

Countermeasures and Tradeoffs for a Class of Covert Timing Channels

James W. Gray, III ¹

Hong Kong University of Science and Technology

Abstract

We identify a class of covert timing channels with the following properties. (1) existing covert timing channel analysis techniques are inappropriate for the channels in this class; and (2) it includes the fastest (i.e., highest capacity) covert channels known to date. Since channels in this class are exploited by *counting* the occurrences of certain events, we call them *counting channels*. We describe a technique for analyzing the capacity of counting channels and use our technique to analyze a known counting channel (viz, the bus contention channel) under two different types of countermeasures: an existing countermeasure called *fuzzy time* and a novel countermeasure called *probabilistic partitioning*. In particular, our analysis provides the only known upper bound on the capacity of the bus contention channel under fuzzy time. Also, for both types of countermeasures, we obtain precise tradeoffs between covert channel capacity and other desirable system properties.

¹A large portion of this work was performed while the author was at the Naval Research Laboratory in Washington, DC. Currently, the author is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, HONG KONG.

Index Terms

- covert channel countermeasures
- engineering tradeoffs
- information-theoretic analysis
- multilevel security
- shared memory multiprocessor architecture

General Terms: Security, Performance, Design

Additional Key Words and Phrases: covert channels, timing channels, information theory, protection, confinement, flow analysis

1 Introduction

Many organizations have a need for computer systems that store, process, and provide access to data that is classified with respect to a multilevel hierarchy (e.g., unclassified, confidential, secret, top secret, etc.). Such systems are commonly referred to as *multilevel secure*. Typically, the set of classifications forms a lattice under a relation called *dominates* [4]. Further, each system user is given a clearance (drawn from the same lattice) and is allowed to read a given piece of data only if the user's clearance dominates the data's classification.²

For simplicity, throughout this paper we refer to two levels, "high" and "low" with the understanding that the high level strictly dominates the low level. That is, low level users are not cleared to see high level data.

The most straightforward approach to implementing a multilevel secure system is to assign classification labels to all objects managed by the system (e.g., files) and clearance labels to all subjects operating within the system (e.g., processes). Then, whenever a subject attempts to obtain read access to an object, the labels are checked and the access is allowed if and only if the subject's label dominates the object's label. This is the approach taken in the well-known "Bell and Lapadula model" [2] and is generally known as *access control* [10].

Despite the intuitive appeal of access control, it has been known since as early as 1973 that access control is not sufficient to ensure that users cannot obtain information for which they do not have the necessary clearance; in particular, any given system typically contains several *covert* communication channels [11] by which information can be transmitted from one level down to a lower level. Note that there is no precise distinction between a covert channel and a normal communication channel; but typically, a channel is called "covert" if it is (1) hidden from the authorities or (2) capable of violating the intended security policy.

In practice, most covert channels arise due to resource sharing. For example, consider 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. Suppose that a high process and a low process are executing concurrently on two distinct processors. We are assuming that access controls are in place that prevent processes from directly reading data for which they are not cleared. Thus, even though the two processes share the same physical memory bank, it would not be possible for the low processor to directly read high data. However, the high process can transmit data to the 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, counts how many of its requests are serviced; if the number is relatively small, it records this as a 1; if the number is relatively high, it records this as a 0. Thus, high data can be sent to the low process at 1000 bits per second. \square

Note that in the above and throughout this paper, we take the system designer's point of view and make the worst-case assumption that processors that are not involved in the covert channel exploitation will not interfere with the communication.³ That is, there is no back-

²A different rule is applied when a user wants to *write* a given piece of data.

³From the *penetrator's* point of view, this is a best-case assumption.

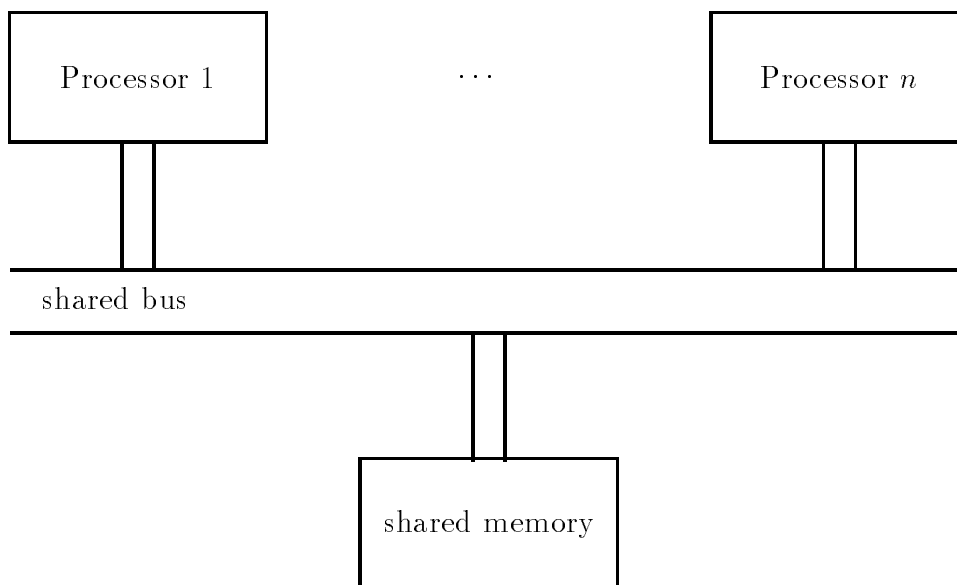


Figure 1: **A Typical Multiprocessor Architecture in which the Bus Contention Channel arises.**

ground noise that may slow down the exploitation of the channel. We believe that making such worst-case assumptions (rather than, say, average-case assumptions) is appropriate in the context of covert channel analysis; that is, to measure the *risk* associated with a given channel, we are interested in finding an *upper bound* on the exploitation rate, rather than the *expected* exploitation rate.

Also note that the threat that we are concerned with is *not* that the users (i.e., the *human* users) of the high process 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 process is actually 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 process. 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.

Generally speaking, whenever there is a shared resource, there is the potential for a covert channel. (This observation forms the basis of the “shared resource” matrix methodology for covert channel detection [9].) In fact, the only way to completely prevent communication over a shared resource is to remove all contention (between security levels) for that resource. This leads to inefficient utilization of resources. Thus, as long as we want to share resources in an efficient manner, we will have covert channels.

Since we expect that users will demand that resources be used efficiently, our approach is to explore various countermeasures designed to reduce the capacity (i.e., the maximum exploitation rate) of covert channels without having too great an impact on system performance.

1.1 Timing Channels, Reference Clocks, and Counting Channels

Typically, covert channels are informally divided into two classes: storage channels and timing channels. A storage channel is one where the high process signals by modulating the value of some storage variable that can be (indirectly) read by the low process; the information being sent over the channel is encoded in the value of the storage variable. On the other hand, a timing channel is one where the high process signals by modulating the timing of an event that is visible to the low process; the information being sent is encoded in the time it takes for an event to occur. This distinction is not entirely precise, but the important thing to note is that in order to exploit a timing channel, the low receiving process needs a reference clock against which it can measure the timing of events.

For example, note that in the above covert channel exploitation scenario we said that “during each millisecond interval” the low process counts how many of its requests are serviced. In saying that, we implicitly assumed that the low process has a clock available that it can use to measure millisecond intervals. In fact, the exploitation of the bus contention channel (as well as many other covert channels) depends crucially on the existence of a usable reference clock. This observation forms the basis of Wray’s “dual clock” covert channel detection method [16].

Another important observation made by Wray is that any sequence of events constitutes a clock. In particular, Trojan horses typically have several clocks available for their use in addition to the normal system clock provided by the operating system. For example, a clock can be set up by issuing a large number of requests to an asynchronous disk controller in such a way that the I/O completion interrupts will arrive (back at the issuing process) at a regular rate, thus providing a usable clock. Similarly, requests to a direct memory access (DMA) controller can be issued in such a way that data arrives in the process’ address space at an extremely regular rate. Generally speaking, any asynchronous controller can be used to provide a reliable reference clock for use in a covert channel exploitation. Therefore, controlling all such reference clocks is crucial to effective covert channel control.

Now, a covert timing channel can be viewed as consisting of two event streams: one event stream serves as the reference clock and the other is the sequence of events whose timing is being modulated by the high sending process. We will refer to this latter stream as the *signaling event stream*. Using these notions, we can identify two distinct (but not all-encompassing) classes of timing channels.

