Buffer Management Improvement in Intrusion Detection/Prevention systems

Omar Claxton

0263899

05/15/2010

Abstract

When using Intrusion Detection/Prevention (IDP) Systems in computer networks, it's important that we detect malicious traffic quickly. In the ideal case, IDP systems should not cause excessive packet drops, which may cause unnecessary retransmission. IDP systems use multiple engines to prevent this issue however there is still communication lacking among them. So it's proposed that we introduce a Management Engine (ME) that will monitor the buffer levels of the engines as well as the state of each flow that was processed. The benefits of the engine is as follows: decreasing overall packet loss, stabilizing quickly with bursty traffic, and minimal interruption when updating the flow state, for the packets arriving out of order or are sent to different engines. An additional function of the ME is monitoring the arriving traffic rate so that it will notify the engines to adjust their parameters for peak overall system performance.

Introduction

Imagine water is flowing from a river to a nearby village. Assuming the villagers dispose of the water as fast as the water arrives, no issues should arise. However, if that water begins to pick up speed and the villagers cannot dispose of it as quickly, flooding can occur. Heavy flooding means everything is put on hold until water level is manageable. A solution will be to place wells in between the river and the village, customized with a heating element placed in the bottom. The heating element will help maintain water levels in the well, allowing it to be able to receive water at any time. Essentially this is similar to the works of queues and a sufficient amount of queues will affect the throughput performance in a computer network. A further improvement would be to have a sensor placed at the river dictating the pace of water heading downstream. This sensor will then communicate with the heating elements so that they can dynamically adjust to match the speed of the water. If the speed of the water is slow then we decrease the amount of heat and if

more than an increase is necessary. Applying this concept in computer networks, we would be able to adjust our engine parameters so that they can adapt to the ingress flows and become more efficient.

Intrusion Detection/Prevention systems are put into place to help policy traffic to seek malicious data and prevent it from doing serious damage. If damage is done, ideally we want to contain and mitigate it quickly. While this is our goal we aim to perform these tasks with little effect to the overall system throughput. If we find viruses, but our system throughput is cut in half, we do not have a very efficient system. Using multiple queues/engines to simultaneously process several flows helps with analyzing multiple flows quickly, however if the engines are not communicating with each other, how do we know what has been processed? When packets arrive out of order, or arrive to different engines; we may suffer a high cost of time in determining which data streams contain the malicious attack. With the use of a management entity, we avoid such issues. The management entity is responsible for flow state monitoring. Whether through passive or aggressive methods, the flow state will always be known regardless of which engine is processing the packet. However the benefits do not end here. The management entity is also responsible for threshold monitoring and incoming arrival rate. These will help decrease total packet loss as well as prevent the underutilization of queues/engines.

Formulation

In a computer network with several arriving flows each having their own rate; there are also several engines with variable rates servicing them. Data arrives periodically and during the off period, the engines will solely service the packets that previously arrived. This system must also be able to handle a burst flow that arrives occasionally. The engines will be bounded so that the next available engine will handle the packets arriving above this boundary. Once an engine has reached

its threshold value, it would no longer accept any new arriving packets until the engine is empty. However the engine can be modified to accept packets prior to it completely emptying in attempts to maintain a high throughput. We will monitor the load of the buffers over a limited time span and compare results

Several questions arise before the experiment is conducted. Should we use smaller capacity engines as opposed to a large engine with a variable threshold? Will the performance vary greatly when the burst traffic arrives? How would the system perform as we increase the amount of flows? If the service rates of the engines change, how does this affect the overall throughput of the system? I feel confident that after the analysis, these questions will be answered.

Parameters

- Packet size b bits
- λ_l arrival rate of flow l
- β_j service rate of engine j
  - For now all engines serve at the same rate
- L buffer size = c*b choose c = 50
  - Buffer size is 50 packets long or generally speaking c packets of size b bits.
- D_{im} i^{th} engine/buffer counter in the m^{th} state

- If more than one engine send to the next engine and stop sending packets to previous engine
- T_i threshold of buffer/ engine i = p%(c)
  - P is the percentage of the max buffer space
- While the amount of packets in the buffer are greater than the threshold
  - Stop receiving packets and just process packets in engine
- While the amount of packets are less than the threshold
  - It is able to receive packets and still processes packets
- If D_{im} greater than T_i increment i and send new flows to next engine

- Periodic Traffic with burst flow every 7th slot

- $\lambda_i = a_i * h$ $\bar{a}$=[a1 ,a2, a3…]
- $\beta_i = u_i * h$ $\bar{u}$=[u1,u2,u3….]

