RBS: Profile-Guided Scheduling for Time-Triggered Applications

Robert Esswein
University of Pittsburgh
Pittsburgh, PA USA
robert.esswein@pitt.edu

Brendan Luksik
NASA Johnson Space Center
Houston, TX USA
brendan.k.luksik@nasa.gov

Andrew Loveless
NASA Johnson Space Center
Houston, TX USA
andrew.loveless@nasa.gov

Alan George
University of Pittsburgh
Pittsburgh, PA USA
alan.george@pitt.edu

2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC) | 978-1-6654-8607-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/DASC55683.2022.9925829

***Abstract*—Time-Triggered networks like Time-Triggered Ethernet, TTP/C, Flexray, and IEEE 802.1Qbv are used in critical embedded systems to provide reliable and predictable message delivery. These networks work by controlling the maximum size and exact rate at which applications can transmit messages. While this approach works well for applications that generate fixed-sized messages at constant rates, it leads to inefficient bandwidth utilization when message sizes and timing are highly variable. For example, to accommodate an application that runs *up to* X times per second, and generates *up to* Y bytes of data per execution, the network needs to be scheduled for the worst-case — a Y-byte message *every* $1/X$ seconds.**

We introduce Reduced Bandwidth Scheduling (RBS), a novel method for reducing the bandwidth allocated to highly-variable applications while still ensuring they meet reliability requirements. RBS models an application's communication pattern as a biased Bernoulli distribution, which we prove represents the worst-case message overhead. With this model, RBS can provide an upper bound on the probability that an application overflows its network buffers, and a lower bound on message reliability. Our evaluation shows that, for representative embedded applications, RBS can reduce bandwidth utilization by up to 329 kbps (43% of the original utilization) while requiring only 2000 bytes of extra buffer space. Moreover, in a realistic case study with a representative audio compression application, RBS reduced bandwidth utilization by 146 kbps (34% of the original utilization) while requiring only 41040 bytes of extra buffer space (20× more) and maintaining reliability greater than 0.999 over 15 years.

***Index Terms*—Real-time networks, real-time systems, safety-critical, TTE**

I. Introduction

Time-triggered architectures are becoming increasingly common in safety and mission-critical embedded systems like spacecraft, airplanes, and automobiles. For example, NASA's Orion and Gateway spacecraft, as well as ESA's Ariane 6 launcher, all use Time-Triggered Ethernet as their main data networks [1]–[5]. Several automobiles use FlexRay, a time-triggered data bus, for communication with braking and steering systems [6]. Additionally, industrial control systems and manufacturing plants are starting to use time-triggered architectures for controlling critical plant processes [7].

This research was supported by the NSF Center for Space, High-performance, and Resilient Computing (SHREC) industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783.

In time-triggered networks, devices (and their applications) are synchronized to a common clock, which may be distributed, and communicate by sending messages according to a global schedule [8], [9]. The schedule is developed offline before the system is deployed, and specifies *time windows* during which each application is allowed to send messages [10]. These time windows prevent applications from ever trying to access shared resources, like switch port buffers, at the same time — which otherwise might lead to dropped messages. However, for this approach to work, the schedule traditionally must be designed to accommodate the worst-case rate at which each application may run, as well as the maximum amount of data it may generate [11], [12].

Because time-triggered schedules are loaded on the system *before* it is deployed, they cannot typically be modified during operation to handle changing network requirements [13]. This means that if an application ever generates less data, or sends messages at a lower rate, than the worst case assumed in the network schedule, that unused bandwidth may be wasted [14]. Some time-triggered networks allow asynchronous traffic, such as standard Ethernet traffic generated by commercial off-the-shelf devices, to reclaim some of the lost bandwidth [15]. However, even in this case, the unused bandwidth cannot be used by other time-triggered applications [16]. This means that by provisioning bandwidth for the worst-case, designers directly decrease their ability to schedule time-triggered applications (and meet deadlines) successfully.

This paper introduces Reduced Bandwidth Scheduling (RBS), a principled method for reducing the bandwidth requirements of highly-variable time-triggered applications, while accepting a small (but bounded) probability of dropped messages. For applications that rarely perform according to their worst case (e.g., data compression and event-driven sensor data), the failure probability can be made extremely low (e.g., 0.001%).

The key idea of RBS is to measure the properties of an application offline and create a stochastic model that represents the application as a biased Bernoulli distribution. This model is then used to select the maximum bandwidth and buffer size allocated to the application when scheduling the network. In times of peak traffic load, when the output from the application exceeds that which can be accommodated by the network schedule, messages are stored locally until network bandwidth

becomes available. This design allows an application to be allocated less bandwidth than its worst-case requirements would normally dictate, while allowing it to make more efficient use of the allocated bandwidth.

With RBS, the cost of reducing bandwidth is an increase in memory usage. However, we consider this a worthwhile trade-off; in modern embedded systems, onboard memory is often much more available than network bandwidth [17], [18]. Also, as we will show in section VI, the extra memory required for significant bandwidth savings is often small.

To evaluate RBS, we used RBS to schedule a wide array of representative embedded applications on a real Time-Triggered Ethernet testbed. Our experimental evaluation shows that RBS can reduce bandwidth utilization by up to 329 kbps, while requiring only 2000 bytes of extra buffer space. Additionally, RBS can achieve these bandwidth savings while introducing only a small failure probability (e.g., 0.001). Our results show that RBS can reduce bandwidth needs by 80 kbps with only 8640 bytes on an audio compression application.

Overall, this paper makes the following contributions:

- RBS: a novel method for reducing the scheduled bandwidth of time-triggered applications while maintaining delivery guarantees.
- A detailed formal analysis of RBS's delivery guarantees with guidance on how to configure RBS to meet specific reliability targets.
- An evaluation of RBS on a real time-triggered network with several representative applications showing significantly reduced scheduled bandwidth.

II. Background

Time-triggered networks are desirable for critical embedded applications because they prevent dropped messages due to resource contention, as well as malicious or faulty devices from monopolizing the network bandwidth and preventing other devices from communicating [9], [15], [19]. To accomplish this goal, devices in time-triggered networks are synchronized to a global clock, and the behavior of the network is controlled by a global schedule [20]. This schedule is designed offline and pre-loaded onto each of the devices. Each device sends and receives messages only as permitted by the schedule. The schedule prevents devices from contending for network resources, like switch buffers, that might lead to dropped messages [16]. It also prevents malicious devices from communicating outside of their preassigned communication windows.

