

# A Space-Optimal Wait-Free Real-Time Synchronization Protocol

Hyeonjoong Cho  
ECE Dept., Virginia Tech  
Blacksburg, VA 24061, USA  
hjcho@vt.edu

Binoy Ravindran  
ECE Dept., Virginia Tech  
Blacksburg, VA 24061, USA  
binoy@vt.edu

E. Douglas Jensen  
The MITRE Corporation  
Bedford, MA 01730, USA  
jensen@mitre.org

## Abstract

*We present a wait-free protocol for the single-writer/multiple-reader problem in small-memory embedded real-time systems. We analytically establish that our protocol requires lesser (or equal) number of buffers than previously best wait-free protocols for this problem. Further, we prove that our protocol is space-optimal — the first space optimality established for wait-free protocols that consider a-priori knowledge of preemptions. Our evaluation studies and implementation measurements using the SHaRK RTOS kernel confirm the protocol’s superiority and effectiveness.*

## 1. Introduction

Most embedded real-time systems involve mutually exclusive, concurrent access to shared data objects, resulting in contention for those objects. Resolution of the contention directly affects the system’s timeliness, and thus the system’s behavior. Mechanisms that resolve such contention can be broadly classified into: (1) lock-based—e.g., Priority Inheritance and Ceiling protocols [10], Stack Resource Policy [3], DASA [6]; (2) wait-free—e.g., NBW protocol [9], Chen’s protocol [4], [1, 8]; and (3) lock-free—e.g., [2].

Lock-based protocols have several disadvantages such as serialized access to shared objects, resulting in reduced concurrency and thus reduced resource utilization [2]. Further, many lock-based protocols typically incur additional runtime overhead due to increased context switching between activities blocked on shared objects (i.e., “blockers”) and activities that hold locks of those objects (i.e., “lock holders”). The increased context switching occurs when lock-based protocols preempt the currently executing blocker, execute the lock holder until the holder releases the lock, and then resume the blocker’s execution. Another disadvantage of using locks is the possibility of deadlocks that can occur when lock holders crash, causing indefinite starvation to blockers. Further, many (real-time) lock-based

protocols require a-priori knowledge of the ceilings of the locks [10, 3], which may be difficult to obtain in some application contexts. Furthermore, OS data structures (e.g., semaphore control blocks) must be a-priori updated with that knowledge, resulting in reduced flexibility (e.g., recompilation to accommodate new activities) [2].

These drawbacks have motivated research on wait-free and lock-free object sharing in real-time systems. Wait-free protocols use *multiple buffers* (e.g., a circular buffer) for writers and readers. For the single-writer/multiple-reader problem, wait-free protocols typically use multiple buffers for the shared object, where the number of buffers used is proportional to the maximum number of times the readers can be preempted by the writer, when the readers are reading. The maximum number of preemptions of a reader bounds the number of times the writer can update the object while the reader is reading. Thus, by using as many buffers as the worst-case number of preemptions of readers, the readers and the writer can continuously read and write in different buffers, respectively, and avoid interference.

Lock-free protocols allow readers to concurrently read while the writer is writing (without acquiring locks), but the readers check whether their reading was interfered by the writer. If so, they read again. Thus, a reader continuously reads, checks, and retries until its read becomes successful. Since a reader’s worst-case number of retries depends upon the worst-case number of times the reader is preempted by the writer, the additional execution-time overhead incurred for the retries is bounded by the number of preemptions.

Both wait-free and lock-free protocols incur additional costs with respect to their lock-based counterparts. Wait-free protocols generally incur additional space costs due to their multiple buffer usage, which is infeasible in many small-memory, embedded real-time systems. Lock-free protocols generally incur additional time costs due to their retries, which is antagonistic to timeliness optimization.

Prior research have shown how to mitigate these space and time costs, so that they are feasible for embedded real-time systems. An excellent survey of this prior research can be found in [8]. To provide context for our work,

we summarize some important efforts here: In [9], Kopetz and Reisinger present one of the earliest wait-free protocols, where buffer sizes in proportional to worst-case preemptions are used. In [2], Anderson *et al.* show how to bound the retry loops of lock-free protocols through judicious scheduling. In [4], Chen and Burns present one of the most space-efficient wait-free protocols, where the worst-case preemptions need not be a-priori known. In [11], Sundell and Tsigas describe a wait-free protocol for the multiple-writer/multiple-reader problem. In [8], Huang *et al.* improve the time and space costs of Chen’s protocol.

In this paper, we focus on wait-free synchronization for the single-writer/multiple-reader problem in small-memory, embedded real-time systems. We focus on wait-free, as opposed to lock-free, as majority of the lock-free protocols have high computational costs [8]. We consider the single-writer/multiple-reader problem, as it occurs in most embedded real-time systems [8]. We present a wait-free protocol for this problem, by focusing on optimizing *space* costs.

We analytically establish that our protocol requires lesser (or equal) number of buffers than what is needed by previously best wait-free protocols for this problem, including Chen’s protocol [4], Improved Chen’s protocol [8], and NBW protocol [9]. In particular, we establish that our protocol subsumes Chen’s and NBW protocols as special cases. We analytically identify the conditions under which our protocol needs less (and equal) number of buffers than other protocols. Further, we prove that our protocol’s buffer requirements is the absolute minimum that is needed to ensure the safety and orderliness of wait-free synchronization.

To determine our protocol’s buffer needs under a broad range of reader/writer scenarios, we conduct numerical evaluations. We also implement our protocol in the SHaRK RTOS [7]. Our evaluations and implementation measurements confirm the protocol’s superiority and effectiveness.

Thus, the paper’s contribution is our wait-free protocol. Among the class of wait-free protocols that consider a-priori knowledge of preemptions, our protocol’s optimal space lower bound is the first such bound.

The rest of the paper is organized as follows: We present our wait-free protocol and establish its space optimality in Section 2. In Section 3, we formally compare our protocol with Chen’s and NBW protocols. We numerically evaluate our protocol in Section 4, and report our implementation experience in Section 5. We conclude the paper in Section 6.

## 2. A Space-Optimal Wait-Free Protocol

A wait-free protocol solves the asynchronous single-writer/multiple-reader problem by ensuring that each reader accesses the shared object without any interference from the writer. To realize the wait-free mechanism, the protocol must hold two properties: safety and orderliness [4]. The

safety property ensures that the shared object does not become corrupted during reading and writing. The orderliness property ensures that all readers always read the latest data that is completely written by the writer.

The basic idea to achieve the two properties is rooted in the three-slot fully asynchronous mechanism for the single-reader/single-writer problem [5]. For this problem, Chen *et al.* show that three buffers are required to keep the latest completely updated buffer for the next reading, while a writer and a reader are occupying buffers respectively. This mechanism allows that a reader can always obtain data from the buffer slot that is last completely updated, while the writer is writing the new version of the shared data [4].

The buffers needed for the single-writer/multiple-reader problem consist of three types: buffers for readers, a buffer for the latest written data, and a buffer for the next write operation. The buffers for readers must satisfy safety—i.e., sufficient buffers must be available to avoid interference between reading and writing. However, this does not imply that we need as many buffers as there are readers.

