On Introducing Noise into the Bus-Contention Channel

James W. Gray, III
Information Technology Division
Naval Research Lab, Washington, DC

Abstract

We explore two approaches to introducing noise into the bus-contention channel: an existing approach called *fuzzy time*, and a novel approach called *probabilistic partitioning*. We compare the two approaches in terms of the impact on covert channel capacity, the impact on performance, the amount of random data needed, and their suitability for various applications. For probabilistic partitioning, we obtain a precise tradeoff between covert channel capacity and performance.

1 Introduction

In [7], Hu describes the *bus-contention channel*, a covert channel that can be exploited at a rate exceeding 1000 bits per second. This covert channel arises in the architecture shown in Figure 1.

In this architecture, multiple processors (which may be processing data at different security levels) access a shared memory bank over a shared bus.[1] When there is contention for the use of the bus (i.e., more than one processor is attempting to use the bus), bus requests are serviced at a slower rate. Since processors can determine (with some degree of accuracy) the rate at which their bus requests have been serviced, it is possible for one processor (e.g., a processor executing a high process) to send data to another processor (e.g., a processor executing a low process) in the following way.

Covert Channel Exploitation Scenario: During each millisecond interval, the high process sends a 1 by flooding the bus with requests, or a 0 by generating no bus requests. The low process generates a constant stream of bus requests and, during each millisecond interval, measures the rate at which its requests are serviced; if the rate is slower than normal, it records this as a 1; if the rate is normal, it records this as a 0. Thus, high data can be sent to the low process at 1000 bits per second. □

Note that throughout this paper, we will make the worst-case assumption that all other processors are not accessing the bus; i.e., there is no background noise that may slow down the exploitation of the bus-contention channel. We believe that making such worst-case assumptions (rather than, say, average-case assumptions) is appropriate in the context of covert channel analysis.

Also note that the threat that we are concerned with is *not* that the users (i.e., the *human* users) of the high processor are attempting to send information to low users. For if they wanted to, they could more easily pass notes in the park and entirely bypass the computer system. Rather, we are concerned that the high processor is executing a Trojan horse (i.e., a program that appears to be something that the users want, but actually contains something else that is entirely undesirable) and that the *Trojan horse* is attempting to send high information to the low processor. This is a legitimate concern since Trojan horses may be introduced into the system by a virus or may even be contained in legitimately installed off-the-shelf software.

In this paper, we explore two approaches to reducing the rate at which the bus-contention channel can be exploited. The first approach, called *fuzzy time* is described by Hu [7]. The second is a novel approach, which we call *probabilistic partitioning*. The remainder of this paper is organized as follows. In Sections 2 and 3, we explore fuzzy time and probabilistic partitioning, respectively. In Section 4, we give some concluding remarks.

2 Fuzzy Time

Hu's approach to reducing the bus-contention channel is based on the observation that exploitation of this channel (or any covert timing channel) requires a reference clock. That is, the low process receives signals over the channel by measuring the rate of its bus request completions *with respect to some reference clock*. Given this, the basic approach of fuzzy time is to make certain that the low process does not have access to an accurate reference clock. There are two parts to the implementation of this approach.

1. The security kernel intercepts and buffers all events that may be used as a reference clock. As described in [7], such events include the timer interrupt, I/O completion interrupts, the arrival of data under the control of Direct Memory Access (DMA) hardware, etc.. Typically, these events include all those that are generated by an asynchronous controller (e.g., a disk controller, terminal controller, or DMA controller).
2. The security kernel delivers all of these various events to the receiving processes on the next tick of

[1] We are assuming that access controls are in place that prevent processors from directly reading data for which they are not cleared. Such access controls are straightforward to implement (see e.g., [3]). Thus, even though the two processors share the same physical memory bank, it would not be possible, e.g., for a low processor to directly read high data.

