Floor the Ceil & Ceil the Floor: Revisiting AIMD Evaluation

Ashwin Rao, Arnaud Legout, Bruno Cessac, Walid Dabbous

Floor the Ceil & Ceil the Floor: Revisiting AIMD Evaluation

Ashwin Rao, Arnaud Legout, Bruno Cessac, and Walid Dabbous
INRIA, France.
Email: {ashwin.rao, arnaud.legout, bruno.cessac, walid.dabbous}@inria.fr

ABSTRACT

Additive Increase Multiplicative Decrease (AIMD) is a widely used congestion control algorithm that is known to be fair and efficient in utilizing the network resources. In this paper, we revisit the performance of the AIMD algorithm under realistic conditions by extending the seminal model of Chui *et al.* [6]. We show that under realistic conditions the fairness and efficiency of AIMD is sensitive to changes in network conditions. Surprisingly, the root cause of this sensitivity comes from the way the congestion window is rounded during a multiplicative decrease phase. For instance, the floor function is often used to round the congestion window value because either kernel implementations or protocol restrictions mandate to use integers to maintain system variables.

To solve the sensitivity issue, we provide a simple solution that is to alternatively use the floor and ceiling functions in the computation of the congestion window during a multiplicative decrease phase, when the congestion window size is an odd number. We observe that with our solution the efficiency improves and the fairness becomes one order of magnitude less sensitive to changes in network conditions.

Keywords

AIMD, Sensitivity, Fairness, Efficiency, Integer Arithmetic.

1. INTRODUCTION

The Additive Increase Multiplicative Decrease (AIMD) [17] is the most popular congestion control algorithm because of its excellent trade-off between fairness and efficiency unlike other algorithms such as additive increase additive decrease (AIAD), multiplicative increase multiplicative decrease (MIMD), and multiplicative increase multiplicative decrease algorithm (MIAD), as shown by Chiu *et al.* [6].

Despite the popularity of their model, Chiu *et al.* [6] acknowledge that it cannot be used for a realistic performance evaluation of the AIMD algorithm. Their model inherently assumes that each source's reaction to the feedback is instantaneous. They also assume homogeneous round trip times among competing flows; this makes the reaction to the feedback synchronous. Such synchronous and instantaneous reaction to feedback does not exist in real network because real networks consists of flows with different round trip times competing for the bottleneck resource. The round trip times introduces a non zero delay in the time to react to the feedback. The heterogeneity in round trip times also ensures that the reaction to the feedback is asynchronous. Another shortcoming of their model is that they assume that systems implementing AIMD can use real numbers. However, either due to protocol restrictions or due to restrictions imposed by the operating system kernels, developers may be restricted to use only integer variables while implementing the AIMD algorithm.

Additive Increase Multiplicative Decrease (AIMD) [17] is the current congestion control algorithm used in TCP [4] when losses are used as the signal of congestion. In addition, the AIMD algorithm has been proposed for ICP [5] to regulate the rate of content requests in Content Centric Networks [18]. We therefore believe that a model to capture the behavior of AIMD under realistic conditions is important and timely.

In this paper, we make the following contributions.

1. We extend the model of Chiu *et al.* [6] to improve its realism by taking into account (a) asynchronous feedback, (b) system variables maintained as integer, and (c) round trip time heterogeneity. We believe this extension can be easily applied to other algorithms such as AIAD, MIAD, and MIMD.

2. We show that the AIMD fairness is highly sensitive to changes in the capacity of the bottleneck and also in the round trip time. We show that the root cause of this sensitivity is the rounding operation performed on the congestion window size during a multiplicative decrease phase.

3. We provide a fix to the implementation of the AIMD algorithm to solve this high sensitivity that (a) improves fairness, (b) reduces the sensitivity of fairness to changes in network conditions by an order of magnitude, and (c) improves the overall

efficiency. We also show the perfect compatibility of our solution with current TCP implementation, which makes an immediate and incremental deployment of our solution possible.

The remainder of this paper is structured as follows. We begin by providing a summary of our reference model, the model of Chiu *et al.* [6], and detail its shortcomings in Section 2. In Section 3, we extend the reference model to abstract realistic conditions that exist in communication networks. We then provide numerical results that highlight issues such as unfairness of AIMD under realistic network conditions. In Section 4, we present our solution to address these issues of AIMD. We also provide supporting numerical results to quantify the benefits of using our solution. We discuss the related work in Section 5 and finally conclude in Section 6.

2. REFERENCE MODEL

In this section, we present the reference model of Chiu *et al.* [6] and show the need to extend it. We henceforth refer to this model as the *reference model*.

2.1 Model Description

Chiu *et al.* [6] consider a network of n flows that share the service of a single bottleneck of capacity X_{goal}. In a given slot of their discrete time model, each user i generates a load given by $x_i(t)$. The total load generated by the n users during the time slot t is $X(t) = \sum_{i=1}^{i=n} x_i(t)$. Congestion at the bottleneck (that is $X(t) > X_{goal}$) is signaled with a binary feedback $y(t) = 1$ that indicates users decrease their respective load. Conversely, when there is no congestion (that is $X(t) \leq X_{goal}$), $y(t) = 0$, which indicates users to increase their load.

In summary, Chiu *et al.* [6] use the following equations to represent the AIMD algorithm.

$$X(t) = \sum_{i=1}^{i=n} x_i(t), \tag{1}$$

$$y(t) = \begin{cases} 0 & \text{when } X(t) \leq X_{goal} \\ 1 & \text{when } X(t) > X_{goal} \end{cases} \tag{2}$$

$$x_i(t+1) = \begin{cases} a_I + x_i(t) & \text{when } y(t) = 0 \\ m_D x_i(t) & \text{when } y(t) = 1. \end{cases} \tag{3}$$

In Equation (3), a_I and m_D respectively represent the additive increase and multiplicative decrease components of the AIMD algorithm. In the rest of the paper, unless explicitly specified, we consider the most popular values of $a_I = 1$ and $m_D = 0.5$ [4, 9, 17].

2.2 Shortcomings of the model

The important shortcomings of this model are as follows.

1. The model assumes that each flow has the same fixed round trip time.
2. As a consequence, all users get notified of a congestion event in the same time slot, that is end-users reaction to a congestion is synchronous.
3. The load generated by each flow is assumed to be a floating point number (see Equation (3)).