The two buffers for writing are required to realize orderliness—i.e., the latest written data must be saved so that a newly activated reader can access it at any time. In addition, the latest written data must be kept until the writer completely writes the next data into another buffer.

We now discuss how to determine the minimum number of buffers that are needed for the single-writer/multiple-reader problem in the following subsections:

### 2.1. Protocol Structure and Task Model

Figure 1 shows a wait-free protocol’s common implementation. W.2 and R.2 show the code sections of the writer and a reader that write and read data, respectively. W.1 is the code section where the writer decides on the buffer for writing, and updates a control variable that indicates the selected buffer. W.3 is the code section where the writer indicates completion of writing and the buffer that has the latest data. In R.1, the reader checks for the latest data to read.

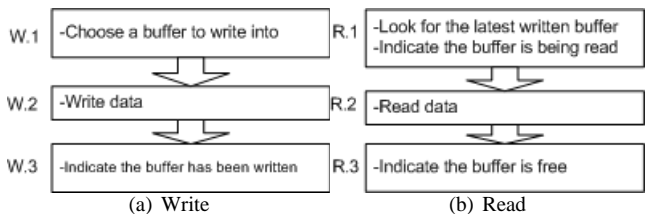


Figure 1. Typical Wait-Free Implementation

The buffer size required for the NBW protocol [9] and the improved protocols in [8] is determined based on the temporal properties of tasks. These prior works consider

the periodic task model, where tasks concurrently share data objects. Aperiodic tasks are handled by a periodic server, so the periodic model is not a limiting assumption. Assuming that all deadlines are met (i.e., during under-load situations and precluding overloads), the maximum number of preemptions of the reader by the writer task in the worst-case can be obtained. We consider the same task model.

## 2.2. Number of Buffers in Use

We introduce some notations for convenience, most of which are similar to those in [4]. We denote the total number of readers as  $M$  and the  $i^{th}$  reader as  $R_i$ . The reader  $R_i$ 's  $j^{th}$  instance of reading is denoted as  $R_i^{[j]}$ . The writer's  $k^{th}$  writing instance is denoted as  $W^{[k]}$ .

$R_i^{[j]}(op)$  stands for a specific operation of  $R_i^{[j]}$ . For example,  $R_i^{[j]}(READING[i] = 0)$  implies the execution of one statement in Chen's algorithm [4].  $W^{[k]}(op)$  also stands for the operation in  $W^{[k]}$ . If  $R_i^{[j]}$  reads what  $W^{[k]}$  writes, we denote it as  $w(R_i^{[j]}) = W^{[k]}$ .

As previously mentioned, safety and orderliness can be achieved with multiple buffers for readers, one buffer for the latest written data, and another buffer for the next writing.

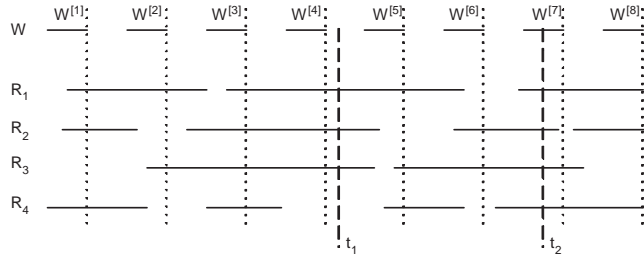


Figure 2. Number of Buffers in Use

Suppose we have 4 readers and 1 writer, as shown in Figure 2. At time  $t_1$ ,  $w(R_1)=W^{[2]}$ ,  $w(R_2)=W^{[2]}$ , and  $w(R_3)=W^{[1]}$ . This implies that two buffers are being used by the readers. In addition, one buffer is required to store and save the latest completely written data by  $W^{[4]}$ , and another is needed for the next writing operation by  $W^{[5]}$ . Thus, four buffers are being used in total, at time  $t_1$ .

At time  $t_2$ ,  $w(R_1)=W^{[6]}$ ,  $w(R_2)=W^{[5]}$ ,  $w(R_3)=W^{[4]}$ , and  $w(R_4)=W^{[6]}$ . The latest written data is by  $W^{[6]}$ , and  $W^{[7]}$  is the next operation. Thus, the total number of buffers used at time  $t_2$  is four, which is the minimum number required at  $t_2$  for ensuring safety and orderliness properties.

The basic intuition for determining the minimum number of buffers is to construct a worst-case where the required number of buffers is as large as possible, when the maximum possible number of interferences of all readers with the writer occurs. We map this problem to a problem called the *Diverse Selection Problem* (or DSP) and then solve it.

## 2.3. Diverse Selection Problem

The DSP denoted as  $D(R, \vec{R}(\vec{x}))$  is defined with the problem range  $R$  and the range vector  $\vec{R}(\vec{x})$  of all elements in the vector  $\vec{x}$ .  $R$  has the lower and upper bounds defined as  $[l, u]$ . Each element  $x_i$  in the vector  $\vec{x}$  has the range  $r_i=[l_i, u_i]$ . The solution to the problem  $D$  is represented as a vector  $\vec{x} = \langle x_1, \dots, x_M \rangle$  where the vector size  $n(\vec{x})$  is  $M$ . Every  $x_i$  must satisfy its range constraint  $r_i$  and the problem range constraint  $R$ . We define  $\{\vec{x}\}$  as a set including all elements of  $\vec{x}$ , but without duplicates. Thus, the size of  $\{\vec{x}\}$ ,  $n(\{\vec{x}\})$ , is less than or equal to  $n(\vec{x})$ . The objective of DSP is to determine the maximum  $n(\{\vec{x}\})$  by selecting  $\vec{x}$ , satisfying all range constraints as diversely as possible.

Given a vector,  $\vec{v} = \langle v_1, \dots, v_i, \dots \rangle$ , we denote the number of  $v_i$ 's having  $k$  value as  $H(\vec{v}, k)$ , and the maximum value among all  $v_i$  elements as  $Top(\vec{v})$ . Given  $D(R, \vec{R}(\vec{x}))$ , the optimal solution of  $D$  when  $R = [t_1, t_2]$  is denoted as  $n_{[t_1, t_2]}^{max}(\{\vec{x}\})$ .

For example, if  $\vec{v} = \langle 1, 2, 2, 2, 6 \rangle$  then,  $H(\vec{v}, 2) = 3$  and  $Top(\vec{v}) = 6$ . An easy approach to solve DSP is by considering all possible cases. The number of all possible cases is  $n(r_1) \times \dots \times n(r_M)$ , where the  $\vec{R}$  of the problem is given as  $\langle r_1, \dots, r_M \rangle$ . By considering all cases, we can select a vector  $\vec{x}$  that maximizes  $n(\{\vec{x}\})$ . However, such an approach would be computationally expensive.

