# An Approach to Flow Control Based on Combined Adaptive Algorithm in Communication Networks

Georgi V. Hristov[1] and Teodor B. Iliev[2]

*Abstract* – **Due to the fundamental end – to – end design principle of the TCP/IP for which the network cannot supply any explicit feedback, today TCP congestion control algorithm implements an additive increase multiplicative decrease (AIMD) algorithm. It is widely recognized that the AIMD mechanism is at the core of the stability of end – to – end congestion control. In this paper we describe a new algorithm we call Adaptive AIMD. The key concept of the adaptive behavior mechanism is to adapt to predicted network resource. We derive a mathematical model of the throughput of the MAIMD, that shows that TCP AIMD is stable, is friendly to Reno and increases the fairness in bandwidth utilization.**

*Keywords* – **AIMD, TCP, Congestion Control,**

## I. INTRODUCTION

Congestion in computer networks is becoming an important issue due to the increasing mismatch in link speeds caused by intermixing of old and new technology. Recent technological advances such as local area networks (LANs) and fiber optic LANs have resulted in a significant increase in the bandwidths of computer network links. However, these new technologies must coexist with the old low bandwidth media such as the twisted pair. This heterogeneity has resulted in a mismatch of arrival and service rates in the intermediate nodes in the network causing increased queuing and congestion [1].

Traditional congestion control schemes help improve performance after congestion has occurred. The point at which the packets start getting lost is called a cliff due to the fact that the throughput falls off rapidly after this point. We use the term knee to describe the point after which the increase in the throughput is small, but when a significant increase in the response time results.

A scheme that allows the network to operate at the knee is called a congestion avoidance scheme, as distinguished from a congestion control scheme that tries to keep the network operating in the zone to the left of the cliff. A properly designed congestion avoidance scheme will ensure that the users are encouraged to increase their traffic load as long as this does not significantly affect the response time, and arc required to decrease them if that happens. Thus, the network load oscillates around the knee [2, 3].

[1]Georgi V. Hristov is with the Department of Communication Systems and Technologies, 8 Studentska Str., 7017 Ruse, Bulgaria, E-mail: ghristov@mbox.contact.bg

[2]Assis. Prof. Ph.D Eng. Teodor B. Iliev is with the Department of Communication Systems and Technologies, 8 Studentska Str., 7017 Ruse, Bulgaria, E-mail: tiliev@ecs.ru.acad.bg

## II. CRITERIA FOR SELECTING CONTROLS

The key criteria are: efficiency, fairness, distributedness, and convergence. We define them formally as follows:

1. ***Efficiency***: The efficiency of a resource usage is defined by the closeness of the total load on the resource to its knee. If $X_{goal}$ denotes the desired load level at the knee, then the resource is operating efficiently as long as the total allocation $X(t)\sum x_i(t)$ is close to $X_{goal}$. Overload ($X(t)>X_{goal}$) or underload ($X(t)<X_{goal}$) are both undesirable and are considered inefficient. We consider both as equally undesirable.

Notice, that efficiency relates only to the total allocations and thus two different allocations can both be efficient as long as the total allocation is close to the goal. The distribution of the total allocation among individual users is measured by the fairness criterion [1].

2. ***Fairness***: The fairness criterion has been widely studied in the literature. When multiple users share multiple resources, the *maxmin* fairness criterion have been widely adopted [2,4,5]. Essentially, the set of users are partitioned into equivalent classes according to which resource is their primary bottleneck. The *maxmin* criterion then asserts that the users in the same equivalent class ought to have the equal share of the bottleneck. Thus, a system in which $x_i(t)=x_j(t)$ $\forall i, j$ sharing the same bottleneck is operating fairly. If all users do not get exactly equal allocations, the system is less fair and we need an index or a function that quantifies the fairness. One such index is [5]:

$$F(x)=\frac{\left(\sum x_i\right)}{n\left(\sum x_j^2\right)} \qquad (1)$$

This index has the following properties:

✓The fairness is bounded between 0 and 1 (or 0% and 100%). A totally fair allocation (with all $x_i$'s equal) has a fairness of 1 and a totally unfair allocation (with all resources given to only one user) has a fairness of $1/n$ which is 0 in the limit as $n$ tends to $\infty$;

✓The fairness is independent of scale, i.e., unit of measurement does not matter.