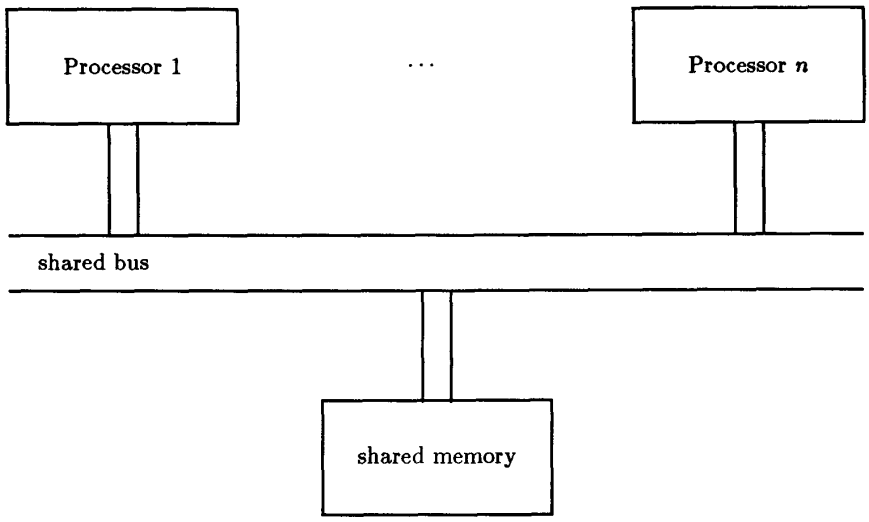

Figure 1: A Typical Multiprocessor Architecture

a fuzzy clock (i.e., a clock with a random clock-tick interval) that is maintained by the security kernel. That is, on each of the fuzzy clock ticks, the security kernel delivers all events (interrupts, data, etc.) that have occurred since the last fuzzy clock tick.

In this way, the fuzzy clock becomes the only (and hence, the most accurate) available reference clock for use in a timing channel exploitation.[2]

2.1 The Effectiveness of Fuzzy Time

The effect of fuzzy time on reducing the bus-contention channel is two-fold. First, the reference clock that is available to the low (receiving) process is slower and less accurate. For example, in the system described by Hu—the VAX security kernel[3]—the fuzzy clock's interval has a mean of 20 milliseconds, as opposed to the one millisecond clock interval needed for the covert channel exploitation scenario described in Section 1. Since the low process receives signals over the covert channel at the rate of its reference clock, the effect of this is that the low process receives fewer signals per second. In the case of the VAX security kernel under fuzzy time, the low process receives 50 signals per second, as opposed to 1000 (or more) signals per second without fuzzy time. Although the signalling processes can try to make up for this reduction in signals per second by enlarging the signalling alphabet, as noted by Hu, "increasing the alphabet size can only increase the bandwidth logarithmically, while reducing the clock rate decreases the bandwidth linearly. Furthermore, in [Hu's] experience, randomization makes most exploitations utilizing large alphabet sizes impractical."[7, §5.5.2] Thus, slowing the reference clock rate results in a significant reduction in the bus-contention channel.

The second effect of fuzzy time is that it impedes (i.e., it slows down) all synchronization between high and low processes. This is due to the fact that each processor has its own, independent, fuzzy clock. In particular, this means that the high process has no way of determining when the low process receives fuzzy clock ticks; and therefore, the high process has no way of knowing when to start and stop sending its signals. Consider, if the high process had access to the low process' fuzzy clock, then it could synchronize its transmission with it (i.e., with the times when the low process receives its signals), thus transmitting one signal per fuzzy clock tick. However, since the high process does *not* have this information, it cannot precisely synchronize with the low fuzzy clock. We therefore expect that the high process must signal at a much slower rate.

2.2 Analyzing Channel Capacity

Hu claims that after implementing fuzzy time, the "... bandwidth [of the bus-contention channel] is less than ten bits per second ..."[7, §7]. This claim is supported by "actual measurements with exploitations"[7, §8]. It would provide a great deal more assurance if these measurements were corroborated by a rigorous mathemat-

[2] Actually, this discussion is slightly simplified from Hu's work. To address all types of clocks, Hu introduces "upticks" (which are used for notification of incoming events) and "downticks" (which are used for notification of outgoing events). However, for the present discussion it is sufficient to consider a single fuzzy reference clock.

[3] VAX is a trademark of Digital Equipment Corporation.

ical analysis of the channel. In particular, we would like to know (in information-theoretic terms) the *capacity* of the bus-contention channel under fuzzy time. In the communication theory community, it has long been believed that the capacity of a given channel is the appropriate measure of the rate at which information can be transmitted over that channel. Unfortunately, Hu does not provide an analysis of the capacity. In private correspondence, Hu stated that this was because they "believed that the problem is intractable". In our own attempts to analyze fuzzy time, we came to the same conclusion. To understand why this is so, let's develop the expression for capacity that is appropriate for the bus-contention channel under fuzzy time.

First, let's precisely describe the inputs and outputs of the channel. We will assume that the low process issues a constant stream of bus requests and counts the number of its requests that complete during each fuzzy clock interval.[4] Also, on each fuzzy clock tick, the low process' system clock register is updated to reflect the (kernel maintained) system time, rounded to the nearest tenth of a second [7, §5.1]; that is, the security kernel maintains the system time to an accuracy of one millisecond (i.e., one thousandth of a second), but for the purpose of fuzzy time, supplies the system time to the processes only on each fuzzy clock tick and rounded to the nearest tenth of a second. Given all this, we will denote a t-millisecond history of low outputs as a sequence:

$$\beta_t = \langle\langle c_1, s_1\rangle, \langle c_2, s_2\rangle, \ldots, \langle c_n, s_n\rangle\rangle$$

where c_i is low's i^{th} bus request completion count and s_i is the value of low's system clock on the i^{th} fuzzy clock tick, *relative* to the starting time of the history (i.e., for simplicity we are translating the system times so that the history starts at time 0); and where $s_{n-1} < t \leq s_n$. We will denote the set of all possible low output histories of length t milliseconds by $\mathcal{O}_t$.