A more efficient approach to solve DSP can be found by an inductive strategy. Consider a DSP  $D(R, \vec{R}(\vec{x}))$ , where  $R = [1, \infty]$  and  $\vec{R} = \langle [1, u_1], \dots, [1, u_M] \rangle$ . If all lower bounds of elements of  $\vec{x}$  are 1, we can define the upper bound vector  $\vec{u} = \langle u_1, \dots, u_M \rangle$  instead of  $\vec{R}$ , for convenience. In the rest of the paper, we call the problem defined by  $D(R, \vec{u})$  as DSP. This is simply because this assumption is well-mapped to the problem of deciding the minimum buffer size for the wait-free protocol.

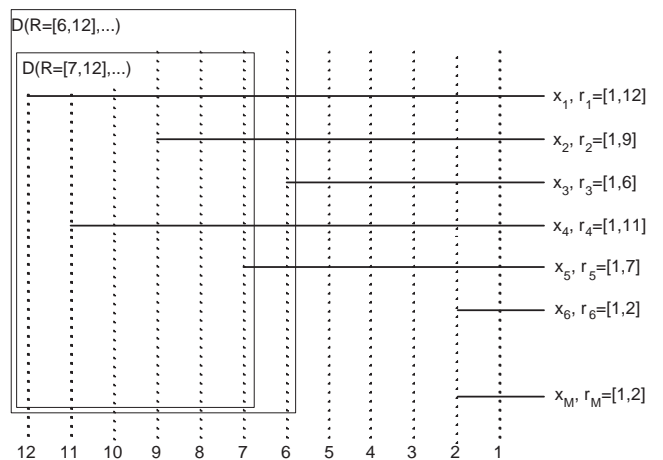


Figure 3. An Inductive Approach to DSP

The solution to the problem  $D(R, \vec{u})$  can be represented as  $n_{[1, Top(\vec{u})]}^{max}(\{\vec{x}\})$ . The idea to decompose the problem is shown in Figure 3. If the solution to the problem  $D([6, 12], \vec{u})$  can be derived from the solution to the problem  $D([7, 12], \vec{u})$ , we can inductively determine the final solution to the problem  $D([1, 12], \vec{u})$ .

**Theorem 1** (DSP for the Wait-Free Protocol). *In the DSP  $D(R, \vec{u})$  with  $R = [1, N]$ ,  $n_{[t+1]}^{max}(\{\vec{x}\}) =$*

$$\begin{cases} n_{[t]}^{max}(\{\vec{x}\}) + 1, & \text{if } \sum_{k=0}^{t+1} H(\vec{u}, N - k) > n_{[t]}^{max}(\{\vec{x}\}) \\ n_{[t]}^{max}(\{\vec{x}\}), & \text{otherwise} \end{cases}$$

where  $N = Top(\vec{u})$ ,  $[t] = [N - t, N]$ , and  $0 \leq t < N$ . When  $t = 0$ , the  $n_{[0]}^{max}(\{\vec{x}\}) = 1$ .

*Proof.* Assume that we have the solution to the problem  $D([N - t, N], \vec{u})$ . When this problem is extended to  $D([N - (t + 1), N], \vec{u})$ , the ranges of several variables  $x_i$  overlap with the problem range  $[N - (t + 1), N]$ . The number of newly added variables that we need to consider is  $H(\vec{u}, N - (t + 1))$ . When the problem range is extended by 1, the maximum possible increment of  $n_{[t+1]}^{max}(\{\vec{x}\})$  is 1. The increment happens only if the number of all  $x_i$  which have their  $r_i$  overlapped with  $[N - (t + 1), N]$  is greater than  $n_{[t]}^{max}(\{\vec{x}\})$ . In other words, this happens when new elements appear in the extended problem scope, or there is an element duplicated within  $[N - t, N]$  at the previous step. Otherwise,  $n_{[t+1]}^{max}(\{\vec{x}\})$  has no change from before. The increment means that the value of one element is determined as diversely as possible. The proof is by induction on  $t$ .

*Basis.* We show that the theorem holds when  $t = 0$ . When the problem is  $D([Top(\vec{u}), Top(\vec{u})], \vec{u})$ , there must be at least one element  $x_i$  with the range  $[1, Top(\vec{u})]$ , and the maximum possible value of  $n_{[0]}^{max}(\{\vec{x}\})$  is 1. Hence, the basis for the induction holds.

*Induction step.* Assume that the theorem holds true when  $R = [N - t, N]$ . We arrive at the optimal solution of  $D([N - (t + 1), N], \vec{u})$  with the optimal solution of  $D([N - t, N], \vec{u})$  as in the base step. Suppose that the derived solution  $n_{[t+1]}^{max}(\{\vec{x}\})$  is not optimal. Then, there must exist another optimal solution  $n_{[t+1]}^{max}(\{\vec{x}'\})$ . Clearly,  $n_{[t+1]}^{max}(\{\vec{x}'\})$  is greater than  $n_{[t+1]}^{max}(\{\vec{x}\})$ . Now, there are two possible cases:

*Case 1.* If  $H(\vec{u}, N - (t + 1)) > n_{[t]}^{max}(\{\vec{x}\})$ , then  $n_{[t+1]}^{max}(\{\vec{x}\})$  is  $n_{[t]}^{max}(\{\vec{x}\}) + 1$ , which is less than  $n_{[t+1]}^{max}(\{\vec{x}'\})$ . Therefore,  $n_{[t]}^{max}(\{\vec{x}\}) < n_{[t+1]}^{max}(\{\vec{x}'\}) - 1$ . This means that there exists another  $\{\vec{x}\}$  that has more than  $n_{[t]}^{max}(\{\vec{x}\})$  elements. This contradicts the assumption that  $n_{[t]}^{max}(\{\vec{x}\})$  is optimal.

*Case 2.* If  $H(\vec{u}, N - (t + 1)) = n_{[t]}^{max}(\{\vec{x}\})$ , then  $n_{[t+1]}^{max}(\{\vec{x}\})$  is  $n_{[t]}^{max}(\{\vec{x}\})$ , which is less than  $n_{[t+1]}^{max}(\{\vec{x}'\})$ . Since no element's range becomes newly overlapped and

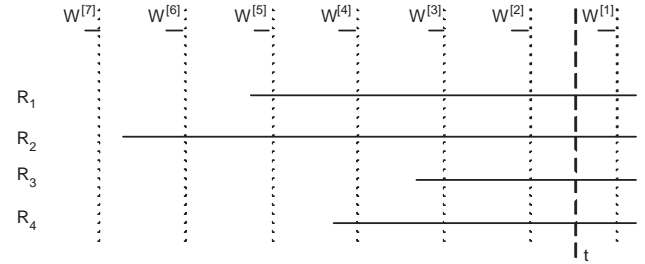
no element has its duplicate,  $n_{[t+1]}^{max}(\{\vec{x}'\}) = n_{[t]}^{max}(\{\vec{x}'\})$ . This means that there exists another  $n_{[t]}^{max}(\{\vec{x}'\})$ , which is greater than  $n_{[t]}^{max}(\{\vec{x}\})$ . This contradicts the assumption that  $n_{[t]}^{max}(\{\vec{x}\})$  is optimal.  $\square$