✓The fairness is a continuous function. Any slight change in allocation shows up in the fairness.

✓If only $k$ of $n$ users share the resource equally with the remaining $n$-$k$ users not receiving any resource, then the fairness is $k/n$.

3. ***Distributedness***: The next requirement that we put on the control scheme is that it be distributed. A centralized scheme requires complete knowledge of the state of the system. For example, we may want to know each individual user's demand

or their sum. This information may be available at the resource. However, conveying this information to each and every user causes considerable overhead, especially since a user may be using several resources at the same time. We are thus primarily interested in control schemes that can be implemented in real networks and, therefore, we assume that the system does the minimum amount of feedback. It only tells whether it is underloaded or overloaded via the binary feedback bits. Other information such as $X_{goal}$ and the number of users sharing the resource are assumed to be unknown by the users. This restricts the set of feasible schemes. We, therefore, describe the set of feasible schemes with and without this restriction [5].

4. ***Convergence***: Finally we require the control scheme to converge. Convergence is generally measured by the speed with which (or time taken till) the system approaches the goal state from any starting state. However, due to the binary nature of the feedback, the system does not generally converge to a single steady state. Rather, the system reaches an "equilibrium" in which it oscillates around the optimal state. The time taken to reach this "equilibrium" and the size of the oscillations jointly determine the convergence. The time determines the responsiveness, and the size of the oscillations determine the smoothness of the control. Ideally, we would like the time as well as oscillations to be small. Thus, the controls with smaller time and smaller amplitude of oscillations are called more responsive and more smooth, respectively, as shown in Fig. 1.

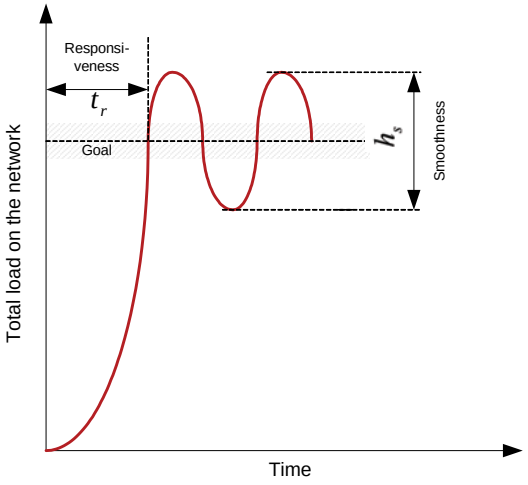


Fig.1 Responsiveness and smoothness

## III. The AIMD control Algorithm

The Additive Increase/Multiplicative Decrease (AIMD) algorithm is described in detail in [6] and is referred as "dynamic window adjustment" in [7]. The basic idea of the algorithm is to reduce the sending rate/window of the flows when the system bandwidth is exhausted and to increase the sending rates/windows when bandwidth is available. As mentioned in the previous section, when bandwidth is available (i.e. the aggregate rates of the flows do not exceed the network threshold: $\sum \omega_i < X_{goal}$ [6]) the system attaches the signal 1 to the acknowledgment of each packet. In response, flows increase by one (packet) their windows. A continuous series of positive signals will cause a linear increase in the flows' rate. Obviously, the increase is not unlimited because the bandwidth is fixed. When flows' rate exceed the bandwidth limit (i.e. $\sum \omega_i \geq X_{goal}$ ) the system attaches the 0 signal to the acknowledgment of each packet and flows respond to congestion by a decrease in their sending rates/windows.

Authors in [6] prove that a linear increase/exponential decrease policy is a condition for the increase/decrease algorithms to set (or converge) quickly the system in a fair state where the load oscillates around some equilibrium. The equilibrium state determines also the fairness and efficiency of the mechanism.

The convergence behavior of a two flow AIMD system is depicted by vectors in a 2- dimensional space oscillating around the efficiency line (or equilibrium) in Fig. 2. Upon each multiplicative decrease, the two windows $x_1$ and $x_2$ move closer to the fairness line ($x_1$=$x_2$). More details on the convergence of AIMD can be found in [6].