- M = 0 D1,0 = 0
- M = 1 $D_{1,1}=D_{j,M-1} + \sum \bar{a}$
- M = 2 $D_{1,2}=D_{j,M-1} + \sum \bar{a} - u_i$
- M=(n/2 ==0) $D_{1,m}=D_{j,M-1} - u_i$
- M=(n/2 !=0) $D_{1,m}=D_{j,M-1} + \sum \bar{a} - u_i$
- M=(n/2 !=0 && n/7 ==0) $D_{1,m}=D_{j,M-1} + 2*\sum \bar{a} - u_i$

Analysis

Figure 1

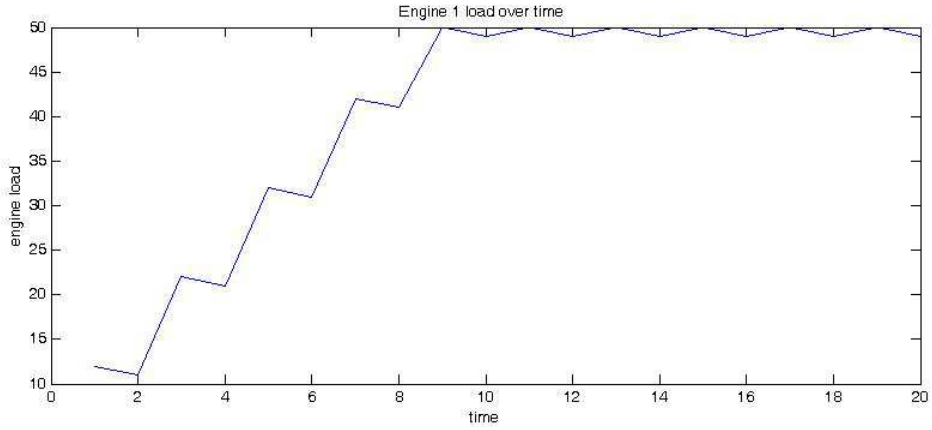

Fig 1. The engine load over time with one engine and threshold set to maximum buffer size which is 50 packets

Figure 2

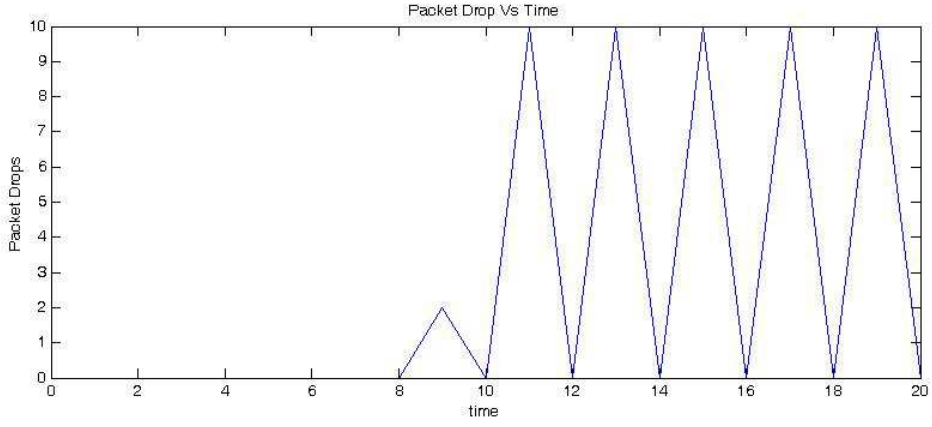

Fig 2. The packet drops experience over time when the engine becomes filled.