**Theorem 2** (Solution Vector for the DSP). *In the DSP  $D(R, \vec{u})$  with  $R = [1, N]$ ,  $\{\vec{x}\}_{[t+1]} = \{\vec{x}\}_{[t]} \cup \{N - (t + 1)\}$  when  $\sum_{k=0}^{t+1} H(\vec{u}, N - k) > n_{[t]}^{max}(\{\vec{x}\})$ , where  $N = Top(\vec{u})$ ,  $[t] = [N - t, N]$ , and  $0 \leq t < N$ . When  $t = 0$ ,  $\{\vec{x}\}_{[0]} = N$ .*

*Proof.* By Theorem 1,  $\{\vec{x}\}$  can be constructed by adding  $\{N - (t + 1)\}$  whenever  $n_t^{max}(\{\vec{x}\})$  increases by 1. Note that this  $\{\vec{x}\}$  is one of the solution vectors.  $\square$

## 2.4. Similarity to WFBP

The DSP has similarity with the *Wait-Free Buffer size decision Problem* (or WFBP). In this problem, we are given  $M$  readers and their maximum interferences as  $\langle N_1^{max}, \dots, N_M^{max} \rangle$ . The objective of WFBP is to determine the worst-case maximum number of buffers.



**Figure 4. A Worst-Case of the WFBP**

Figure 4 illustrates how to construct the worst-case where the required number of buffers are as large as possible with an example. For convenience, the index of the writer is reversed compared with Figure 2. In this example,  $R_1$ 's maximum interference is 5, which is illustrated in a line. It means that  $w(R_1)$  may belong to the set  $\{W^{[1]}, \dots, W^{[6]}\}$ . We assume that the worst-case happens at time  $t$  between  $W^{[2]}$  and  $W^{[1]}$ , where  $W^{[2]}$  writes the latest completely written data, and  $W^{[1]}$  is the next writing operation for which another buffer is needed.

For this reason, we restate WFBP as determining  $\vec{x} = \langle w(R_1), \dots, w(R_M) \rangle$  that will maximize  $n(\{\vec{x}\} \cup \{W^{[1]}, W^{[2]}\})$ , where  $w(R_i) \in \{W^{[1]}, \dots, W^{[N_i^{max} + 1]}\}$ . If we abbreviate  $W^{[j]}$  as  $j$ , the problem is redefined as determining  $\vec{x} = \langle x_1, \dots, x_M \rangle$  that will maximize  $n(\{\vec{x}\} \cup \{1, 2\})$ , where  $x_i \in \{1, \dots, N_i^{max} + 1\}$ .

This is equivalent to DSP except that  $n(\{\vec{x}\} \cup \{1, 2\})$  is used as the objective to maximize, instead of  $n(\{\vec{x}\})$ .

Therefore, the final solution  $\{\vec{x}\}$  of a given WFBP is obtained with a sum of the solution from a mapped DSP and a set  $\{1, 2\}$ . We claim that this is correct, because the algorithm for DSP that we propose is designed to find  $\{\vec{x}\}$  which does not have 1 and 2 as its elements, if possible. We can guarantee that in this way, even if the solution from DSP is summed with  $\{1\}$  or  $\{2\}$ , it is still for the worst-case.

**Corollary 3** (Space Optimality). *If a solution to the WFBP can be obtained, then it must be the minimum and space-optimal buffer size that satisfies the two properties, safety and orderliness.*

*Proof.* The solution is the number of buffers needed in the worst-case of the given problem. Even with one less buffer than the obtained solution, we cannot realize all reading and writing, and still satisfy safety and orderliness. Hence, the solution to the WFBP is the minimum and space-optimal.  $\square$

## 2.5. Algorithm for WFBP

We now present an algorithm, Algorithm 1, to solve the WFBP based on the previous sections. The algorithm inputs include the number of readers  $M$  and the maximum interference  $N^{max}[i]$ . The *sum* and the function *doesExist*( $t$ ) correspond to  $\sum H(\vec{u}, \dots)$  and  $H(\vec{u}, t)$  in Theorem 1.

---

### Algorithm 1: Algorithm for WFBP

---

```

1 input : # of readers M; max interference  $N^{max}[M]$ 
2 output : required buffer size  $n$ 
3  $sum = n = 0;$ 
4  $on_1 = on_2 = false;$ 
5 for  $i = 0$  to  $M-1$  do  $N^{max}[i]++;$ 
6  $sort(N^{max}[1, \dots, M]);$ 
7 for  $t = N^{max}[0]$  to  $1$  do
8    $sum += doesExist(t, N^{max}[1, \dots, M]);$ 
9   if  $sum > n$  then
10      $n++;$ 
11     if  $t = 2$  then  $on_2 = true;$ 
12     if  $t = 1$  then  $on_1 = true;$ 
13 if  $on_2 = false$  then  $n++;$ 
14 if  $on_1 = false$  then  $n++;$ 

```

---

To reduce the time complexity of *doesExist*( $t$ ), we sort all  $N^{max}[i]$  before the main loop. *doesExist*( $t$ ) uses a static variable, and does not search the entire array  $N^{max}[i]$  each time. The flag  $on_i$  indicates whether or not the DSP solution includes  $i$ . If it does not include 1 or 2, the required buffer size for the WFBP solution,  $n$ , is incremented.

The time complexity of this algorithm is  $O(M \log M + N^{max})$ . We believe that this cost is reasonable, as the algorithm is run off-line for determining the buffer needs.

## 2.6. A Wait-Free Implementation

The NBW protocol uses a circular buffer to realize wait-free synchronization. The idea behind the circular buffer is that while a writer circularly accesses the buffers, the readers follow the writer. However, we cannot use the circular type of buffer because a writer in our protocol needs to determine a safe buffer, which can be any of the buffers.

---

### Algorithm 2: Modified Chen's Protocol for Writer

---

```

1 Data: BUFFER [1, ..., NB]; - NB: # of buffers
2 Data: READING [1, ..., n]; - n : # of readers
3 Data: LATEST
4 GetBuf ()
5 begin
6   bool  $InUse[1, \dots, NB];$ 
7   for  $i = 1$  to  $NB$  do  $InUse[i] = false;$ 
8    $InUse[LATEST] = true;$ 
9   for  $i = 1$  to  $n$  do
10     $j = READING[i];$ 
11    if  $j \neq 0$  then  $InUse[j] = true;$ 
12     $i = 1;$  while  $InUse[i]$  do  $++i;$ 
13    return  $i;$ 
14 end
15 Writer ()
16 begin
17   integer  $widx, i;$ 
18    $widx = GetBuf();$ 
19   Write data into BUFFER [ $widx$ ];
20   LATEST =  $widx;$ 
21   for  $i = 1$  to  $n$  do
22     Compare-and-Swap(READING [ $i$ ], 0,  $widx$ );
23 end

```

---

The same situation arises with Chen's protocol, where the writer can access anywhere. Thus, to implement our protocol, we slightly modify Chen's protocol. Our implementation scheme is shown in Algorithms 2 and 3. In Algorithms 2 and 3, the *GetBuf*() function searches the empty buffer to write to the buffers assigned by Algorithm 1.

