Considerations for Scheduling Periodic Burst Data Transfers

Problem Statement

In a system supporting a multitude of data streams, where the aggregate data transfer bandwidth is much larger than the bandwidth of a single stream, an individual stream's data is transferred in *bursts* which occur at regular intervals. This paper describes work on an algorithm for organizing burst data transfers so as to optimize overall throughput and ensure regular arrival of data.

If all streams have identical data rates, streams can be interleaved with simple time multiplexing, the order of bursts is preserved, and bursts do not overlap (or collide). When streams of different data rates are mixed, multiplexing streams becomes more complicated. This paper describes work on an algorithm for scheduling numerous streams of disparate data rates by assigning stream start times, so that contemporaneous resource utilization is avoided.

Initial goals of this development are

1. Develop a framework for describing and analyzing these data transfers.
2. Describe a heuristic algorithm for allocating transfer starting times.
3. Explore an optimized allocation algorithm.

In the system, each data transfer (burst) consists of a single block of the same fixed size. One way of visualizing this is with a wave form such as shown in Figure 1.

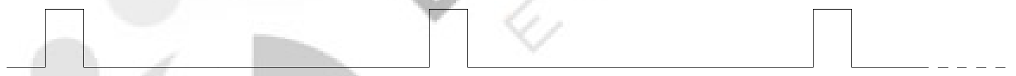


Figure 1 Burst Transfer

Within this context, a periodic data stream can also be illustrated as a series of flags indicating the *timeslices* during which a burst of data occurs. For purposes of discussion, we define the *headspring* of the data stream as the numbered timeslice in which it first occurs.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
	1				1				1				1				1	. . . .

Figure 2 Sample Stream

Scheduling a stream consists of assigning its headspring. When multiple streams have identical periods, simply assigning them different headsprings guarantees that they will never occupy the same timeslice (as long as the number of streams is less than the period). When multiple streams have different periods, scheduling requires more analysis.

In the target system, data blocks contain payload data, overhead, and padding. The type and amount of payload data determine how often a block is transferred, *i.e.*, the *period* of a given data stream. The amount of padding can be adjusted, so that it is possible to shorten the period.

Background

The following analysis makes use of fundamental number theory and abstract algebra. The number of examples and the detail of development could be considered overkill, but the intent is to get the idea across to a wide audience. The concepts needed to fully understand the algorithm and its implementation are described in the last section, Underlying Theory.

Steam Definitions

Consider the data stream shown in Figure 2. It has period 4 and its headspring in timeslice 1. It therefore occupies timeslices 1, 5, 9, 13, 17…. In general, the data stream can be represented by the set

$$\left\{a_0 + k \cdot P\right\}_{k=0}^{\infty} \tag{1.1}$$

where P is the period of the stream and a_0 its headspring. (Note in passing that this is the definition of an arithmetic progression.) In the case of Figure 2, $a_0 = 1$ and $P = 4$, giving

$$\left\{1 + k \cdot 4\right\}_{k=0}^{\infty} \text{ or } \{1,5,9,13,17,21,25,29\ldots\}.$$

Another way to express this is as a linear congruence,

$$k \equiv 1 \pmod{4}$$

which gives us the values of *k* in a single column in the integers mod 4 as shown below.

$$\begin{array}{cccc}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15 \\
\vdots & \vdots & \vdots & \vdots
\end{array}$$

So, with the same generality as (1.1), a data stream is represented by

$$k \equiv a_0 \pmod{P} \tag{1.2}$$

Stream Pair Collisions

In order to schedule multiple streams, it is necessary that no two streams occupy the same timeslice. Two streams occupying the same timeslice is called a *collision*. The following sections define stream collisions using algebraic relations.

Collisions with Diophantine Equations

In order for two streams not to collide, they obviously must have different headsprings. Beyond that, what condition(s) cause two streams to collide (or better, not to collide)?

Given two streams S_1 and S_2 as sets,

$$S_1 : \left\{ h_1 + k \cdot P_1 \right\}_{k=0}^{\infty} \text{ and } S_2 : \left\{ h_2 + j \cdot P_2 \right\}_{j=0}^{\infty} \text{ where } h_1 < h_2, P_1 < P_2$$

a collision occurs if and only if there exits some integers *j* and *k* such that

$$h_1 + k \cdot P_1 = h_2 + j \cdot P_2 ,$$

so the collision condition can be rewritten as a linear Diophantine equation

$$k \cdot P_1 - j \cdot P_2 = h_2 - h_1 \qquad (1.3)$$

Applied to the collision condition, Bézout's Identity states that there exits a pair of integers (*j*,*k*) such that

$$\gcd(P_1, P_2) = kP_1 - jP_2 = h_2 - h_1 \qquad (1.4)$$

so if $h_1 - h_2 = \gcd(P_1, P_2)$, then the collision equation has a solution (for some integers *k*,*j* satisfying (1.3)), meaning that a collision occurs. The collision will land in the k^{th} cycle of stream 1 and the j^{th} cycle of stream 2, numbering from zero.

Collisions with the Chinese Remainder Theorem

Another way to look at this is to write the two streams as a pair of linear congruences.

$$\begin{aligned} S_1 &: k \equiv h_1 (\operatorname{mod} P_1) \\ S_2 &: k \equiv h_2 (\operatorname{mod} P_2) \end{aligned} \qquad (1.5)$$

It is sometimes more convenient, especially then dealing with more than two streams, to use a congruence representation. The Chinese Remainder Theorem (CRT) states that a solution to a system of linear congruences (such as (1.5)) can be found when the moduli (P_1 and P_2) are relatively prime.