In order to create the schedule, the rate and maximum message sizes corresponding to all time-triggered applications traditionally must be known. If too much bandwidth is required by all of the applications, the scheduling process fails and no schedule is generated. When this happens, either the network must be redesigned to alleviate the bottleneck or the applications must be redesigned to use less bandwidth [10].

One major trade-off with time-triggered networks is that flexibility is sacrificed for reliability [13], [21]. Because the schedule is predefined, an application cannot use more bandwidth than it was allocated. Thus, to avoid message drops, applications must be allocated bandwidth according to their worst-case needs. This approach is acceptable for applications with consistent and predictable bandwidth requirements. However, for applications with unpredictable requirements, it leads to inefficient network utilization and schedulability problems [10], [22].

In RBS, we introduce a principled method for reducing the bandwidth allocated to time-triggered applications, while not exceeding a bounded probability of message drops.

III. Model

We assume that applications generate data periodically according to a synchronous clock. This assumption reflects the capabilities of typical embedded applications in spacecraft, aircraft, and energy generation plants, where tasks are executed according to a static cyclic schedule, which is synchronized to the time-triggered network [23]–[26]. We also assume that applications can buffer messages locally for a bounded amount of time before they are transmitted. This buffering could be performed either by the network interface card (NICs) or in host memory. Modern time-triggered NICs already have the ability to buffer hundreds of messages [27].

Applications are modelled according to a Bernoulli distribution, in which, at each execution, an application either generates no data or a predefined maximum amount of data. This distribution will be known as the biased Bernoulli distribution. As we will show in subsection IV-A, the biased Bernoulli distribution results in the application behavior that is most likely to cause buffer overflows. Let D be the *maximum* amount of data generated in one execution (the bias), and μ be the *mean* amount of data generated in one execution. Let p be the probability the application produces data in a given execution. Note that $\mu = Dp$. The bandwidth allocated to the application by RBS is a function of the mean: bandwidth $= b\mu$, where b is a scaling factor. The size of the buffer allocated to the application is buffer size $= S\mu$. In subsection IV-B, we describe how to calculate the parameters b and S in order to meet a certain reliability target.

IV. Design

In this section, we describe how RBS models applications and selects usable resource parameters to achieve a given reliability. RBS saves network bandwidth by buffering the variable data outputs of an application to fit within the bandwidth allocated to the application, which is potentially much smaller than what would be required to accommodate the worst-case scenario. Using a stochastic model of an application, an appropriately-sized buffer can be selected to ensure all data is transmitted over the reduced bandwidth reservation to a configurable certainty.

A. Application Model Creation

For an application to have its bandwidth reduced, a model for accurately estimating its output characteristics and applying

them to a reliability analysis is needed. This model should provide a conservative estimation in order to capture the widest range of application behaviors. The model should also be applicable to any application's traffic distribution. For these reasons, RBS models application traffic loads as following the biased Bernoulli distribution.

The biased Bernoulli is a two-state distribution modelling an application that outputs only either a volume of data D (the bias) with probability p, or 0 in a given execution. Any application whose output follows other statistical distributions can map to the biased Bernoulli if their mean output, μ, can be determined, and if they have a known worst-case output. This will allow us to map the application to a biased Bernoulli model where D is the worst-case output and $p = D/\mu$.

The biased Bernoulli distribution is only useful for RBS if it is the distribution that captures the greatest number of failure states for any given application. To help compare the number of states captured by different distributions, we can draw upon the Central Limit Theorem. In this case, each "failure state" is a sample trial of the application where a resource was overused. The Central Limit Theorem demonstrates that as the number of runs increases, the more the outcomes form a Normal distribution. Since this is the case no matter what distribution is used to model the application, all candidates can be compared by checking the number of trials in which some selected $S\mu$ is exceeded, which can be done by comparing statistical variances. We will show that the biased Bernoulli distribution naturally leads to the highest variance of any distribution — and thus is the most conservative model.

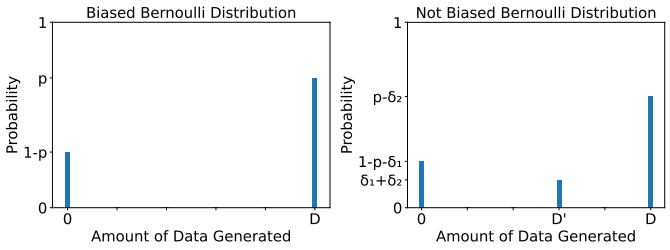

Fig. 1. Left: the probability distribution for an application following the biased Bernoulli distribution. Right: a probability distribution for an application that does not follow the biased Bernoulli distribution.

Lemma 1. *If two applications, one outputting data according to a biased Bernoulli distribution, and the other according to any other distribution, share a common worst-case output, D, and mean output, μ, then the biased Bernoulli application always has the higher variance.*

Proof. Consider two applications of common D and μ, but with different output distributions, a biased Bernoulli (App 1) and some other output distribution (App 2). An example is shown in Fig. 1. App 2 must have at least one additional output state between 0 and D, D', to not trivialize to App 1. Because of this extra state, some executions that would have been D or 0 for App 1 must now be D'; the difference in these probabilities is captured as δ_1 and δ_2, respectively. So, the probability of App 2 generating D data is $p - \delta_2$, the probability of App 2 generating no data is $1 - p - \delta_1$, and the probability of App 2 generating D' data is $\delta_1 + \delta_2$.