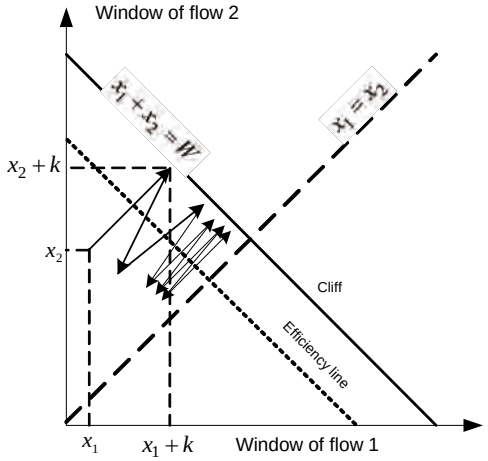


Fig.2 Vectorial representation of two-flow convergence to fairness.

Fig. 2 shows the convergence behaviour of a two-flow AIMD system. It can be seen that the vector that traces the sum of the windows is in parallel with the fairness line when the flows increase their rates, and points towards the origin of the axes when flows apply exponential decrease. Another tip that we can grasp from this figure is that the projections of the vector parallel to the fairness line on the *x* and *y* axis are equal. The practical importance of this is that during linear increase phase both flows widen their windows by the same amount.

Assume a two flow system with capacity $X_{goal}$ and let $W$=$X_{goal}$/*MSS* (MSS is the packet size), be the maximum number of packets that the system can store per step or RTT ($W$ coincides with the cliff line in the Fig.2). Let the flows $f_1$ and $f_2$ have $x_1$ and $x_2$ initial resources ($x_1$, $x_2 \in N$) respectively. Without loss of generality we assume that $x_1$<$x_2$ and $x_2$+$x_2$<$W$. Let the additive parameter be $a_I$=1 and multiplicative decrease parameter be $b_D$=1/2. A simple convergence scenario follows:

Flow $f_1$ Flow $f_2$

$x_1$ $x_2$

$x_1+1$ $x_2+1$

$x_1+1+1$ $x_2+1+1$

$x_1+1+1+\cdots+1$ ($k_1$ terms) $x_2+1+1+\cdots+1$ ($k_1$ terms)

$\sum(w+k)\geq X$ State: $x_1+x_2+2k_1\geq W$

***Action***: Multiplicative decrease

$\frac{x_1+k_1}{2}$ $\frac{x_2+k_1}{2}$

$\frac{x_1}{2}+\frac{k_1}{2}+1$ $\frac{x_2}{2}+\frac{k_1}{2}+1$

$\frac{x_1}{2}+\frac{k_1}{2}+1+\cdots+1$ ($k_2$ terms) $\frac{x_2}{2}+\frac{k_1}{2}+1+\cdots+1$ ($k_2$ terms)

$\sum(w+k)\geq X$ State: $\frac{x_1}{2}+\frac{x_2}{2}+k_1+2k_2\geq W$

***Action***: Multiplicative decrease

$\frac{x_1}{4}+\frac{k_1}{4}+\frac{k_2}{2}$ $\frac{x_2}{4}+\frac{k_1}{4}+\frac{k_2}{2}$

M M

$\frac{x_1}{2^{\lg x_2}}+\frac{k_1}{2^{\lg x_2}}+\frac{k_2}{2^{\lg\frac{x}{2}}}+\ldots+k_j$ $\frac{x_2}{2^{\lg x_2}}+\frac{k_1}{2^{\lg x_2}}+\frac{k_2}{2^{\lg\frac{x}{2}}}+\ldots+k_j$

It can be seen from this numerical example that two flows running the AIMD algorithm will converge to fairness after $1+\lg x_2$ cycles. In general, if the multiplicative decrease parameter is $b_D=1/\beta$ and $x_1$ and $x_2$ are the initial windows of two flows then these flows will converge to fairness in $\log_\beta(\max(x_1,x_2))+1$ cycles or $O(W\log_\beta W)$ steps because of the linear increase.