We know from the CRT that a solution exists when the moduli are pairwise relatively prime. Therefore we may state that a necessary condition for a set of non-colliding congruences not to have a solution is that each and every pair (P_j, P_k) of the moduli $\{P_1 \ldots P_n\}$ must **not** be relatively prime. This is the same result obtained by applying

Bézout's Identity to the Diophantine stream equations, but is more easily extended to three or more streams.

Two Examples

An example illustrating a stream collision is given in the next section. Following that a new condition governing stream collisions is developed with reference to yet another example.

Example 1- Streams with Relatively Prime Periods

Consider the following streams A and B, where A has period 5 and headspring 0, and B has period 3 and headspring 1. (To shorten this, we'll adopt the notation A_5^0 and B_3^1.)

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A					A					A					A					A					A
	B			B			B			B			B			B			B			B			B

The illustration shows that the streams collide in timeslices 10 and 25. Applying (1.3) to these streams gives

$$\gcd(5,3) = 5k - 3j = 1 \tag{1.6}$$

and Bézout's Identity says that these integers *k* and *j* exist. It's easy to verify that *k*=2, *j*=3 is a solution. And in fact, the illustration shows a collision in cycle 2 of stream A and cycle 3 of stream B (numbering cycles starting with zero).

It is important to note that the difference in headsprings does not matter, at least not in this case. Consider A_5^0 and B_3^2.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A					A					A					A					A					A
		B			B			B			B			B			B			B			B		

Again, applying the collision condition (1.3),

$$k \cdot P_1 - j \cdot P_2 = 5k - 3j = h_2 - h_1 = 2 \tag{1.7}$$

a collision at *k*=1, *j*=1 is predicted and is confirmed by the illustration. It seems that Bézout's Identity does not apply, because $\gcd(5,3) = 1$, not 2. In this case, Bézout's Identity asserts that

$$\exists(j,k): 5k - 3j = 1$$

and indeed (3,2) satisfies the equation. The collision condition asks if

$$\exists(j',k'): 5k' - 3j' = h_2 - h_1 = 2\,.$$

If we let $k' = 2k$ and $j' = 2j$, both Bézout's Identity and the collision condition are satisfied. Generalizing then, we could say that for <u>any</u> headspring difference $\delta = h_2 - h_1$, defining $k' = \delta k$ and $j' = \delta j$ maps the solution guaranteed by Bézout's Identity to the collision condition. Therefore, when two stream periods are relatively prime (their gcd is by definition 1), they will collide no matter how the headsprings are chosen.

Example 2 – Streams whose Periods are not Relatively Prime

Since we know that, to avoid collisions, streams cannot have periods that are relatively prime numbers, the next thing to investigate is how this analysis applies when the stream periods are not relatively prime. When two periods are not relatively prime, they have a greatest common denominator greater than one. Application of (1.4) still holds, only this time, it does not hold for any difference of headsprings.

Consider A_6^0 and B_8^2, and the illustration shown below.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A						A						A						A						A	
		B								B								B							

From the illustration, a collision in timeslice 18 is evident. Using (1.4),

$$\exists(k, j): \gcd(P_1, P_2) = kP_1 - jP_2 = h_2 - h_1$$

$$6k - 8j = 2$$

and solutions are seen for (k,j) = (3,2) which coincide in slice number 18 (cycle 3 of stream A, cycle 2 of stream B). Note that, as before, we could multiply the equation by integer values, $12k - 16j = 4$, $18k - 24k = 6$, …, and calculate more collision occurrences.

The above can be summarized by saying that, when stream periods are relatively prime, the difference in headsprings must not be an integer multiple (including 1) of the greatest common divisor of their periods.

Next, look at A_6^0 and B_8^1.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A						A						A						A						A	
	B								B								B								B

The illustration shows no collision. Since $\gcd(6,8) = 2 \neq h_2 - h_1 = 1$ (the headspring offset is not equal to the greatest common denominator, or an integer multiple thereof), (alternatively, gcd(6,8) does not divide 1), so Bézout's Identity does not guarantee a solution.

Another way to say this is that if $\gcd(P_1, P_2) \mid h_2 - h_1$ (read the greatest common divisor of the periods evenly divides the headspring difference), then the streams collide. There are two formulations of this:

$$h_2 - h_1 \equiv 0 \mod \gcd(P_1, P_2) \tag{1.8}$$

and the original

$$\gcd(P_1, P_2) \mid h_2 - h_1 \tag{1.9}$$

Summary

A successful scheduling of data transfers avoids collisions. What's needed from all this is a guarantee that collisions do not occur. These can be stated for pairs of streams, and extended to more than two streams by pairwise application. The three conditions developed above are summarized below.

Condition 1

The headspings of two streams, no matter their periods, must not be equal. (This should be obvious.)

Proof If $h_1 = h_2$ then $h_1 - h_2 = 0$ in (1.3) and a collision occurs when *(j,k) = (0,0).*

Condition 2

When the periods of two streams differ, their periods must not be relatively prime numbers.

Proof If the p If $P_1 \neq P_2$ and P_1 and P_2 are relatively prime, by definition, their greatest common divisor is 1. Applying Bézout's Identity to the collision condition in (1.3) gives

$$k \cdot P_1 + j \cdot P_2 = h_1 - h_2 = \gcd(P_1, P_2)$$

which says that a collision occurs in the k^{th} period of S_1 and j^{th} period of S_2 at timeslice $h_1 - h_2$. Any integral offset $h_1 - h_2$ in (1.3) is a multiple of 1, so, no matter how the headsprings are chosen, according to Bézout's Identity, there will be some pair of integers *k,j* where there is a collision.