Now consider a case where

$$Var(\text{App 1}) \leq Var(\text{App 2})$$

which can be expressed equivalently as

$$\begin{aligned} p(D-\mu)^2 + (1-p)\mu^2 \leq & (p-\delta_2)(D-\mu)^2 \\ & + (\delta_1+\delta_2)(D'-\mu)^2 \\ & + (1-p-\delta_1)\mu^2 \end{aligned} \tag{1}$$

Now consider that for both applications, μ must be the weighted sum of its output states.

$$\mu = pD = (p-\delta_2)D + (\delta_1+\delta_2)D'$$

which lets us isolate δ_1,

$$\delta_1 = \frac{\delta_2(D-D')}{D'} \tag{2}$$

Since, by definition, $D > D'$, it follows that $0 < \delta_1 \leq (1-p)$ and $0 < \delta_2 < p$. Thus, substituting (2) into (1) gives

$$\delta_2((D-\mu)^2 - (D'-\mu)^2) \leq \frac{\delta_2(D-D')}{D'}((D'-\mu)^2 - \mu^2) \tag{3}$$

Which can be simplified to

$$D \leq D' \tag{4}$$

So, in order for $Var(\text{App 1}) \leq Var(\text{App 2})$, D' must be greater than D, which is a contradiction. Therefore, the distribution of data generated by an application following the biased Bernoulli distribution has a larger variance than any other distribution. □

Lemma 2. *For two Normal distributions $f_1(x)$ and $f_2(x)$ with means $\mu_1 = \mu_2 = \mu$ and variances $\sigma_1^2 > \sigma_2^2$ respectively, then for some arbitrary $(\alpha = S\mu) > \mu$, $P(f_1(x) > \alpha) > P(f_2(x) > \alpha)$.*

Proof. Consider a Normal distribution with mean μ and variance σ^2. Next, consider some $\alpha > \mu$. Since this is a normal distribution, calculating the probability that a sample taken from this distribution is larger than α is equivalent to calculating the probability that a sample taken from the standard Normal distribution is larger that $(\alpha - \mu)/\sigma$. The probability of a sample taken from the standard Normal distribution is equal to the area under the probability density function on the interval $((\alpha - \mu)/\sigma, \infty)$. Since this interval is larger for larger σ, increasing the variance of a Normal distribution increases the probability of a sample taken from the distribution being greater than $\alpha > \mu$. Thus, for two Normal distributions with the same mean, the distribution with

the larger variance is more likely to produce a sample greater than some $\alpha > \mu$. □

Theorem 1. *Messages sent according to the biased Bernoulli distribution result in the highest probability of buffer overflow on the sending device.*

Proof. Consider again the two applications from Lemma 1. Assume that both applications execute for some interval n times. By the Central Limit Theorem, the collection of n outcomes for both applications converge to Normal distributions. Lemma 1 implies that App 1 must have a greater variance than App 2 over these trials. Lemma 2 implies that because App 1 must have the higher variance, then for any value $\alpha > \mu$, App 1 statistically contains more states than App 2 that exceed that threshold. Because any state where the output is $> \alpha$ causes a buffer overflow, App 1 is always most likely to fail. □

B. Estimating the Probability of Failure

In the following section, we explain how the probability of failure is bounded from the input parameters b and S. To do this, we use the biased Bernoulli distribution to model the amount of data generated by the application. As discussed in subsection IV-A, any application that does not follow the biased Bernoulli distribution can be modelled as a biased Bernoulli distribution, with its mean μ and maximum D.

Before determining what the parameters b and S must be, the system designer must decide what the reliability of the application should be. Depending on the application, the reliability might need to be greater than 0.99 or 0.999. RBS provides a method for estimating the probability of failure at each execution of the application given that it has not failed before, which will be known as $f(t)$. The reliability of the application over its lifetime is equal to one minus the probability of failure at every execution in its lifetime. This can be found by summing $f(t)$ over every execution in its lifetime. RBS guarantees a known $f(t)$ based on S and b, so several iterations of adjusting these parameters and recalculating $f(t)$ might be necessary to reach a reliability requirement.

The application will fail, i.e., lose data, when the amount of data in the buffer exceeds the size of the buffer. This condition is met when the application outputs more data over some number of executions than the available bandwidth can transmit over the same time. In the worst case where D data is generated at every execution:

$$S\mu < Dt - b\mu(t-1)$$

This allows us to find the first execution, t_{min}, that a failure is possible:

$$t_{min} = \lfloor \frac{S\mu - b\mu}{D - b\mu} \rfloor + 1 \tag{5}$$

This does not apply to instances where even a single execution generates less than the maximum amount of data. In any other case, for some $t > t_{min}$, there exists a myriad of combinations of generating and non-generating executions, only some of which will ultimately satisfy the failure condition. To track these, we introduce a new concept: the *failure interval*, which is depicted in Fig. 2. The failure interval is defined as some number of consecutive executions, i, where the buffer usage begins at 0. The failure interval ends at execution t, where the buffer usage exceeds $S\mu$ for the first time. Further, we stipulate that the buffer usage can never be zero at some intermediate execution (if it was, a new failure interval would begin). The failure interval provides a means to count all ways that the failure condition can be met at some execution t.

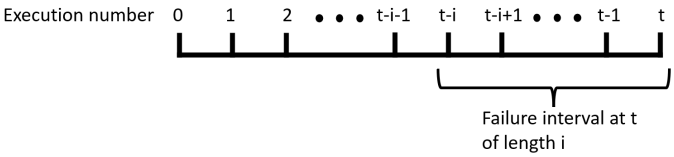

Fig. 2. Visualization of a failure interval. The buffer must be empty at the beginning of the failure interval.

Because all applications are modelled as biased Bernoulli generators, we can determine the output of some interval of length t, where actually only $n \leq t$ executions generated data, as nD. From that, we can compute how many executions within a failure interval must generate data in order to guarantee a failure at the end of the interval, n_t.

$$n_t = \lfloor \frac{S\mu}{D} + \frac{(t-1)b\mu}{D} \rfloor + 1 \tag{6}$$

This captures how much data must be generated in just one of many unique failure intervals ending at t. All these failure intervals can be examined to calculate $f(t)$. A failure interval of length i might have numerous different combinations of generating and non-generating (i.e., executions that generate D data or no data) executions, all of which result in a failure at the end of the interval. By considering the probability that any failure interval ending at execution t occurs, $f(t)$ can be calculated. Finding $f(t_{min})$ is simple because there can only be one failure interval to that point, the worst case, which is found by:

$$f(t_{min}) = p^{(t_{min})} \tag{7}$$

For $t > t_{min}$, multiple failure intervals are possible at each t.