In the first class, the reference clock rate is much faster than the signaling event rate. In this case, each event in the signaling event stream can be individually timed and thus can carry some information. In recent work by Millen [12] and Moskowitz and Miller [13] the authors analyze the capacity of various timing channels from this class. Typically, in the channels that they analyze, the high (sending) process either interferes with (and slows down) a particular low-level signaling event or does not interfere with that event. When the high process does not interfere, the signaling event takes 1 (one) time unit and when the high process does interfere, it takes $1 + \alpha$ time units. Implicit in their analysis is that the low process has a reference clock that is sufficiently accurate to detect the difference between the two. In fact, they implicitly assume that the *exact* timing of the signaling event is known by the low process. This assumption is realistic only in the case where the reference clock rate is much faster than the signaling event rate.

In the second class of timing channels, the signaling event rate is much faster than the

reference clock rate. In this case, a single signaling event cannot be accurately timed. Rather, the channel is exploited as follows. For an entire reference clock interval, the high sending process maintains a certain level of interference with the signaling event stream. During that same reference clock interval, the low receiving process *counts* the number of signaling events that occur. When the reference clock interval completes, it is this event count that carries information. In fact, since there is no way to time individual events in the signaling stream, the event counts are the only data that the low process receives over the channel. For this reason, we refer to covert channels in this class as *counting channels*. Outside of our own early reports on the present work (viz, [5, 6]) there have been no published analyses of counting channels.

1.2 Outline of the Paper

In Section 2 we review an approach (due to Hu) for implementing reference clock controls in an operating system. In Section 3 we describe our general approach to analyzing the information-theoretic capacity of counting channels.

In the remainder of the paper, we explore two approaches to reducing the capacity of counting channels. The first approach, called *fuzzy time* was first described by Hu [7, 8]. The second is a novel approach, which we call *probabilistic partitioning* and which was first described in [5]. In Sections 4 and 5, we explore fuzzy time and probabilistic partitioning, respectively. For each, we obtain a tradeoff between covert channel capacity and another desirable system property. In Section 6, we give some concluding remarks.

2 Implementing Reference Clock Control

As mentioned above, controlling all of the reference clocks that are available to Trojan horses for use in covert channel exploitations is crucial to effective covert channel control. We will therefore briefly describe an approach to implementing such reference clock control, which was first described by Hu in [7]. 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 [8], 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 a clock that is maintained by the security kernel. That is, on each of the system controlled clock ticks, the security kernel delivers all events (interrupts, data, etc.) that have occurred since the last clock tick.

In this way, the security-kernel-controlled clock becomes the only (and hence, the most accurate) available reference clock for use in a covert channel exploitation.⁴

⁴Actually, 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

There is one reference clock that is of particular concern. That is, the clock provided by a separate processor writing to shared memory. Consider, for example, the situation where there are three processors on the bus. One processor is running the high sending process; a second processor is running the low receiving process; and the third processor is running a low reference clock process. This third process alternately writes ones and zeros to some (low) memory location that it shares with the low receiving process. The receiving process periodically reads this memory location and when it sees a change, it interprets it as a clock tick. In this way, the third process acts as a fast and accurate reference clock. Clearly, this reference clock must be controlled along with all other reference clocks in the system.

The solution we envision is that shared memory would be virtualized in much the same way that a cache virtualizes memory. That is, a write to “shared memory” is implemented by a write to some local memory buffer. Then, on each (system-controlled) reference clock tick, all of the local memory buffers that have been modified will be copied to the actual shared memory location. In this way, shared memory cannot be used to implement a reference clock that is faster than the system-controlled reference clock.

Clearly, for applications that make heavy use of shared memory, this approach will have a significant performance impact. Further, performance concerns will lead system designers to make the system-controlled reference clock as fast as possible. However, security concerns (viz, requirements on covert channel capacity) will lead designers to make the clock as slow as possible. Indeed, later in this paper, we will see that we can make precise tradeoffs between the reference clock rate and covert channel capacity.

3 Capacity Analysis for Counting Channels

In this section, we give our general approach to analyzing the capacity of a counting channel. We will defer to later sections all of the details about applying the model to analyze realistic counting channels.

For our purposes, a countermeasure is a means of introducing noise into a covert channel. We think of the system choosing a parameter, k , that allows the two processes to have a certain amount of resource contention (and thus a certain amount of communication) during each reference clock interval. In order for the choice of this parameter to constitute “noise”, this choice is made randomly.⁵ For all the cases that we will consider in this paper, the choice for each interval is made independently and identically distributed (i.i.d.). Thus, it is sufficient to model a countermeasure as a set of countermeasure parameters, K , and a probability distribution, $R(k)$, over that set. Intuitively, $R(k)$ should be regarded as the probability that the system will choose parameter k during any given clock tick interval.

Let I be the high process’ input alphabet; that is, I is the set of all possible actions that the high process can take during a given reference clock interval. Typically, I will include only those actions that can have some effect on the low process’ event count during a given reference clock interval.

notification of outgoing events). However, for the present paper it is sufficient to consider a single reference clock.

⁵Actually, it is made pseudo randomly. The important thing is that the processes attempting to signal over the covert channel cannot predict the choice any better than if it were truly random.

We describe the statistical properties of the low process' event count as a conditional probability function $P(j | i, k)$ where $j \in \mathbf{N}$ is a natural number representing the low receiving process' event count, $i \in I$ is the high sending process' input, and $k \in K$ is the parameter chosen by the system. That is, $P(j | i, k)$ describes the probability distribution on event counts j given that i was input from high and the countermeasure parameter is k . This probability distribution would be used to model the effect of low-level system implementation details that affect the low process' event count, but are not under the control of Trojan horses; e.g., using separate asynchronous machine clocks for each CPU on the bus may cause bus response times to appear to be nondeterministic. The description of P would reflect such nondeterminism. Thus, we think of P as describing the noise that occurs "naturally" in the covert channel, as opposed to the noise that is artificially injected by the countermeasure $R(k)$. We envision that in practice, the description of P would be based on statistical tests of the actual machine under consideration.

Now, we can define the channel transition matrix, $q(j | i)$, i.e., the probability of the low process' event count being j given that the high process sends an i .

$$q(j | i) = \sum_k R(k)P(j | i, k) \quad (1)$$

We can rigorously justify this equation under the assumption that the high process' input, i , and the system supplied parameter, k , are chosen *independently*. We expect this to be the case for the following reasons. First, the two choices are made simultaneously and the high process should not be able to predict the system's choice in advance. Thus, the high process will not be able to base its choice on the system's choice. Second, since it is desirable (from the system designer's perspective) for the two to be independent, the system's choice would be made independently of high processing.

We now rigorously justify Equation 1 as follows. Let $P(i, j, k)$ be the joint probability distribution on the input, i , the event count, j , and the parameter, k . Let $P(i)$, $P(k)$, $P(i, k)$, $P(j, i)$ be marginal distributions calculated from $P(i, j, k)$ in the standard way; e.g.,

$$P(i, k) = \sum_j P(i, j, k)$$

We will assume that this joint probability distribution is consistent with our model, viz, $P(k) = R(k)$ and $P(j | i) = q(j | i)$. Our assumption that i and k are chosen independently is formalized by the following equality, which holds for all i and k .

$$P(i, k) = P(i)P(k)$$

Then, we have the following.

$$\begin{aligned} \sum_k R(k)P(j | i, k) &= \sum_k R(k)P(i, j, k)/P(i, k) \\ &= \sum_k R(k)P(i, j, k)/(P(i)P(k)) \\ &= \sum_k P(i, j, k)/P(i) \end{aligned}$$

$$\begin{aligned}
&= P(i, j)/P(i) \\
&= q(j | i)
\end{aligned}$$

Thus, Equation 1 is justified.

Finally, we can define the capacity of the channel (in terms of bits per clock tick) in the standard way for discrete memoryless channels [14]. That is,

$$C = \max_{p(i)} \sum_{i,j} p(i)q(j | i) \log \frac{q(j | i)}{\sum_{i'} p(i')q(j | i')} \quad (2)$$

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

In the following two sections, we apply this approach to the analysis of the bus contention channel under various countermeasures.

4 Fuzzy Time

Hu’s approach to reducing the capacity of the bus contention channel is to introduce noise into the reference clock. That is, the security kernel controlled reference clock is given a random interval. In this way, all covert channels whose exploitation depends on measuring the timing of events with respect to a reference clock will be mitigated. Additionally, since this technique does not affect resource usage or resource contention resolution policies, the performance impact (outside of the impact due solely to reference clock control) of this technique is small.