Now, what are the high process' inputs? For the purpose of the present analysis, we will assume that the high process uses only two input values—generate constant bus requests (to send a 1) or generate no bus requests (to send a 0), and that it chooses the duration of its inputs to be some integer multiple of one millisecond.[5] That is, the high process' inputs are of the form "generate (constant/no) bus requests for n milliseconds", where n is a positive integer.

Note that the high process can precisely control the duration of its inputs—even when they do not start or end on a fuzzy clock tick—in the following way. Since the low process' strategy is (by assumption) to generate a constant stream of bus requests, and the other processors are (by assumption) generating no bus requests, the high process knows the precise load on the bus, and therefore can calculate the number of bus requests it needs to make in order for its input to last for a given duration.[6]

Given this description of the high process' inputs, we will denote a t-millisecond history of high inputs as a sequence:

$$\alpha_t = \langle d_1, d_2, \ldots d_t\rangle$$

where $d_i \in \{0, 1\}$ is the high input during the i^{th} millisecond. We will denote the set of all t-millisecond high input histories by $\mathcal{I}_t$.

Now, given a description of how the bus works and of the probability distribution on fuzzy clock ticks, we can define a conditional probability function $q(\beta_t \mid \alpha_t)$ that gives the probability of the low output history β_t given the high input history α_t. In particular, to define q, note that for a given t-millisecond high input history, α_t, and a given t-millisecond history of fuzzy clock tick intervals, say $\gamma_t = \langle t_1, t_2, \ldots, t_n\rangle$, the resulting t-millisecond low output history, β_t, is functionally determined. Let's say that $\beta_t = f(\alpha_t, \gamma_t)$, where the function f is determined by the particular characteristics of the bus. Then, q can be defined as follows.

$$q(\beta_t \mid \alpha_t) = \sum_{\gamma_t} \begin{cases} P(\gamma_t), & \text{if } f(\alpha_t, \gamma_t) = \beta_t; \\ 0, & \text{otherwise.} \end{cases} \qquad (1)$$

where $P(\gamma_t)$ is the probability of the fuzzy clock tick history, γ_t, occurring, and is determined by the probability distribution with which fuzzy clock ticks are generated.

Now, we can define the capacity of the bus-contention channel as

$$C \triangleq \lim_{t\to\infty} C_t \qquad \text{bits per second} \qquad (2)$$

where

$$C_t \triangleq \max_{p(\alpha_t)} \left(\frac{1000}{t}\right) I(\mathcal{I}_t, \mathcal{O}_t)$$

where the maximization is taken over all possible probability distributions, $p(\alpha_t)$, on t-millisecond high input histories; and $I(\mathcal{I}_t, \mathcal{O}_t)$ is the mutual information between $\mathcal{I}_t$ and $\mathcal{O}_t$, defined as:

$$I(\mathcal{I}_t, \mathcal{O}_t) \triangleq \sum_{\alpha_t, \beta_t} p(\alpha_t) q(\beta_t \mid \alpha_t) \log \frac{q(\beta_t \mid \alpha_t)}{\sum_{\alpha'_t} q(\beta_t \mid \alpha'_t) p(\alpha'_t)}$$

Some notes are in order.

- The reason we need to consider the mutual information between *sequences* of inputs and outputs

[4] Note that we may have already lost some generality in our analysis; although the above assumption seems to be a good strategy for the low process to use in an exploitation of the bus-contention channel, it may not be the optimal strategy. But if the analysis in this restricted case turns out to be intractable, then our case for the intractability of the general problem will be even stronger—for the general analysis of the bus-contention channel entails maximizing over all possible low strategies.

[5] Note that we have again lost some generality; the optimal high strategy may involve some violation of this assumption.

[6] In contrast, the low process cannot use this trick since it does not know, a priori, how much bus traffic will be generated by the high process.

(rather than between single inputs and outputs) is that the bus-contention channel under fuzzy time is, in information theoretic terms, a *channel with memory*. Namely, at any given millisecond t, the state of the channel consists of the time t' at which the low process received its last fuzzy clock tick (for this determines the probability distribution over when it will receive its next fuzzy clock tick) and the current bus request completion count.

- The above definition of the capacity of a channel with memory follows that of Gallager [6, §4.6] with one exception; Gallager considers the effect of the initial state of the channel. However, he also shows that if a channel is *indecomposable*[7], then the initial state has no effect on the channel's capacity [6, Theorem 4.6.4]. Since the bus-contention channel is indecomposable (proof left to the reader), we have omitted all references to the initial state from the above.