Multiple executions of the application corresponds to multiple samples from the biased Bernoulli distribution. Therefore, the amount of data generated by the application follows the Binomial distribution, multiplied by the bias term, D. Naively, to calculate the probability of a failure interval of a particular length, we could calculate this probability according to a Binomial distribution:

$$f(t) = \sum_{i=t_{min}}^{t} p^{n_i}(1-p)^{i-n_i} \binom{i}{n_i} \tag{8}$$

However, this overcounts the number of failure intervals since it includes those that terminate prior to t and failure intervals

that have empty buffers part way through the interval. So, we introduce three terms to tailor the $f(t)$ more accurately: $C(t)$, P_0, and $f_{dec}(t)$.

First, we construct $C(t)$, which we use to eliminate unreachable failure intervals. We observe that since any useful configuration of RBS satisfies $b\mu < D$, by (6), n_t is at most $n_{t-1}+1$. In a vacuum, we can intuit this to mean that when a failure interval extends by one execution, it requires *at most* one additional generating execution to fail at that point.

However, a failure interval of length $t > t_{min}$ implies the existence of smaller failure intervals. Therefore, if we have previously counted the failure intervals of length $t-1$, then the failure intervals where $n_t = n_{t-1}+1$ cannot actually fail at t. This is because failure intervals must end on a generating execution, and a failure at t would mean that at execution $t-1$, there must be at least n_t-1 generating executions. This means that if $n_t = n_{t-1}+1$, then all combinations that would satisfy the failure interval of length t would also satisfy the failure interval of length $t-1$, making the former impossible to occur.

Now, we can construct a correction function, $C(t)$. As just mentioned, when $n_t = n_{t-1}+1$, t cannot have a failure interval, so $C(t) = 0$. But otherwise, we return to a binomial approximation of the true count, $C(t) = \binom{t}{n_t}$. This results in $C(t)$ being a piece-wise function where:

$$C(t) = \begin{cases} 0 & \text{if } n_t = n_{t-1}+1 \\ \binom{t}{n_t} & \text{if } n_t = n_{t-1} \end{cases} \tag{9}$$

This counts many cases that would cause a failure before t. We accept this inaccuracy because the result is that the estimate for the probability of failure is higher than it actually is. This representation also counts many cases in which the buffer empties completely during the failure interval, which is a case captured by a smaller failure interval. Again, this results in an overestimate of the probability of failure, so this is acceptable.

To provide some insight into this equation, we take the derivative of (6):

$$\frac{dn_t}{dt} = \frac{b\mu}{D} = bp \tag{10}$$

This value represents the frequency in which $n_t = n_{t-1}+1$, which is the proportion of executions in which $C(t) = 0$. When $C(t) = 0$, $f(t)$ decreases from $f(t-1)$ because there are no new failure intervals. Because the probability of an empty buffer at any execution after the first execution is less than 1, the probability of a failure interval of length $t-1$ ending at execution t is less than the probability of a failure interval of length $t-1$ ending at execution $t-1$.

Next, we construct a method to estimate the probability that the buffer is empty at a particular execution. This is necessary because failure intervals always begin with an empty buffer. Let the probability of the buffer being empty at any given execution be P_0. At execution $t = 0$, $P_0 = 1$, since the buffer starts empty by definition. Then, at $t = 1$, $P_0 = 1-p$, and so on, following the fact that biased the Bernoulli model either adds D or 0 to the buffer at every t. As t becomes large, $P_0(t)$ converges to a constant which we can apply uniformly, found by:

$$P_0 = \lim_{t\to\inf} \frac{tb\mu - tpD}{tb\mu} = 1 - \frac{1}{b} \tag{11}$$

This probability is valid in the range $1 < b < \frac{1}{p}$, which is acceptable for RBS. Selecting some $b \leq 1$ means that the mean bandwidth is the allotted bandwidth, removing any margin for the application to avoid failing. And selecting $b \geq \frac{1}{p}$ implies that application has been provisioned with enough bandwidth to handle its worst-case, so RBS is not needed. Thus, when RBS is useful, b will always be in this range.

Finally, we construct $f_{dec}(t)$ to count failure intervals of length $t_{min} \leq i < t$. Because the probability of failure at execution t is equal to the probability of all failure intervals of length $i \leq t$ occurring, $f_{dec}(t)$ can be defined as a function of $f(t-1)$. Since $f(t-1)$ has a failure interval starting at the first execution, this failure interval is not multiplied by P_0. However, to calculate $f(t)$, the failure interval of length $t-1$ no longer starts at the first execution. Therefore, the probability of a failure interval of length $t-1$ times the probability of the buffer not being empty must be subtracted out of $f(t-1)$ in order to get $f_{dec}(t)$. Thus:

$$f_{dec}(t) = f(t-1) - (1-P_0)p^{n_{t-1}}(1-p)^{t-1-n_{t-1}}C(t-1) \tag{12}$$

The intuition for $f_{dec}(t)$ reasons that at every t we've computed $f(t-1)$ and can use P_0 to avoid counting those intervals where the buffer reached back to 0. However, the final failure interval of length $t-1$ is not weighed against P_0, counting extra probability for that interval. $f_{dec}(t)$ is just a term that removes the cases where the buffer was *not* empty from the final failure interval of length $t-1$.

Now, the probability of failure at any execution t can be approximated with the following equation by adding a new term to (12):

$$f(t) = f_{dec}(t) + p^{n_t}(1-p)^{t-n_t}C(t) \tag{13}$$

The new term is the probability of getting the exact number of data generating executions to cause a failure at t, which converges to 0. Therefore, this equation converges to some value > 0. Since $f(t)$ converges, $f_{dec}(t)$ also converges.

The reliability of an application over its lifetime is equal to one minus the probability of failure before the end of its life. Because $f(t)$ represents the conditional probability that the application fails on execution t given that it has not failed before t, this equation becomes:

$$R = 1 - \sum_{i=0}^{t} f(i) \tag{14}$$

Based on how long a mission is expected to take, the total number of executions that an application will perform can be calculated. To simplify calculations, $f(t)$ will be taken as a constant term at the converging value. This simplification

underestimates the actual reliability because $f(t)$ approaches the convergent value from below. Using (14), the reliability of the application over the entire interval during which it is expected to operate can be bounded to be above a certain threshold.