Based on the above example, below is presented a pseudocode for AIMD algorithm and an example of a distributed algorithm for an AIMD-based system of *m* flows. A new feature of this pseudocode is that it distinguishes the amount by which the window of the flow has widened during additive increase phase. This amount is symbolized as *k* and it can be easily noticed in Fig. 2 and in the above example. Resources consumed by the flows (i.e. congestion window) are represented by the vector/tuple ($\omega_1,\omega_2,...,\omega_m$), where the $i^{th}$ element of the vector represents the congestion window of flow *i*. Note that the AIMD-System's pseudocode is used to describe the system behaviour and is not executed by one single entity.

```
        P1(w,k,dw){
                while (feedback==1)do
                        {
                                k:=k+a;
                        }
                dw:=1/2(w+k);
                w:=dw;
                return (w,k,dw);
                }
System ((x1,x2,...,xm),m,n)
        i:=1
        if (i<m)
                { w[i]:=x[i];
                i:=i+1; }
                j:=1; i:=1;
        if (j<n)
                { if (i<m)
                {w[i]:=P1(w[i]);
                i:=i+1;}
        j:=j+1;}
return (w1,w2,...,wm)
```

In Table 1 we show the notation of AIMD algorithm.

TABLE I
AIMD ALGORITHM NOTATION.

| | |
|---|---|
| $x_i$ | Initial window of flow *i*. |
| $k$ | Resources consumed by additive increase |
| $\omega$ | The value of the window immediately after the multiplicative decrease |
| $\omega+k$ | Current window of a flow ($k\geq 0$) |
| $n$ | Integer. Represents the number of cycles towards convergence. |
| $m$ | Integer. Represents the number of flows. |
| $a_I$ | The additive increase rate ($a_I=1$) |
| $b_D$ | The multiplicative decrease ratio ($b_D=1/2$) |

Based on this observation, we can define fairness in the context of the AIMD system functionality:

A system of *m* flows $S(f_1, f_2,\ldots,f_m)$, where $f_i$ is the flow *i* and $\omega_i$ is its corresponding instantaneous throughput, converges to fairness in *n* cycles if $\omega_1,\omega_2,...,\omega_m$ become equal exactly at the $n^{th}$ cycle.

## IV. MODIFIED AIMD ALGORITHM

The AIMD mechanism has been proved in practice to be reliable (at least from congestion avoidance perspective), a little attention has been given to the dynamics of this algorithm. It is suggested that the bandwidth utilization of this algorithm be improved with buffer provisioning at the routers. Furthermore, these suggested methods do not consider the problem of how fast the link can be filled with data. The responsiveness and smoothness of this algorithm is studied in [6] and a little attention is given to each component of this mechanism.

The first component, multiplicative decrease, releases from the window those estimated shares that are not known to other flows (i.e. a sequence of multiplicative decreases will nullify those shares). The second component, additive increase, guarantees that the new resources that are used are estimated fairly.

Assume that two flows $f_1$ and $f_2$ at time *t* enter the system with windows $x_1$ and $x_2$ ($x_1<x_2$ and $x_1+x_2<W$). The flows start consuming resources (additively) from the system and at time $t+\delta t$, the system notifies the flows to release resources ($x_1+x_2+2k\geq W$). Since both flows $f_1$ and $f_2$ evolve with the same additive increase parameter, from time *t* to time $t+\delta t$ they consume exactly *k* resource units, each. When the system resources are exhausted the flows essentially release resources (i.e. multiplicative decrease) from the initial windows $x_1$ and $x_2$ which were allocated unfairly. So, our algorithm suggests to decrease multiplicatively (to half the previous size) the windows $x_1$ and $x_2$ alone. On the base of this conclusion we can modified the AIMD algorithm.

The algorithm of the Modified AIMD (MAIMD) system can be described as follows:

```
P2(w,k,dw){
        while (feedback==1)do
                {
                        k:=k+a;
                }
        dw:=1/2w+k;
        w:=dw;
        return (w,k,dw);
        }
System ((x1,x2,...,xm),m,n)
        i:=1
        if (i<m)
                { w[i]:=x[i];
                i:=i+1; }
                j:=1; i:=1;
        if (j<n)
                { if (i<m)
                {w[i]:=P2(w[i]);
                i:=i+1;}
        j:=j+1;}
return (w1,w2,...,wm)
```