However, there is a high round trip time heterogeneity in the Internet leading to asynchronous feedback loops. In addition, integer variables are used in AIMD algorithm implementations either because of protocol restriction [5] or because popular kernels, such as the Linux kernel, strongly recommend against using floating point operations in the kernel [7]. We believe considering integer variables is important because the multiplicative decrease operation performed using integers introduces rounding errors whose impact on the performance of AIMD is not known.

We would like to point out that though Linux and BSD implementations maintain the TCP congestion window as integer variables, the units used by these kernels differ. The BSD kernel maintains the congestion window as bytes while the Linux kernel maintains the congestion window as full sized segments. The IETF RFC for TCP [4] suggests keeping the congestion window in bytes, however, it acknowledges that some implementations maintain the window in units of full sized segments or packets; this RFC also acknowledges the use of integer arithmetic while performing the multiplicative decrease. We shall later see the impact of using packets instead of bytes.

In summary, in this section, we described the reference model of Chiu *et al.* [6] and highlighted its shortcomings. In the next section, we present an extension to address the shortcomings.

3. EXTENSION TO REFERENCE MODEL

In this section, we first present our extension to the reference model. We then present the metrics we use for the numerical resolution of our reference model. Finally, we analyze the numerical resolution and identify the root cause of a high sensitivity of the AIMD algorithm to network conditions.

3.1 Description of Extension

We now extend the reference model to address its shortcomings. To take into account the notion of round trip time, we assume that the load generated by the i-th source remains in the system for a *round* that lasts for a duration of d_i slots; we call d_i the *round duration* of the i-th user. For our analysis we assume that the round duration of each user is fixed[1]. During the first slot of a

[1]The model can be extended to support a round duration that varies with time. However, in this paper, we restrict ourselves to a round duration that does not change with time.

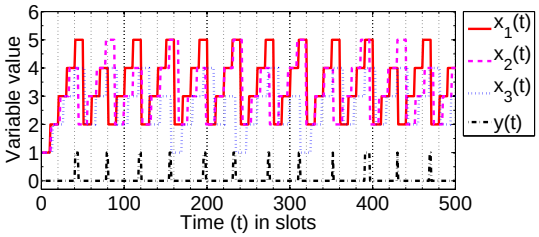


Figure 1: Numerical resolution of a system of three flows with round duration of 10, 11, and 13 slots when $X_{goal} = 12$. *We observe that the flows respond asynchronously to the feedback.*

round, the i-th user updates its load depending on the feedback received during the last round of d_i slots. The i-th user increases its load only if the system was not overloaded during the last round.

Our extension to the reference model keeps Equation (1) and Equation (2) and modifies Equation (3) as follows.

$$x_i(t+1) = \begin{cases} x_i(t) & \text{when } t \not\equiv 0 (mod\, d_i) \quad (4a) \\ a_I + x_i(t) & \text{when } t \equiv 0 (mod\, d_i) \text{ and} \\ & \sum_{j=0}^{d_i-1} y(t-j) = 0 \quad (4b) \\ max(\lfloor m_D x_i(t) \rfloor, 1) & \text{otherwise} \quad (4c) \end{cases}$$

In this extension, Equation (4a) abstracts the notion of round trip times per flow, Equation (4b) and Equation (4c) ensure that the response time to a congestion notification is a function of the round duration (d_i); thus the flows react to a congestion asynchronously. Equation (4c) restricts the minimum load to be 1, the smallest non zero integer; similar restrictions exist in TCP [4]. For example, the following code snippet shows that the TCP implementation in Linux kernel version 3.5.2, one of the latest kernel versions[2], uses the floor function and restricts the minimum congestion window to at least two TCP segments on a multiplicative decrease.

```
u32 tcp_reno_ssthresh(struct sock *sk)
{
  const struct tcp_sock *tp = tcp_sk(sk);
  return max(tp->snd_cwnd >> 1U, 2U);
}
```

We would like to point out that our extension can be used for realistic evaluation of other algorithms such as AIAD, MIAD, and MIMD, however in this paper we restrict ourselves to the popular AIMD algorithm.

We now use Figure 1 to validate that our extension indeed addresses the shortcomings presented in Section 2.2. Figure 1 shows the time evolution of the load generated by three flows that share a single bottleneck of capacity $X_{goal} = 15$. The three flows have a round duration of 10, 11, and 12 slots respectively; all the three flows start with a load of 1. We see that each source updates its load during different time slots. This shows that the reaction to a feedback is asynchronous and that it depends on the round duration. We also see that the flow 1 ($d_1 = 10$) dominates the other flows. Considering the dominance of flow 1, it is desirable to know if the flows are indeed receiving their fair share and if the system is efficient. In the next section we present the metrics we use to evaluate the fairness and efficiency.

[2]Proportional rate reduction [8] ensures that congestion window is set to the slow start threshold after loss recovery.

3.2 Metrics To Evaluate Performance of AIMD Algorithm

We now present the metrics to evaluate the system performance, and study the trade-off between fairness and efficiency of the AIMD algorithm.

1) Mean Load ($\bar{x}_i$). The mean load generated by the i-th flow is $\bar{x}_i = \frac{\sum_{t=1}^{t=T} x_i(t)}{T}$, where T represents the total time for which the flow was active. We use the mean load to compute the following metrics.

2) Load Ratio and Sensitivity of Load Ratio. We use the ratio of the load generated by two flows as an indicator of the fairness between these two flows. We compute the load ratio between flow i and flow j as follows, $L_{ij} = \frac{\bar{x}_i}{\bar{x}_j}$. By using the technique used by Mathis *et al.* [20], one can show that the desired value of the load ratio is $L_{ij} = \frac{d_j}{d_i}$.

In the following, we define the sensitivity of the load ratio to a change in the round duration and in X_{goal}. When all the system variables other than the round duration of flow k are constant, a change in the round duration d_k of flow k by Δd_k results in the load ratio between flow i and flow j to be modified by ΔL_{ij}. We define the sensitivity of the load ratio $L_{i,j}$ to the change in round duration of flow k as $|\frac{\Delta L_{ij}}{\Delta d_k}|$. Similarly, when all the system variables other than X_{goal} are constant, a change in X_{goal} by an amount ΔX_{goal} results in the load ratio between flow i and flow j to be modified by a value given by ΔL_{ij}. We define the sensitivity of the load ratio $L_{i,j}$ to the change in X_{goal} as $|\frac{\Delta L_{ij}}{\Delta X_{goal}}|$.