C. Real-Time Characteristics

Because data using RBS is buffered, the real-time guarantees provided by time-triggered networks do not apply in the same way. However, given the guarantees of the network, a calculation for maximum delay can be developed. To start, some characteristics of time-triggered traffic must be defined. Let f be the frequency at which messages for a particular application are sent, l be the maximum message size, and d be the maximum transmission delay. All of these terms are set or calculated at design time, so the calculation for maximum data delay can also be done at design time. Trivially, we know that the greatest delay will occur when the buffer is full. Therefore, the maximum data delay is given by:

$$\Delta T = \frac{1}{f}\lceil\frac{S\mu}{l}\rceil + d \tag{15}$$

V. Implementation

To evaluate RBS, we used a real testbed consisting of a Time-Triggered Ethernet (TTE) network switch and two TTE network interface cards from TTTech Computertechnik. The network cards were connected to Dell PCs with 4-core Intel i3-450 processors running Ubuntu 14.04 LTS with kernel 3.13.0-36-generic x86 64. All physical links were 100Base-TX. The network equipment was configured to support a single time-triggered data stream whose bandwidth characteristics were manipulated for each experiment.

On this testbed, we implemented both a generic traffic-generator and an application that transmits compressed audio. The traffic generator runs according to a cyclic executive, periodically producing data according to a Biased Bernoulli distribution. We refer to this as the Bernoulli application, which can be parameterized by its bias, D, and the probability of producing the bias, p. The compressed audio application uses the WavPack compression algorithm [28], which is commonly used in real-time audio transmission [29].

VI. Evaluation

In this section, we answer the following questions:

1) How effective is RBS at reducing bandwidth utilization?
2) How reliable are applications scheduled with RBS?
3) How well does RBS perform when used with a real audio streaming application?

A. Bandwidth Utilization

Experimental Setup. In this experiment, we examined how much RBS reduces bandwidth utilization. We examined the impact of changing three parameters of an application: the amount of data generated, target reliability, and frequency F. The cases tested were $max(f(t)) = \{10^{-6}, 10^{-10}\}$ while $F = 40$ Hz and $D = 500$ bytes, $F = \{5, 200\}$ Hz while $max(f(t)) = 10^{-10}$ and $D = 500$ bytes, and $D = \{250, 1000\}$ bytes while $max(f(t)) = 10^{-10}$ and $F = 40$ Hz. These values are based on those found in typical embedded applications [23], [26], [30], [31].

Results. Our results are shown in Fig. 3. Bandwidth is shown on the y-axis and buffer length on the x-axis, with several different graphs for different values of p. Each case could apply to many different applications, so long as they have the same D and μ parameters.

The results show that RBS can significantly reduce bandwidth utilization. For example, in the case with $p = 0.1$, $F = 200$ Hz, $max(f(t)) = 10^{-10}$, and $D = 500$ bytes, bandwidth is reduced by 660 kbps (82.5%) with a buffer length of 9900 bytes (19.8 times what is required with worst-case bandwidth). Similarly, in the case with $p = 0.1$, $F = 40$ Hz, $max(f(t)) = 10^-6$, and $D = 500$ bytes, bandwidth is reduced by 108 kbps (67.5%) with a buffer length of 2535 bytes (5.07 times what is required with worst-case bandwidth).

Briefly, we also note that there is a diminishing return on increasing buffer length. In other words, increasing the buffer length by more than $\times 20$ the the length of the buffer needed with the maximum bandwidth does not significantly decrease the bandwidth utilization.

B. Reliability

Experimental Setup. To examine the impact RBS has on reliability, we performed two experiments on the same Bernoulli application running at 100 Hz. In each experiment, we ran the application until failure occurred (i.e., data was dropped). Both experiments used $D = 2048$ bytes and $p = 0.5$. μ was determined by measuring 100 executions of the application offline. From this profile, we found $\mu = 1004$. In the first experiment, the target probability of failure was 10^{-9} and we set the bandwidth to 988 kbps, which resulted in a buffer length of 46148 bytes. In the second experiment, the target probability of failure was 10^{-3} and we set the bandwidth to 924 kbps, which resulted in a buffer length of 23092 bytes. In the first experiment, because the reliability was so high, this test was not feasible to run in real time. Instead, we simulated it at $\times 100000$ speed in software. The second experiment was executed it on real hardware.

In both experiments, we ran the applications until they failed. We repeated this process (10 times for high probability of failure, 100 times for low probability of failure) in order to approximate the Mean-Time-To-Failure (MTTF) of the application. We then compared the measured MTTF against the expected MTTF, which was determined using the probability of failure found from the model.

Results. Our results are shown in Fig. 4 and Fig. 5. Fig. 4 depicts the time to failure of 100 trials of experiment one. Using (13), the MTTF for this application was calculated to be 1.68×10^9 executions. On average, this setup experienced a time to failure of 1.18×10^{10} executions, an order of magnitude better. Similarly, experiment two's setup was reliable beyond the bounds of the model. RBS predicted at least 744 executions before failure. Over 10 trials, the MTTF was $11,200$