Condition 3

When the periods of two streams differ, if the periods of two streams are not relatively prime and if the difference between their headsprings is evenly divisible by the periods' greatest common divisor, a collision will result.

Proof From (1.9), if

$$d = \gcd(P_1, P_2) \mid h_2 - h_1 \qquad ,$$

there is an integer *m* such that

$$m \cdot d = h_2 - h_1 .$$

Applying (1.3) to two streams, the question is, do there exits integers *j,k* such that

$$k \cdot P_1 - j \cdot P_2 = h_2 - h_1 .$$

Since, $h_2 - h_1 = m \cdot d$, $k \cdot P_1 - j \cdot P_2 = m \cdot d$, and $m \cdot \gcd(P_1, P_2) = k \cdot P_1 - j \cdot P_2$.

Bézout's Identity says that there exist integers *s* and *t* such that $\gcd(P_1, P_2) = s \cdot P_1 + t \cdot P_2$. Let $k = m \cdot s$ and $j = m \cdot t$. Then $m \cdot \gcd(P_1, P_2) = k \cdot P_1 - j \cdot P_2$, so a collision occurs.

Condition 4

If the periods of two streams differ, and the two streams' periods are not relatively prime, and the difference of their headsprings is not evenly divisible by the periods' greatest common divisor, the two streams will not collide.

Proof Using (1.3) to again state the collision condition, we want to show that

$$k \cdot P_1 - j \cdot P_2 = h_2 - h_1$$

has no solution. Let $d = \gcd(P_1, P_2)$. By definition, $d \mid P_1$ and $d \mid P_2$. So, we can let $P_1' = \frac{P_1}{d}$, and $P_2' = \frac{P_2}{d}$, so that $d(k \cdot P_1' - j \cdot P_2') = h_2 - h_1$. From this equation, you can see that unless $d \mid h_2 - h_1$, the equation has no integer solution.

More than Two Streams

The preceding analysis divides the stream allocation problem in two pieces, sort of. So far, it has been shown that two streams' periods must not be relatively prime numbers. An additional constraint was placed on the difference between two streams' headsprings, that it not be evenly divisible by their periods' greatest common divisor. For more than two streams, these conditions may be extended by application to all pairs of streams.

Stream Period GCDs

Beyond relative primality, another pairwise property of stream periods has a significant impact on stream collisions. Up to this point, analysis has focused on whether or not a collision occurs at all. As was shown, to avoid collisions, stream periods must be pariwise relatively prime, *i.e.*, their greatest common divisor must be greater than one. The magnitude of the GCD determines *how many* streams of differing periods can coexist without collision.

Consider streams of periods 44 and 64. $\gcd[44, 64] = 4$, so the periods are not relatively prime. Next, assume that three streams of period 44 are placed with headsprings 0, 1, and 2, as shown in the following illustration.

00	01	02	03	04	05	06	07	08	09	10	11	12
A_{44}^0	A_{44}^1	A_{44}^2										

Beginning with the first unoccupied position (timeslice 3), attempt to place as many instances of period 64 streams as possible, using Condition 4 as a test for collision with existing streams. Applying Condition 4 gives

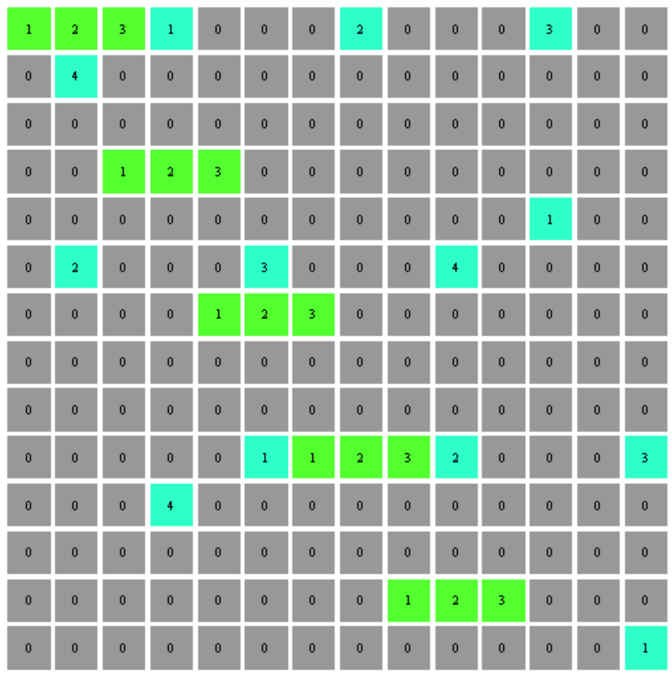
$$\gcd[44,64] \mid h_2 - h_1 \quad or \quad 4 \mid h_2 - h_1 .$$

The following table shows Condition 4 as applied to placing a number of streams with period 64 starting at headspring 3.