3) System Efficiency (ρ) and Sensitivity of System Efficiency. We define the efficiency of a system with n flows competing for a bottleneck resource with a capacity X_{goal} to be $\rho = \frac{\sum_{i=1}^{n} \bar{x}_i}{X_{goal}}$.

In the following, we define the sensitivity of the efficiency to changes in round duration and changes in X_{goal}. When all the system variables other than the

round duration of flow k are constant, then a change in the round duration d_k of flow k by an amount Δd_k results in the efficiency to be modified by a value given by $\Delta\rho$. We define the sensitivity of the system efficiency ρ to the change in round duration of flow k as $|\frac{\Delta\rho}{\Delta d_k}|$. Similarly, when all the system variables other than X_{goal} are constant, then a change in X_{goal} by an amount ΔX_{goal} results in the efficiency to be modified by a value given by $\Delta\rho$. We define the sensitivity of the system efficiency ρ to a change in X_{goal} as $|\frac{\Delta\rho}{\Delta X_{goal}}|$.

In summary, we use the load ratio L_{ij} and the efficiency ρ to evaluate the trade-off between system fairness and system efficiency. We use the load ratio and not the Jain fairness index [2,6] for evaluating the fairness because the load ratio gave us a better picture on the sensitivity of fairness. We use the sensitivity of the load ratio and the sensitivity of the efficiency to quantify the system sensitivity to changes in the network conditions. One can also extend this study of sensitivity to changes in the initial conditions of the load generated by the flows. However, based on our results we present in the following section, we believe that the parameters we use give a clear picture of the behavior of the system.

3.3 Numerical Evaluation

For the sake of clarity, in this section, we only present results for a network of two flows. We performed numerical evaluation for a network of up to four flows and our conclusions hold for all the different networks we evaluated.

The goal of our evaluation is to assess the impact of integer arithmetic on the fairness and efficiency of an AIMD algorithm. To reach this goal, we compare the numerical evaluation of our model using *integer arithmetic* with the numerical evaluation of our model using *floating point arithmetic*, that is Equation (4c) is replaced by $x_i(t) = max(m_D x_i(t), 1)$.

In the following, we start presenting the parameters used for the numerical evaluation of our model, then we discuss its fairness properties and finally its efficiency.

3.3.1 Parameters

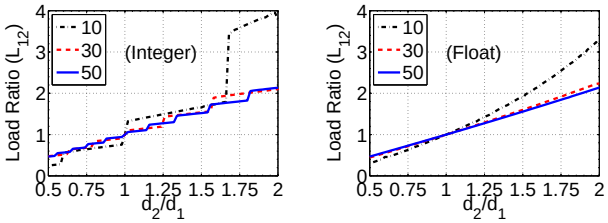
In our extension to the reference model, the round duration abstracts the round trip time in communication networks. About 70% of the round trip times observed by iPlane measurements are less than 150 ms [19]. Based on their results, we select the round duration of flow 1, d_1, in the interval $[50, 75]$ slots, and the round duration for flow 2, d_2 in the interval $[25, 150]$ slots.

The load generated by each flow abstracts the notion of congestion window. We now use the following calculations to determine the range of X_{goal} values. Akamai reports that 73% of the global links to Akamai servers have a connection speed of less than 5 Mbps [3]. The same report also mentions that 27% of connections from India to Akamai servers use a link speed of less than 256 kbps. A one way delay of 75 ms (round trip time of 150 ms) for a link speed of 5 Mbps results in a congestion window of 32 packets of 1500 bytes. Therefore, for each pair of (d_1, d_2) we performed numerical evaluations using an X_{goal} in the interval $[10, 75]$.

For our numerical evaluation, we assume that each flow begins with a load of 1. We stop the evaluation after $0.5 * 10^6$ slots. In this section, unless explicitly specified we consider $a_I = 1$ and $m_D = 0.5$.

Variable	Description
n	The number of flows.
d_i	The round duration of i-th source.
$x_i(t)$	The load generated by i-th flow.
$X(t)$	The total load generated by the n flows.
X_{goal}	The capacity of bottleneck resource.
$y(t)$	The feedback.
L_{ij}	The ratio of load generated by flow i and flow j.
ρ	The system efficiency.
a_I	Additive increase component of AIMD
m_D	Multiplicative decrease component of AIMD

Table 1: Summary of variables used in our extension to the reference model.



(a) $x_i(t)$ as an integer. (b) $x_i(t)$ as a floating point number.

Figure 2: Load Ratio for X_{goal} of 10, 30 and 50. Round duration was 50 slots for flow 1, and 50 slots to 100 slots for flow 2. *Using integer arithmetic for the load generated makes the system more sensitive to changes in the round duration.*

3.3.2 Fairness

Figure 2 shows the impact of integer and floating point arithmetic on the fairness of an AIMD algorithm under realistic network conditions. In this figure, we plot the load ratio for $d_1 = 50$, d_2 in the interval $[50, 100]$, and X_{goal} in the set $10, 30, 50$. The desired value for the load ratio L_{ij} is d_2/d_1. However, in Figure 2 (a) we observe that with integer arithmetic the load ratio increments in steps as d_2/d_1 increases. These steps indicate that one of the two flows gets more than its desired fair share. Furthermore, we observe that the curves differ