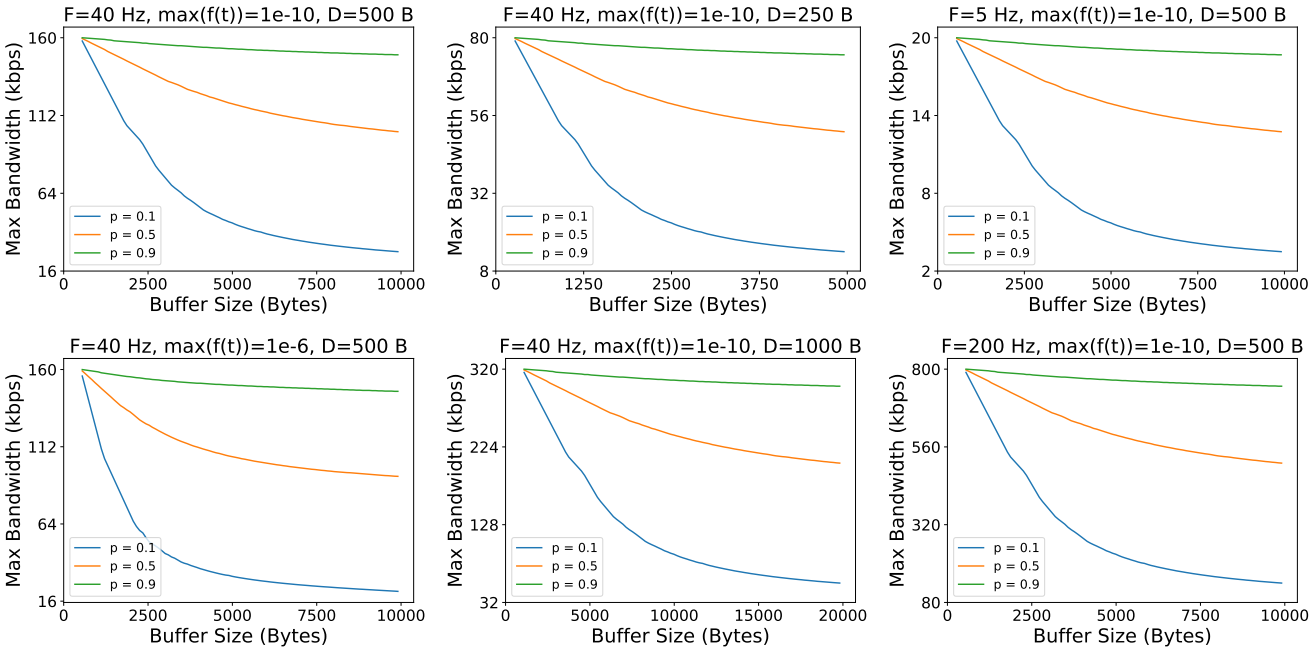

Fig. 3. Different parameters and their effects on the required bandwidth and buffer length to achieve less than a predefined probability of failure.

executions, with no exceptions below the predicted minimum. Fig. 5 depicts the trial which failed earliest, showing that buffer usage at each cycle was safely within limits until well beyond the MTTF.

Fig. 4. Execution time before first failure for experiment one in subsection VI-B (target probability of failure: 10^{-9}).

C. Case Study: Audio Compression

Experimental Setup. We used a compressed audio application to test how RBS performs on real software. The application sent audio data over the network using the WavPack compression algorithm [28]. This compression algorithm was used because it allows for compressing small blocks of audio data, making it a common choice for real-time communication [29]. The data used was an audio recording of Wikipedia's article on Parallel Computing [32], which was selected because of its length (54 minutes, 15 seconds) and because speech is a common example of real-time data in embedded systems [33].

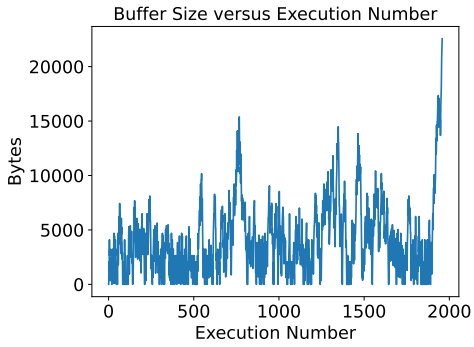

Fig. 5. The buffer usage at each execution for experiment two in subsection VI-B (target probability of failure: 10^{-3}).

The sample rate used by this recording was 44.1 kHz, a standard sampling frequency [34]. The application sent compressed data at 25 Hz, meaning each transmission contained 1764 samples. Throughout the entire audio recording, there were a total of 81375 executions, from which we used the first 100 to estimate the parameters of the application.

We measured the mean amount of data per execution to be $\mu = 1052.92$ bytes and the maximum amount of data to be $D = 2160$ bytes. These measurements were used as the parameters for the application's model, resulting in $p = 0.487$. The worst-case bandwidth for compressed audio data was calculated to be 433 kbps.

We targeted a reliability of at least 0.999 over 15 years, which matches the reliability requirements of satellite components [35]. Two cases were tested. The first used a buffer

length 5 times the maximum amount of data that could be generated, or 10800 bytes. The second used a buffer length 20 times the maximum amount of data that could be generated, or 43200 bytes.

Results. In order to achieve a reliability of 0.999, $f(t)$ at each execution must be less than $(1-R)/(\text{total executions}) = 8.45 \times 10^{-14}$. For the first case with a buffer of 10800 bytes, the bandwidth allocated to the application was 387 kbps, 89.4% of the worst-case bandwidth. For the second case with a buffer of 43200 bytes, the bandwidth allocated to the application was 287 kbps, 66.3% of the worst-case bandwidth. For the first application, the maximum buffer usage was 4204 bytes or 38.9% of the maximum buffer length. For the second application, the maximum buffer usage was 19822 bytes or 45.9% of the maximum buffer length. These results show that neither application came close to dropping data. The graphs are shown in Fig. 6.

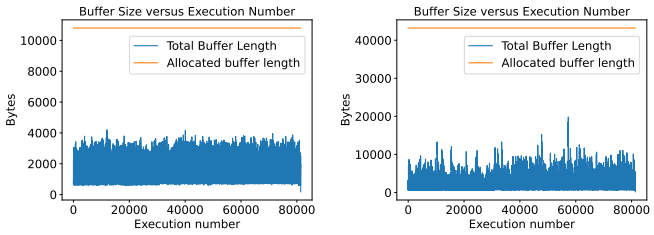

Fig. 6. Buffer length at each execution, Left: buffer length = 10800 bytes, Right: buffer length = 43200 bytes.

VII. Related Work

Improving Network Performance Many works focus on improving the performance of real-time networks. Often, these works introduce ways to reduce the worst case traversal times of messages [36], [37], or increase the resilience of message routing [38], [39]. For example, many software-defined networks use separate output queues to minimize worst-case queuing delays at switch egress ports [40]. In general, RBS is orthogonal to these works — instead attempting to reduce bandwidth utilization. Some works have similar goals, but require coding to split messages into fragments, and redundant paths to forward fragments through the network. In contrast, RBS works without requiring modifications to the applications, and works even in non-redundant networks.