	A_{44}^{0}		A_{44}^{1}		A_{44}^{2}	
	$h_2 - h_1$	$4 \mid h_2 - h_1$	$h_2 - h_1$	$4 \mid h_2 - h_1$	$h_2 - h_1$	$4 \mid h_2 - h_1$
A_{64}^{3}	3	no	2	no	1	no

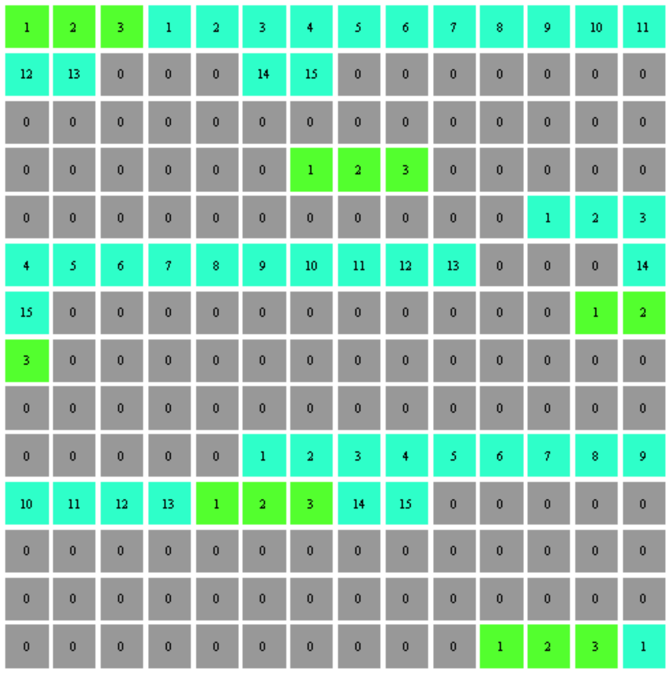
So, the stream A_{64}^{3} can be added, and another stream A_{64}^{4} can be tested. As the table below shows, it fails placement, as do A_{64}^{5}, A_{64}^{6}, and A_{64}^{8}. Stream A_{64}^{7} works because although the headspring difference divides $\gcd[44,64]$, it is being compared to a stream of identical period.

	A_{44}^{0}		A_{44}^{1}		A_{44}^{2}		A_{64}^{3}	
	$h_2 - h_1$	$4 \mid h_2 - h_1$	$h_2 - h_1$	$4 \mid h_2 - h_1$	$h_2 - h_1$	$4 \mid h_2 - h_1$	$h_2 - h_1$	$4 \mid h_2 - h_1$
A_{64}^{4}	4	yes	3	no	2	no	1	no
A_{64}^{5}	5	no	4	yes	3	no	2	no
A_{64}^{6}	6	no	5	no	4	yes	3	no
A_{64}^{7}	7	no	6	no	5	no	4	n/a
A_{64}^{8}	8	yes	7	no	6	no	5	no



Continuing the process will result in placing a total of four period 64 streams: A_{64}^{3}, A_{64}^{7}, A_{64}^{11}, A_{64}^{15}. The figure at left shows these streams of period 64 in turquoise and the three period 44 streams in green, for a bit more than three complete cycles of the pattern.

In terms of overall utilization, the results are pretty dismal. (All of the grey squares are unoccupied.) Examining the table tells us the reason: the magnitude of $\gcd(44,64)$ is small. For streams of differing periods, any time the difference in headsprings is a multiple of 4, a stream collision occurs.



1	2	3	1	2	3	4	5	6	7	8	9	10	11
12	13	0	0	0	14	15	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	2	3	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	2	3
4	5	6	7	8	9	10	11	12	13	0	0	0	14
15	0	0	0	0	0	0	0	0	0	0	0	1	2
3	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	3	4	5	6	7	8	9
10	11	12	13	1	2	3	14	15	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	2	3	1

Repeating the process with periods of 48 and 64, the performance is better. The figure at left shows three streams of period 48 and fifteen streams of period 64. Note that the $\gcd(48, 64)$ is 16. This seems to have everything to do with how many dissimilar streams can be placed.

The preceding example implies that, when choosing stream periods, in addition to the requirement for pariwise non-primality, the numbers should be such that the pariwise greatest common divisors are as large as possible. This statement can be quantified by examining the formulation for Condition 4 given in (1.8).

$h_2 - h_1 \equiv 0 \mod \gcd(P_1, P_2)$, or for simplicity $h_2 - h_1 \equiv 0 \mod d$ occurs with frequency $\frac{1}{d}$, assuming we are trying to occupy all headsprings, since a single element of every residue class modulo *d* maps to zero. Therefore, increasing *d* reduces the frequency of occurrence, thus the occurrence of collisions when trying to fit streams in all possible headspring locations.

Choosing Stream Headsprings

As stated at the outset, scheduling a stream consists of assigning its headspring. Given a set of stream periods, there are two simplistic approaches to assigning headsprings.

Fixed Band Allocation

The first of these is to determine a fixed number of streams for each period (or bitrate), called *bands*. This is approximately what was done above, by first placing three streams of one period (44 and 48), and then placing streams of a second period.

When doing exhaustive allocation for a set of bit rates, fixed band allocation can give priority to certain bit rates by placing them higher in the list of bands to allocate.

Interleaved Allocation

This method consists of iteratively allocating a stream of each period until either all streams have been allocated or no more streams can be allocated.

Example Using Real World Parameters

Consider four bitrates, a 0.025 second timeslice and a 512KB transfer block. Stream periods are calculated in the following manner.

The read period for data blocks is determined by the stream bitrate and the payload content (content less any metadata and fill), as

$$R_C = \frac{(Block\ Size - Metadata - Fill)\times 8}{Data\ Stream\ Bit\ Rate} = \frac{seconds}{block} \tag{1.10}$$

.

An expression of timeslice is $t_s = \frac{seconds}{timeslice}$. Combining the two gives

$$P = \frac{R_C}{t_s} = \frac{timeslices}{data\ block} \tag{1.11}$$

.

Some preliminary calculations for data periods are shown below. There are four bitrates used in the example. A 512 KB data block is used, with 8KB of metadata and 1KB of fill assumed for purposes of illustration. The real numbers can differ, but this is of no consequence, as the fill number will increase anyway as periods are adjusted.

