Technical Report

Covert Channel Elimination Protocols

Nick Ogurtsov
Hilarie Orman
Richard Schroeppel
Sean O'Malley
Oliver Spatscheck

Department of Computer Science
Gould-Simpson Building
University of Arizona
Tucson, AZ 85721

Abstract

With the increasing growth of electronic communications, it is becoming important to provide a mechanism for enforcing various security policies on network communications. This paper discusses our implementation of several previously proposed protocols that enforce the Bell LaPadula security model. We also introduce a new protocol called "Quantized Pump" that offers several advantages, and present experimental results to support our claims.

1 Introduction

With the explosive growth of electronic communications over the past few years, secure communication and data security in general are becoming increasingly important. In particular, there is the question of how to prevent sensitive data from falling into the wrong hands. Various security policies and security systems have been proposed that address this issue. One of the simplest and most common security policies is the Bell-LaPadula security model [1], which can be summarized as "no read up, no write down," where "up" is an entity with a high security level and "down" is an entity with a low security level.

This paper examines various proposed methods for enforcing the Bell-LaPadula security model in the context of network communications. We implemented all of the major protocols discussed here. Comparative results derived using TCP over an ethernet are presented in this paper. Our studies are concerned with four parameters:

- Overall throughput — how much data can be sent from the low security side to the high security side per unit of time;
- Covert channel throughput — how much data can an intruder on the high security side send to the low security side per unit of time;
- Space utilization — how much buffer space is required to adequately control the covert channel;
- Reliability — is the communication reliable?

Starting with the simplest and most straightforward protocols, we discuss the basic problems and pitfalls of enforcing one-directional data flow in section 2. We also examine the basic notion of covert timing channels and why they are inherent in reliable communications.

In sections 3, 4, and 5, we move on to more complex protocols: SAFP (Store and Forward Protocol), the Pump, and the Upwards Channel. We discuss their distinct features and analyze covert channel capacities. We then discuss our implementation of these protocols, mention various implementation pitfalls to avoid, and compare experimental performance of these protocols.

In section 6, we introduce a new protocol called "Quantized Pump" that offers a number of advantages. Its design goals were to make it extremely easy to analyze and easy to control. Covert channel bandwidth can set to *precisely* any value desired. We present three different versions of the Quantized Pump that have different throughput/space tradeoffs. We discuss our implementation of all three versions and the results of our experimental measurements of the protocols.

The following table shows a brief summary of features of various protocols we discuss in this paper:

Protocol	Covert Channel	Precise Bound	Reliable	Throughput	Buffer Size
SAFP	Yes	No	Yes	1	—
Pump	Yes	No	Yes	0.9	—
Upwards Channel	No	Yes	No	—	—
Quantized Pump	Yes	Yes	Yes	1	Quadratic
Log. Quantized Pump	Yes	Yes	Yes	0.9	$N \log N$
Linear Quantized Pump	Yes	Yes	Yes	0.4	Linear

Protocol Summary

Most of the column names in the above table are self-explanatory. The Precise Bound column refers to whether or not it is possible to set the covert channel bandwidth to an *exact* value. The Throughput column lists relative throughputs which are expressed as a fraction of SAFP's throughput. The Buffer Size column describes the space complexity of the protocols with respect to the maximum rate at which the sender on the low security network can send data. The "—" signs are present in the entries where the respective parameter is preset at startup time and thus cannot be meaningfully compared to the other entries in the table.

2 Simple Solutions

There are several simple ways to enforce the Bell-LaPadula security policy. Note that we want to be able to do this without modifying any programs on the sender side or on the receiver side. The reason for this is quite simple: we do not trust either the sender or the receiver; after all, that is why we are concerned with enforcing the security policy in the first place. In this section we discuss some of these methods and also discuss the reasons why they are inadequate for most general scenarios.

2.1 Isolation Method

One easy way of enforcing the Bell-LaPadula security model is physically isolating the sensitive information from the outside world by dividing networks into high security and low security classes that are not interconnected. There is no exchange of information between the two classes of networks and thus no danger of unauthorized information flow.

While the simplicity of such a setup is certainly a virtue, it is by no means an ideal solution in most cases. Most applications require *some* information exchange. We should mention however, that this is one of the preferred methods of the military where security concerns outweigh everything else.