Quality Aware Frame Skipping. One solution for transmitting data over resource-constrained channels is quality aware frame skipping (QAFS), where some data is purposefully not sent during times of high utilization to prevent network congestion [41], [42]. QAFS was originally developed for streaming video in asynchronous networks, but the same principle could be used in time-triggered networks as well. In contrast to QAFS, does not require frames to be periodically skipped, and provided a calculable bound on the probability that frames are dropped. RBS also allows designers to save significantly more bandwidth (e.g., $> 20\%$) than what can be achieved with QAFS, without significantly degrading quality.

Profile-Guided Optimization. A common method for improving performance is to use profile-guided optimization, in which traces of real executions of a system are processed offline and used to inform the behavior of the system at runtime [43]–[45]. One way to use this technique in time-triggered networks would be to measure the bandwidth requirements of an application, and directly use it to determine how much bandwidth to allocate for the application in the network. RBS takes a more principled approach, in which a representative model is created from the profile. This model is used to select the amount of bandwidth and the buffer space required to reach a reliability target.

VIII. Conclusion

This paper presented RBS, a method for reducing the amount of bandwidth allocated to time-triggered applications. RBS leverages the observation that many applications have a worst-case that is both rare and significantly worse than the average case. RBS exploits this observation by buffering data that otherwise would far exceed the allocated network bandwidth, and allowing this data to be transmitted over a longer period of time. Our experiments show that RBS is effective and efficient across a broad range of representative embedded applications. Moreover, RBS reduced the bandwidth required by a real audio compression application by 34%, while achieving a reliability greater than 0.999 over 15 years.

References

[1] Kerry Timmons, Kathleen Coderre, William D. Pratt, et al. “The Orion spacecraft as a key element in a deep space gateway”. In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–12. DOI: 10.1109/AERO.2018.8396769.

[2] Paul Muri, Svetlana Hanson, and Martin Sonnier. “Gateway Avionics Concept of Operations for Command and Data Handling Architecture”. In: *2021 IEEE Aerospace Conference (50100)*. 2021, pp. 1–7. DOI: 10.1109/AERO50100.2021.9438539.

[3] Nahman Tariq, Ivan Petrunin, and Saba Al-Rubaye. “Analysis of Synchronization in Distributed Avionics Systems Based on Time-Triggered Ethernet”. In: *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. 2021, pp. 1–8. DOI: 10.1109/DASC52595.2021.9594327.

[4] Andrew Loveless. *On Time-Triggered Ethernet in NASA’s Lunar Gateway*. July 2020.

[5] Raphaël Polonowski and Rémi Clavier. “Ariane Launchers Digital Engineering: stakes and challenges”. In: *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. 2019, pp. 1–5. DOI: 10.1109/MECO.2019.8760090.

[6] Jae-Sung Yang, Suk Lee, Kyung Chang Lee, et al. "Design of FlexRay-CAN gateway using node mapping method for in-vehicle networking systems". In: *2011 11th International Conference on Control, Automation and Systems*. 2011, pp. 146–148.

[7] Stefan Fuchs, Hans-Peter Schmidt, and Stefan Witte. "Test and on-line monitoring of real-time Ethernet with mixed pysical layer for Industry 4.0". In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2016, pp. 1–4. DOI: 10.1109/ETFA.2016.7733518.

[8] Astrit Ademaj and Hermann Kopetz. "Time-Triggered Ethernet and IEEE 1588 Clock Synchronization". In: *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. 2007, pp. 41–43. DOI: 10.1109/ISPCS.2007.4383771.

[9] *FlexRay Communication System Electrical Physical Layer Specification*. Rev 3.0.1. FlexRay Consortium. 2010.

[10] Wilfried Steiner. "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks". In: *2010 31st IEEE Real-Time Systems Symposium*. 2010, pp. 375–384. DOI: 10.1109/RTSS.2010.25.

[11] Ching-Chih Han, K.G. Shin, and Chao-Ju Hou. "Synchronous bandwidth allocation for real-time communications with the timed-token MAC protocol". In: *IEEE Transactions on Computers* 50.5 (2001), pp. 414–431. DOI: 10.1109/12.926157.

[12] Insik Shin and Insup Lee. "Periodic resource model for compositional real-time guarantees". In: *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*. 2003, pp. 2–13. DOI: 10.1109/REAL.2003.1253249.

[13] Zonghui Li, Hai Wan, Zaiyu Pang, et al. "An enhanced reconfiguration for deterministic transmission in time-triggered networks". In: *IEEE/ACM Transactions on Networking* 27.3 (2019), pp. 1124–1137.

[14] H. Kopetz and G. Grunsteidl. "TTP - A time-triggered protocol for fault-tolerant real-time systems". In: *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*. 1993, pp. 524–533. DOI: 10.1109/FTCS.1993.627355.

[15] H. Kopetz, A. Ademaj, P. Grillinger, et al. "The time-triggered Ethernet (TTE) design". In: *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*. 2005, pp. 22–33. DOI: 10.1109/ISORC.2005.56.

[16] K. Nikishin and N. Konnov. "Schedule Time-Triggered Ethernet". In: *2020 International Conference on Engineering Management of Communication and Technology (EMCTECH)*. 2020, pp. 1–5. DOI: 10.1109/EMCTECH49634.2020.9261540.

[17] Roberto Giorgi. "Scalable Embedded Systems: Towards the Convergence of High-Performance and Embedded Computing". In: *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*. 2015, pp. 148–153. DOI: 10.1109/EUC.2015.34.

[18] Rui Wang and Wenming Cao. "Universal Information Coverage for Bandwidth-Constrained Sensor Networks". In: *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2007, pp. 904–907. DOI: 10.1109/ROBIO.2007.4522283.

[19] Hermann Kopetz. "The Rationale for Time-Triggered Ethernet". In: *2008 Real-Time Systems Symposium*. 2008, pp. 3–11. DOI: 10.1109/RTSS.2008.33.

[20] *Time-Triggered Ethernet*. Society of Automotive Engineers International. 2011.

[21] Brendan Luksik, Andrew Loveless, and Alan D. George. "Gatekeeper: A Reliable Reconfiguration Protocol for Real-Time Ethernet Systems". In: *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. 2021, pp. 1–10. DOI: 10.1109/DASC52595.2021.9594416.