Now we are in a position to discuss why evaluating the channel capacity (i.e., Equation 2) is so difficult. First of all, we cannot directly compute the capacity from its definition. This is due to the fact that the definition involves taking the limit (of C_t) as t approaches infinity. Trying to compute this directly would take infinite time. We are therefore left with two options: find an analytic solution or compute a numerical approximation. Finding an analytic solution would probably involve exploiting the structure of the conditional probability function, q. However, as shown in Equation 1 above, the expression for q involves a summation, which is typically difficult to deal with analytically unless a closed-form expression for it can be found. Our own attempts in this direction have thus far failed.

On the other hand, since inputs to the bus-contention channel occur at such a high rate (one thousand per second), numerical approximations of any significance involve long sequences of inputs and hence an intractable optimization problem. For example, an attempt to compute C_{100} (which represents the amount of information that can be transmitted over the channel in 100 milliseconds—i.e., one tenth of a second), requires us to find the optimal probability distribution over the set $\mathcal{I}_{100}$, which has 2^{100} elements (i.e., greater than 10^{30} elements). This is far beyond the realm of tractable optimization problems.

In the above, we have tried to indicate why analyzing the capacity of the bus-contention channel under fuzzy time is difficult. We'd like to emphasize that this is not to say that it can't be done. Rather, it is an open research problem.

2.3 Exploitation Rates

We have already noted that Hu did not provide a capacity analysis for the bus-contention channel under fuzzy time. What then, did Hu mean when he claimed that the "bandwidth is less than ten bits per second"? In private correspondence, John Wray and Wei-Ming Hu explained that the covert channel analysis team developed an exploitation—i.e., a high (sending) process and a low (receiving) process—and tried it out on the actual system to see how fast it ran. This is certainly a useful test. However, the results of this test do not provide any evidence that faster communication is not possible. Rather, this test gives us a *lower bound* on capacity. That is, exhibiting an exploitation that transmits data at a given rate, say n bits per second, shows that the capacity of the channel is *at least* as high as n bits per second.

Supposing that it is too difficult to evaluate the capacity of a given covert channel, then this type of ad hoc testing of various exploitations may be the best possible approach. However, we need to be aware of the limitations of such testing. In particular, such testing does *not* establish an upper bound on the rate of communication over the channel. We simply have to hope that system penetrators will not be more clever in their exploitations than the system analysts were.

2.4 Other Disadvantages of Fuzzy Time

Besides the fact that the effect of fuzzy time has not yet been mathematically analyzed, it has several disadvantages. We describe these here. Some of these disadvantages may seem rather obvious; however, we believe it is worth stating them explicitly, just so we have an accurate picture of the advantages and disadvantages of fuzzy time.

1. Processes do not have access to an accurate time source. This may render fuzzy time totally unacceptable for certain applications—e.g., real-time systems.

2. As discussed in Section 2.1, the effectiveness of fuzzy time relies partly on slowing all synchronization between processes at different levels; this includes synchronization *from low to high*. For example, if we were to allow a high process to have read access to a segment of memory for which a low process had write access, then this would allow the low process to quickly signal to the high process (e.g., as soon as it receives each of its fuzzy clock ticks). This would allow the high process to synchronize its sending activities to the low process' receiving activities and would thus negate much of the effectiveness of fuzzy time. Therefore, in the VAX security kernel, the timing of all reads to shared memory are fuzzed.[8] Unfortunately, fast reading down may be important in certain applications. For example, recent algorithms for multilevel-secure database concurrency control require such reading down (see [1] and [5]).

[7] The interested reader is referred to [6, §4.6] for the definition of *indecomposable*. Knowing the definition is not necessary in what follows.

[8] Actually, in the latest implementation of the VAX security kernel, shared memory (between virtual machines) was not implemented at all. But before the project was cancelled, the developers had planned to implement (fuzzy) shared memory in a future version.

3. All of the discussion in [7] and [8] regarding the performance impact of fuzzy time seems to focus on throughput. For example, in [8, §VI.E], it is stated that the "performance degradation due to fuzzy time was only 5-6% of CPU usage on multiprogrammed benchmarks". While the minimal impact of fuzzy time on (bus and CPU) throughput is one of its major advantages, it is unclear how it affects response time. For example, every I/O request is delayed until the fuzzy clock tick after it is submitted. Then, any replies to this request are delayed until the fuzzy clock tick after the reply arrives. These delays must have some impact on the overall response time of the system—especially for I/O intensive applications such as database systems. However, this impact is not analyzed or discussed in [7] or [8].