2.2 ACK Filter

Another solution is to connect a low security network to a high security network with a special gateway called "ACK Filter." This gateway forwards any data from the low security network to the high security network but only forwards acknowledgments of data receipt — which are necessary for reliable communication — from the high to the low network. This seemingly solves the problem of protecting highly sensitive data in the high security network, since the data cannot actually be sent to the low security network — only the acknowledgments — but there is a problem: covert channels.

An intruder on the high security network can use the times of acknowledgments to signal the low security network. For instance, the intruder might acknowledge some packets immediately and use that as a code for 0, and acknowledge other packets after half a second delay and use that as a code for 1. This is known as a *covert timing channel*, since it is the time that is used to convey unauthorized information.

The problem of covert channels is a well known and relatively well studied problem that appears in various places in computer security. Detailed description and analysis of such channels can be found in [2, 3, 4, 5].

Data acknowledgments are necessary to insure reliable communication and yet they introduce a covert channel, so we have a conflict between security and reliability. Fortunately, there are ways to minimize such covert channels.

2.3 Blind Write-Up

We can sacrifice reliability and use the *blind write-up* method, i.e. eliminate all acknowledgments and hope that all data gets through. This eliminates the covert channel, but now the data is not guaranteed to be received. Some data may be lost due to congestion, it may be sent to a machine that is currently down, the receiving machine may not be fast enough to process the data, etc., so this method is not really acceptable in practice.

It turns out however, that this approach can be improved significantly by introducing a buffer to enhance reliability. This protocol is the "Upwards Channel" or "One-Way Forwarder" and is discussed in section 5.

3 Store and Forward Protocol (SAFP)

The Store and Forward Protocol is the simplest conventional protocol for reliable communication across two networks. Its usefulness for eliminating covert channels is limited at best; we mention it here because it is a useful benchmark against which to compare the performance of more complex protocols.

The idea of this protocol is very simple: we have a gateway between a low security network and a high security network. When the gateway receives a packet from the low side, it stores it in a buffer and sends an acknowledgment back to the low side notifying it that the packet has been successfully received. Then it transmits the packet to the high side and waits for an acknowledgment. When it gets the acknowledgment back, it deletes the message from its buffer. If it doesn't receive the acknowledgment or it receives a negative acknowledgment, it can retransmit the message because it still has it in its buffer. All data except for acknowledgments from the high security network is ignored and is *not* forwarded to the low security network. (Fig. 1)

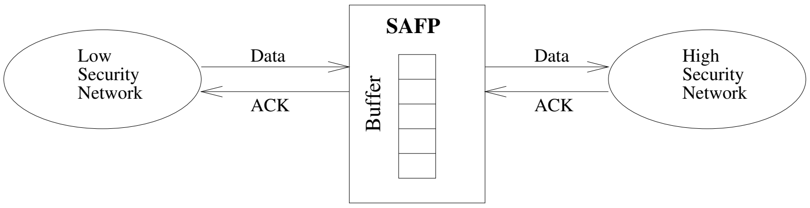

Figure 1: SAFP

3.1 The Covert Channel

The covert channel appears when the SAFP buffer fills up. This will happen if the high side processes messages slower than the low side is sending them. When the buffer is full, SAFP cannot store the messages in the buffer immediately upon receiving them and must wait for space to become available before it can store the message and send an acknowledgment to the low side. In other words, the time it takes to free the buffer space (i.e. the time it takes to receive an acknowledgment from High) is directly related to the time it takes SAFP to send an acknowledgment to Low. The high side can control the rate of acknowledgments to Low by keeping the buffer full and varying the time of its acknowledgments.

Fig. 2 plots acknowledgments versus the time it takes to get that acknowledgment as seen from the point of view of the sender on the low security network. This illustration has no covert channel and hence all of the times are uniformly and randomly distributed near zero.

Fig. 3 shows the same communication but with a covert channel present. In this case, High is trying to send alternating zeros and ones to Low, encoded by "no delay" — **0** and "500 ms delay" – 1. This illustration was produced by running SAFP over TCP Reno.