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—the fuzzy clock’s interval has a mean of 20 ms, as opposed to the one ms clock interval needed for the covert channel exploitation scenario described above. 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 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, [Hu’s] experience has shown that randomization makes most exploitations utilizing large alphabet sizes impractical.”[8, §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 inhibits 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 *doesn't* 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.

This second effect of fuzzy time (inhibiting the synchronization between processes), while being beneficial to the goal of covert channel capacity reduction, has the undesirable effect of complicating the capacity analysis. In fact, our own attempts to analyze the precise capacity of the bus contention channel under fuzzy time failed. (See [6] for a description of these attempts.) The problem is that the two processes can synchronize to a certain (but unknown) extent. In particular, to determine to what extent the two processes can simultaneously synchronize and signal, we must optimize over all possible synchronizing and signalling *strategies*.⁶ And the space of all such strategies, even when we limit it to strategies of a relatively short (finite) length is astronomical in size.

To solve this problem, we simply assume that the low process can send *instantaneous* synchronization signals to the high process. This is a worst-case assumption in the following sense. The covert channel capacity we obtain under this assumption will be an *upper bound* on the actual covert channel capacity (when instantaneous synchronization signals are not possible). This is due to the fact that any communication that is achievable using impeded synchronization would also be achievable using the instantaneous synchronization. Thus, from the point of view of securing the system, the worst-case scenario is where the two processes can instantaneously synchronize.

Fortunately, from the point of view of security, we often are interested in such worst-case scenarios. That is, we are interested in determining an upper bound on the risk associated with a particular system. This is precisely what our analysis provides—an upper bound on the capacity of the actual covert channel.

The assumption of instantaneous synchronization manifests itself in our model in the following way. Every time the low process receives a reference clock tick (and thus begins a new event count), it notifies the high process of this fact. The high process can thus begin sending its next input signal right at the start of the new reference clock interval. That is, the high process can choose a new input signal for each reference clock tick interval and can send that signal during the entire reference clock tick interval. In this case, we can model the bus contention channel under fuzzy time as follows.

The set of countermeasure parameters K will correspond to the set of reference clock tick lengths. In all of the following examples, these lengths will be expressed in microseconds, abbreviated μs . Thus, for example, if the system always chooses the clock interval between 1 and 18 ms, then the set K will be $\{1000, 1001, 1002, \dots, 18000\}$.

The high process' input alphabet, I , will be a finite subset of the real interval $[0, 1]$. The interpretation of input symbol $i \in I$ is as follows. During any given interval, the high process' bus usage is some fraction of its total possible bus usage. When the high process chooses input i , that fraction will be i . For example, if the high process chooses input 0 during a given interval, then it will not use the bus at all during that interval. On the other hand, if the high process chooses input 1, then it will use the bus as much as it can during the given interval.

⁶Here, the term *strategies* is meant in the game-theoretic sense.

Consider some examples.

Example 4.1 Lets consider a realistic example based on the system described in [8]. We need to specify the input alphabet (I), the set of countermeasure parameters (K) and the associated probability distribution ($R(k)$), and the conditional probability distribution on bus request completion counts (P). We'll let $I = \{0, 1\}$. (For this example, a larger input alphabet is of no use to the high process. After completing the description of the present example, we'll describe how we determine an appropriate input alphabet.)

Our fuzzy clock will have the same probability distribution as in [8]. In particular, Hu says that the “VAX interval-timer is a counter that increments at one-microsecond intervals and generates interrupts when the counter overflows” [8, §5.1]; he also says that the “interval between two successive ticks is a uniformly distributed random variable” and that clock ticks vary “randomly between 1 and 19 milliseconds, with a mean of 10 milliseconds” [8, §5.3]. In other words, for each fuzzy interval, the kernel loads a uniformly-distributed random number between 1,000 and 19,000 (inclusive) into the interval timer. Thus, there are 18,001 different possibilities for the length of a given clock-tick interval. Further, Hu describes how successive ticks are used alternately for outgoing events (which Hu calls *downticks*) and incoming events (which Hu calls *upticks*); according to his description, a given process can see only its upticks, which occur on *every other* tick of the underlying clock. Therefore, the probability distribution on the *process-visible* clock tick is determined by the probability distribution on two successive (independent and identically distributed) ticks of the *underlying* clock.

That is, we have the following.

$$p_v(n) = \sum_{\substack{i,j \text{ such that} \\ i+j=n}} p_u(i)p_u(j) \quad (3)$$

where $p_v(n)$ denotes the probability distribution on process-visible clock ticks and $p_u(n)$ denotes the probability distribution on the underlying clock ticks.

Also, since p_u is uniformly distributed, we have the following.

$$p_u(n) = \begin{cases} 1/(18,001), & \text{if } 1,000 \leq n \leq 19,000; \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Before we give the general form for p_v , we'll consider a few special cases. Since the shortest possible interval of the underlying clock is 1,000 microseconds, the probability of getting a process-visible clock tick of 2,000 microseconds is equal to the probability of getting two successive underlying clock ticks of 1,000 microseconds. That is,

$$p_v(2,000) = p_u(1,000)p_u(1,000) = 1/(18,001)^2$$

The probability of getting a process-visible clock tick of 2,001 microseconds is equal to the probability of getting underlying clock ticks of 1,000 and then 1,001 or underlying clock ticks of 1,001 and then 1,000. That is,

$$\begin{aligned} p_v(2,001) &= p_u(1,000)p_u(1,001) + p_u(1,001)p_u(1,000) \\ &= 2/(18,001)^2 \end{aligned}$$

In general, from Equations 3 and 4 we have the following.

$$p_v(n) = \begin{cases} \frac{(n - 1,999)}{(18,001)^2}, & \text{if } 2,000 \leq n \leq 20,000; \\ \frac{(38,001 - n)}{(18,001)^2}, & \text{if } 20,001 \leq n \leq 38,000; \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the set of countermeasure parameters is $K = \{2000, 2001, \dots, 38000\}$ and the associated probability distribution is $R(k) = p_v(k)$.

Now we need to describe P , the conditional probability distribution on bus request completion counts. In practice, this description should be based on statistical tests of the actual machine under consideration. Since we do not have such statistics available, we will make some assumptions. (In a later example, we will consider the effect of different assumptions; in particular, we will consider the worst-case assumption where there is no additional noise introduced by P .) Let's suppose that the statistical tests yield the following results.

First of all, we'll assume that when there is no contention for the bus, a bus request will take approximately $10 \mu s$ to complete. That is, the *minimum bus response time* is $10 \mu s$. (In a later example, we consider the effect of faster bus response times.) In particular, given that the system chooses countermeasure parameter k (i.e., a clock tick interval of $k \mu s$) and the high process chooses to send input 0, the expected output (i.e., bus completion count) will be $\text{round}(k/10)$, that is, $k/10$ rounded off to the nearest integer. Further, we will assume that the distribution looks like a triangle centered around $\text{round}(k/10)$, tapering off linearly and symmetrically on both sides. For example, for $k = 11,000$, we have the distribution shown in Figure 2. More precisely, we have the following. We will use the integer n , given as follows, to normalize the distribution.

$$n = \sum_{j=\text{round}(k/10)-9}^{\text{round}(k/10)+9} (10 - |\text{round}(k/10) - j|) \quad (5)$$

The conditional distribution P is then given by the following.

$$P(j | 0, k) = \begin{cases} \frac{10 - |\text{round}(k/10) - j|}{n}, & \text{if } |\text{round}(k/10) - j| < 10; \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

We'll also suppose that statistically, on input 1 and with a $k \mu s$ clock-tick interval, the expected bus-request completion count is half of that for input 0 and is distributed in the same way. Thus, we have the following.

$$P(j | 1, k) = \begin{cases} \frac{10 - |\text{round}(k/20) - j|}{n}, & \text{if } |\text{round}(k/20) - j| < 10; \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

Given the above descriptions of I , K , R , and P , it is straightforward (albeit tedious) to compute the channel transition matrix, $q(j | i)$ (i.e., the probability of the low process