Figure 3

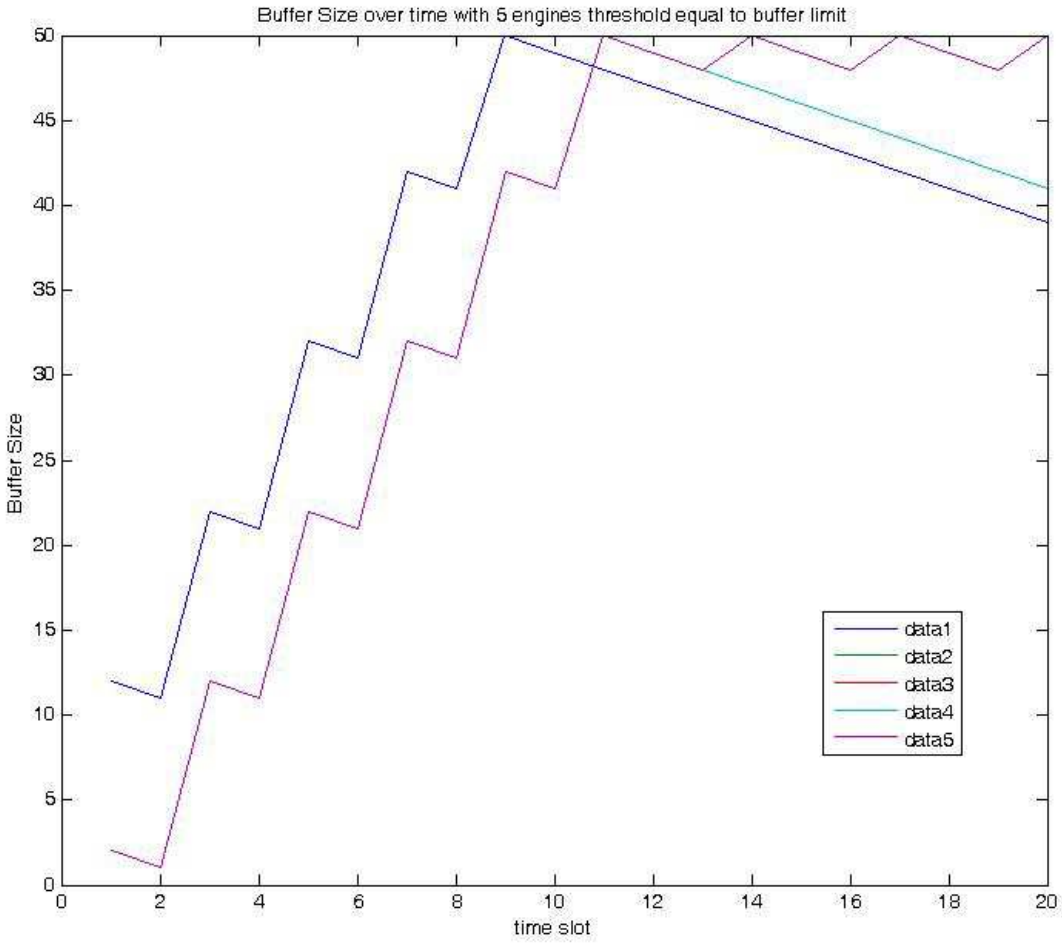

Fig 3. Five engines with the threshold set to the maximum buffer size.