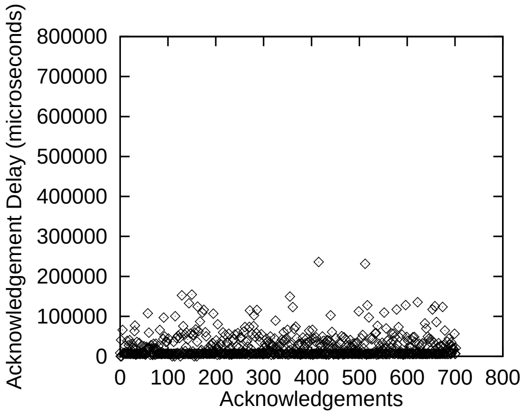

Figure 2: SAFP, no covert channel

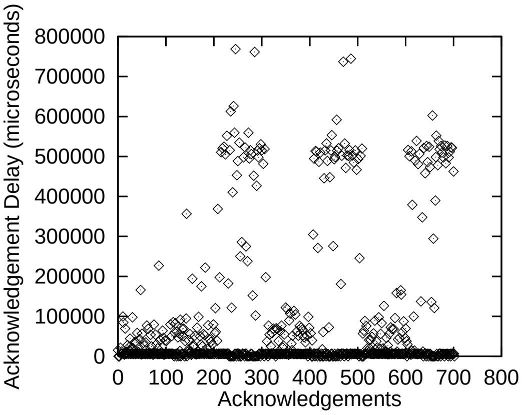

Figure 3: SAFP, covert channel present

The covert channel in SAFP was substantially analyzed in [6], section 3.1. We approached the analysis of this channel from a slightly different perspective, from the perspective of average traffic analysis in our implementation of the protocol, and we were able to get some concrete numbers for the maximum capacity of the covert channel.

Assume the simplest possible situation, where High is trying to signal Low by using long acknowledgment delays for **1** and no delay for **0**. For the sake of simplicity, let's further assume that it takes exactly twice as much time to send a **1** as it takes to send a **0**. (This ratio of 2:1 was found in our experiments to be the lowest ratio where 1s are still reliably detectable.) Then, since the "cost" (time) to send a **1** is different from the cost of sending a **0**, the optimal encoding no longer consists of 50/50 ratio of **0**s to **1**s.

We can find this optimal ratio as follows: let the cost of sending a 0 be C and the cost of sending a **1** be $2C$; also, let the probability of a **0** be p and the probability of a **1** be $q = 1 - p$. The average information transmitted per symbol is $-(p \log p + q \log q)$. The average cost per transmitted symbol is $Cp + 2Cq$, so the amount of information transmitted per unit time is $I = \frac{-(p \log p + q \log q)}{Cp+2Cq}$. The maximum of this expression is found at $p =$61.8%.

From experimental results obtained by running SAFP over TCP Reno, we found that the error per acknowledgment of transmitting a **0** is very low: less than 1%, where as the error in transmitting a **1** is very high: close to 75% (i.e. only one of every four acknowledgments was carrying sufficient timing information to identify it as a code for 1). This is caused by number of factors, but one of the most important ones is the influence of the TCP sliding window. (See Section 3.2.3 for further details on the TCP window.)

Now we can find the exact maximum bandwidth of the covert channel by using the standard formula for binary information transfer in noisy channels (see [7], section 4.2): $I(xy) = \sum_x \sum_y p(xy) \log \frac{p(xy)}{p(x)p(y)}$, where x refers to 1s and y refers to 0s. When we apply this formula to our previous results, we get that the information content of an acknowledgment from the high side is approximately 0.1 bits, i.e. it takes about 10 acknowledgments to send 1 bit of data. The average transmission rate in our experiments was about 3 milliseconds/acknowledgment (see Table 2 and section 3.3 for more details), so the covert channel present in our system was about 33 bits/second.

This is a fairly high bandwidth covert channel but it might be acceptable in some situations. The virtue of SAFP is that it is simple to implement and easy to analyze. The drawback is that it allows a high bandwidth covert channel.

3.2 Implementation

The Store and Forward Protocol was implemented as an xkernel protocol above TCP. Its implementation is as simple as its description. Fig. 4 demonstrates the protocol stack used on the gateway.