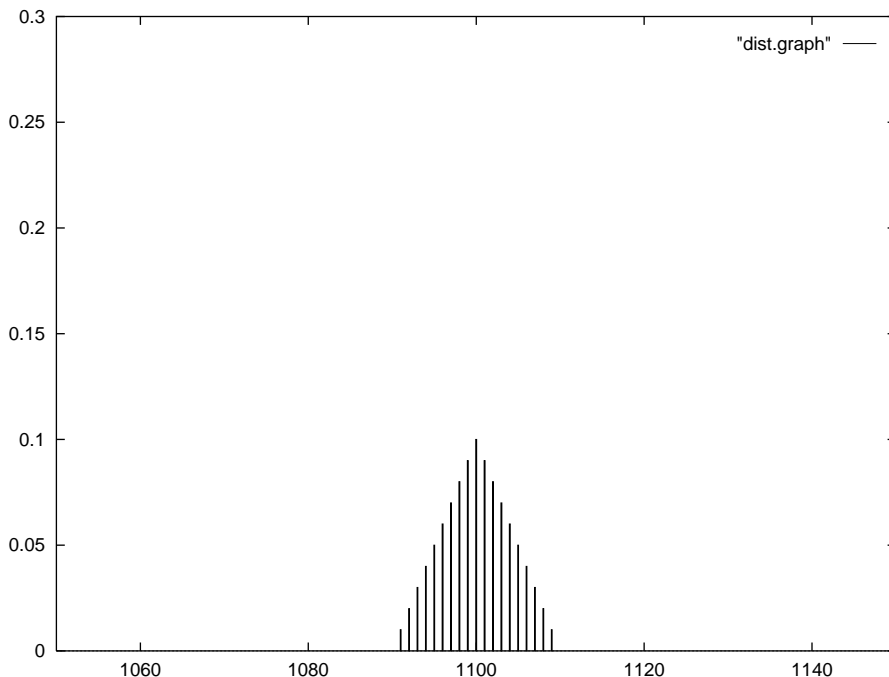


Figure 2: **Distribution of Naturally Occurring Noise.** The horizontal axis is the low process' completion count; the vertical axis is the probability of obtaining that completion count, given a particular clock tick interval and high input. This graph shows the distribution due to naturally occurring (background) noise that we use in some of our analyses.

completing j bus-requests given that the high process sends i), using Equation 1. We wrote a computer program to do this, and then, using the algorithm due to Arimoto [1] and Blahut [3], to compute the resulting capacity.⁷ The resulting capacity is 22.7 bits per second. \square

The significance of this number—22.7—is that it gives us an *upper bound* on the rate at which data can be sent over the channel. It gives a quantitative measure of the effectiveness of the countermeasure under consideration. This information can be used in a risk analysis to help determine whether the system is usable for a given application.

This is in contrast to the approach used by Hu to evaluate the effectiveness of fuzzy time [8]. His approach was to experiment with the channel using actual exploitations and measure the rate achieved. (In personal communications, Hu reported that in these experiments, he was able to achieve a rate of two bits per second.) While Hu's approach can provide some qualitative indications that our analysis can't (such as the *difficulty* of achieving a given rate; i.e., how sophisticated would the Trojan horse be?), it does not provide any evidence that penetrators cannot achieve higher rates.

Therefore, we see our analysis as being complementary to Hu's evaluation approach. 22.7 bits per second represents a theoretical limit on what penetrators can achieve, whereas 2 bits per second represents what penetrators could be expected to achieve given the amount of effort expended by Hu.

Now, before going on to the next example, we describe how we determine the set of channel inputs, I . Theoretically, the set of possible inputs is enormous. In particular, each possible

⁷Details of the computer program are available from the author.

pattern of bus usage that the high (sending) process can produce is a distinct input. If we tried to include all possible inputs in our model, the set I would be so large that computing the resulting capacity would be intractable. Fortunately, in practice (and in our model) the large majority of these inputs are indistinguishable to the low (receiving) process. We therefore want to obtain a set of inputs that is small enough that the resulting capacity calculation is tractable, yet covers all inputs that are distinguishable by the low process. We choose this input alphabet in the following ad hoc (but, we believe, effective) way.

Recall that for the present analyses, the set of inputs, I , is a finite subset of the real interval $[0, 1]$ and an input $i \in I$ is interpreted as the fraction of time (during the given clock tick interval) that the high process uses the bus.⁸ Clearly, the two most distinctive inputs are 0 (the high process doesn't use the bus at all) and 1 (the high process uses the bus as much as it can). So, we start out with a small set of equally spaced inputs that includes 0 and 1. For example, if we start out with 6 inputs, the set I would be $\{0, 1/5, 2/5, 3/5, 4/5, 1\}$. Now, given such an initial set, we can describe our ad hoc procedure as follows.

We perform the capacity calculations with the initial set I then

1. enlarge the set I ;
2. recalculate the capacity;
3. if the capacity has changed significantly, go to step 1; otherwise, terminate the procedure and use the previous set I .

The results of a typical application of this ad hoc procedure are shown in Figure 3, where we plot the calculated capacity on the vertical axis, versus the alphabet size of the model on the horizontal axis. From the figure, we see that the calculated capacity increases until the alphabet size gets to about 26, after which it levels off. We infer that regardless of how much larger we make the input alphabet, the capacity of the model will not increase. So we take the channel capacity of the model with an input alphabet of 26 symbols to be the capacity of the actual covert channel.

Example 4.2 The Effect of the Fuzziness of the Clock

We would like to see how different distributions on clock-tick intervals affect the capacity of the channel. In particular, it would be nice if we could obtain a tradeoff between clock accuracy and channel capacity. Toward this end, let's generalize the previous example in the following way.

We will use the same scheme for generating clock-tick intervals as is used on the VAX security kernel [8]; that is, we will assume that Equation 3 still holds. We will however generalize Equation 4 to the following.

$$p_u(n) = \begin{cases} 1/(u_u - l_u + 1), & \text{if } l_u \leq n \leq u_u; \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

That is, the underlying clock tick interval is uniformly distributed over the range from l_u microseconds to u_u microseconds (inclusive). We can now describe p_v for this more general situation as follows.

⁸A more precise interpretation of inputs is given as part of the next example.

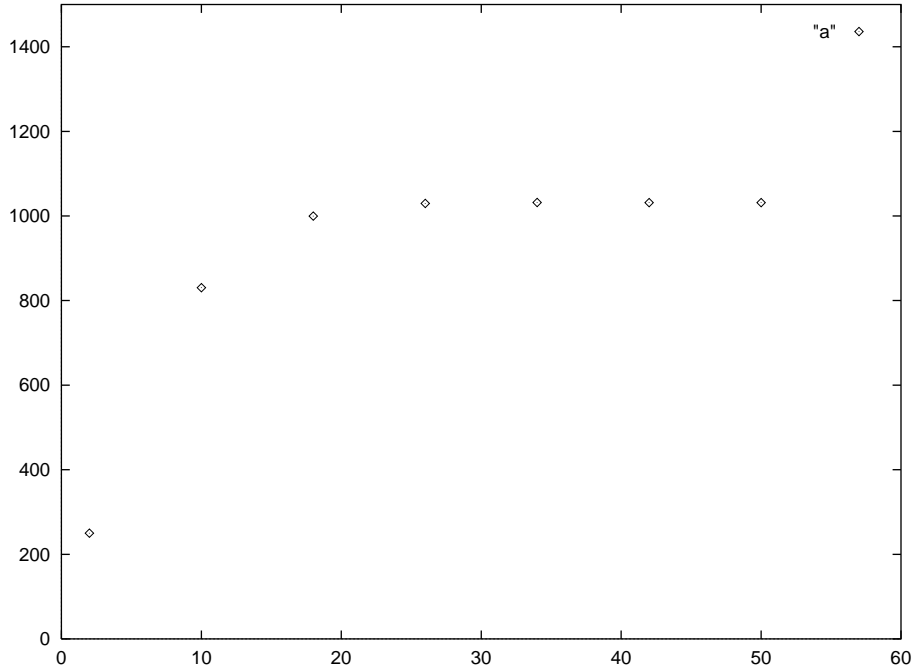


Figure 3: **Choosing an input alphabet.** The vertical axis is the calculated capacity and the horizontal axis is the input alphabet size used in the calculation. This graph indicates that capacity levels off fairly quickly and further increases in the alphabet size have no effect. It is therefore straightforward to determine the maximum usable input alphabet.