Additionally, data is assumed to be interleaved across four independent machines and scheduling for collision avoidance is to take place on each one. An aggregate period is obtained by truncating the base period and multiplying it by the number of machines.

Bitrate (bps)	Base Period	Group Period
400K	412.06	1648
1.2M	137.35	548
2.5M	65.93	260
3.75M	43.95	172

The numbers for group periods are "starting numbers" in the sense that they need to be tested and adjusted downward (within reason) so that

- they are not pariwise relatively prime, and
- their greatest common divisor is large.

One such set of periods is shown below, along with their integer factorizations.

172	$43^1 \times 2^2$
258	$43^1 \times 3^1 \times 2^1$
516	$43^1 \times 3^1 \times 2^2$
1591	$43^1 \times 37^1$

Band	Period	Streams
1	172	14
2	258	14
3	516	14
4	1591	11

So, the adjusted periods have a GCD of 43. Interleaved allocation using these parameters results in the performance shown in the table at left. This could be improved using more disks (groups), but that is a drastic measure to accommodate 53 streams.

An alternative is to try further adjustment of the periods. A new set of periods with integer factorizations, and the number of streams that can be place using interleaved allocation, is shown in the table below.

Band	Period	Factorization	Streams
1	166	$2^1 \times 83^1$	22
2	249	$3^1 \times 83^1$	24
3	498	$2^1 \times 3^1 \times 83^1$	65
4	1162	$37^1 \times 83^1$	61

Using these periods, a total of 172 streams can be placed, a more than threefold improvement over the previous numbers.

Another way to influence the allocation is to limit the number of streams in one or more bands. Interleaved allocation simply allocates a stream from each band in turn, until a stream from a band cannot be allocated, and continues until no stream from any band can be allocated. By limiting the number of streams from certain bands, we can see if those limits will allow for more streams of the other bands. Assuming that the Band 1 and 2 (3.75 and 2.5 Mb/s) streams are the ones we would like to increase, and noting that they stopped at about 20 streams apiece, a severe limitation of Bands 3 and 4 should give an indication of overall limitations on Bands 1 and 2.

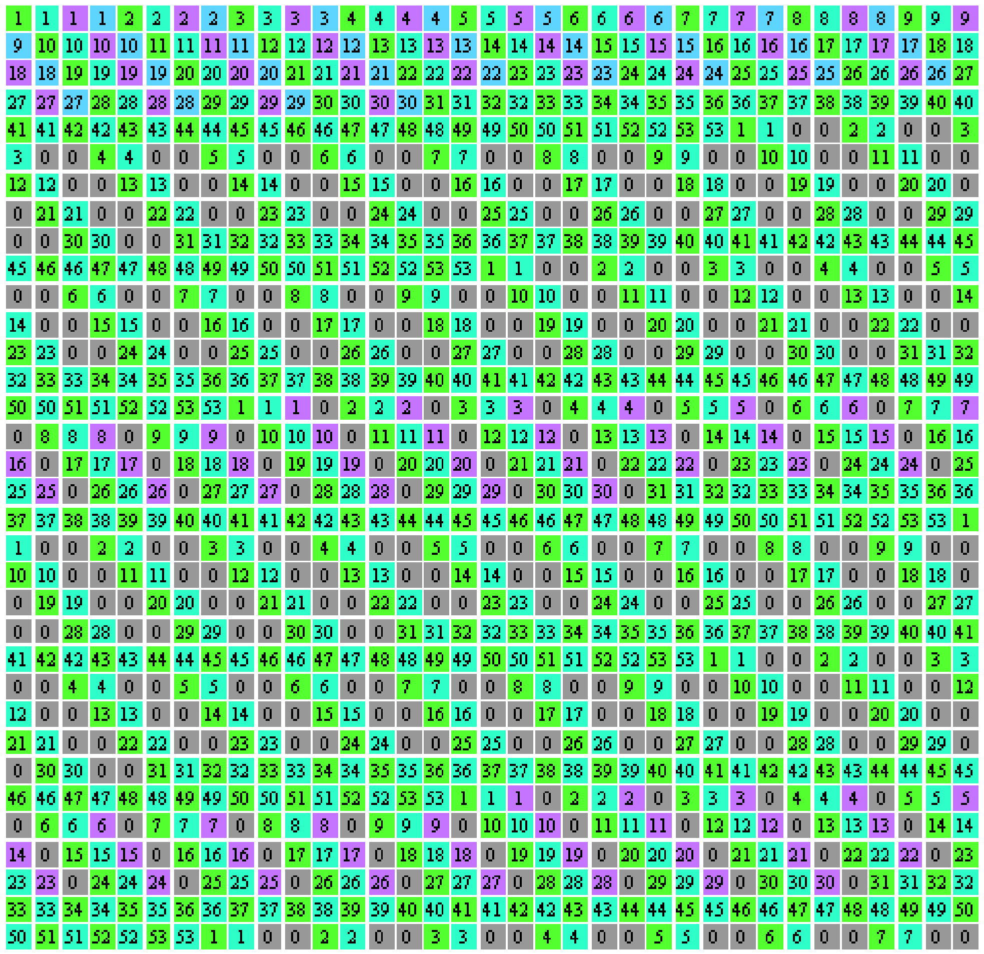
Band	Period	Streams
1	166	43
2	249	38
3	498	4
4	1162	4
1	166	45
2	249	42
3	498	0
4	1162	0

Setting the limit on Bands 3 and 4 to four streams, then zero streams results in the two allocations shown in the table at left. The results show that even when Bands 3 and 4 are removed entirely, the total number of Band 1 and 2 streams that can be allocated is $45 + 42 = 87$.