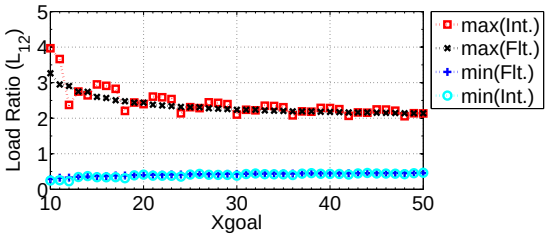


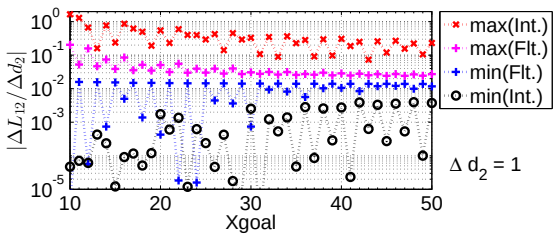
Figure 3: Maximum and minimum load ratio when $10 \leq X_{goal} \leq 50$, $50 \leq d_1 \leq 75$, and $0.5d_1 \leq d_2 \leq 2d_1$. *The maximum load ratio is significantly larger when using integer variables compared to when floating point variables are allowed.*

vastly for different values of X_{goal}. We observe that the steps become steeper, indicating a higher unfairness, when X_{goal} is smaller. The explanation is that the steps become steeper because a decrease in the input values for Equation (4c) causes the rounding errors due to the floor function in Equation (4c) to increase. In addition Figure 2 (b) shows that the fairness of the AIMD algorithm with floating point arithmetic, does not contain the steps observed in Figure 2 (a). This validates our intuition that the rounding errors introduced by Equation (4c) are responsible for the steps observed in Figure 2 (a). .

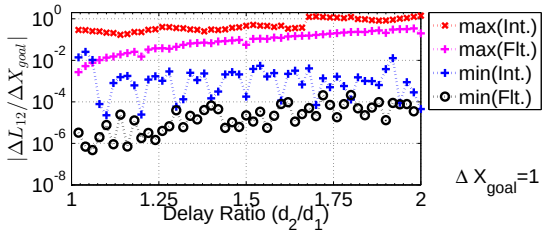
We now specifically focus on the impact of X_{goal} on the fairness of an AIMD algorithm with integer (Int.) and floating point (Flt.) arithmetic in Figure 3. In this figure, for each value of X_{goal} selected in the interval $[10, 50]$, we consider d_1 ranging from 50 to 75, and for each value of d_1, we choose d_2 ranging from $0.5d_1$ to $2d_1$. For each value of X_{goal}, the desired value of the load ratio is d_2/d_1. *We therefore expect the maximum load ratio to be 2 and the minimum load ratio to be 0.5 for all values of* X_{goal}. In Figure 3, we plot the minimum and maximum load ratio observed for each X_{goal} for all d_1 and d_2.

When $x_i(t)$ is a floating point number we observe that the maximum and minimum load ratios converge to their desired values of 2 and 0.5 respectively when X_{goal} is large. However, for smaller values of X_{goal} we observe a significant difference between the desired values and the observed values. This difference is because the amount of overload due to an additive increase of 1 ($a_I = 1$) increases when X_{goal} decreases. When $x_i(t)$ is an integer, we observe a non-monotonous behavior. This is because of the rounding errors introduced by the floor function in Equation (4c).

We quantify the sensitivity of the load ratio to variations in the round duration and X_{goal} in Figure 4 (a) and Figure 4 (b) respectively. In Figure 4 (a) we use a semilog scale to plot the maximum and minimum value of the sensitivity to the round duration d_2 for each value



(a) Sensitivity to round duration



(b) Sensitivity to X_{goal}.

Figure 4: Sensitivity of load ratio. *Sensitivity when using integers is an order of magnitude larger compared to when floating point numbers are used to maintain the load.*

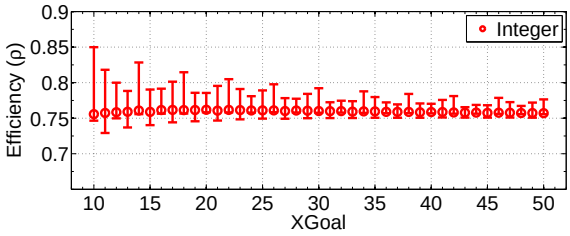
of X_{goal}; we use $\Delta d_2 = 1$ for this plot. The values of d_1, d_2, and X_{goal} were chosen from the same set used to generate Figure 3. In Figure 4 (a) we observe that the maximum sensitivity with floating point arithmetic is an order of magnitude smaller than with integer arithmetic for all the values of X_{goal} considered. We also observe that the difference between the maximum and minimum sensitivity when using floating point numbers is visibly smaller than when using integer arithmetic. Figure 4 (a) thus shows that using floating point arithmetic makes the system less sensitive to changes in round duration for a given value of X_{goal}.

In Figure 4 (b), we use a semilog scale to plot the sensitivity $|\frac{\Delta L_{12}}{\Delta X_{goal}}|$, when $d_1 = 50$, and X_{goal} was varied from 10 to 50 in steps of 1. In this figure we consider d_2 from 50 to 100 giving $1 \leq d_2/d_1 \leq 2$. As in the case of Figure 4 (a), in Figure 4 (b) we observe that using floating point arithmetic results in a smaller maximum sensitivity compared to when integer arithmetic is used. We also observe that the difference between the maximum and minimum sensitivity when using floating point numbers is visibly smaller than when using integer arithmetic.

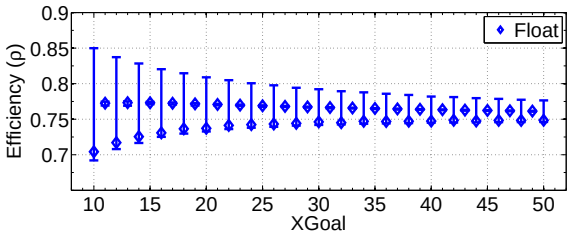
In conclusion, *a system that uses integer variables is more sensitive to changes in the round duration and* X_{goal} *compared to a system that uses floating point variables.*

3.3.3 *Efficiency*

We have seen that integer arithmetic adversely im-

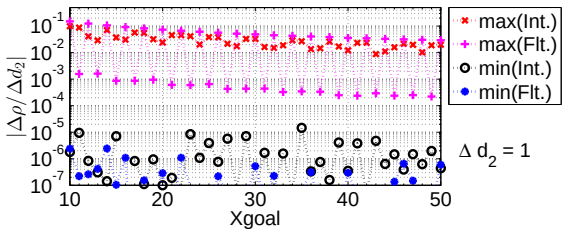


(a) System variables as integers.