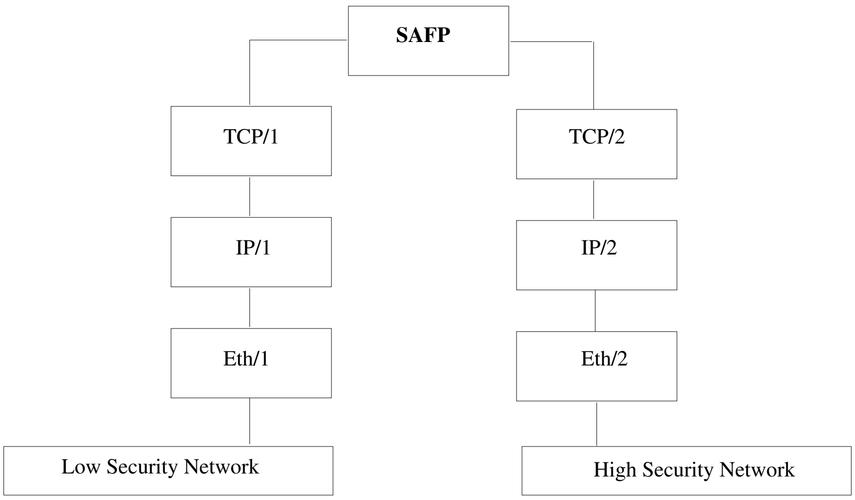

Figure 4: SAFP gateway protocol stack

3.2.1 TCP Gateway

Both TCP and IP protocols were modified to play the role of a gateway. The main modification was to make both of them direct *all* packets up the stack of protocols to SAFP, even if the IP number does not match the IP number of the gateway machine and even if TCP does not know of the port being connected to. They were also modified to "spoof" IP addresses and TCP port numbers. In essence, SAFP is pretending to be High to Low and pretending to be Low to High. This provides a clean and completely transparent paradigm for both High and Low. In fact, neither High nor Low is aware of the fact that there is a gateway in between.

3.2.2 Reliable Open

There is a problem with this spoofing of IP addresses and TCP ports: performing a reliable open. When Low opens a connection to High, the gateway running SAFP will intercept the message, and establish the connection with Low while masquerading as High. Then it will connect to High, pretending to be Low. The problem is in doing this reliably.

Since SAFP is running above TCP, by the time SAFP is notified about an incoming connection, TCP has already established the connection with Low. Moreover, Low's TCP can send data to the gateway immediately afterwards and before SAFP has anything to say about it. Now suppose that when SAFP tries to connect to High, the connection is refused. By this time, Low may have already sent some data to SAFP, since SAFP already established a connection with Low, and that data already had been acknowledged by gateway's TCP. Reliability is lost because the data that was acknowledged will never be delivered to High.

There is not much that can be done about this problem as long as SAFP is running above TCP. It is possible to prevent Low from sending any data until SAFP establishes a connection to High by setting the size of the TCP receive buffer to zero. But even the fact that the connection has been established is already a violation of a reliability issue. In our implementation we decided that since it is impossible to achieve a completely reliable open without running at TCP level, we might as well use it to our advantage to speed up data transfers by accepting the data from Low at the highest possible rate while the connection to High is being opened. By the time SAFP finally establishes the connection to High, Low may have already sent many kilobytes of data to SAFP. If it is a small data transfer, Low may even be completely done before High is even ready to receive any data, then SAFP turns in the Big Buffer scheme of [13].

3.2.3 Effects of the TCP Sliding Window

TCP's sliding window comes in handy in combating covert channels. The sliding window allows the sender to send several packets in a row, without waiting for any acknowledgments, until the window is filled up. Likewise, the receiver can acknowledge several packets at once.

The window helps to add timing noise to the covert channels in the following way. When the sender sends several packets back-to-back and the receiver acknowledges them all with one ACK, we have only one piece of useful timing information for several packets. For every group of packets sent, only one will get an acknowledgment that carries useful timing information. This is conceptually the same thing as timing noise. Hence, with a standard TCP window we were able to get the figure of about 75% timing noise in our tests, i.e on average only 1 out of every 4 packets got an acknowledgment with useful timing information. (This result will vary with the network environment and the details of the TCP implementation.) A direct corollary of the observation is that by increasing the size of the TCP window — applications have full control over the receive buffer within TCP — we may increase the amount of noise on the covert channel, which in turn will decrease the amount of information that can be sent over the covert channel per packet of data.