4. The implementation of fuzzy time requires a lot of random data. Typically, better random number generators (i.e., better in terms of how difficult they are to predict) take longer to produce random numbers. Therefore, random data should be considered a resource. An approach, such as fuzzy time, that uses random data at a high rate can be expensive to implement if really good random data is required. In [7, §6.2], Hu describes how the VAX Security Kernel designers sacrificed the quality of their random number generator in order to obtain the necessary amount of random data. Naturally, this is another area that is subject to attack—for if the penetrators can predict (or even determine at a *later* time) the sequence of fuzzy clock intervals, then fuzzy time loses much of its effectiveness.

3 Probabilistic Partitioning

In this section, we describe and analyze an alternative approach to introducing noise into the bus-contention channel. This approach has the following advantages.

1. It allows processes to have accurate reference clocks.
2. It does not depend on the lack of fast synchronization from low to high.
3. It is analyzable using information theory.
4. It is parameterizable and provides a precise tradeoff between covert channel capacity and performance.
5. It requires less random data than fuzzy time.
6. It can be used to completely close the bus-contention channel.

3.1 The Mechanism

The basic idea is that the bus interface controller (BIC) (i.e., the mechanism that serves as the interface between a given processor and the bus) will have two modes of operation—secure mode and insecure mode—and during normal operation of the bus, with some probability distribution, the bus controller will switch back and forth between these two modes.

Consider the architecture shown in Figure 2. In this architecture, there is a central clock that supplies clock ticks to all processors and to a central random number generator. For our later analysis, we will say that this clock has a fixed interval of 10 milliseconds (i.e., clock ticks are produced at a rate of 100 per second). The processors will buffer all asynchronous events (using the techniques described by Hu) and deliver them to their respective processes on the arrival of the central clock ticks. In this way, the central clock will be the fastest and most accurate reference clock available for covert channel exploitations. In contrast with Hu's work, this clock will not be fuzzy.

Also on each central clock tick, the random number generator will produce a single random bit and send it to all of the BICs. If the bit is a 0, then the BICs will operate in secure mode during the current central clock interval; if the bit is a 1, then they will operate in insecure mode. Clearly, the distribution on 0's and 1's produced by the random number generator parameterizes this scheme. In fact, we envision this distribution being software configurable (e.g., via the system security officer's trusted interface to the security kernel). We will analyze the scheme for various distributions below.

Now, what are the two modes? For insecure mode, we can use any standard resource contention scheme. That is, in insecure mode, the processors will contend for the bus in the standard way; both performance and covert channel capacity are at their peak.

For secure mode, the BICs can implement a fixed-time-slice, round-robin allocation policy. That is, each processor will be allowed to access the bus during its fixed time slice; there will be no contention and time slices may go unused even when there are other processors waiting to use the bus. In this mode, each processor will be permitted to use, at most, one n^{th} of the bus' total capacity; but, the covert channel capacity will be zero.

We call this mechanism *probabilistic partitioning* because the bus is partitioned (i.e., in secure mode) at certain times according to some probability distribution.

Before we analyze probabilistic partitioning, a few notes are in order.

1. If every memory access was mediated by the security kernel, then it would be possible to implement probabilistic partitioning in software. However, for performance reasons, it is preferable that the security kernel only mediate requests to open a file (i.e., to bring a file into a process' address space). Once the file has been opened, all security checks have been performed and the process can proceed to access it without any intervention from the security kernel. This is the approach used in the VAX security kernel. Given this, the functionality of our special-purpose BICs cannot be placed in (the soft-

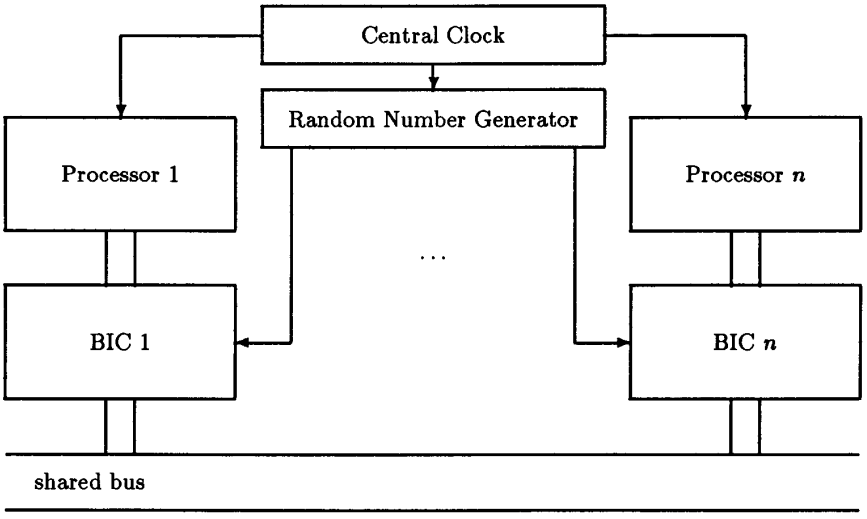

Figure 2: Architecture for Probabilistic Partitioning

ware part of) the security kernel; it must be placed in hardware.