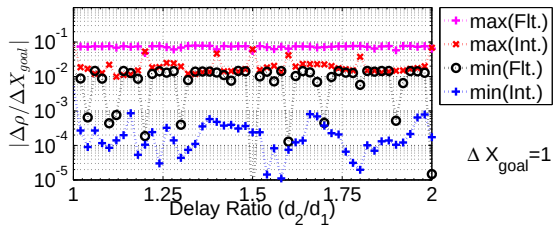


(b) System variables as floating point numbers.

Figure 5: Efficiency when $10 \leq X_{goal} \leq 50$, $50 \leq d_1 \leq 75$, and $0.5d_1 \leq d_2 \leq 2d_1$. *Using integers as system variables provides better system efficiency compared to using floating point numbers.*



(a) Sensitivity to round duration



(b) Sensitivity to X_{goal}.

Figure 6: Sensitivity of efficiency. *Sensitivity when using floating point numbers is an order of magnitude larger compared to changes in X_{goal} when integers are used to maintain the load.*

pacts the fairness of an AIMD algorithm, we will now explore its impact on efficiency. In Figure 5, we plot the median efficiency of a system for a given value of X_{goal}, the error bars indicate the minimum and maximum efficiency of the system. In Figure 5 we observe that the median efficiency when using integer arithmetic is higher than the median efficiency when using floating point arithmetic for even values of X_{goal}. When X_{goal} is odd we observe that median efficiency is close to the maximum and minimum efficiency when floating point arithmetic is used.

The explanation for these observations is as follows. When the system is overloaded, $X(t) > X_{goal}$, integer arithmetic causes the total load to exceed X_{goal} by a load of at least 1 and at most n ($1 \leq X(t) - X_{goal} \leq n$). Floating point arithmetic enables to signal an overload even when $X_t - X_{goal} < 1$; the range of values that can cause an over load is therefore $0 < X(t) - X_{goal} \leq n$ when floating point arithmetic is used. Therefore, *the flows can be more aggressive when integer arithmetic is used compared to when floating point arithmetic is used.* Furthermore, when X_{goal} is a multiple of the number of flows, floating point arithmetic ensures that $0 < X(t) - X_{goal} \leq 1$ when the round duration of at least one pair of flows is different. Thus, for even values of X_{goal} in Figure 5, we observe that the median efficiency when using integer arithmetic is higher than the median efficiency when using floating point arithmetic. The system attains maximum efficiency when the round duration of each of the flows is the same. This is because all the flows are equally aggressive in attaining their respective fair share. When X_{goal} is odd (for $n = 2$), floating point arithmetic ensures that the $0 < X_t - X_{goal} \leq 1$ for all combinations of round duration in the steady state. Therefore we observe that the median efficiency is close to the maximum and minimum efficiency.

.

We present the sensitivity of the efficiency to delay and X_{goal} variations in Figure 6. The range of input values used to plot Figure 6 are the same as the ones Figure 4. We observe in Figure 6 (a) that, when X_{goal} is odd, the maximum sensitivity is smaller with floating point arithmetic than with integer arithmetic, but the sensitivity in both cases is similar when X_{goal} is even. In Figure 6 (b) we observe a smaller difference between the maximum and minimum sensitivity when integer arithmetic is used compared to when floating point arithmetic is used. In Figure 6 we observe that *the efficiency of a system that uses integer variables is an order of magnitude less sensitive to changes in X_{goal} compared to a system that uses floating point numbers.*

3.3.4 Summary

To recap, during the numerical evaluation of our extension to the reference model we made the following observations.

1. With integer arithmetic, the load ratio (and hence fairness) is sensitive to the round duration of the competing flows and the value of X_{goal}. We observe that the fairness is an order of magnitude

more sensitive with integer than with floating point arithmetic.

2. With integer arithmetic, the efficiency of the system is higher than with floating point arithmetic. We also observe that the efficiency of the system using integers arithmetic is an order of magnitude less sensitive to changes in X_{goal} than with floating point arithmetic.

3.4 Discussion on Realistic Implementation of AIMD

The numerical resolution of our extension to the reference model portrays the classical trade-off between efficiency and fairness. On the one hand we observe that the system is more efficient when it uses integer variables while on the other hand the system suffer from poor fairness.

We also observe that the error introduced by the floor function performed during the multiplicative decrease amplifies when the round duration of competing flows are different. The amplification of the error makes the fairness of the system sensitive to changes in the round duration and X_{goal}. Furthermore, we also observe that the unfairness increases as the bottleneck resource capacity, X_{goal}, decreases.

In our extension we assume that the floor function is used when performing the multiplicative decrease. We make similar observations when we use the ceiling function instead of the floor function in Equation (4c). Similarly, in this section we present results for a system of two flows. We have performed numerical evaluations for a system of up to four flows and we make similar observations with respect to fairness and efficiency and their sensitivity to round duration and X_{goal}.

It is important to remark that the issue we observe with integer arithmetic is an implementation issue because of protocol restrictions or because of restrictions by the operating systems. There is a rounding error, when the floor operation is performed on an odd congestion window during a multiplicative decrease phase. Therefore, we qualify this issue as an implementation bug that needs to be solved. In the following section, we present our solution to this rounding error bug.

4. FIXING THE ROUNDING ERROR BUG

In this section, we first present a fix to the bug causing rounding errors on a multiplicative decrease with integer arithmetic. Then, we show that our fix solves the observed unfairness issue without sacrificing efficiency.

4.1 Description of the fix

Our fix simply consists in alternating between the floor and the ceiling functions on successive calls to the multiplicative decrease procedure when its input is an odd number. The insight for this fix is the following. The bug we observed in Section 3.3 is due to a rounding error. By alternating the floor and ceil functions, we expect to compensate for the rounding errors. In addition, in Equation (4c), the multiplication by $m_D = 0.5$ produces an error only if the input for the multiplicative decrease is an odd number; there is no error when the multiplicative decrease is performed for even numbers. We therefore alternate between the floor and ceiling functions only when a multiplicative decrease is performed when the load is an odd numbers. The implementation of this fix is simple. We maintain a state in variable a which indicates whether the floor ($a = 0$) or the ceiling function ($a = 1$) needs to be called during the multiplicative decrease. The pseudo-code for our fix is the following.

if $x(t)$ is even **then**
  $x(t+1) \leftarrow max(x(t)/2, 1)$
else
  $x(t+1) \leftarrow max(x(t)/2 + a, 1)$
  $a \leftarrow (a+1) \bmod 2$
end if

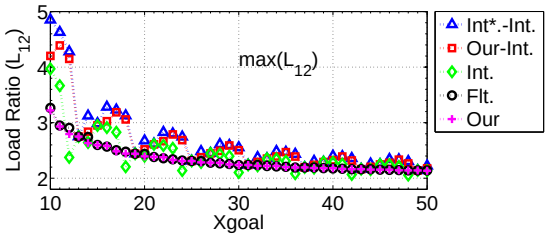
We would like to point out that this fix is specific to an AIMD algorithm with $m_D = 0.5$. However, for other values of m_D similar fixes can be applied.