Let $l_v = 2l_u$ and $u_v = 2u_u$ be the lower and upper bounds, respectively, on the visible clock-tick interval (in microseconds). Let $\mu = (u_v - l_v)/2$ be the mean visible clock-tick interval (in microseconds). We can now describe the distribution on visible clock-tick intervals as follows.

$$p_v(n) = \begin{cases} \frac{(n - l_v + 1)}{(u_u - l_u + 1)^2}, & \text{if } l_v \leq n \leq \mu; \\ \frac{(u_v - n + 1)}{(u_u - l_u + 1)^2}, & \text{if } \mu + 1 \leq n \leq u_v; \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

To account for larger sets of inputs we also need to generalize Equations 6 and 7. To do so, we need to describe the set I and our interpretation of an input $i \in I$. We'll use $|I|$ to denote the size of I and we'll assume that I is the set $\{0/(|I|-1), 1/(|I|-1), \dots, (|I|-1)/(|I|-1)\}$. As previously discussed, when the high process "inputs" $i \in I$ for a given interval, we take this to mean that the high process makes use of the fraction, i , of the bus that is available to it. Thus, when the bus is in insecure mode, $i/(1+i)$ of the bus will be used by the high process and $1/(1+i)$ of the bus will be used by the low process (since the low process uses all of the bus that is available to it).

Now we let \mathbf{I} , \mathbf{J} , and \mathbf{K} denote the random variables corresponding to the high input, the low output, and the clock tick interval, respectively and we describe the expectation on the

low output, given input i and clock interval k as follows.

$$E[\mathbf{J} \mid \mathbf{I} = i, \mathbf{K} = k] = \text{round} \left(\left(\frac{k}{10} \right) \left(\frac{1}{1+i} \right) \right) \quad (10)$$

(where the denominator of the left-hand fraction, 10, is the expected time (in microseconds) for the bus to service a single request.) Thus, $E[\mathbf{J} \mid \mathbf{I} = i, \mathbf{K} = k]$ is simply the total number of requests that are expected to be processed during a $k \mu\text{s}$ interval, multiplied by the fraction that will be taken by the low receiver.

Now we model the naturally-occurring noise just as in Example 4.1. We use the same normalizing factor, n , given in Equation 5. The conditional distribution P is then given by the following.

$$P(j \mid i, k) = \begin{cases} \frac{10 - |E[\mathbf{J} \mid \mathbf{I} = i, \mathbf{K} = k] - j|}{n}, & \text{if } |E[\mathbf{J} \mid \mathbf{I} = i, \mathbf{K} = k] - j| < 10; \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

Note that for $|I| = 2$, this distribution is equivalent to that given by Equations 6 and 7. Also, it is straightforward to verify that, as expected,

$$E[\mathbf{J} \mid \mathbf{I} = i, \mathbf{K} = k] = \sum_j P(j \mid i, k)$$

We will use the same definition of $q(j \mid i)$ (i.e., Equation 1). Further, our previous definition of channel capacity (i.e., Equation 2) still applies. Thus, the only things we've generalized in the present example are the range of the underlying clock tick interval and the size of the high input alphabet.

We calculated the capacity for a variety of underlying clock tick interval ranges. In particular, we chose all of the underlying clock tick ranges used in this example so that the resulting mean (visible) clock tick interval would be 20 milliseconds. Rather than plotting the resulting capacity with respect to the *underlying* clock tick interval ranges, we plotted capacity with respect to the mean deviation.

Note: the *mean deviation* (d) of the visible clock tick interval is calculated as follows.

$$d = \sum_n p_v(n) |n - \mu|$$

For our purposes, the mean deviation of the clock tick interval is a measure of the “fuzziness” of the clock—the higher the mean deviation, the fuzzier the clock.

The resulting plot is shown in Figure 4. This graph illustrates the tradeoff between covert channel capacity and clock accuracy. Such tradeoffs are important in applications where the accuracy of available clocks is a significant concern, e.g., in real-time control applications.

□

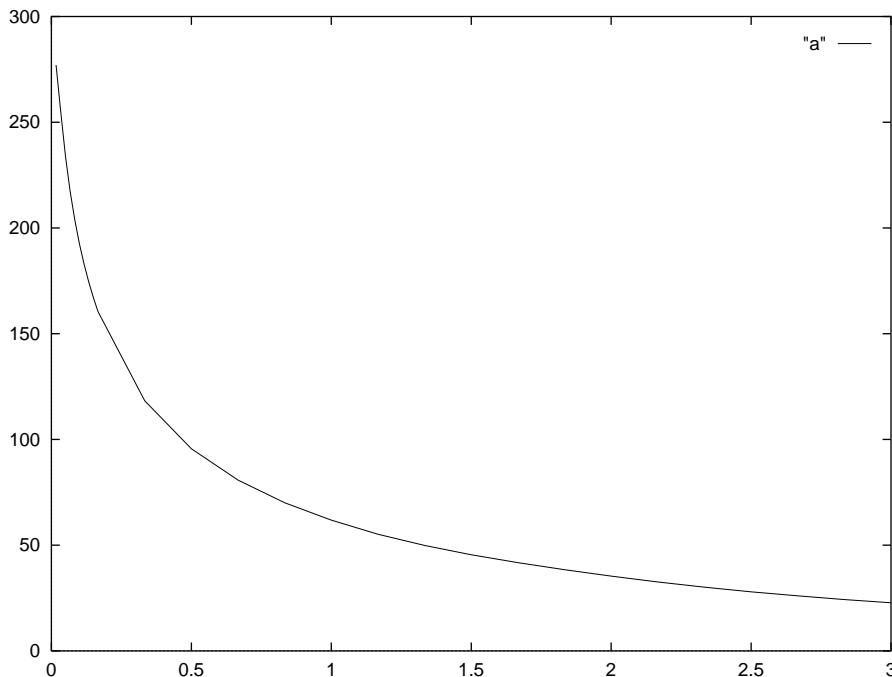


Figure 4: **The Effect of Varying the Mean Deviation of the Clock Tick Interval.** The horizontal axis is the mean deviation of the clock tick interval (i.e., the “fuzziness” of the clock) in milliseconds. The vertical axis is the resulting capacity in bits per second. This graph illustrates the clear tradeoff between clock accuracy and covert channel capacity that is provided by fuzzy time.

Example 4.3 The Effect of Naturally Occurring Noise

In this example we consider the effect of naturally occurring noise. In real systems, this is noise introduced by low level nondeterminism that is not under the control of either the Trojan horse processes or the security kernel. For example, nondeterminacies in the timing of bus accesses due to asynchronous clocks would fall in this category. In terms of our model, naturally occurring noise is the noise introduced by the conditional distribution $P(j | i, k)$.

In this example, we recalculate the capacity for the same situation used in the previous example: The mean clock tick interval is 20 milliseconds, the range of the mean deviation on the clock tick interval is the same, and the minimum bus response time is 10 microseconds. This time however, we will use no natural noise. More precisely, we will use the following definition of the conditional probability distribution $P(j | i, k)$.

$$P(j | i, k) = \begin{cases} 1, & \text{if } j = E[\mathbf{J} | \mathbf{I} = i, \mathbf{K} = k]; \\ 0, & \text{otherwise.} \end{cases}$$

From the point of view of securing a given system, this is, again, a worst-case assumption. For if the covert channel’s capacity is X under the assumption that there is no naturally occurring noise, then the capacity of the covert channel in a real system (where there may be *some* naturally occurring noise) can only be lower.

This results of these calculations are plotted in Figure 5. For comparison, we also include the results of the previous example (Graph 4). Note that the results are nearly identical. In fact,

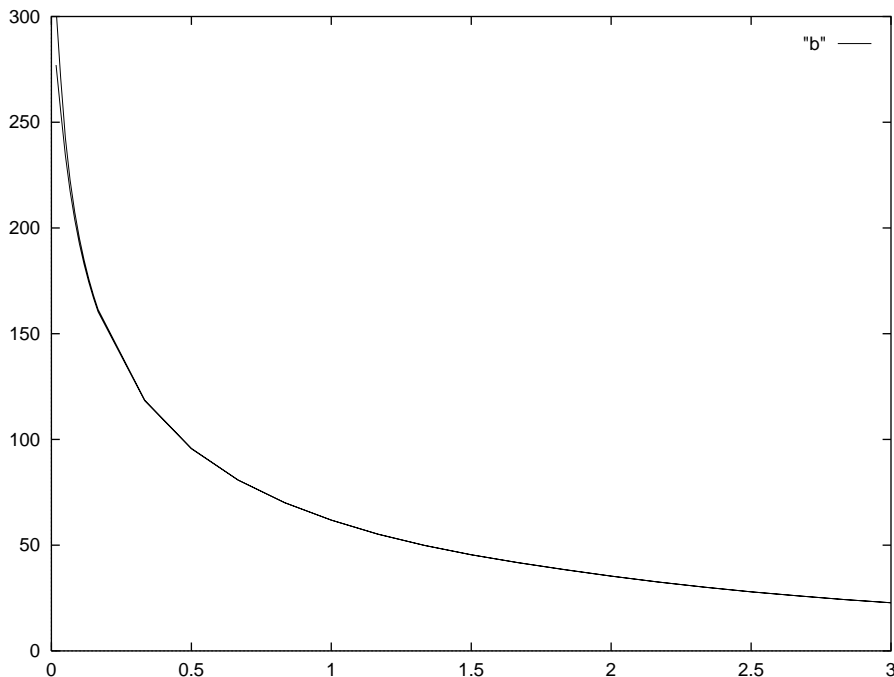


Figure 5: **The Effect of Naturally Occurring Noise.** The horizontal axis is the mean deviation of the clock tick interval in milliseconds. The vertical axis is the resulting capacity in bits per second. Two curves are plotted—one using a model that incorporates background noise and one that does not. The graph shows that for fuzzy time, the additional background noise is, for the most part, negligible.