---

### Algorithm 3: Modified Chen's Protocol for Reader

---

```

1 Data: BUFFER [1, ..., NB]; - NB: # of buffers
2 Data: READING [1, ..., n]; - n : # of readers
3 Data: LATEST
4 Reader ()
5 begin
6   integer  $ridx;$ 
7   READING [ $id$ ] = 0;
8    $ridx = LATEST;$ 
9   Compare-and-Swap(READING [ $id$ ], 0,  $ridx$ );
10   $ridx = READING[id];$ 
11  Read data from BUFFER [ $ridx$ ];
12 end

```

---

Compared with the implementation in [8], our approach does not need to implement separate protocols for "fast"

readers and “slow” readers. Additionally, we achieve the speed improvement by reducing the required buffer size, which reduces the number of iterations in *GetBuf()*’s loop, compared with the original Chen’s protocol [4].

### 3. Formal Comparison with Chen’s and NBW

#### 3.1. Special Case Behavior

The buffer size that the NBW protocol [9] requires depends on the maximum number of interferences that a reader can suffer from the writer. It does not depend on the number of readers, because simultaneous reading by the readers accesses the same buffer, irrespective of the number of readers. On the other hand, the buffer size that the Chen’s protocol [4] requires is directly proportional to the number of readers, and is independent of the number of interferences. We now show that our protocol subsumes both Chen’s protocol and the NBW protocol as special cases.

**Lemma 4.** *The buffer size for Chen’s protocol [4] is a special case of the WFBP solution given in Algorithm 1.*

*Proof.* Assume that we are given  $M$  readers and no information about interferences. We can map this problem to DSP, by setting  $R$  as  $[1, \infty]$  and the upper-bounds of  $\vec{x}$  as  $\langle \infty, \dots, \infty \rangle$ . According to Theorem 2,  $n(\{\vec{x}\})$  cannot exceed  $n(\vec{x})$ . Thus, the worst-case buffer size is obtained as  $(M + 2)$ , that is  $n(\vec{x}) + n(\{1, 2\})$ . This is exactly the same value as that obtained by Chen’s protocol.  $\square$

**Lemma 5.** *The buffer size for NBW protocol [9] is a special case of the WFBP solution given in Algorithm 1.*

*Proof.* Assume that we are given infinite number of readers with a knowledge of  $Top(\vec{u}) = N^{max}$ . This problem can be modeled as the problem with  $R = [1, N^{max} + 1]$  and  $\forall i, u_i = N^{max} + 1$  for the worst-case. By Theorem 1,  $H(\vec{u}, N) = \infty$ , and whenever  $t$  increases,  $n(\{\vec{x}\})$  increases by 1 until  $t$  and  $n(\{\vec{x}\})$  reaches to  $N^{max}$  and  $N^{max} + 1$ , respectively. Thus, the worst-case buffer size is obtained as  $N^{max} + 1$ , i.e.,  $n(\{1, \dots, N^{max} + 1\} \cup \{1, 2\})$ . This is exactly the same value as that obtained by NBW protocol.  $\square$

**Theorem 6** (Upper Bound of the WFBP solution). *In the WFBP,  $n^{max}(\{\vec{x}\}) \leq \min(M + 2, N^{max} + 1)$ , where  $M$  is the number of readers and  $N^{max}$  is the maximum number of interferences that a reader can suffer.*

*Proof.* Proof follows directly from Lemmas 4 and 5.  $\square$

Chen’s protocol is attractive because the number of interferences need not be known a-priori. On the other hand, NBW has the advantage that the required number of buffers

can be further reduced if the number of interferences are much smaller than the number of readers. Additionally, we note that the number of buffers needed by our algorithm is less than or equal to that of Chen’s or NBW protocol.

#### 3.2. Buffer Size Conditions

According to Theorem 6, our wait-free protocol always finds the number of required buffers which is less than or equal to that of Chen’s protocol or the NBW protocol.

We now identify the precise conditions under which the required buffer size of our protocol is equal to that of Chen’s or NBW. To derive the conditions, we observe two properties in the WFBP. In the following theorem, we introduce a notation  $\{\{\vec{x}\}\}$ , which denotes the set including all possible solutions  $\{\vec{x}\}$  for the given DSP.

**Theorem 7** (Chen’s Tester). *When the number of readers in the wait-free buffer size decision problem is  $M$  and  $N^{max} > M$ ,  $\{3, \dots, M + 2\} \in \{\{\vec{x}\}\}$ , if and only if  $n^{max}(\{\vec{x}\}) \geq M + 2$ .*

*Proof.* We prove both necessary and sufficient conditions.

*Case 1.* Assume that when  $\{3, \dots, M + 2\} \in \{\{\vec{x}\}\}$ ,  $n^{max}(\{\vec{x}\}) < M + 2$ . Since the size of the optimal solution is less than  $M + 2$ , the size of  $\{\vec{x}\}$  cannot exceed  $M + 2$ . This contradicts our assumption that  $\{1, 2\} \cup \{3, \dots, M + 2\}$  is a solution.

*Case 2.* Assume that the set  $\{\vec{x}\}$  is  $\{x_3, \dots, x_{M+2}\}$ , in which  $x_i$ ’s are different between each other and aligned in increasing order. Now, all  $x_i$  must not be 1 or 2, otherwise  $n^{max}(\{\vec{x}\})$  is less than  $M + 2$ . Therefore,  $x_3$  should be greater than or equal to 3, and  $x_4$  is greater than  $x_3$ . Inductively,  $x_{i+1} \geq x_i + 1$ , where  $3 \leq i < M + 2$ . In other words, since  $x_i \geq x_{i-1} + 1 \geq x_{i-2} + 2 \geq \dots$ , the inequality  $u_i \geq x_i \geq i$  holds. For this reason,  $\{3, \dots, M + 2\}$  satisfies the range constraints of all elements.  $\square$

By Theorem 7,  $n^{max}(\{\vec{x}\}) < M + 2$ , if  $\{3, \dots, M + 2\} \notin \{\{\vec{x}\}\}$ . This means that by checking if  $\{3, \dots, M + 2\}$  is feasible for the problem, we can determine whether or not it requires  $M + 2$  buffers that Chen’s protocol needs.

**Theorem 8** (NBW Tester). *When the number of readers in the wait-free buffer size decision problem is  $M$  and  $N^{max} \leq M$ ,  $\{2, \dots, N^{max} + 1\} \in \{\{\vec{x}\}\}$ , if and only if  $n^{max}(\{\vec{x}\}) \geq N^{max} + 1$ .*

*Proof.* We prove both necessary and sufficient conditions.