This is worse performance than padding 2.5 Mbs streams to effectively become 3.75 Mbs streams on the back end, in which case the total number streams would be 166 (or greater, since 166 was a reduction from the original 172).

Band	Period	Factorization	Streams
1	166	$2^1 \times 83^1$	53
2	166	$2^1 \times 83^1$	53
3	498	$2^1 \times 3^1 \times 83^1$	30
4	1494	$2^1 \times 3^2 \times 83^1$	30

It turns out that this seemingly extreme measure leads to a set of periods that allows close to optimal allocation, as shown in the table at right. Note that the total number of streams is $53 + 53 + 30 + 30 = 166$, meaning that an entire cycle of the shortest stream period is completely occupied. Figure 3 shows the allocation map for this scenario. As you can see, the utilization is very high. This is due to the fact the periods' GCD is 166, as can be readily seen from the factorizations, since each contains 2×83.



1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	9	9	9
9	10	10	10	10	11	11	11	11	12	12	12	12	13	13	13	13	14	14	14	14	15	15	15	15	16	16	16	16	17	17	17	17	18	18
18	18	19	19	19	19	20	20	20	20	21	21	21	21	22	22	22	22	23	23	23	23	24	24	24	24	25	25	25	25	26	26	26	26	27
27	27	27	28	28	28	28	29	29	29	29	30	30	30	30	31	31	32	32	33	33	34	34	35	35	36	36	37	37	38	38	39	39	40	40
41	41	42	42	43	43	44	44	45	45	46	46	47	47	48	48	49	49	50	50	51	51	52	52	53	53	1	1	0	0	2	2	0	0	3
3	0	0	4	4	0	0	5	5	0	0	6	6	0	0	7	7	0	0	8	8	0	0	9	9	0	0	10	10	0	0	11	11	0	0
12	12	0	0	13	13	0	0	14	14	0	0	15	15	0	0	16	16	0	0	17	17	0	0	18	18	0	0	19	19	0	0	20	20	0
0	21	21	0	0	22	22	0	0	23	23	0	0	24	24	0	0	25	25	0	0	26	26	0	0	27	27	0	0	28	28	0	0	29	29
0	0	30	30	0	0	31	31	32	32	33	33	34	34	35	35	36	36	37	37	38	38	39	39	40	40	41	41	42	42	43	43	44	44	45
45	46	46	47	47	48	48	49	49	50	50	51	51	52	52	53	53	1	1	0	0	2	2	0	0	3	3	0	0	4	4	0	0	5	5
0	0	6	6	0	0	7	7	0	0	8	8	0	0	9	9	0	0	10	10	0	0	11	11	0	0	12	12	0	0	13	13	0	0	14
14	0	0	15	15	0	0	16	16	0	0	17	17	0	0	18	18	0	0	19	19	0	0	20	20	0	0	21	21	0	0	22	22	0	0
23	23	0	0	24	24	0	0	25	25	0	0	26	26	0	0	27	27	0	0	28	28	0	0	29	29	0	0	30	30	0	0	31	31	32
32	33	33	34	34	35	35	36	36	37	37	38	38	39	39	40	40	41	41	42	42	43	43	44	44	45	45	46	46	47	47	48	48	49	49
50	50	51	51	52	52	53	53	1	1	1	0	2	2	2	0	3	3	3	0	4	4	4	0	5	5	5	0	6	6	6	0	7	7	7
0	8	8	8	0	9	9	9	0	10	10	10	0	11	11	11	0	12	12	12	0	13	13	13	0	14	14	14	0	15	15	15	0	16	16
16	0	17	17	17	0	18	18	18	0	19	19	19	0	20	20	20	0	21	21	21	0	22	22	22	0	23	23	23	0	24	24	24	0	25
25	25	0	26	26	26	0	27	27	27	0	28	28	28	0	29	29	29	0	30	30	30	0	31	31	32	32	33	33	34	34	35	35	36	36
37	37	38	38	39	39	40	40	41	41	42	42	43	43	44	44	45	45	46	46	47	47	48	48	49	49	50	50	51	51	52	52	53	53	1
1	0	0	2	2	0	0	3	3	0	0	4	4	0	0	5	5	0	0	6	6	0	0	7	7	0	0	8	8	0	0	9	9	0	0
10	10	0	0	11	11	0	0	12	12	0	0	13	13	0	0	14	14	0	0	15	15	0	0	16	16	0	0	17	17	0	0	18	18	0
0	19	19	0	0	20	20	0	0	21	21	0	0	22	22	0	0	23	23	0	0	24	24	0	0	25	25	0	0	26	26	0	0	27	27
0	0	28	28	0	0	29	29	0	0	30	30	0	0	31	31	32	32	33	33	34	34	35	35	36	36	37	37	38	38	39	39	40	40	41
41	42	42	43	43	44	44	45	45	46	46	47	47	48	48	49	49	50	50	51	51	52	52	53	53	1	1	0	0	2	2	0	0	3	3
0	0	4	4	0	0	5	5	0	0	6	6	0	0	7	7	0	0	8	8	0	0	9	9	0	0	10	10	0	0	11	11	0	0	12
12	0	0	13	13	0	0	14	14	0	0	15	15	0	0	16	16	0	0	17	17	0	0	18	18	0	0	19	19	0	0	20	20	0	0
21	21	0	0	22	22	0	0	23	23	0	0	24	24	0	0	25	25	0	0	26	26	0	0	27	27	0	0	28	28	0	0	29	29	0
0	30	30	0	0	31	31	32	32	33	33	34	34	35	35	36	36	37	37	38	38	39	39	40	40	41	41	42	42	43	43	44	44	45	45
46	46	47	47	48	48	49	49	50	50	51	51	52	52	53	53	1	1	1	0	2	2	2	0	3	3	3	0	4	4	4	0	5	5	5
0	6	6	6	0	7	7	7	0	8	8	8	0	9	9	9	0	10	10	10	0	11	11	11	0	12	12	12	0	13	13	13	0	14	14
14	0	15	15	15	0	16	16	16	0	17	17	17	0	18	18	18	0	19	19	19	0	20	20	20	0	21	21	21	0	22	22	22	0	23
23	23	0	24	24	24	0	25	25	25	0	26	26	26	0	27	27	27	0	28	28	28	0	29	29	29	0	30	30	30	0	31	31	32	32
33	33	34	34	35	35	36	36	37	37	38	38	39	39	40	40	41	41	42	42	43	43	44	44	45	45	46	46	47	47	48	48	49	49	50
50	51	51	52	52	53	53	1	1	0	0	2	2	0	0	3	3	0	0	4	4	0	0	5	5	0	0	6	6	0	0	7	7	0	0