3.2.4 Gateway Performance

If the SAFP protocol is running on a workstation and instead of a dedicated gateway, the workstation has to be at least twice as fast as the sending and the receiving workstations to achieve optimal performance. SAFP has to do roughly twice as much work as either the sender or the receiver in the same time period: SAFP has to receive the data from one network and simultaneously send the data to the other network, thus employing two TCP/IP stacks at once, while both the sender and the receiver use just one TCP/IP stack at the same time.

3.3 Experimental Results

Our experimental tests were run on Intel 80486 processors with 66Mhz clock speed running Mach operating system version 3.0 [8]. The protocols were running in the xkernel environment version 3.2 [9]. The xkernel processes which contained the gateway protocols were running at a high priority level. The sender and receiver protocols were running on identical machines with an identical setup as the gateway.

Data Transferred	360KB	720KB	1.4MB
Transfer Time	37 s	71 s	152 s
Transfer Rate	9.7 KB/s	10.1 KB/s	9.2 KB/s
Buffer Size	102400 bytes	102400 bytes	102400 bytes
Average Packet Size	1382 bytes	1483 bytes	1477 bytes
Average Delay	25 ms	28 ms	31 ms

Table 1: SAFP, covert channel present

Table 1 summarizes the performance of SAFP with a covert channel present. The covert channel was created by sending a random set of bits. It used a 250 millisecond delay on an ACK to signal a **1** and no delay to signal a **0**. Such a long delay was chosen to make the behavior of the protocols with a covert channel more pronounced and easier to analyze. As a point of reference, the minimum useful delay for covert channel transmissions seemed to be around 25-50 milliseconds.

The data rates are quite low — only about 10 Kilobytes/second. Such low rates are due to several factors: a fairly slow processor, inefficiencies caused by running the protocols on Mach OS as a user process, and the fact that a covert channel is present. We do not consider such slow data rates to be a big drawback in our experiments, since all the rest of the protocols we implemented and examined were running in exactly the same environment and thus we can still effectively compare their performance.

The TCP send and receive buffers were the standard 4K buffers. The test application was giving TCP a continuous stream of data blocks, each 1024 bytes long. By comparison, the average packet size of actual TCP packets that were going onto the wire was larger, as shown on the table, all due to TCP repackaging. Still, the ratio of the TCP buffer sizes to the size of the packets was nearly 3:1, which means that the arguments of the previous section about the influence of the TCP window on the covert channel bandwidth fully apply here.

Data Transferred	360 KB
Transfer Time	12 s
Transfer Rate	30 KB/s
Buffer Size	102400 bytes
Average Packet Size	1334 bytes
Average Delay	3 ms

Table 2: SAFP, no covert channel

The next table, Table 2, presents the summary of SAFP behavior without a covert channel. The covert channel introduced a factor of three slowdown into the channel as compared to Table 1. We were unable to get much above the rate of 30K/s for data transfer in our experiments.

	No Covert Channel	Covert Channel Present
Transfer Time	2 s	29 s
Transfer Rate	180 KB/s	12 KB/s
Data Transferred	360 KB	360 KB

Table 3: No gateway between sender and receiver

Finally, Table 3 provides a basis for comparison of gateway protocols to communication between the sender and receiver when there is no gateway in between. Our SAFP protocol was 6 times slower than optimal without any covert channels but only 1.3 times slower than optimal with the covert channel present.

This shows that in our tests when a covert channel is present, much of the slowdown is due to artificial delays involved in creating a covert timing channel and not necessarily to the gateway software.

Using native Mach OS TCP in place of the xkernel TCP gave virtually identical performance.

4 The Pump

The Pump protocol (M. Kang and I. Moskowitz [6, 10, 11]) substantially reduces covert channel bandwidth from that of SAFP.