Figure 4

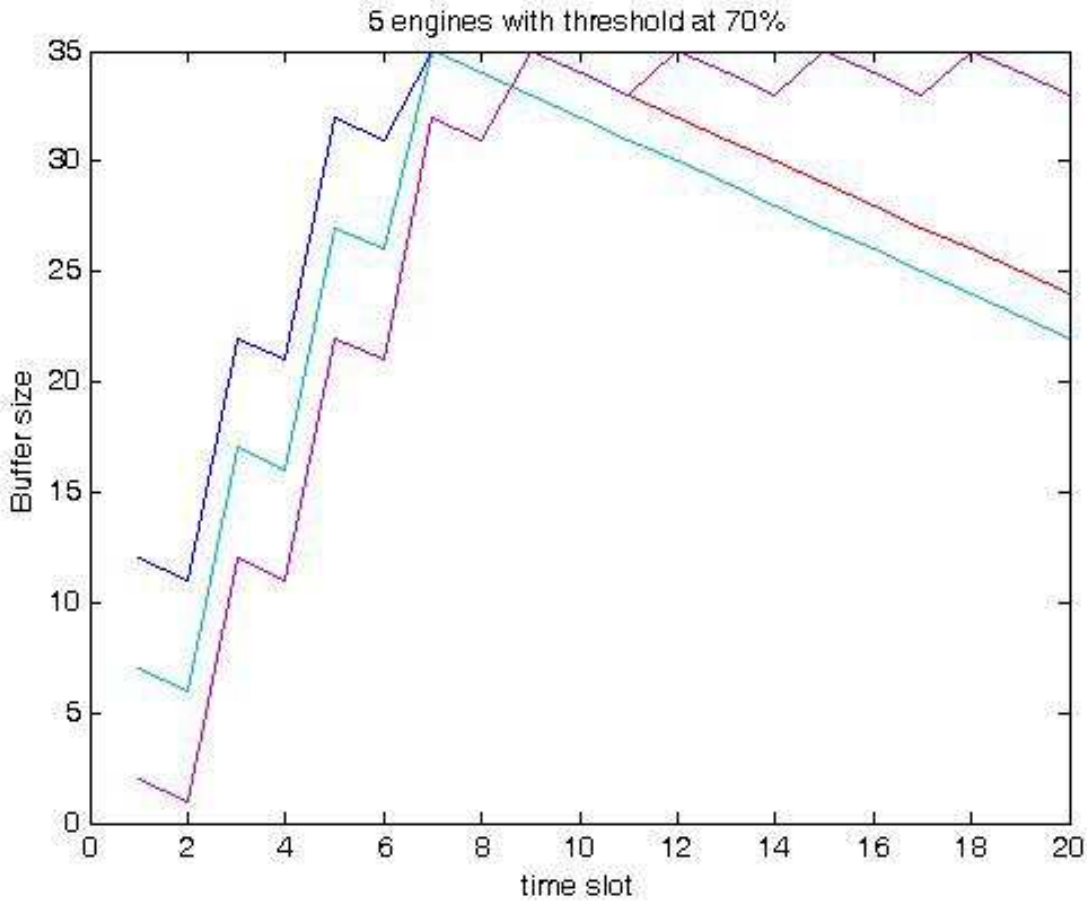

Fig 5. Five engines with threshold set to 70% of maximum buffer size

Figure 5

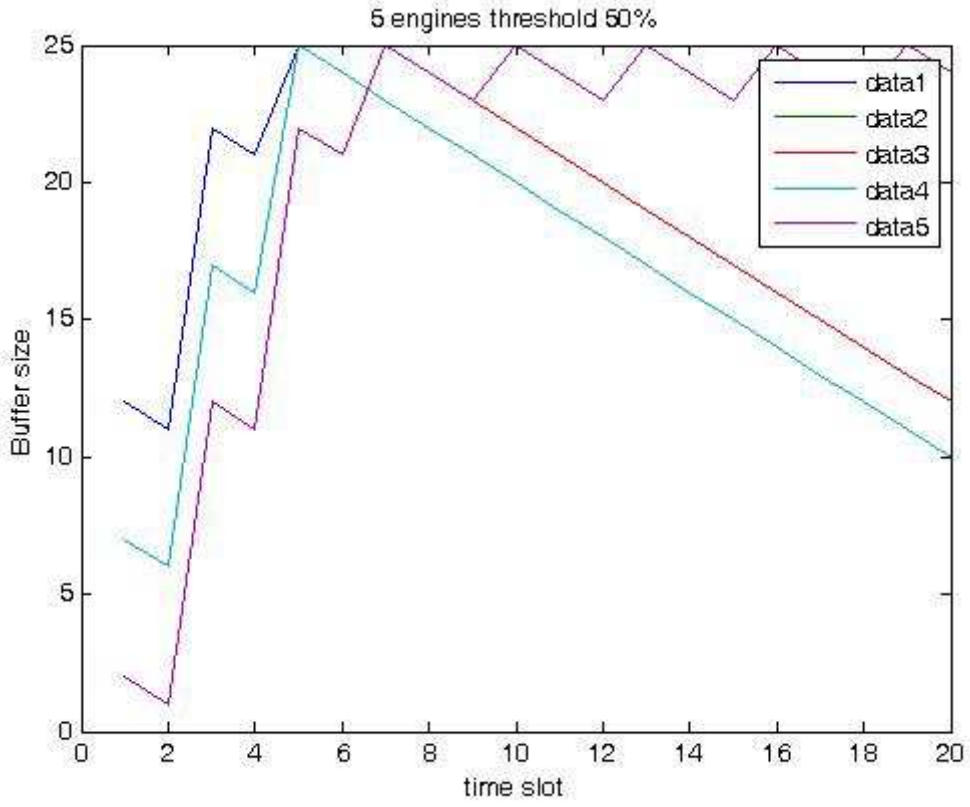

Fig 5. Five engines with threshold set to 50% of the maximum buffer size.