4.2 Numerical Evaluation

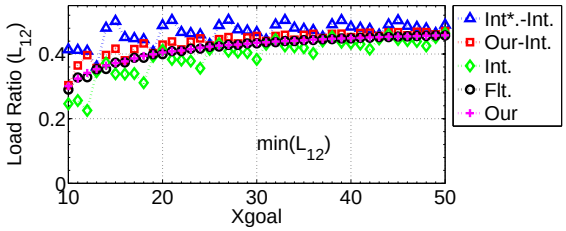
We evaluate in the following the efficiency and fairness achieved by our fix, and the possibility to incrementally deploy a patch. During incremental deployment flows that use our fix shall compete with flows that use the floor function while performing the multiplicative decrease (see Equation (4c)). For qualitative assessment of the impact of such a deployment, we also consider a system with two flows where one flow uses the floor function and the other uses the ceiling function while performing the multiplicative decrease. We use the following notations in the legend of the figures in this section.

- *Our:* The two flows use our fix.
- *Int.:* The two flows use integer arithmetic with the floor function (see Equation (4c)).
- *Flt.:* The two flows use floating point arithmetic.
- *Our-Int.:* Both the flows use integer arithmetic, however, one of the two flows uses our fix while the other uses the floor function. We use this result to assess the impact of incremental deployment of our fix.
- *Int.*-Int.:* Both the flows use integer arithmetic, however, one of the two flows uses the ceiling function while the other uses the floor function.

In Figure 7, we present the maximum and minimum load ratio between two flows; the range of values for

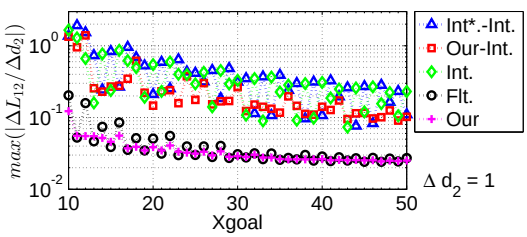


(a) Max Load Ratio.

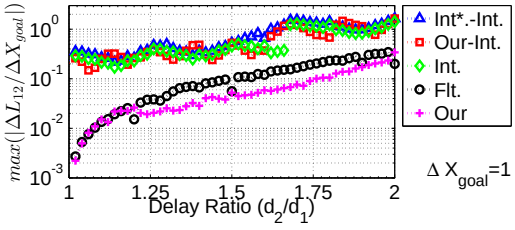


(b) Min Load Ratio.

Figure 7: Maximum and minimum load ratio when $10 \leq X_{goal} \leq 50$, $50 \leq d_1 \leq 75$, and $0.5d_1 \leq d_2 \leq 2d_1$. *We observe that when using our fix the load ratio monotonously converges to the desired values.*



(a) Sensitivity to round duration.

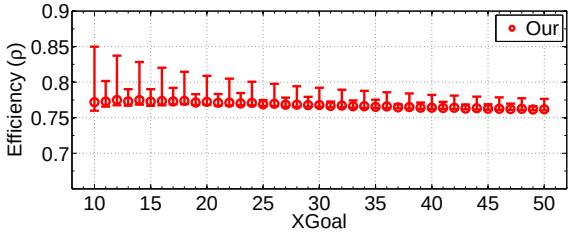


(b) Sensitivity to X_{goal}.

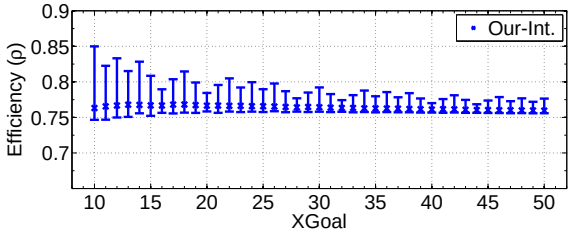
Figure 8: Sensitivity of the load ratio to the round duration and to X_{goal} when $10 \leq X_{goal} \leq 50$, $50 \leq d_1 \leq 75$, and $0.5d_1 \leq d_2 \leq 2d_1$. *We observe that with our fix the system achieves the low sensitivity exhibited when using real numbers.*

X_{goal}, d_1, and d_2 are the ones we used for Figure 3 in Section 3.3. We observe that values for the maximum load ratio with our fix are very close to the ones with floating point arithmetic. During incremental deployment (*Our-Int.*), we observe that the maximum load ratio is not a monotonous function of X_{goal}; the load ratio is significantly larger compared to when both the flows use our fix. This is because the flow that uses our fix performs the ceiling function, which is more aggressive that the floor function, during at most half of the multiplicative decrease events. However, the maximum load ratio during incremental deployment (*Our-Int.*) does not exceed the maximum load ratio observed when one flow uses the ceiling function and the other uses the floor function(*Int*.-Int.*). We make similar observations for the minimum load ratio in Figure 7 (b).

We show in Figure 8 that *by using our fix the load ratio, and thus system fairness, is an order of magnitude less sensitive* to changes in both the round duration and X_{goal}. We use the same range of values used to plot Figure 4 (a) and Figure 4 (b) in Section 3.3. In Figure 8 (a), we observe that with our fix (Our) the sensitivity of the load ratio is close to the one obtained with floating point arithmetic (Flt.), that is much better than with integer arithmetic without our fix (Int.). In a system where one flow uses our fix and the other one uses the floor function during the multiplicative decrease (Our-Int.), the sensitivity is similar to that observed when both the flows use the floor function. In Figure 8 (b), we also observe that our fix solves the sensitivity issue of the load ratio to X_{goal} variations.