*Case 1.* Assume that when  $\{2, \dots, N^{max} + 1\} \in \{\{\vec{x}\}\}$ ,  $n^{max}(\{\vec{x}\}) < N^{max} + 1$ . Since the size of the optimal solution is less than  $N^{max} + 1$ , the size of  $\{\vec{x}\}$  cannot exceed  $N^{max} + 1$ . This contradicts our assumption that  $\{1, 2\} \cup \{2, \dots, N^{max} + 1\}$  is a solution.

Case 2. Assume that the set  $\{\vec{x}\}$  is  $\{x_2, \dots, x_{N^{max}+1}\}$ , in which  $x_i$ 's are different between each other and aligned in increasing order. Now, all  $x_i$  must not be 1, otherwise  $n^{max}(\{\vec{x}\})$  is less than  $N^{max} + 1$ . Therefore,  $x_2$  should be greater than or equal to 2, and  $x_3$  is greater than  $x_2$ . Inductively,  $x_{i+1} \geq x_i + 1$  where  $2 \leq i < N^{max} + 1$ . In other words, since  $x_i \geq x_{i-1} + 1 \geq x_{i-2} + 2 \geq \dots$ , the inequality  $u_i \geq x_i \geq i$  holds. For this reason,  $\{2, \dots, N^{max} + 1\}$  satisfies the range constraints of all elements.  $\square$

We can also investigate if a given WFBP needs  $N^{max} + 1$  buffers or less by checking feasibility with  $\{2, \dots, N^{max} + 1\}$ . We call  $\{3, \dots, M+2\}$  and  $\{2, \dots, N^{max} + 1\}$  as ‘‘Chen’s tester’’ and ‘‘NBW tester,’’ respectively.

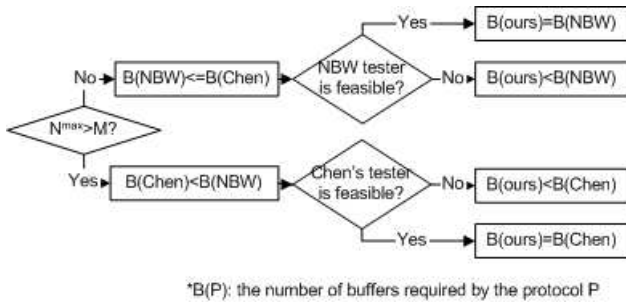


Figure 5. Decision Procedure

From Theorems 7 and 8, we derive a decision procedure that determines the wait-free protocol with the lowest buffer size. Figure 5 shows this procedure. To illustrate it, we use the WFBP example in [8], which is also shown in Table 1.

Table 1. Task Set

Task	$N^{max}$
Reader 0	2
Reader 1	2
Reader 2	2
Reader 3	3
Reader 4	3
Reader 5	14
Reader 6	49

Table 2. Asymptotical Time Complexities

Wait-Free Protocol	Read	Write
NBW	$O(1)$	$O(1)$
Chen’s	$O(1)$	$O(n)$
Improved Chen’s	$O(1)$	$O(n)$
Ours	$O(1)$	$O(n)$

By our decision procedure, since  $N^{max} > M$ , Chen’s protocol requires smaller number of buffers than NBW. The next step is determining whether Chen’s tester, which is  $\langle 3, 4, \dots, 9 \rangle$  in this problem, is feasible. It turns out that it is not feasible, as the second element 4 in the tester is out of the range  $[1, 3]$  of reader 1. Hence, we expect to find smaller number of required buffers than that of Chen’s protocol.

Algorithm 1 determines that we need 6 buffers for this problem. We determine a vector  $\{\vec{x}\} = \{1, 2, 3, 4, 15, 50\}$  as a worst-case candidate for the WFBP from Theorem 2. As mentioned earlier, the solution means that one of

the worst-cases occurs when we need buffers for writers  $\{W^{[1]}, W^{[2]}, W^{[3]}, W^{[4]}, W^{[15]}, W^{[50]}\}$ .

### 3.3. Comparison with Improved Chen’s Protocol

In [8], Huang *et al.* suggest a transformation mechanism to reduce the buffer size needs of a given wait-free protocol. The transformation is applied to many wait-free protocols including Chen’s protocol. The transformed Chen’s protocol is called Improved Chen’s protocol in [8].

We cannot formally compare our protocol with Improved Chen’s protocol in terms of space overhead, because no analytical foundation is given for the transformation mechanism in [8]. Due to the lack of an analytical underpinning, we regard Improved Chen’s protocol as a heuristic strategy. Consequently, a formal comparison is not possible, and only an experimental comparison is possible, where the two protocols can be compared for as many cases as possible. We do this in Section 4. Our experiments in Section 4 reveal that the buffer size needs of our protocol and Improved Chen’s are the same, for all the cases that we consider.

Of course, this does not imply that Improved Chen’s and ours always need the same number of buffers, because it is impossible that our evaluation studies in Section 4 cover all the cases. Nevertheless, we explain why Improved Chen’s and ours require the same number of buffers in Section 4. Note that with Corollary 3, we guarantee that the buffer size needed for wait-free cannot be reduced any further.

### 3.4. Comparison of Time Complexity

Implementation of NBW and Chen’s protocols require the Compare-And-Swap (CAS) instruction. The CAS instruction is used to atomically modify control variables of the wait-free protocol by combining comparison and swap operations into a single instruction. The instruction is available in many modern processors and takes constant time.

NBW has no loop within both write and read operations. However, Chen’s protocol has 3 loops within the write operation and no loop within the read operation. With  $n$  buffers, the time complexity of Chen’s writing operation is  $O(n)$ .

Improved Chen’s protocol and our protocol are variations of Chen’s protocol, and hence have similar time complexities as that of Chen’s writing and reading. According to Theorem 6, the loop iteration in our protocol’s write operation cannot exceed  $M + 2$ . Thus, the time complexity of our protocol is  $O(n)$ , which is the same as that of Chen’s. Since the asymptotical speeds are therefore similar, a speed improvement can be obtained (for Chen’s, Improved Chen’s, and ours) by reducing the buffer size. Table 2 summarizes the asymptotical time complexities of the protocols.

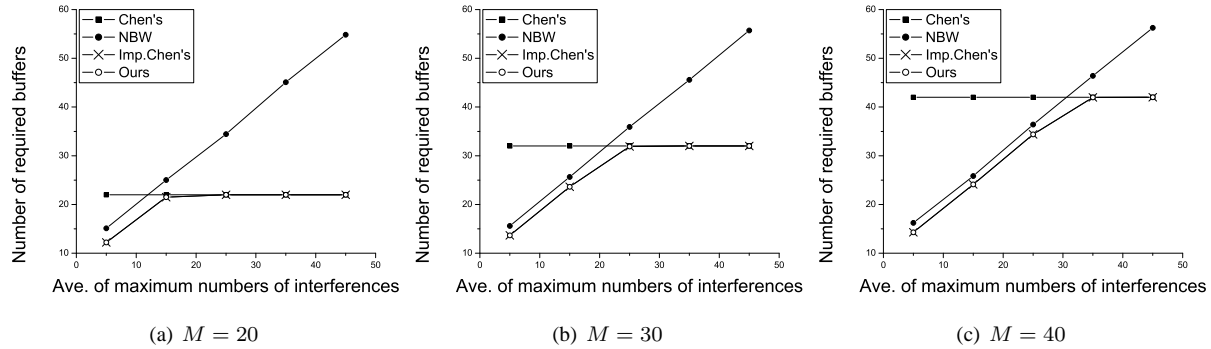


Figure 6. Buffer Sizes Under Increasing Interferences With Normal Distribution for  $N_i^{max}$

## 4. Numerical Evaluation Studies

We conduct numerical evaluations to evaluate the buffer size needs of our protocol under a broad range of reader/writer conditions, including increasing maximum interferences and readers. We also consider NBW, Chen’s, and Improved Chen’s protocols for comparative study. We consider Improved Chen’s protocol among all protocols in [8], because it is the most space-efficient protocol in [8].