Figure 6

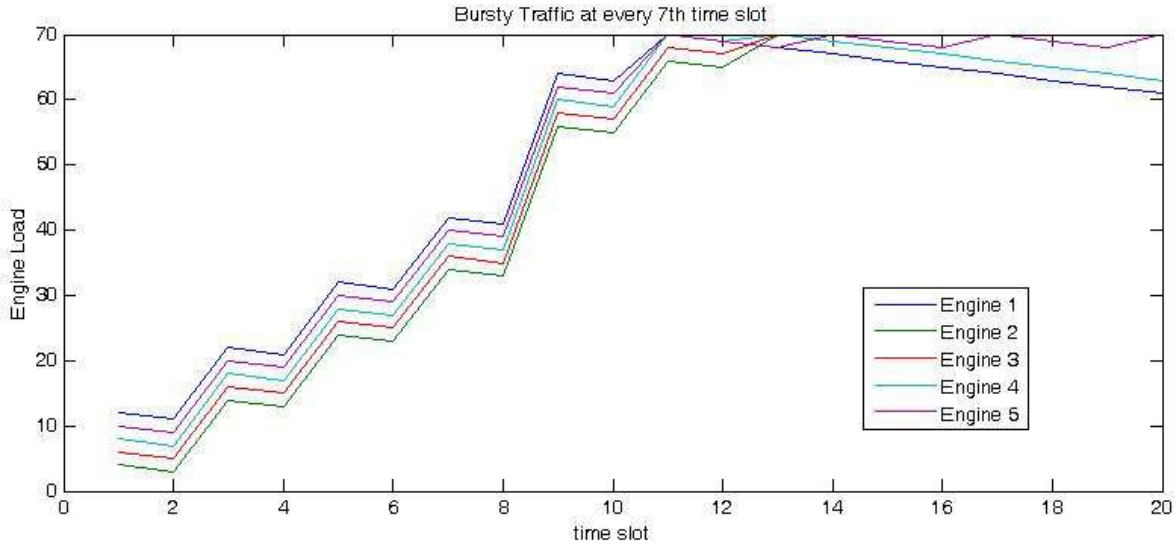

Fig 6. 5 engines with threshold set as the maximum buffer size wit the introduction of burst traffic

Figure 7

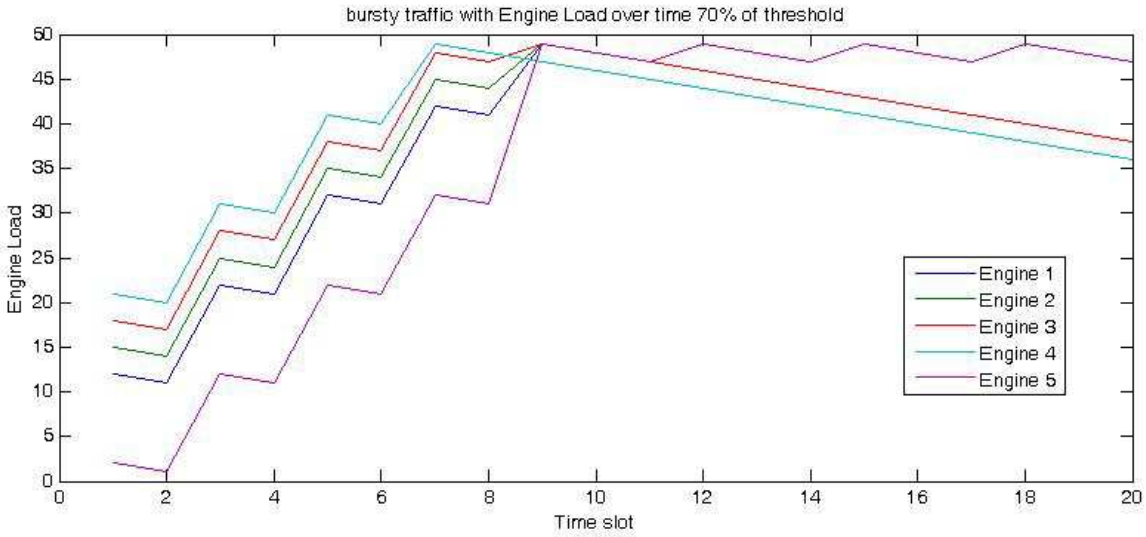

Fig 7. 5 engines with threshold set as 70 % the maximum buffer size wit the introduction of burst traffic

Discussion

Figures 1 and 2 show normal traffic conditions with only one engine. Notice the periodic curve for the packets that are dropped. The threshold for this engine is the maximum capacity of the buffer. New arriving packets will be dropped and will not be accepted until the engine's queue is fully depleted. Fortunately there was an off period during data arrival, which helped ease some congestion. If no off period were present, we would suffer a higher packet loss. Too many dropped packets means the sender is forced to reduce its sending rate, decreasing system performance. Additional engines will delay packet loss, but ultimately the last engine will experience similar results as the system with only one engine present.