The idea is to improve the throughput and reduce the covert cannel bandwidth by using a historic average of High's rate of acknowledgments as the rate of sending acknowledgments to Low. The Pump consists of three parts: trusted low process, trusted high process, and the communication buffer.(Fig. 5)

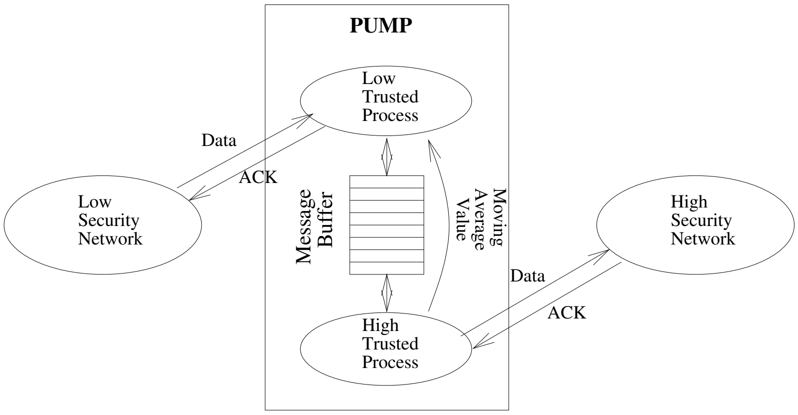

Figure 5: Pump

When the trusted low process receives a message from Low, it will insert the message into the buffer, consult the current value of the moving average of the last m acknowledgment times from High, use a probabilistic function to add some random timing noise to the channel, then delay the acknowledgment by that amount of time. By doing this, the low trusted process forces the low side to send messages at the rate that the high side can receive them but it does that indirectly and with random noise.

The trusted high process of the Pump simply sends messages from the buffer to the high side and computes the moving average based on the acknowledgment times.

4.1 Covert Channel

The Pump is a big improvement over SAFP in terms of covert channel reduction. The moving average increases the time it takes for High to go to a different rate of acknowledgments, thus slowing down the covert channel. The probabilistic function that introduces randomness into the channel also helps to reduce covert bandwidth.

However, despite being an improvement over SAFP, the Pump still has covert channels. Fig. 6 illustrates one such channel. The communication is exactly the same as illustrated on Fig. 3 for SAFP but now we have the Pump protocol in place of SAFP. Here, the size of the moving average was chosen to be deliberately low (30 packets) as compared to the number of packets per covert bit (approximately 100 packets), thus allowing a big and pronounced covert channel, in order to better illustrate its appearance.

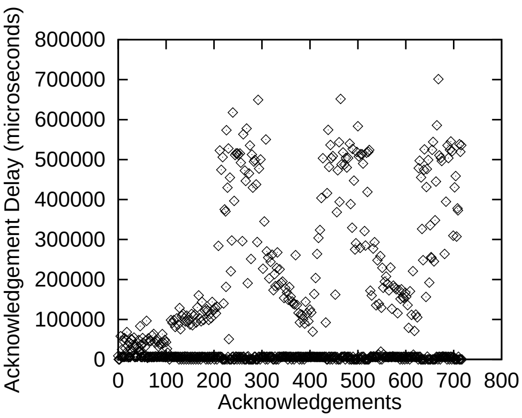

Figure 6: Pump, covert channel present

So the question is: what is the covert channel bandwidth of the Pump? Kang and Moskowitz [6] have analyzed three possible exploitation strategies in terms of dependency on two variables: the number of packets in the moving average and the size of the buffer. Unfortunately, this does not say much about the total covert channel bandwidth possible with the Pump, since it is entirely possible that other strategies might be found or various combinations of these strategies might yield higher covert channel bandwidths.

There is a substantial amount of information flowing from the high trusted process to the low trusted process on every packet. It consists of two pieces:

- Buffer full/not full — this is necessary to make communication reliable;
- The moving average — an n bit quantity of information passed from the high trusted process to the low trusted process on every packet.

This multi-bit quantity is flowing from the high security network to the low security network on every acknowledgment from the Pump! This is a violation of the Bell-LaPadula policy because this information originates in the high security network and makes it all the way down to the low security network. The violation is diminished because High cannot arbitrarily change this multi-bit value and has to follow certain rules in changing it, but the fact remains: many bits of data flow from High to Low on every packet. The random noise helps to reduce the amount of information flowing down but certainly does not eliminate it completely. Because we have this multi-bit quantity flowing from High to Low, we can conclude that the covert channel is *no greater* than that many bits per packet but that is too high an upper bound to be of any use. (A corollary to the above observation is that the more precision we have in the clock, the more bits flow from the trusted high process to the trusted low process.) The exact bound is hard to calculate in general because we have a continuous channel with a gradual, limited change behavior. The authors of the Pump protocol simply considered several possible attacks, concluded that the covert channel bandwidth can be adequately controlled for those attacks by setting the size of the moving average and the size of the buffer, and left it at that.