We exclude the Double Buffer protocol [8] from our study as it needs nearly two times the buffer space than Chen’s protocol. (The Double Buffer protocol trades off space for time.) Thus, our protocol will clearly outperform the Double Buffer protocol in terms of buffer needs.

### 4.1. Increasing Interferences

We consider a task set with 1 writer and multiple readers whose maximum number of interferences  $N_i^{max}$  is randomly generated with a normal distribution (with a fixed standard deviation of 5), and by varying the average. The protocols are evaluated by their buffer size needs — the actual amount of needed memory is the number of buffers times the message size in bytes. Each experiment is repeated 100 times to determine the average buffer sizes.

Figure 6 shows the buffer size needs of each protocol as the average  $N_i^{max}$  is increased from 5 to 45, for 20, 30, and 40 readers. From the figure, we observe that as  $N_i^{max}$  increases, the buffer size needs of NBW increases, whereas that of Chen’s protocol remains the same (for a given reader size), since its buffer needs is proportional only to the number of readers. As the number of readers increases from 20 to 40, Chen’s protocol needs increasing number of buffers. Meanwhile, the number of buffers that our protocol requires never exceeds that of Chen’s and NBW’s, as Theorem 6 holds. Interestingly, the number of buffers that Improved Chen’s protocol requires is exactly the same as that of ours. Note that no analysis on the buffer size needs of Improved

Chen’s is presented in [8], whereas Theorem 6 gives the upper bound on the buffer size needs of our protocol.

We observed exact similar trends for other fixed standard deviations for  $N_i^{max}$ ’s distribution, and other distributions for  $N_i^{max}$ . We omit these results here due to space limitations. These results confirm that the trend in Figure 6 is independent of  $N_i^{max}$ ’s distributions and standard deviations.

### 4.2. Heterogenous Readers in Multiple Groups

From Figure 6, we also observe that when most readers have small  $N_i^{max}$ , the number of buffers needed by our protocol approaches that of NBW’s. Moreover, when most readers have larger  $N_i^{max}$ , the number of buffers needed by our protocol approaches that of Chen’s protocol’s.

This motivates us to study the buffer size needs of our protocol under two groups of readers, one that has small  $N_i^{max}$ ’s and the other that has large  $N_i^{max}$ ’s. (A similar evaluation is conducted in [8], where readers are classified as “fast” and “slow.”) We divide tasks into the two groups whose averages of the (normal) distribution for  $N_i^{max}$ ’s are fixed as 5 and 45, respectively. We then vary the ratio of the two groups. For example, 3:1 in the X-axis in Figure 7(a) means that the readers having smaller  $N_i^{max}$  are 3 times more than the readers having larger  $N_i^{max}$ .

Figure 7 shows the buffer sizes of each protocol as the ratio is varied from 3:1 to 1:3, for 20, 30, and 40 readers. We observe that the buffers needed for NBW, Improved Chen’s, and our protocol increase as the readers with larger  $N_i^{max}$  increases. This result is consistent with that in [8], where Improved Chen’s is shown to require less buffers, as fast readers with smaller  $N_i^{max}$  increases. The results confirm that ours and Improved Chen’s require the minimum buffer size when considering two heterogenous reader groups.

We now consider a more complex scenario with three reader groups, called “fast,” “slow,” and “medium,” which are not considered in [8]. The averages of the (normal) distribution for  $N_i^{max}$ ’s for the three groups are fixed as 5, 25,

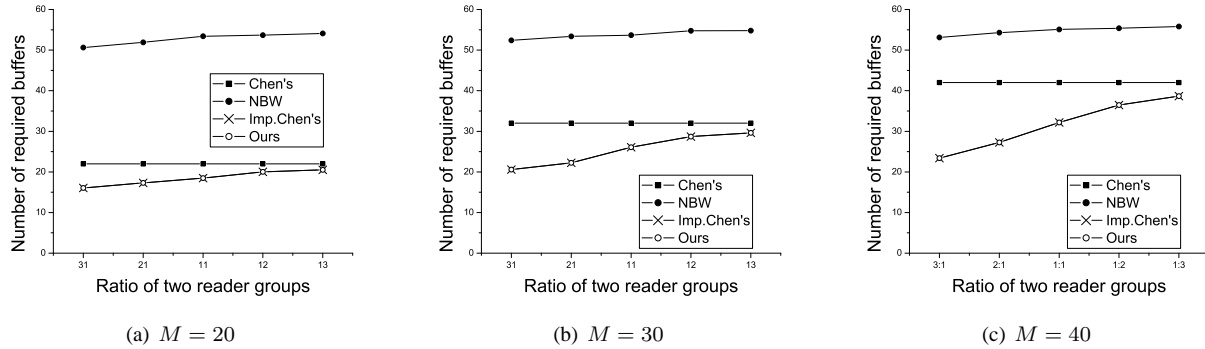


Figure 7. Buffer Sizes With 2 Reader Groups Under Varying Reader Ratio, for 20, 30, and 40 Readers

and 45, respectively, and the ratio of the three groups are varied from 6:3:1 to 1:3:6. Figure 8 shows the results.

From the figure, we observe that as the number of fast readers increases, the number of buffers needed decreases. Further, we observe that the buffer size required by Improved Chen's is the same as that of ours even when we include the "medium" reader group in our evaluation.

## 5. Implementation Experience

A wait-free protocol's practical effectiveness is determined by its space and time costs. In developing a wait-free protocol, we focus on optimizing space costs, and we establish the space optimality of our protocol. Although reducing the protocol's time costs is not our goal, we now determine the time costs to establish our protocol's effectiveness.

Our wait-free protocol (Algorithms 2 and 3) is a modification of Chen's protocol, augmented with the buffer size computed by Algorithm 1. Thus, we expect that our protocol incurs at most as much time overhead as that of Chen's. Moreover, the higher space efficiency that our protocol enjoys can lead to higher time efficiency, because it reduces the search space for determining the protocol's safe buffer—e.g., `GetBuf()`'s loop in Algorithm 2.

To evaluate the actual time costs of our protocol, we implement our protocol in the SHaRK (Soft Hard Real-Time Kernel) OS [7], running on a 500MHz, Pentium-III processor. Similar to Section 4, we also implement Chen's, Improved Chen's, and NBW protocols for a comparative study. We also consider lock-based sharing in this study.