2. Hu did not consider mechanisms such as probabilistic partitioning since he (explicitly) wanted to use off-the-shelf hardware. (As discussed above, probabilistic partitioning requires special-purpose BICs.)

3. The accuracy of the clock available to a given process in probabilistic partitioning is ten times better than that available under fuzzy time; viz, probabilistic partitioning provides an accurate 10 millisecond clock interval whereas fuzzy time provides the system time rounded to the nearest tenth of a second (100 milliseconds).

4. The rate of random data used by probabilistic partitioning is one bit per 10 milliseconds (i.e., 100 bits per second), whereas fuzzy time uses (on average) $\log_2(18001)$ bits per 10 milliseconds (i.e., greater than 1400 bits per second).[9]

5. The effectiveness of probabilistic partitioning relies on reducing bus contention, rather than on destroying the synchronization between processes. Therefore, allowing high to read down (or allowing any other means of synchronization from low to high for that matter) does not reduce the effectiveness of the mechanism. Further, by eliminating all bus contention, the covert channel capacity will be reduced to zero (although at a rather high performance penalty).

[9] The number $\log_2(18001)$ is obtained as follows. According to Hu, the "VAX interval-timer is a counter that increments at one-microsecond intervals and generates interupts when the counter overflows"[7, §5.1]. Further, on the VAX security kernel, clock ticks vary "randomly between 1 and 19 milliseconds, with a mean of 10 milliseconds"[7, §5.3]. In other words, for each fuzzy interval, the kernel loads a randomly generated integer between 1,000 and 19,000 (inclusive) into the interval timer. Thus, there are 18,001 different possibilities, which means that $log_2(18001)$ bits must be generated for each fuzzy clock tick.

3.2 Capacity Analysis

In contrast with fuzzy time, the information-theoretic analysis of the bus-contention channel under probabilistic partitioning is straightforward. Since the two processes are now synchronized by the central clock and there is no memory in the channel from one clock tick to the next, we can model the channel as a *discrete memoryless channel*[10] where one channel use in the model represents one central clock interval. This allows us to dispense with any consideration of high and low strategies and also simplifies the definition of capacity. We will only need to make one assumption; that is, we must make some assumption regarding the viable alphabet size for a covert channel exploitation using the given reference clock.

For example, let's suppose that the expected bus-request service time varies between 1 and 2 microseconds, depending on the current bus traffic. Now, in a 10 millisecond reference clock interval, the low process will complete between 5,000 and 10,000 bus requests. We might say that this gives us an alphabet size of 5,001. However in practice, the high process will not be able to reliably control (via its own bus requests) the *exact*

[10] See e.g., [6, §4.2] for a treatment of discrete memoryless channels.

number of bus requests that the low process can complete during a given clock interval. For our analysis, we will assume that the exploitation can make use of an alphabet with 100 symbols. Our analysis will not, however, be tied in any way to the particular alphabet size; that is, we can easily carry out the following analysis for any (finite) alphabet size. In practice, this value would be determined through testing or examination of the actual system.

Now, how do we model the bus-contention channel under probabilistic partitioning? We'll start by denoting the set of input letters by $\{a_1, a_2, \ldots, a_{100}\}$ and the set of output letters by $\{b_1, b_2, \ldots, b_{100}\}$. Now, the channel works as follows. During each central clock interval, the low process submits a constant stream of bus requests and counts the number of completions. We'll assume that if the bus is operating in secure mode (i.e., the random number generator produced a zero), then the low process will complete the same number of requests as if the bus were fully loaded.[11] Let's suppose that this particular output letter is b_1. Thus, when the bus is in secure mode, regardless of the high process' input letter, the low output letter will be b_1. On the other hand, when the bus is in insecure mode, the low output letter will correctly reflect the high input letter; that is, on input a_i, the output will be b_i.

Now we need to describe the probability distribution of the random number generator. The case that we will analyze is where the numbers generated are independent and identically distributed (i.e., each random number in the sequence is independent of all others and each is generated according to the same distribution). In this case, we can simply specify the probability of generating a one (i.e., the probability of being in insecure mode), which we'll denote $r(1)$. Then, the probability of generating a zero (i.e., secure mode) is given by $r(0) = 1 - r(1)$.