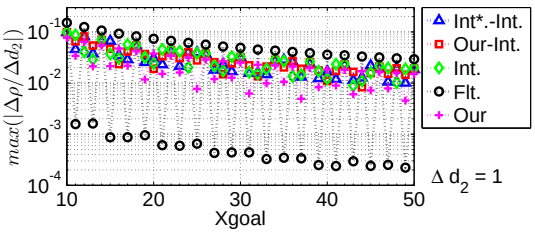
(a) Both flows use our fix.



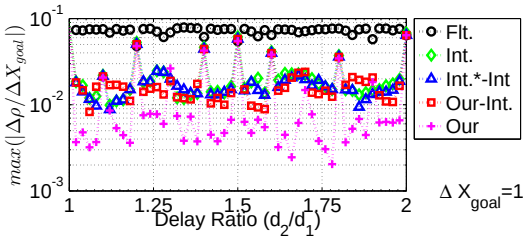
(b) One flow uses our fix while the other uses the floor function.

Figure 9: Efficiency when $10 \leq X_{goal} \leq 50$, $50 \leq d_1 \leq 75$, and $0.5d_1 \leq d_2 \leq 2d_1$. *We observe that by using our fix the system does not loose out on efficiency.*

We now show that *our fix significantly improves efficiency compared to the case with integer arithmetic without the fix.* In Figure 9 (a), we plot the efficiency of the system when both flows use our fix. By comparing Figure 5 (a) and Figure 5 (b) with Figure 9 (a), we observe that the median efficiency with our fix is better



(a) Sensitivity to round duration.



(b) Sensitivity to X_{goal}.

Figure 10: Sensitivity of the efficiency when $10 \leq X_{goal} \leq 50$, $50 \leq d_1 \leq 75$, and $0.5d_1 \leq d_2 \leq 2d_1$. *We observe that, despite using integers, by using our fix the system becomes less sensitive to changes in round duration compared to when using only the floor function.*

than the one observed with integer arithmetic or even with floating point arithmetic. We also observe that the minimum efficiency can be higher by up to 2.5% with our fix than with integer arithmetic without our fix. In Figure 9 (b), we observe that the system efficiency does not suffer when one flow uses our fix while the other uses integer arithmetic without our fix (that is the floor function only) during the multiplicative decrease phase.

We now use Figure 10 to show that *the sensitivity of efficiency to changes in round duration and X_{goal} does not increase with our fix* compared to when integer arithmetic is used with the floor function only. In Figure 10 (a), we plot the sensitivity of the efficiency to changes in the round duration. We observe that for odd values of X_{goal}, the sensitivity with our fix is more than the low sensitivity observed with floating point arithmetic; however, it does not exceed the sensitivity observed when using only integer arithmetic with the floor function. When X_{goal} is even, our fix mimics the behavior of using integer arithmetic and we observe a maximum sensitivity that is less than that observed when floating point arithmetic is used to maintain the system variables. Furthermore, when a source that uses our fix competes with a source using the floor function the maximum sensitivity does not increase to value beyond those exhibited when using either integer arithmetic or floating point arithmetic. In Figure 10 (b) we observe that our fix produces the smallest values for maximum sensitivity of efficiency to X_{goal}. We also observe that during incremental deployment of our fix the maximum sensitivity does not exceed the maximum sensitivity observed when floating point arithmetic is used.

To recap, we observe that with our fix the system fairness improves to compared to when only integer arithmetic is used. We also observe that during incremental deployment of our fix the system fairness is never worse than the fairness exhibited when one flow uses the ceiling function and the other uses the floor function. Our results show that the gains in fairness do not result in poor efficiency. We observe that by using our fix the system is able to maintain the levels of efficiency exhibited when integer arithmetic is used with the floor function.

4.3 Summary

In the previous section, Section 3, we observed that the AIMD algorithm is the fairest with floating point arithmetic and the most efficient with integer arithmetic. In this section, we proposed a fix and showed that our fix achieves both the best fairness and efficiency. We observe that, despite using integers to maintain system variables, our fix makes the algorithm fairness an order of magnitude less sensitive to changes in the network condition without a decrease in system efficiency.

We also observe that during incremental deployment of our solution the efficiency and fairness is similar to that when the competing flows use integer arithmetic with the floor function. Apart from alternating between the floor and ceiling functions we also analyzed the impact of randomly selecting between a floor and ceiling function when the input for the multiplicative decrease is an odd number. We do not present these results to avoid redundancy. We observed that the results in the average case are close to what we observe on successively alternating between the floor and ceiling function, however in the worst case the behavior is like that of using the floor function.

5. RELATED WORK

We would like to point out that we are not the first to extend the model of Chiu *et al.* [6]. Gorinsky *et al.* [16] have extended the model of Chiu *et al.* [6] to highlight some of the issues with the AIMD algorithm. The authors then extend their work and propose Multiplicative Additive Increase Multiplicative Decrease (MIAMD) [15]. Though the authors support sources having different delays to the bottleneck link they do not incorporate the concept of a *round duration* in their paper. In their paper, during the first slot of each round the sources react to the feedback received only during a fraction of the slots of the round duration. These two works by Gorinsky *et al.*are the closest to our work in extending the reference model of Chiu *et al.* [6] used in this paper. Indeed, in this paper we show that AIMD with $a_I = 1$

and $m_D = 0.5$ can have some serious shortcomings with respect to fairness under realistic conditions. However, we also show that AIMD (with $a_I = 1$ and $m_D = 0.5$) implemented using our solution can converge to the desired values of fairness and yet be efficient.

Floyd *et al.* [9] have studied the fairness of AIMD algorithms for different values of a_I and m_D. They show a flows with $a_I = 1$ and $m_D = 0.5$ will be fair only to flows that have $a_I = \frac{3m_D}{2 - m_D}$. In this paper we restrict ourselves to $a_I = 1$ and $m_D = 0.5$ because these values of a_I and m_D are the most widely used values. Furthermore, we are not the only ones restricting to these values of a_I and m_D, most of the analysis of fairness of TCP flows available in the literature either explicitly mention or inherently $a_I = 1$ and $m_D = 1$ [10,11,17,23].