it is only for small mean deviations that there is any significant difference. This indicates that for the fuzzy time countermeasure, naturally occurring noise is not particularly significant. For this reason, in the remainder of the examples in this section, we will not use any natural noise. \square

Example 4.4 The Effect of Varying the Mean Clock Tick Interval

In this example we explore the effect of varying the mean clock tick interval (i.e., the reference clock rate). We use the same model as in the previous example: there is no naturally occurring noise and the minimum bus response time is 10 microseconds. However, in this example, we calculate the capacity for a variety of mean clock tick intervals. In particular, in Figure 6 we show the channel capacity for five different mean clock tick intervals: 4, 8, 12, 16, and 20 milliseconds. Each mean clock tick interval is represented as a line, where, as in the above examples, the capacity is plotted with respect to the mean deviation on the clock tick interval.

This graph illustrates the kind of tradeoffs that can be made between the (system-controlled) reference clock rate and covert channel capacity. In Section 2 we pointed out that in many applications, performance will be closely tied to the reference clock rate. For such applications, system designers will want to use the shortest possible clock tick interval. Figure 6 shows that even with a clock tick interval as short as four milliseconds, we can achieve significant reduction in covert channel capacity. Further, it shows the additional capacity reduction that can be obtained by moving to, say, an eight millisecond clock tick interval. \square

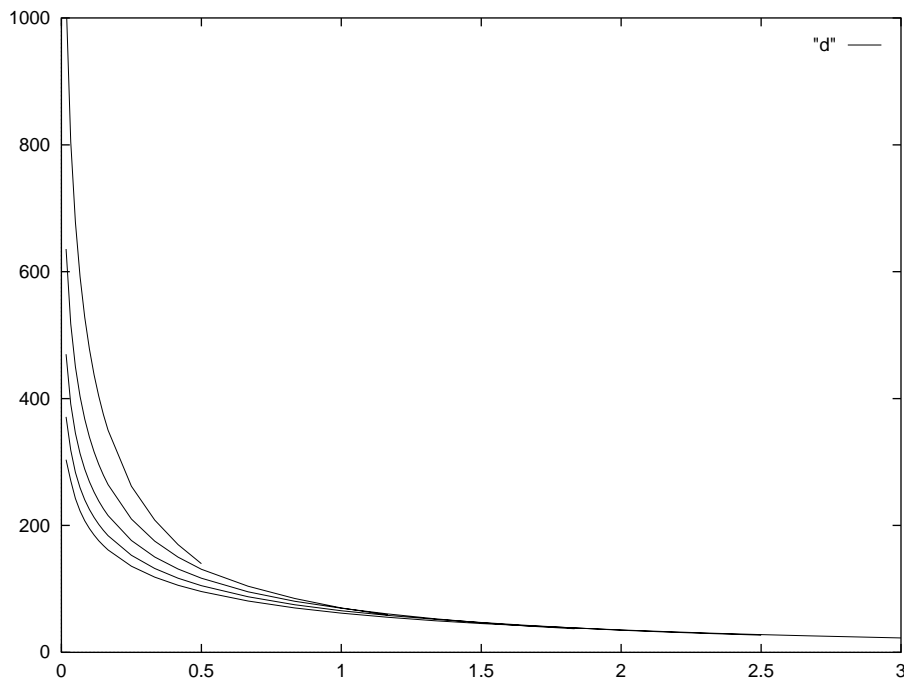


Figure 6: **The Effect of Varying the Mean Clock Tick Interval.** Each curve in this plot corresponds to a different *mean* of the (visible) clock tick interval. Starting with the topmost line, the lines correspond to 4, 8, 12, 16, and 20 millisecond means, respectively. The horizontal axis is the mean deviation of the clock tick interval in milliseconds. The vertical axis is the resulting capacity in bits per second. Note that the lowest curve is the only one that goes all the way out to the right side of the graph. This is due to the particular way that the clock distribution is constructed. For example, due to this construction, it is not possible for a clock with a 2 millisecond mean to have a 1 millisecond mean deviation. Therefore, most of the lines stop short of the right side of the graph.

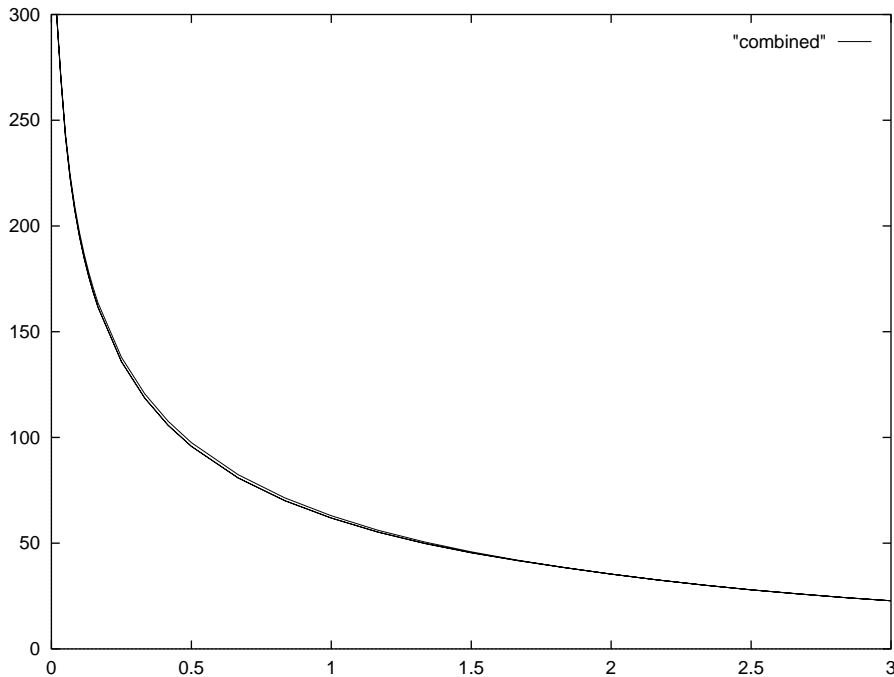


Figure 7: **The Effect of the Bus Response Time.** The horizontal axis is the mean deviation of the clock tick interval in milliseconds. The vertical axis is the resulting capacity in bits per second. There are actually three curves plotted for three different bus response times— $1 \mu\text{s}$, $5 \mu\text{s}$, and $10 \mu\text{s}$. However, the curves for $5 \mu\text{s}$ and $10 \mu\text{s}$ bus response times are too close together to see the difference in this plot. The graph shows that the effectiveness of fuzzy time is largely invariant over widely varying bus response times.

Example 4.5 The Effect of Varying the Bus Response Time

In this example, we consider the effect of faster bus response times on the capacity of the bus contention channel. In particular, we use the same model as was used to generate the results in Figure 5; that is, we use a mean clock tick interval of 20 milliseconds, the mean deviation varies in the same way, and we use no naturally occurring noise. However, in this case, we generate two additional curves using faster minimum bus response times, namely, 5 microseconds and 1 microsecond as opposed to 10 microseconds. The results are graphed in Figure 7. The results from Figure 5 (for the case of no natural noise) are also included for comparison.