Figure 3 - Allocation Map 166, 166, 498, 1494

To work backwards and see how much fill data is implied by a given (shortened) data period, the following relationships are used.

$$Period = Time\ Slices\ Between\ Subchunks$$

$$Subchunk\ Time\ Interval = Period \times Time\ Slice\ Amount$$

$$Payload\ Data = \frac{Video\ Bitrate \times Subchunk\ Time\ Interval}{8}$$

Applying these to the periods obtained in the preceding allocation results in the following amounts for fill. Again, there is an assumed figure for metadata. Fill can be adjusted accordingly for the real number.

Answered and Unanswered Questions

1. This study began with a goal of developing a heuristic method for allocating streams. It has succeeded and exceeded in the following sense.

   The two headspring assignment methods – Fixed Band Allocation and Interleaved Allocation are such heuristics. However, the bulk of the work here, the underlying algorithm for predicting (or avoiding) stream collisions, is absolute, and applicable to any assignment scheme, even an optimal one.

2. The two headspring assignment methods, band by band and interleaved, are the obvious ones. There may be others useful for different applications. Additionally, we may want to look at a hybrid of the two, Fixed then Interleaved, to allocate a fixed number of "important" streams, and then some mix of others.

   Also, it should be pointed out that these two methods are not optimal. The preceding example with release 2 numbers may be able to achieve better utilization. The Interleaved heuristic came close. We might be able to do better without a great deal of additional effort, at least in the case where we have such a cooperative GCD.

3. The section entitled Stream Period GCDs, while stating fact, may not be the end of the story. For instance, its argument for maximizing $\gcd(P_1, P_2)$ is based on the goal of occupying every headspring. While that's a nice idea, it is an impossible goal without contrived periods and numbers of streams. There could be additional strategies for choosing sets of stream periods that I do not yet understand.

Underlying Theory

Although an attempt was made to illustrate and convince through examples wherever possible, it is not wise to assume that the proposed algorithm works on that basis alone. Therefore, proofs were given as well. A brief explanation of the underlying mathematics follows, with references for those who are curious.

Division

Integer division of two numbers a and $b \neq 0$ is defined by the existence of two integers *r* and *q* such that $a = q \cdot b + r$, with $0 \leq r < |b|$. *r* is the remainder of $a \div b$ and *q* is the quotient. *r* is commonly written as $\mathrm{mod}[a,b]$. When $\mathrm{mod}[a,b] = 0$, *b* divides *a* (evenly). This is written as $b \mid a$.

Greatest Common Divisor

The Greatest Common Divisor of two integers *(a,b)*, commonly expressed as $\gcd(a,b)$ is the largest integer that divides both *a* and *b*. In other words, if $d = \gcd(a,b)$ then *d* is the largest possible integer which satisfies

$$a = d \cdot m$$
$$b = d \cdot n$$

where m and n are positive integers. If *d* is 1, *a* and *b* are *relatively prime*.

The Euclidean Algorithm

The Euclidean algorithm solves $d = \gcd(a,b)$. This ancient factorization algorithm is used in a variety of computations and is often a convenient method of proof. There are countless references for this information in number theory and abstract algebra texts.

Division

For integers *a* and *b*, with *b*>0, there exist integers *q* and *r* (quotient and remainder) such that $a = q \cdot b + r$, with $0 \le r < b$. Gallian[1] gives a proof.

GCD Algorithm

For any pair of positive integers *a* and *b*, the *greatest common divisor* of *a* and *b*, gcd(*a,b*), can be found by repeated application of integer division to obtain a sequence of decreasing integer remainders $r_1 > r_2 > \cdots$ in the sequence shown in general form (taken from Gallian) below. The arrows highlight how the remainders are carried forward in the sequence.

$$\begin{array}{ll}
a = bq_1 + r_1 & 0 < r_1 < b, \\
\quad\;\; \swarrow & \\
b = r_1q_2 + r_2 & 0 < r_2 < r_1, \\
\swarrow \quad \swarrow & \\
r_1 = r_2q_3 + r_3 & 0 < r_3 < r_2, \\
\swarrow \quad \swarrow & \\
r_2 = r_3q_4 + r_4 & 0 < r_4 < r_3, \\
\vdots & \\
r_{k-3} = r_{k-2}q_{k-1} + r_{k-1} & 0 < r_{k-1} < r_{k-2}, \\
\quad \swarrow \quad \swarrow & \\
r_{k-2} = r_{k-1}q_k + r_k & 0 < r_k < r_{k-1}, \\
\quad \swarrow \quad \swarrow & \\
r_{k-1} = r_kq_{k+1} + 0. &
\end{array}$$

One reason that the above iteration works is because $\gcd(a,b)$ is a linear combination of *a* and *b*, *i.e.*, you can find integers *s* and *t* such that $\gcd(a,b) = as + bt$. (This is Bézout’s Identity, *infra*. See Gallian for a rigorous proof that follows this line of reasoning. Weisstein[2] provides an argument for the validity of this iteration that is somewhat easier to understand.) A more basic argument follows.