6. DISCUSSION AND CONCLUDING REMARKS

In this paper we revisit the performance of AIMD, the cornerstone algorithm used in internet congestion control. We extend the reference model of Chiu *et al.* [6] to address its shortcomings which includes synchronous response to system feedback. We use our extension to show that the AIMD algorithm under realistic conditions suffers from the classical trade-off between efficiency and fairness.

Under realistic conditions we observe that AIMD suffers from not only poor fairness but also high sensitivity of fairness to changes in round duration or bottleneck capacity. We also observe that the unfairness increases as the bottleneck capacity decreases. We identify the source of the problem to be the rounding errors introduced by the multiplicative decrease operation. These rounding errors arise because of restrictions to use integers; these restrictions are imposed either by the protocol using AIMD or the operating system kernels. We the provide a simple fix to address this issue and show that by using our fix the system fairness and efficiency improves.

One argument against low resource availability could be the ever increasing network capacities. However, Akamai reports that about 44% of connection links to the Akamai servers have a data rate of less than 5 Mbps [3]. Furthermore, our model abstracts the available congestion window as the resource and not the bottleneck bandwidth. Recent advances in queuing techniques such as CoDel [22] are aimed at operating at the optimal value of the congestion window. CoDel ensures that the TCP connections use a congestion window which significantly lower than those observed when the queues suffer from *Bufferbloat* [13]. On the same lines, applications are also explicitly limiting the congestion windows used by their TCP connections [14, 24].

The implementation of TCP in the Linux kernel maintains the congestion window in units of full sized segments while the BSD implementations use units of bytes. By switching from packets to bytes, for the same congestion window, the rounding error decreases by several orders of magnitude. In this paper we show that the rounding errors tend to amplify the unfairness and make the system highly sensitive to changes in network conditions. Based on our observations we recommend maintaining the congestion window in units of bytes. For other protocols that use AIMD we recommend using an appropriate unit for the load that minimizes the rounding error.

We tested our solution using ns3 [1] simulations. We considered a topology where n sources transfer data to n destinations. These n TCP flows pass through a common bottleneck router. We used the actual linux kernel stack for our simulations [1] and considered up to 10 flows and different bottleneck capacities from 256 kbps to 5 Mbps. When no background traffic was passing through the bottleneck router we observe phase effects to be the dominating cause of unfairness [10, 12]. We could not observe any noteworthy improvement by using our solution when using the drop-tail and CoDel queue. With stochastic fair queuing [21] the system was fair even without our fix and on adding our solution we observed an efficiency of no more than 1%. For the CoDel and drop-tail queues, adding a background traffic that randomly generated a load of 10% to 15% of the bottleneck capacity desynchronized the flows. This desynchronization also abstracted the random selection between the floor and ceiling function; we therefore cannot confidently quantify the impact of our solution for existing TCP implementation. We would like to point out that we did not observe any side-effects or unexpected behaviors on using our solution in the Linux kernel.

7. REFERENCES

[1] ns3 network simulator.

[2] Aggarwal, A., Savage, S., and Anderson, T. Understanding the Performance of TCP Pacing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2000), vol. 3, Ieee, pp. 1157 –1165 vol.3.

[3] Akamai. The State of the Internet.

[4] Allman, M., Paxson, V., and Blanton, E. RFC5681: TCP Congestion Control, 2009.

[5] Carofiglio, G., Gallo, M., and Muscariello, L. Icp: Design and evaluation of an interest control protocol for content-centric networking. 304 –309.

[6] Chiu, D.-M., and Jain, R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer*

Networks and ISDN Systems 17, 1 (1989), 1–14.

[7] Corbet, J., Rubini, A., and Kroah-Hartman, G. *LINUX DEVICE DRIVERS*. 2005.

[8] Dukkipati, N., Mathis, M., Cheng, Y., Ghobadi, M., and Nandita Dukkipati Matt Mathis, Y. C. M. G. Proportional Rate Reduction for TCP. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement 2011* (2011).

[9] Floyd, S., Handley, M., Padhye, J., and Widmer, J. Equation-based congestion control for unicast applications. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (New York, NY, USA, 2000), SIGCOMM '00, ACM, pp. 43–56.

[10] Floyd, S., and Jacobson, V. On Traffic Phase Effects in Packet-Switched Gateways. Tech. rep., Lawrence Berkeley Laboratory, 1991.

[11] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw. 1*, 4 (Aug. 1993), 397–413.

[12] Floyd, S., and Paxson, V. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking 9*, 4 (2001), 392–403.

[13] Gettys, J., and Nichols, K. Bufferbloat: Dark buffers in the internet. *ACM Queue* (2011), 1–15.

[14] Ghobadi, M., and Mathis, M. Trickle : Rate Limiting YouTube Video Streaming. *USENIX ATC* (2012).

[15] Gorinsky, S., Georg, M., Podlesny, M., and Jechlitschek, C. A Theory of Load Adjustments and its Implications for Congestion Control. *JOURNAL OF INTERNET ENGINEERING 1*, 2 (2007), 82–93.

[16] Gorinsky, S., and Vin, H. Extended analysis of binary adjustment algorithms. Tech. rep., 2002.

[17] Jacobson, V. Congestion Avoidance and Control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols* (New York, NY, USA, 1988), ACM, pp. 314–329.

[18] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking Named Content. *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies* (2009), 1–12.

[19] Madhyastha, H. V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., and Venkataramani, A. iPlane: An information plane for distributed services. In *Proc. of OSDI* (Nov. 2006), pp. 367–380.

[20] Mathis, M., Semke, J., Mahdavi, J., and Ott, T. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Computer Communication Review 27*, 3 (1997), 67–82.

[21] McKenney, P. Stochastic fairness queueing. 733 –740 vol.2.

[22] Nichols, K., and Jacobson, V. Controlling Queue Delay. *ACM QUEUE 10*, 5 (May 2012).

[23] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of the ACM SIGCOMM'98 conference on Applications, technologies, architectures, and protocols for computer communication* (1998), ACM, pp. 303–314.

[24] Rao, A., Lim, Y.-s., Barakat, C., Legout, A., Towsley, D., and Dabbous, W. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies on - CoNEXT '11* (New York, New York, USA, Dec. 2011), ACM Press, pp. 1–12.