click_chatter("%s: no drop",
     declaration().c_str());
#endif
    return false;
} else if (0.9*Q > avg) {
        _count = -1;
#if SDRED_DEBUG
  click_chatter("%s: drop,
    over max_thresh",
    declaration().c_str());
#endif
        return true;
    }
    if(_max_thresh > avg) {
    _max_thresh = _max_thresh;
    b = true;
    }
    else if(0.7*Q > avg) {
    _max_thresh = _max_thresh + 0.1*Q;
    b = true;
    }
    else if(0.8*Q > avg) {
    _max_thresh = _max_thresh + 0.2*Q;
    b = true;
    }
    else if(0.9*Q > avg) {
    _max_thresh = _max_thresh + 0.3*Q;
    b = true;
    }

    set_C1_and_C2();
int p_b;
if (b)
    p_b = ((_C1*_size.scaled_average())>>
            QUEUE_SCALE)-_C2;
else
    p_b =((_G1*_size.scaled_average())>>
            QUEUE_SCALE)-_G2;

_count++;
if (_count>0&&p_b>0&&_count>
       _random_value/p_b){
#if SDRED_DEBUG
    click_chatter("%s: drop,
       random drop (%d, %d, %d, %d)",
       declaration().c_str(), _count, p_b,
  _random_value, _random_value/p_b);
#endif
    _count = 0;
    _random_value=(random()>>5)&0xFFFF;
    return true;
}

if (_count == 0)
    _random_value=(random()>>5)&0xFFFF;

#if SDRED_DEBUG
    click_chatter("%s: no drop",
      declaration().c_str());
#endif
return false;
}
```

```
inline void
SDRED::handle_drop(Packet *p)
{
  if (noutputs() == 1)
      p->kill();
  else
      output(1).push(p);
  _drops++;
}

void
SDRED::push(int, Packet *p)
{
  if (should_drop())
      handle_drop(p);
  else
      output(0).push(p);
}

Packet *
SDRED::pull(int)
{
  while (true) {
    Packet *p = input(0).pull();
      if (!p)
        return 0;
      else if (!should_drop())
        return p;
      handle_drop(p);
  }
}
```

4 Conclusion

As a first phase of the task, two module for Click were written which implement SDRED DSRED algorithms. A method of Click modules creation was also described.

References:

[1] S. Floyd, V. Jacobsonl, Random Early Detection Gateways for Congestion Avoidance, 2003.

[2] B. Zheng and M. Atiquzzaman, DSRED: An Active Queue Management Scheme for Next Generation Networks // IEEE.. — 2000.

[3] I. Ryoo, A State Dependent RED: An Enhanced Active Queue Management Scheme for Real-Time Internet Services

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti: The Click Modular Router // ACM SIGOPS Operating Systems Review, v. 33 , Issue 5 (December 1999), p. 217–231.

[5] FAQ about Click, `http://read.cs.ucla.edu/click/faq`

[6] Element Class Reference, `http://www.read.cs.ucla.edu/click/doxygen/classElement.html`