Figures 3, 4 and 5 show multiple engines with threshold monitoring. As we decrease our threshold, the engines stop receiving packets earlier and newly arriving packets are sent to the next engine. While the next engine fills up, the previous engine is processing the remaining packets in its buffer. When this engine processes all remaining packets it will now be able to accept new arrivals. The faster the engine reaches its threshold; less time is needed before the engine is available again.

Figures 6 and 7 show higher capacity engines to accommodate the newly introduced burst in traffic. The addition of sudden increase in traffic makes the engine approach threshold faster. It is here we experience a trade off, we cannot accept as much packets as in the initial round; but with so many engines we suffer minimal loss.

As we increase the amount flows arriving into the system or experience a burst of traffic, our engines are able to reach the end of their boundaries faster. The benefit, although indirectly is that we start accepting new packets faster. Accepting new packets faster means packet dropping is kept to a minimum. We can further improve these engines by starting accepting packets earlier than when the queue is

fully depleted. That would lead to even fewer packet drops and converge to a high throughput quickly.

Conclusion / Future Work

There are several things to consider when choosing a threshold size. What kind of traffic will be experienced? Would we have a steady stream of arriving data? Would there be periods of time when no traffic is arriving? In my analysis I presented both periodic arrivals along well a burst arrival. However I did noticed a pattern existed. If the arriving data had no intervals of off periods, the buffers would fill up faster, however an equilibrium state will be quickly reached.

This concept through simulation has been theoretically proven, however I feel it could be taken a step further. If we use variable service rates for the engine along with adaptive threshold monitoring, system throughput should increase. Large buffer engines may not be needed using a combination of the two. I believe less buffer space is required when using a combination of both techniques. This will lead to cheaper cost without sacrificing our throughput.

Also further analysis of flow state needs to be done. When we have intermingled flows, there is no guarantee that flows will be processed together in the same engine, and if the engines do not communicate how would we know what part of a flow was processed. Also if it's possible, we need to gain access to that information quickly. If we make the ME solely responsible for this, whether through passive or aggressive updating, we will be successful at operating near or at line rate.

Appendix

```
function [D] = arrival3(c, a, u,p, j, m )
%ARRIVAL more than one engine
```

```
%   Detailed explanation goes here
%ar = a * b; % arrival rate
%sr = u * b; % service rate
L= c; %* b;% L the buffer capacity with c the amount of packets of
size b
T= p * L; % T is the threshold with p a percentage of the Buffer
capacity

D= zeros(j,m);
Pd=zeros(j,m);
norma=a;
bursta=2*a;


e=1;% initializing engine
D(1,1)= sum(a);
s = 2;% initializing state
maxit = 0; %maximum iterations
while maxit < (m-1) %go trhough a maximum iteration of m
    if mod(maxit,2)~=0 % the mths state traffic is arriving and
being processed

        if mod( maxit, 7)==0%every 7th state traffic doubles in rate
            a=bursta;
            if D(e,(s-1))< T
            D(e,s) = D(e,(s-1))+ sum(a)-u(e);
            s=s+1;
            end
        end
        if mod (maxit,7)~=0% normal traffic flow
            a=norma;
            if D(e,(s-1))< T
            D(e,s) = D(e,(s-1))+ sum(a)-u(e);
            s=s+1;
            end
        end



        if D(e,(s-1)) > T
```

```
            counter =D(e,(s-1))-T;
            D(e,s-1)=T ;
            for i= s:m
                D(e,i)= D(e,(i-1))-u(e);
            end

            if e< j
                e = e+1;
                s= 1;
                D(e,s)=counter;% preparing new engine
            end


            if s<m
                s=s+1;
                maxit=0;
            end

        end

    end

    if mod(maxit,2)== 0% the traffic is not arriving but just being
processed

        D(e,s) = D(e,(s-1))-u(e);
        s=s+1;

    end

    maxit = maxit + 1;

end

plot((1:m),D((1:e),(1:m)));
```