In section 6, we propose a new protocol based on the Pump which is easily analyzable, has an easily controlled covert channel bandwidth, and can be proven to have that covert channel bandwidth and no more.

4.2 Implementation

The Pump was implemented as an application layer protocol over TCP in the xkernel environment. The original Pump was described as having the trusted high process and the trusted low process as separate entities and we talked about them as such in the preceding section. Logically they are a part of the same protocol, so in our implementation we made them into two separate threads instead of separate processes. Communication between the trusted high process (thread) and the trusted low process (thread) is accomplished by a shared variable which conveys the value of the moving average from the trusted high process to the trusted low process.

Fig. 7 illustrates the protocol stack used for the pump.

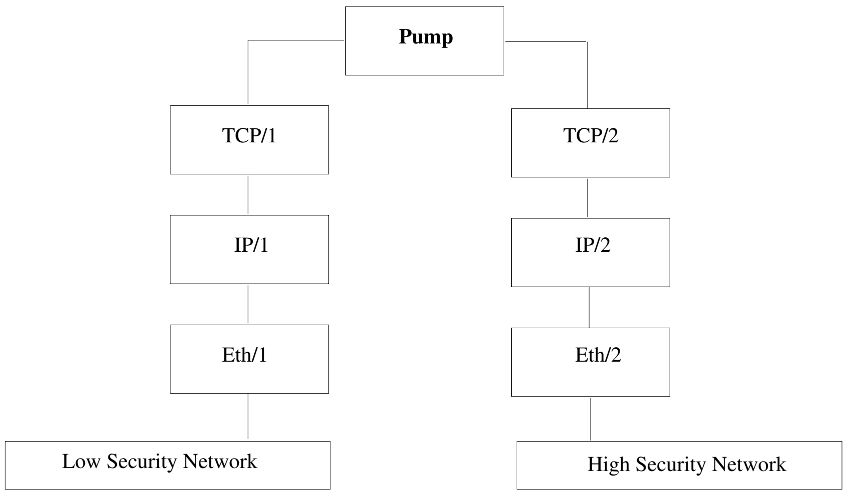

Figure 7: Pump gateway protocol stack

The Pump runs above TCP. Both TCP and IP protocols were modified as with SAFP. Since TCP automatically acknowledges all packets that it receives *before* it forwards them up to the application layer, the Pump has to rely on setting the size of the sliding window for flow control, i.e. opening the TCP window instead of explicitly sending an acknowledgment to Low.

The original protocol described in [6] does not deal with the fact that the Pump may run over a sliding window protocol such as TCP. As we pointed out in our discussion of SAFP, the sliding window may actually introduce additional timing noise into the covert channel and is thus a useful artifact that should not be neglected.

TCP's sliding window is denominated in bytes rather than messages. Extra care needs to be taken in opening the window in order to avoid introducing a new covert channel. The window should be opened either by some fixed constant number of bytes or by the number of bytes of the last message, not by the number of bytes that High just acknowledged. Otherwise a new covert channel of up to 16 bits per packet is opened.

4.3 Experimental Results

Tables 4, 5, and 6 summarize behavior of our implementation of the Pump with various parameters changing. In all three cases there is a covert channel present. As with SAFP, **1** was encoded with a 250 millisecond delay and **0** was encoded with no delay.

Data Transferred	360 KB	720 KB	1.4 MB
Transfer Time	41 s	78 s	173 s
Transfer Rate	8.8 KB/s	9.2 KB/s	8.1 KB/s
Buffer Size	102400 bytes	102400 bytes	102400 bytes
Moving Average	60 packets	60 packets	60 packets
Average Packet Size	1362 bytes	1373 bytes	1379 bytes
Average Delay	23 ms	30 ms	31 ms

Table 4: PUMP, covert channel present

Comparing Table 1 to Table 4, we see that the Pump's throughput is lower than SAFP by 10% to 12%. It is also 1.4 as slow as the no-gateway throughput. This is an acceptable slowdown, since the Pump is a substantial improvement over SAFP in terms of reducing the covert channel bandwidth.