Note that the three graphs are nearly identical. This is a surprising result given that the curves are generated for bus response times ranging over an order of magnitude. This leads us to believe that our results are largely invariant under different bus response times and in fact, will remain useful even as bus response times are greatly improved in the years to come. \square

In this section we have used our approach to answer various questions about the effectiveness of fuzzy time as a countermeasure for the bus contention channel. There are, of course, many more questions that could be addressed, both within our current model and beyond. One interesting question that will require some additional mathematical analysis is the following. As mentioned previously (see Section 2), for applications that make heavy use of shared memory, the mean clock tick interval is an important parameter for performance (viz, shorter

intervals are better). Thus, for a given mean clock tick interval, it would be interesting to find the countermeasure distribution that *minimizes* the resulting covert channel capacity. That is, there is no compelling reason to limit our countermeasure distributions (as we have done in the present work) to those produced using the approach taken in the VAX security kernel effort (viz, Equations 8 and 9).⁹ A more general approach is to find the *optimal* distribution given a particular constraint on the mean clock tick interval. Further, finding such optimal distributions (and the associated capacities) over a range of mean clock tick intervals would provide us with the most favorable tradeoff between this important performance parameter and channel capacity. This is a possible topic for future work.

5 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 can be used to completely close the bus-contention channel.

5.1 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 multiple modes of operation ranging between a completely secure mode (i.e., the covert channel has been reduced to zero capacity) and a completely insecure (i.e., no covert channel countermeasures are used) mode. To introduce noise into the bus contention channel, the bus controller randomly switches between these modes during the operation of the system.

Consider the architecture shown in Figure 8. In this architecture, there is a central clock that supplies clock ticks to all processors and to a central random number generator. In our later analysis, we will assume that this clock has a fixed interval of 20 milliseconds (i.e., clock ticks are produced at a rate of 50 per second). The processors will buffer all asynchronous events (using the techniques described by Hu and reviewed in Section 2 above) 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 the fuzzy time approach, this clock will not be fuzzy.

Also on each central clock tick, the random number generator will produce a random number and send it to all of the BICs. This number is the countermeasure parameter that designates which mode the bus should operate in for the present clock interval. We will analyze the scheme for various distributions on the countermeasure parameter below.

Now, what are the modes? All of the modes are built up from two basic modes: *secure mode* and *insecure mode*. 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.

⁹In fact, there are reasons why we may want to *avoid* the VAX security kernel’s approach; see [15].

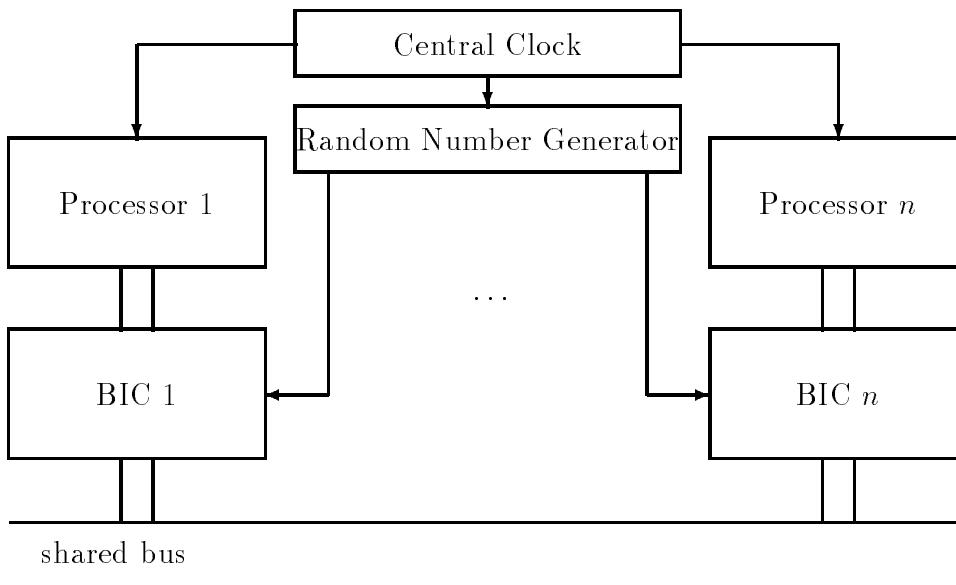


Figure 8: **An Architecture for Probabilistic Partitioning.**

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 $1/n$ of the bus' total capacity (where n is the number of processors on the bus); but, the covert channel capacity will be zero.

For modes in between these two extremes, the BICs alternate between the two basic modes during the clock tick interval. Suppose we have μ different modes. In this case, the set of countermeasure parameters, K , is given by

$$\begin{aligned}
 K &= \{0/(\mu - 1), 1/(\mu - 1), 2/(\mu - 1), \dots, (\mu - 1)/(\mu - 1)\} \\
 &= \{0, 1/(\mu - 1), 2/(\mu - 1), \dots, 1\}
 \end{aligned}$$

and when countermeasure $k \in K$ is chosen, the bus is in insecure mode for the fraction k of the clock interval.¹⁰

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

Note that since (in a typical multilevel secure system) all memory accesses are mediated by the security kernel, it may be possible to implement probabilistic partitioning in software.

¹⁰Note that the alternation between modes in a given mode, k , should be done in such a way that if the sending and receiving processes were to make use of any subinterval of the clock interval, then statistically, the bus will be in insecure mode for that fraction, k , of the subinterval. That is, the alternation must be applied uniformly over the entire interval. For example, we cannot implement mode $k = 1/2$ by statically placing the bus in secure mode during the first half of the clock interval and in insecure mode during the second half. For in this case, the sending and receiving processes might be able to achieve a higher exploitation rate by making use of some subinterval rather than the entire interval. A suitable alternation can be achieved by randomly switching (with a suitable distribution) between the two basic modes during the interval.

However, we expect that in order to achieve sufficiently fast switching between the two basic bus modes, it will be necessary to implement the bus modes and mode switching functionality in hardware.

5.2 Analyzing Capacity under Probabilistic Partitioning

We would like to obtain a tradeoff for probabilistic partitioning between covert channel capacity and some measure of system performance. In our investigations, we have found the most appropriate measure to be a notion we call the bus *availability*. So, before describing our model of the bus contention channel under probabilistic partitioning, we introduce this notion.

The term *availability* is meant to be a measure of how much (i.e., what fraction) of the given resource could possibly be obtained by a given process. It represents the *maximum* fraction of the resource that can be obtained, in the sense that within the limits of its availability, a process may still need to contend with other processes for the resource. Thus, it is likely that a given process will not be able to obtain all of the resource that is “available”.

For example, in a typical operating system, a user may have a disk quota (i.e., upper limit) of 16 megabytes in a system that has 64 megabytes of disk storage. In this case, we would say that the user’s disk availability is 1/4. However, it may also be the case that there are 5 users of this system, all of whom have a disk quota of 16 megabytes. In this case, not all of the users will be able to obtain all of the resource that is available to them—they will have to *contend* for the resource.

Since in our case, the resource allocation scheme is stochastic, we are interested in the *expected* bus availability. We can define this with respect to a given countermeasure distribution, $R(k)$, as follows.

Definition 5.1 (Bus Availability) For a given countermeasure $R(k)$, the *expected bus availability*, $A(R)$, is given by:

$$A(R) = \sum_k R(k)(k + (1 - k)/m)$$

(where m is the number of processors on the bus.) □

To understand this definition note that when the bus is in mode k :

1. the fraction of time that the bus is in insecure mode is k , during which the bus availability is 1 (i.e., all processors are free to access the bus continuously.); and
2. the fraction of time that the bus is in secure mode is $(1 - k)$, during which the bus availability is $1/m$ (i.e., the bus is partitioned among the m processors on the bus).

As with the analysis of fuzzy time, we will assume that the number of processors on the bus is 2, but it is straightforward to generalize our results to larger numbers.

Now we describe our model of the bus contention channel under probabilistic partitioning. In particular, the sets of inputs, I , and outputs, J , will be the same as in our analysis of fuzzy

time; we already described the set of countermeasure parameters, K , in the previous section. It remains to describe the conditional probability distribution on outputs, $P(j | i, k)$, and the distribution on countermeasure parameters, $R(k)$.

In the present analysis, we use a minimum bus response time of 10 microseconds (i.e., the same as what we used for our analysis of fuzzy time) and we consider the case where the number of bus modes, μ , is 256. We use the same distribution on naturally occurring noise as in the analysis of fuzzy time in Examples 4.1 and 4.2. Thus, to describe $P(j | i, k)$, we need only describe the expectation on the output, j , given input i and countermeasure k , which we do as follows.

$$E[\mathbf{J} | \mathbf{I} = i, \mathbf{K} = k] = \text{round} \left(\frac{20 \text{ ms}}{10 \text{ } \mu\text{s}} \left(\frac{1 - k}{2} + \frac{k}{1 + i} \right) \right)$$

To understand this definition, note that $(20 \text{ ms})/(10 \text{ } \mu\text{s})$ is the number of bus requests completed by *all* processors during a single clock tick interval. Also, during a clock interval where countermeasure k is chosen:

1. the bus is in secure mode for a fraction, $(1 - k)$, of the interval, during which the low process gets half of the bus (since there are, by assumption, two processors on the bus); and
2. the bus is in insecure mode for a fraction, k , of the interval, during which the low process gets $1/(1 + i)$ of the bus (where i is the high process' input signal).