Given this, we can describe the probability of a particular output letter, b_j, given a particular input letter, a_i, by a conditional probability function, q, described as follows.

$$q(b_j \mid a_i) = \begin{cases} r(1) & \text{if } j = i \text{ and } i \neq 1; \\ 1 & \text{if } j = i \text{ and } i = 1; \\ r(0) & \text{if } j \neq i \text{ and } j = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Now, the capacity of the bus-contention channel under probabilistic partitioning is given by the standard definition of the capacity for discrete memoryless channels as follows.[12]

$$C \triangleq \max_{p(a_i)} \sum_{i,j} p(a_i) q(b_j \mid a_i) \log \frac{q(b_j \mid a_i)}{\sum_{i'} p(a_{i'}) q(b_j \mid a_{i'})}$$

where the maximization is taken over all possible probability distributions, $p(a_i)$, on high input.

We used the algorithm due to Arimoto [2] and Blahut [4] to compute this capacity for various values of $r(1)$, the results of which are plotted in Figure 3. Some notes regarding this graph are in order.

- From this figure, we see that the channel capacity is essentially a linear function of $r(1)$. (Note: there is a slight curve at the top of the plot that is essentially negligible—except in the case of a small alphabet size.)
- We see that when $r(1)$ is 1 (i.e., the bus is always in insecure mode), the capacity of the bus-contention channel is less than 700 bits per second. In contrast, Hu claims that without the fuzzy time countermeasure in place, the bus-contention channel can be exploited at a rate exceeding 1000 bits per second. Primarily, this discrepancy is due to the fact that even when $r(1)$ is 1, (under the probabilistic partitioning countermeasure) the exploitation of this channel is slowed down to the rate of the central clock (100 signals per second); for it is still the best available reference clock. In Hu's exploitation of the channel, a much faster reference clock is used.

Now how does probabilistic partitioning affect performance? A thorough analysis would require experiments with an actual implementation (or perhaps a simulation) of probabilistic partitioning. We have not performed such experiments. We can however analyze one (admittedly simplistic) measure of performance, which gives us some indication of how probabilistic partitioning affects performance. Namely, we will consider the maximum bus throughput that is available to any one processor.

Note that probabilistic partitioning does not reduce the overall throughput attainable on the bus; for in the case where all processors are constantly attempting to access the bus,[13] even in secure mode the bus will be in constant use. Rather, probabilistic partitioning reduces the maximum throughput attainable *by any one processor.*

The bus throughput that is available to a given processor, henceforth called *single-processor bus throughput*, is bounded by a linear function of $r(1)$; viz, we can define this bound as a function, $T(r(1))$, as follows.

$$T(r(1)) \triangleq (1 - r(1))T_1 + r(1)T_2$$

where T_1 and T_2 are constants representing the single-processor bus throughput when the bus is operating in secure mode and insecure mode, respectively.

Thus, we see that probabilistic partitioning provides us with a linear tradeoff between channel capacity and single-processor bus throughput. This precise tradeoff

[11] This seems like a realistic assumption. Further, if it does not hold in the actual implementation, then the following analysis can easily be changed without significantly affecting our results.

[12] See e.g., [6, §4.2] for the standard definition of capacity for discrete memoryless channels.

[13] We do not mean to imply that this case occurs often in practice; it does not.

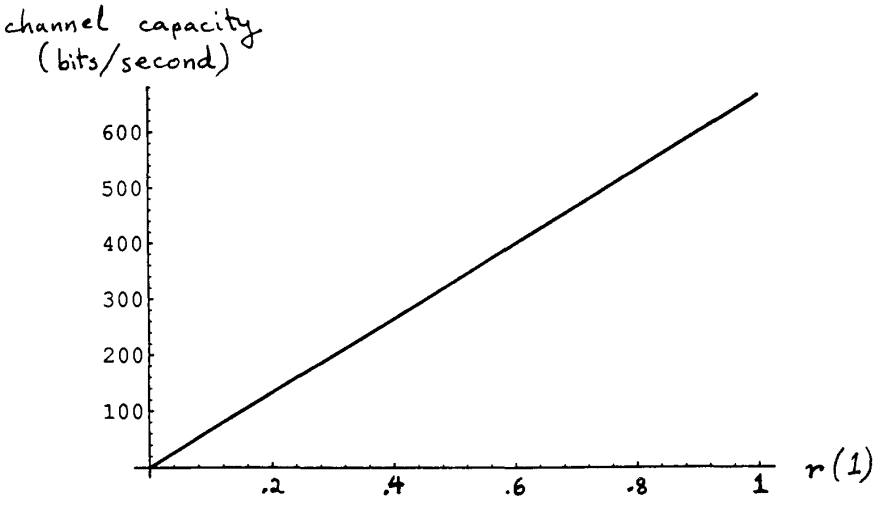

Figure 3: Capacity of the bus-contention channel under probabilistic partitioning

is perhaps the main advantage of probabilistic partitioning over fuzzy time. To the extent that single-processor bus throughput gives system designers an indication of overall system performance, the use of probabilistic partitioning gives system designers intellectual control over (1) the capacity of the bus-contention channel and (2) the impact of the covert channel countermeasure on system performance.