Properties of Integer Division

- If $d \mid a$ and $d \mid b$ then $d \mid b-a$.
- If $d \mid a$ and $d \mid b-a$ then $d \mid b$.

Proof: If $d \mid a$, then there is some integer *m* such that $a = d \cdot m$. If $d \mid b$, then there is some integer *n* such that $b = d \cdot n$. So $b - a = d \cdot n - d \cdot m = d \cdot (n-m)$. Therefore, $d \mid b-a$.
As before, if $d \mid a$, $a = d \cdot m$. If $d \mid b-a$, then there is some integer *n* such that $(b-a) = n \cdot d$. So, $b - m \cdot d = n \cdot d$, $b = n \cdot d + m \cdot d = d(n+m)$. Therefore, $d \mid b$.

Application to Greatest Common Divisors

- $\gcd(a,b) = \gcd(b, a-b)$.

Proof: By definition, $d = \gcd(a,b)$ means that $d \mid a$ and $d \mid b$. By the first Property of Integer Division, above, then $d \mid b-a$. So d is a common divisor of *a*, *b*, and $b-a$. Without loss of generality, we can consider $b-a$ equivalent to $|b-a| = a-b$. So $d \mid a-b$. So there exist integers *t*, *u*, and *v* such that

$$d = t \cdot a = u \cdot b = v \cdot (a-b).$$

The integers *t* and *u* are, by definition, such that *d* is the largest possible solution.

Linear Diophantine Equations

Linear Diophantine Equations are 1st degree or lower polynomial equations where the variables are restricted to be integers. There are two approaches to dealing with such equations:

Treating them as ordinary algebraic curves but restricting interest to their integer lattice points.

Using number theoretic properties such as integer divisibility to attempt solution.

This analysis uses the latter approach. There are many interesting Diophantine Equations. Diophantine Equations in the form of Bézout's Identity (below) are used here to describe periodic streams. For more information, see Weisstein.[3]

Bézout's Identity

Bézout's Identity is a result in number theory attributed to the French mathematician Étienne Bézout. It states that if *a* and *b* are both integers > 0, then there exist integers *u* and *v* such that $\gcd(a,b) = a \cdot u + b \cdot v$. The integers *u* and *v* are called Bézout Numbers.

Another way of saying this is to state that the greatest common divisor of two integers *a* and *b* is a linear combination of those two numbers: $\gcd(a,b) = ua + vb$ (where u and v are integers).

This identity is applied to stream equations in a modified form, *viz*., $\gcd(a,b) = a \cdot u - b \cdot v$. Substituting $b' = -b$ puts the expression back into the form stated above.

Bézout's Identity can be proved *via* application of the division algorithm. Again, refer to Gallian and Weisstien.

The Chinese Remainder Theorem

Multiple streams may be expressed as a set of simultaneous linear congruences,

$$\begin{gathered} k \equiv a_{1_0} \pmod{P_1} \\ k \equiv a_{2_0} \pmod{P_2} \\ \vdots \\ k \equiv a_{n_0} \pmod{P_n} \end{gathered}$$

As with the simpler two stream case, a solution to this system of congruences implies a stream collision. The Chinese Remainder Theorem (CRT) states that such a set of congruences is known to have a solution when $P_1, P_2, \ldots P_n$ are relatively prime. References for this famous theorem abound. One is found in Bressoud.[4]

The general statement of the Chinese Remainder Theorem is as follows.
Let $n_1 > 1, n_2 > 1, \ldots n_j > 1$ be pairwise relatively prime integers, and let $n = n_1 \cdot n_2 \cdot \ldots n_j$. Then $k \equiv a_1 (\text{mod } n_1), k \equiv a_2 (\text{mod } n_2), \ldots k \equiv a_j (\text{mod } n_j)$ has one simultaneous solution $k \in \{0, 1, \ldots n-1\}$. Furthermore, if k, k' are two solutions, then $k' \equiv k \pmod{n}$.

The converse of the CRT, which is also true, says that if a solution to a system of linear congruences exists, then the the moduli are pariwise relatively prime. Proof of the converse is not trivial. Dobbs[5] gives four different approaches to proof of the converse employing group theory and tensor products.

[1] Gallian, Joseph A. *Contemporary Abstract Algebra*. New York: Houghton Mifflin, 2006.
[2] Weisstein, Eric W. "Euclidean Algorithm." From *MathWorld*--A Wolfram Web Resource.
[3] Weisstein, Eric W. "Diophantine Equation." From *MathWorld*--A Wolfram Web Resource.
[4] Bressoud, David, and Stan Wagon. *A Course in Computational Number Threory*. New York: Springer-Verlag, 2000.
[5] Dobbs, D.E., "Four Proofs of the converse of the Chinese Remainder Theorem", International Journal of Mathematical Education in Science and Technology, 39:1. 104-109.