Given this definition of $E[\mathbf{J} | \mathbf{I} = i, \mathbf{K} = k]$, we can define $P(j | i, k)$ exactly as in Examples 4.1 and 4.2 by using Equations 5 and 11.

Now, we need to decide on the countermeasure distribution $R(k)$. In fact, to obtain our tradeoff, we will describe a family of distributions, $R_n(k)$, $0 \leq n \leq 1$. Based on some ad hoc analysis and experimentation we chose this family as follows.

- For $n \leq \mu/(2(\mu - 1))$
 - $R_n(0) = 1 - 2n$;
 - $R_n(k) = 2n/(\mu - 1)$, for $k = 1/(\mu - 1), 2/(\mu - 1), \dots, (\mu - 1)/(\mu - 1)$.
- For $n > \mu/(2(\mu - 1))$
 - $R_n(k) = (\mu - 1)(\mu - n')/(n'\mu(\mu + 1))$, for $k = 0/(\mu - 1), 1/(\mu - 1), \dots, (n' - 1)/(\mu - 1)$;
 - $R_n(n'/(\mu - 1)) = 1/\mu$;
 - $R_n(k) = (\mu - 1)(n' + 1)/(\mu(\mu + 1)(\mu - n' - 1))$, for $k = (n' + 1)/(\mu - 1), (n' + 2)/(\mu - 1), \dots, (\mu - 1)/(\mu - 1)$.

where $n' = \text{round}(n(\mu - 1))$.

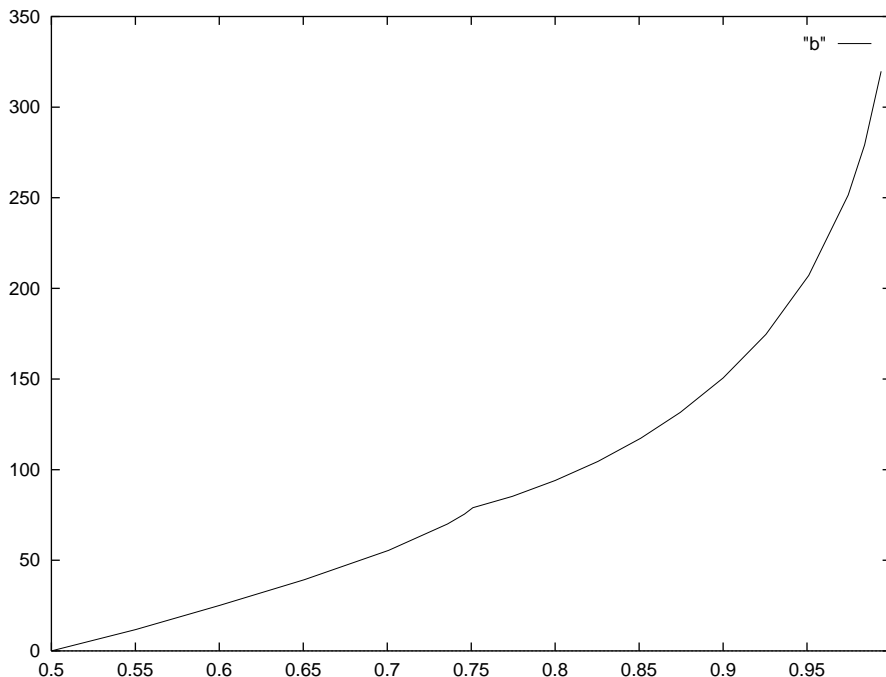


Figure 9: **Analysis of Probabilistic Partitioning.** The horizontal axis is the expected bus availability (as a fraction of the total bus). The vertical axis is the resulting capacity in bits per second. This graph illustrates the clear tradeoff between covert channel capacity and bus availability that is provided by probabilistic partitioning.

That is, for our tradeoff, we chose two different kinds of distributions—one described by the top bullet (for $n \leq \mu/(2\mu - 2)$), and one described by the bottom bullet (for $n > \mu/(2\mu - 2)$). It is straightforward to check that for any real number n , $0 \leq n \leq 1$,

$$\sum_k R_n(k) = 1$$

That is, R_n is a probability function.

We calculated the channel capacity of the channel under the family of countermeasure distributions, $R_n(k)$. The results are shown in Figure 9. In this figure, we plot the covert channel capacity under countermeasure distribution R_n with respect to the *expected bus availability* $A(R_n)$. From the graph, we see that probabilistic partitioning provides us with a trade off between the expected bus availability and covert channel capacity.

Note that there is a small discontinuity at $A(R) = 0.75$. This occurs at the point between the two different countermeasure distributions described above. Most likely, this indicates that the distributions we chose are not optimal in terms of making the best possible use of a given bus availability. It may be useful (and a possible subject for future work) to find such optimal distributions. In that way, we could determine the most favorable tradeoff between bus availability and covert channel capacity provided by probabilistic partitioning. It should also be possible to explore (with respect to the capacity of the bus-contention channel under probabilistic partitioning) the impact of various model parameters as we did for fuzzy time.

6 Comparison of the two Approaches

The following summarizes some of the advantages and disadvantages of the two countermeasures discussed in this paper.

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.
3. Since the effectiveness of fuzzy time relies on not allowing processes to have an accurate clock, its use may not be possible in certain applications; e.g., real time systems.
4. The effect of fuzzy time on bus performance (both throughput and response time) is minimal; whereas the adverse effect of probabilistic partitioning on bus performance, as illustrated in Figure 9 can be significant in cases where covert channels need to be reduced to levels near zero.
5. By reducing the amount of bus contention down to zero, probabilistic partitioning can reduce the channel capacity to zero, whereas fuzzy time cannot.

7 Concluding Remarks

We have identified a class of covert channels, called *counting* channels, and described techniques for analyzing their information-theoretic capacity. We have demonstrated the utility and generality of these techniques by applying them to analyze the bus contention channel under two types of countermeasures in a wide variety of circumstances. For the first of these countermeasures, viz, fuzzy time, our analysis provides the only known upper bound on the channel capacity. Further, our analysis provided precise tradeoffs between covert channel capacity and other system properties for both types of countermeasures.

Now that we have provided precise analyses of the effects of probabilistic partitioning and fuzzy time, 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.

Another avenue for future research is the optimization of countermeasure parameters. For example, it would be useful to answer questions such as “for a given average clock tick interval, what is the best covert channel reduction that can be achieved by fuzzy time?”

References

- [1] 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.
- [2] 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.
- [3] Richard E. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, IT-18(4):460–473, July 1972.
- [4] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.
- [5] James W. Gray, III. On analyzing the bus-contention channel under fuzzy time. In *Proc. Computer Security Foundations Workshop VI*, pages 3–9, Franconia, NH, June 1993.
- [6] James W. Gray, III. On introducing noise into the bus-contention channel. In *Proc. 1993 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1993.
- [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] Wei-Ming Hu. Reducing timing channels with fuzzy time. *The Journal of Computer Security*, 1(3):233–254, 1992.
- [9] Richard A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, 1(3):256–277, August 1983.
- [10] Butler W. Lampson. Protection. In *Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, Princeton University, March 1971. Reprinted in *Operating Systems Review*, vol. 8, no. 1, January, 1974, pp. 18–24.
- [11] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10), October 1973.
- [12] Jonathan K. Millen. Finite state noiseless covert channels. In *Proceedings of the Computer Security Foundations Workshop II*, Franconia, NH, 1989.
- [13] Ira S. Moskowitz and Allen R. Miller. The channel capacity of a certain noisy timing channel. *IEEE Transactions on Information Theory*, 38(4):1339–1344, July 1992.

- [14] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, July 1948. Republished in: C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, IL 1949.
- [15] Jonathan T. Trostle. Modelling a fuzzy time system. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 82–89, Oakland, CA, May 1993.
- [16] John C. Wray. An analysis of covert timing channels. *The Journal of Computer Security*, 1(3):219–232, 1992.