3.3 Disadvantages of Probabilistic Partitioning

There are, of course, some disadvantages of probabilistic partitioning in comparison with fuzzy time. We describe these here.

1. Probabilistic partitioning addresses only the bus-contention channel; that is, it does not reduce the capacity of any other covert channels that may be present in the system. Therefore, separate countermeasures must be introduced for each covert channel. In contrast, fuzzy time is a general-purpose covert channel countermeasure; it reduces the signalling rate (and, we would expect, the capacity) of all covert channels in the system.
2. Since probabilistic partitioning relies on special-purpose hardware, it is not as portable as fuzzy time. This may be a significant factor in whether or not system designers will adopt its use.

4 Conclusions

Naturally, the two approaches explored in this paper give rise to a wide range of possibilities. The two approaches can be used in conjunction, and each approach can be tailored in terms of the particular probability distribution used in the randomization. If we can provide a precise analysis of the effect of fuzzy time, then it may be fruitful to explore a combined approach.

For example, suppose system designers face a requirement (for real-time control purposes) to provide processes with a clock that has a clock-tick interval with a 5 millisecond mean and a 0.5 millisecond variance (or less). Suppose further that they have a requirement that all covert channels have capacities below 10 bits per second. If (under the given clock accuracy requirement) the fuzzy time countermeasure cannot sufficiently reduce the capacity of the bus-contention channel, then a combined approach may be able to satisfy both requirements while minimizing the performance impact due to probabilistic partitioning.

We would like to point out that the intention of Section 2 is not to discredit fuzzy time; on the contrary, we think fuzzy time is an excellent idea. Rather, the intent is to point out some of the disadvantages of the approach—something which Hu did not really do.

Here is a summary of the major comparisons between the two approaches.

Impact on Covert Channel Capacity: Clearly, fuzzy time reduces the capacity of the bus-contention channel a great deal. However, it is not possible, within the current state of the art, to give a precise analysis of the affect of fuzzy time on channel capacity. In contrast, the effect of probabilistic partitioning on covert channel capacity is straightforwardly analyzed. This may be very important in applications where tight tradeoffs are necessary. Also, it is not clear whether fuzzy time can reduce the channel capacity to zero, whereas probabilis-

tic partitioning can.

Impact on Performance: Fuzzy time has little or no negative impact on bus throughput or response time. However, it seems likely that the delays introduced to slow down the available reference clocks would have a significant impact on system response time for I/O intensive applications. However, this has not yet been analyzed. On the other hand, probabilistic partitioning can have a serious negative impact on the bus throughput for a single processor. So much so, that for certain applications, it may be desirable to design the system with a much higher capacity bus in order to offset this impact.

Suitability for Applications: Fuzzy time is probably not suitable for real-time applications requiring fine grain control with respect to time. Nor is it suitable when fast reading down or writing up (in security level) is required. In contrast, these applications present no problems for probabilistic partitioning.

Applicability: Fuzzy time is effective against all covert channels, whereas probabilistic partitioning is effective only against the bus-contention channel.

Portability: Fuzzy time is a software solution, and thus can be highly portable. In contrast, probabilistic partitioning requires special-purpose hardware (viz, bus interface controllers) and, therefore, can be used only on hardware platforms that provide such hardware.

Acknowledgements

I would like to thank Wei-Ming Hu of DEC, Paul Syverson of NRL, Jonathan Trostle of MITRE, John Wray of DEC, and the anonymous referees for their comments on a previous draft of this paper.

References

[1] Paul Ammann, Frank Jaeckle, and Sushil Jajodia. A two snapshot algorithm for concurrency control in multi-level secure databases. In *Proceedings of the 1992 Computer Society Symposium on Research in Security and Privacy*, pages 204–215, Oakland, CA, May 1992.

[2] Suguru Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, IT-18(1):14–20, January 1972.

[3] D. E. Bell and L. J. LaPadula. *Secure Computer System: Unified Exposition and Multics Interpretation, Technical Report MTR-2997 Rev. 1.* The MITRE Corporation, March 1976.

[4] Richard E. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, IT-18(4):460–473, July 1972.

[5] Oliver Costich and John McDermott. A multilevel transaction problem for multilevel secure database systems and its solution for the replicated architecture. In *Proc. 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 192–203, Oakland, CA, May 1992.

[6] Robert G. Gallager. *Information Theory and Reliable Communication.* John Wiley and Sons, Inc., New York, 1968.

[7] Wei-Ming Hu. Reducing timing channels with fuzzy time. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, 1991.

[8] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A retrospective on the vax vmm security kernel. *IEEE Transactions on Software Engineering*, 17(11):1147–1165, November 1991.