The AIMD itself can not be applied when the window is equal to one byte/segment/packet. Consider a single flow system, the decrease window of this flow is $\omega$ and assume that prior to congestion $k \times a_I$ resources were allocated in additive increase. Therefore, $\omega + k \times a_I \geq W$ . The ensuing phase of multiplicative decrease will produce a reduction of resource utilization at:

$$w \leftarrow \frac{w}{2} + k \times a_I \qquad (2)$$

MAIMD can only be applied if:

$$w \geq 2a_I \qquad (3)$$

## V. Simulation Result

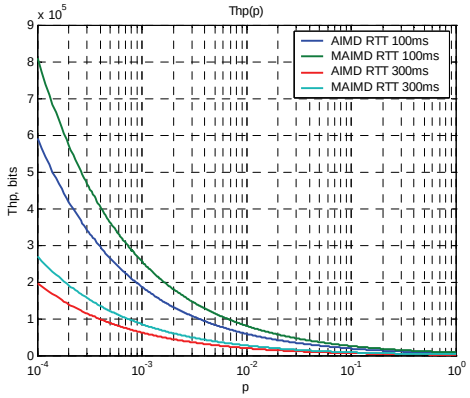


Fig.3 Performance of the TCP AIMD and TCP MAIMD

In our experiments we have used two TCP flows with AIMD and MAIMD algorithms. On Fig.3 we show the performance of the TCP MAIMD and TCP AIMD with different RTT. Fig.4 shows the investigation where we adjust TCP MAIMD parameters, and keep TCP AIMD at the default values as a reference for comparison.

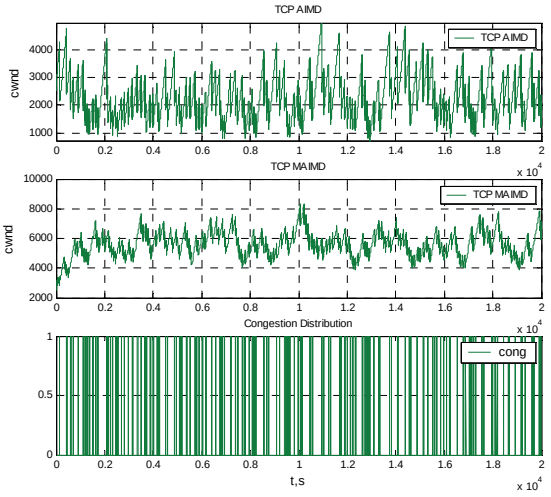


Fig.4 Congestion window evolution

## VI. Conclusion

Placing side by side the multiplicative decrease functions of AIMD and MAIMD we notice that MAIMD augments its window by a well-known factor: ½*k*. This improves its fairness and efficiency and suggests that augmenting the windows after multiplicative decrease, by a well-known increase factor, leads to enhanced efficiency and faster convergence to fairness. From the conducted simulation we can see that the efficiency of MAIMD is 8% higher than the efficiency of AIMD.

## Acknowledgement

This work is a part of the research project BY-TH-105/2005 of Bulgarian Science Fund at Ministry of Education and Science.

## References

[1] E. Gafni and D. Bertsekas, “Dynamic control of session input rates in communication networks”, IEEE Trans., Automation Control, vol.28, pp. 1090 – 1096, 1984

[2] H, Hayden, Voice Flow Control in Integrated Packet Networks, MIT, M.S. Thesis, MIT Technical Report LIDS-TH-1152, 1981.

[3] B. P. Tsankov, A. A. Aliazidi, “Connection Control in ATM Networks”, TELECOM’92, Conference Proceeding, pp. 205-210, Varna, Bulgarian, 1993

[4] J.M. Jaffe, “Bottleneck Flow Control”, IEEE Trans. on Communications, vol. 29, pp. 954-962, 1981

[5] K.K. Ramakrishnan, D.M, Chiu and R. Jain, Congestion Avoidance in Computer Networks with a Connectionless Network Layer, Part IV-A Selective Binary Feedback Scheme for General Topologies, Technical Report DEC TR-509, Digital Equipment Corporation, 1987.

[6] D. Chiu and R. Jain. “Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks”. Journal of Computer Networks and ISDN, 17(1), pp.1–14, 1989.

[7] V. Jacobson. “Congestion Avoidance and Control”, SIGCOMM’88, Conference Proceedings, pp. 314–329, 1988.