[22] Karsten Albers and Frank Slomka. "Efficient Feasibility Analysis for Real-Time Systems with EDF Scheduling". In: *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*. DATE '05. USA: IEEE Computer Society, 2005, pp. 492–497. ISBN: 0769522882. DOI: 10.1109/DATE.2005.128. URL: https://doi-org.pitt.idm.oclc.org/10.1109/DATE.2005.128.

[23] Antony Gillette, Brendan O'Connor, Christopher Wilson, et al. "Spacecraft mission agent for autonomous robust task execution". In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–8. DOI: 10.1109/AERO.2018.8396796.

[24] Thomas Henties, James J Hunt, Doug Locke, et al. "Java for safety-critical applications". In: *2nd international workshop on the certification of safety-critical software controlled systems (SafeCert 2009)*. Citeseer. 2009.

[25] Stuart R. Faulk and David L. Parnas. "On Synchronization in Hard-Real-Time Systems". In: *Commun. ACM* 31.3 (Mar. 1988), pp. 274–287. ISSN: 0001-0782. DOI: 10.1145/42392.42397. URL: https://doi-org.pitt.idm.oclc.org/10.1145/42392.42397.

[26] J.R. Ellis and S.A. Von Edwins. "Controlling large cyclic avionics software systems written in Ada". In: *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference*. 1988, 727–731 vol.2. DOI: 10.1109/NAECON.1988.195087.

[27] Michael Birner, Reinhard Exel, Stefan Moedlhamer, et al. *TTE-Controller ASIC Interface Control Document*. TTTech Computertechnik. July 2017.

[28] David Bryant. *The WavPack Codec*. URL: https://www.wavpack.org/.

[29] Zefir Kurtisi and Lars Wolf. "Using wavpack for real-time audio coding in interactive applications". In: *2008 IEEE International Conference on Multimedia and Expo*. 2008, pp. 1381–1384. DOI: 10.1109/ICME.2008.4607701.

[30] Ting Zhang, Chunxiu Xu, Muqing Wu, et al. "Implementation of Ad Hoc Network management system based on embedded ARM-Linux platform". In: *2010 International Conference on Networking and Digital Society*. Vol. 1. 2010, pp. 167–170. DOI: 10.1109/ICNDS.2010.5479602.

[31] Jeff Brown, Bill Shipman, and Ron Vetter. "SMS: The Short Message Service". In: *Computer* 40.12 (2007), pp. 106–110. DOI: 10.1109/MC.2007.440.

[32] Mangst. *Spoken Article: Parallel Computing*. Aug. 2013. URL: https://en.wikipedia.org/wiki/File:En-Parallel_computing.ogg.

[33] N. Nukaga, R. Kamoshida, K. Nagamatsu, et al. "Scalable Implementation Of Unit Selection Based Text-To-Speech System For Embedded Solutions". In: *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*. Vol. 1. 2006, pp. I–I. DOI: 10.1109/ICASSP.2006.1660154.

[34] P.M. Lane, R. Van Dommelen, and M. Cada. "Compact disc players in the laboratory: experiments in optical storage, error correction, and optical fiber communication". In: *IEEE Transactions on Education* 44.1 (2001), pp. 47–60. DOI: 10.1109/13.912710.

[35] Jean-Francois Castet and Joseph H. Saleh. "Satellite and satellite subsystems reliability: Statistical data analysis and modeling". In: *Reliability Engineering and System Safety* 94.11 (2009), pp. 1718–1728. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2009.05.004. URL: https://www.sciencedirect.com/science/article/pii/S0951832009001094.

[36] Fabien Geyer, Alexander Scheffler, and Steffen Bondorf. "Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis". In: *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2021, pp. 157–170. DOI: 10.1109/RTAS52030.2021.00021.

[37] *Aircraft Data Network Part 7: Avionics Full-Duplex Swithced Ethernet Network-ARINC Specification 664 P7-1*. ARINC. 2009.

[38] Minh Bui, Brigitte Jaumard, and Chris Develder. "Anycast end-to-end resilience for cloud services over virtual optical networks". In: *2013 15th International Conference on Transparent Optical Networks (ICTON)*. 2013, pp. 1–7. DOI: 10.1109/ICTON.2013.6603032.

[39] Rutvij H. Jhaveri, Sagar V. Ramani, Gautam Srivastava, et al. "Fault-Resilience for Bandwidth Management in Industrial Software-Defined Networks". In: *IEEE Transactions on Network Science and Engineering* 8.4 (2021), pp. 3129–3139. DOI: 10.1109/TNSE.2021.3104499.

[40] Rakesh Kumar, Monowar Hasan, Smruti Padhy, et al. "End-to-End Network Delay Guarantees for Real-Time Systems Using SDN". In: *2017 IEEE Real-Time Systems Symposium (RTSS)*. 2017, pp. 231–242. DOI: 10.1109/RTSS.2017.00029.

[41] D. Isovic and G. Fohler. "Quality aware MPEG-2 stream adaptation in resource constrained systems". In: *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.* 2004, pp. 23–32. DOI: 10.1109/EMRTS.2004.1310994.

[42] Anand Kotra and Gerhard Fohler. "Resource aware real-time stream adaptation for MPEG-2 transport streams in constrained bandwidth networks". In: *2010 IEEE International Conference on Multimedia and Expo*. 2010, pp. 729–730. DOI: 10.1109/ICME.2010.5583196.

[43] Lu Ding, Adrian Lizarraga, Susan Lysecky, et al. "Accuracy-Guided Runtime Adaptive Profiling Optimization of Wireless Sensor Networks". In: *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. 2013, pp. 82–91. DOI: 10.1109/ECBS.2013.22.

[44] Andrei Homescu, Steven Neisius, Per Larsen, et al. "Profile-guided automated software diversity". In: *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2013, pp. 1–11. DOI: 10.1109/CGO.2013.6494997.

[45] R. Gupta, D.A. Berson, and J.Z. Fang. "Path profile guided partial redundancy elimination using speculation". In: *Proceedings of the 1998 International Conference on Computer Languages (Cat. No.98CB36225)*. 1998, pp. 230–239. DOI: 10.1109/ICCL.1998.674173.