We consider a task set with 20 readers and a writer, and use a message size of 8 bytes for an inter-process communication (or IPC). We measure the average-case execution time (or ACET) for performing an IPC. The execution time for an IPC is the time needed for executing the code segment that accesses the shared object. With traditional lock-based sharing, this code segment is the critical section. Note that

a wait-free protocol's IPC execution time includes times for controlling protocol's variables, accessing the shared object, and potential interference from other tasks.

In Section 4.2, we varied the ratio of two reader groups whose averages of the (normal) distribution for  $N_i^{max}$ 's are fixed as 5 and 45, respectively. We now select two cases from which the ratio of readers having smaller and larger  $N_i^{max}$  are 4:1 and 1:4, respectively. These two cases can be represented as 16 fast and 4 slow readers, and 4 fast and 16 slow readers, respectively, for the purpose of Improved Chen's [8], since that protocol needs the readers to be classified as "slow" and "fast". We fix the writer's period as 0.2 msec and let the writer invoke 6,000,000 times during our experiment interval for computing the ACETs. The period of the 20 readers ranges from 400 usec to  $\sim 10msec$ .

Figure 9 shows the measurements from our implementation. We observe that NBW has the smallest ACET, lock-based sharing has the largest ACET, and Chen's, Improved Chen's, and our protocol have almost the same ACET. NBW has the smallest ACET, because its implementation does not have any loop (and thus less computational costs) inside both the reader and writer functions. Lock-based sharing has the largest ACET due to its blocking times. Further, accessing and releasing locks in SHaRK is done through system calls, which takes longer than wait-free protocols (which are implemented without system calls).

It is not surprising to observe that Chen's, Improved Chen's, and ours have almost the same ACET, as Improved Chen's and ours are variations of Chen's. However, note that in [8], when the number of fast readers are increasing, the ACET of Improved Chen's tends to be shorter because the needed buffer size decreases. This trend does not appear in our experiments, for both Improved Chen's and ours. This is because the expected speed improvement is only (approximately) 0.1 usec. This difference is small enough to be affected by the OS type, code optimizations, and measurement methodology, among other factors.

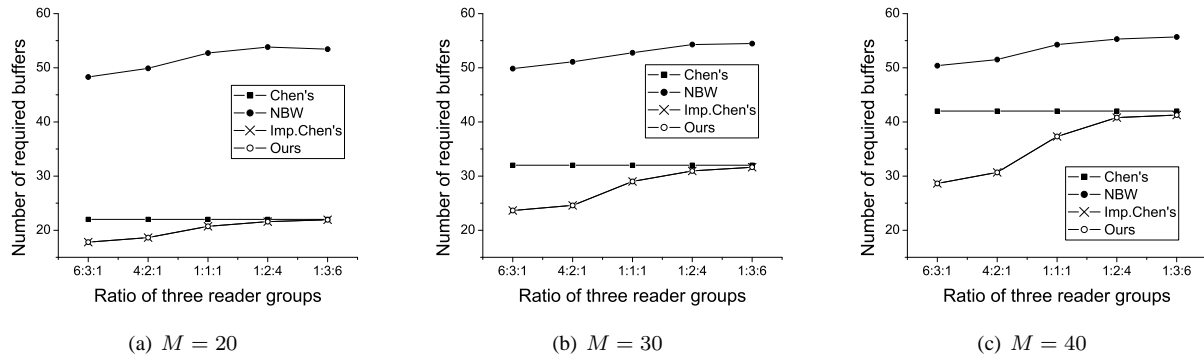


Figure 8. Buffer Sizes With 3 Reader Groups Under Varying Reader Ratio, for 20, 30, and 40 Readers

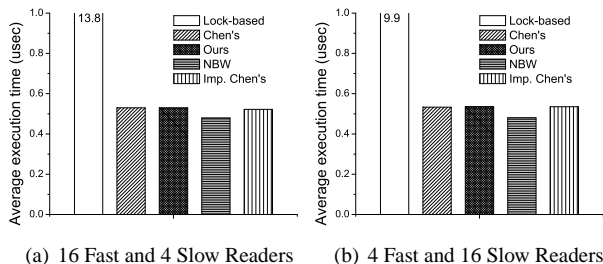


Figure 9. ACET of Read/Write in SHaRK RTOS

Thus, our implementation measurements verify our protocol's effectiveness in the time dimension—i.e., our protocol's time costs is comparable to that of previous best protocols including Chen's and Improved Chen's.

## 6. Conclusion

In this paper, we consider the single-writer/multiple-reader problem that occurs in embedded real-time systems. We develop a wait-free protocol for this problem, and establish that our protocol requires lesser (or equal) number of buffers than what is needed by previously best wait-free protocols including Chen's, NBW, and Improved Chen's protocols. Further, we prove that our protocol needs the absolute minimum number of buffers, and yet, it is safe and orderly — this is the first such bound established for wait-free protocols that consider a-priori knowledge of preemptions. Our evaluation studies and implementation measurements confirm the protocol's superiority and effectiveness.

Several aspects of the work are directions for further research. Examples include extending the protocol for the multiple-writer/multiple-reader problem, and complex concurrent objects such as (non-blocking) stacks and queues.

## 7. Acknowledgements

This work was sponsored by the US Office of Naval Research under Grant N00014-00-1-0549 and The MITRE Corporation under Grant 52917.

## References

- [1] J. H. Anderson, R. Jain, and S. Ramamurthy. Wait-free object-sharing schemes for real-time uniprocessors and multiprocessors. In *IEEE RTSS*, pages 111 – 122, Dec. 1997.
- [2] J. H. Anderson, S. Ramamurthy, and K. Jeffay. Real-time computing with lock-free shared objects. *ACM TOCS*, 15(2):134–165, 1997.
- [3] T. P. Baker. Stack-based scheduling of real-time processes. *Real-Time Systems*, 3(1):67–99, Mar. 1991.
- [4] J. Chen and A. Burns. A fully asynchronous reader/writer mechanism for multiprocessor real-time systems. Technical Report YCS-288, University of York, May 1997.
- [5] J. Chen and A. Burns. A three-slot asynchronous reader/writer mechanism for multiprocessor real-time systems. Technical Report YCS-186, University of York, 1997.
- [6] R. K. Clark. *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, 1990.
- [7] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time systems development. In *ECRTS*, pages 199–206, 2001.
- [8] H. Huang, P. Pillai, and K. G. Shin. Improving wait-free algorithms for interprocess communication in embedded real-time systems. In *USENIX Annual Technical Conference*, pages 303–316, 2002.
- [9] H. Kopetz and J. Reisinger. The non-blocking write protocol nbw: A solution to a real-time synchronisation problem. In *IEEE RTSS*, pages 131–137, 1993.
- [10] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.
- [11] H. Sundell and P. Tsigas. Space efficient wait-free buffer sharing in multiprocessor real-time systems based on timing information. In *IEEE RTCSA*, pages 433 – 440, 2000.