Moving Average	60 packets	120 packets	240 packets
Transfer Time	41 s	39 s	38 s
Transfer Rate	8.8 KB/s	9.2 KB/s	9.5 KB/s
Data Transferred	360 KB	360 KB	360 KB
Buffer Size	102400 bytes	102400 bytes	102400 bytes
Average Packet Size	1362 bytes	1388 bytes	1361 bytes
Average Delay	23 ms	26 ms	26 ms

Table 5: PUMP, covert channel present

In Table 5 we show what happens when the number of packets in the moving average is increased. Contrary to what we expected, this slightly improved the performance. The small improvement is due to the fact that with longer moving average it takes a lot longer to slow down the connection. Without changing the size of the covert bits, the value of the moving average never reaches the highs that it did with a shorter moving average. This behavior is particularly pronounced at the very beginning of a connection and exists only until enough packets have gone through the gateway to construct a complete moving average. This phenomenon is only noticeable if the size of the moving average is comparable to the amount of data transfered through the gateway. When more data is transfered, the phenomenon disappears.

Buffer Size	102400 bytes	204800 bytes	309600 bytes
Transfer Time	41 s	37 s	35 s
Transfer Rate	8.8 KB/s	9.7 KB/s	10.3 KB/s
Data Transferred	360 KB	360 KB	360 KB
Moving Average	60 packets	60 packets	60 packets
Average Packet Size	1362 bytes	1375 bytes	1368 bytes
Average Delay	23 ms	14 ms	6 ms

Table 6: PUMP, covert channel present

Table 6 shows the behavior of the Pump when the buffer size is changed. As expected, the throughput of the protocol is increased with the increase in buffer space. This is primarily due to the fact that our implementation allows Low to send data while the Pump establishes communication with High. The more data can be sent through during that time, the better the performance. Again, this effect is really noticeable only when the size of the buffer is comparable to the amount of data that goes through the gateway for any particular connection.

5 The Upwards Channel (One-Way Forwarder)

The Upwards Channel protocol is a protocol that eliminates all covert channels by sacrificing some reliability. This protocol is similar to "blind write-up," except here the use of a buffer and preset rate control mechanisms provide some assurance of delivery. It was first published by David Goldschlag [12]; a similar protocol was implemented by Andy Bavier under the direction of Sean O'Malley at the University of Arizona prior to the publication of [12]. This protocol is also similar to the Big Buffer scheme of [13] except in this case it does not require any trusted components.

The idea of this protocol is very simple: to eliminate all covert channels, we physically isolate the high and low network and put a special gateway in between. This gateway comes in two parts: one computer is on the low network and the other computer is on the high network. They are connected by an *optical link*, which is a commercially available fiber-optic based device in which data can only flow in one direction.(Fig. 8)

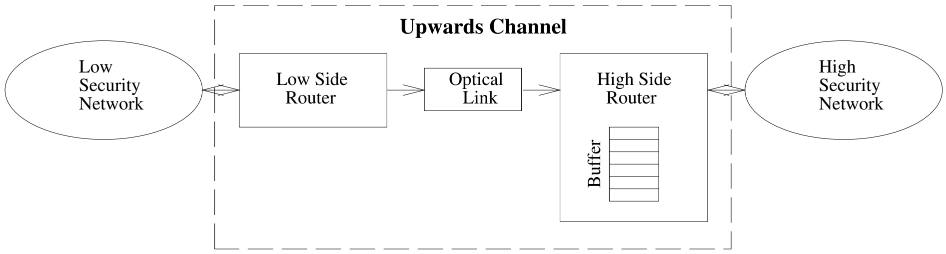

Figure 8: Upwards Channel

When the low side of the gateway receives a message from Low, it simply forwards it on to the high side via the optical link and acknowledges receipt of the packet. The high side of the gateway places the message into a buffer and forwards messages from that buffer to High.

The obvious problem with this protocol is that if a message gets corrupted or the buffer becomes full, some data will be lost with no possibility of automatic recovery. However, this may be an acceptable protocol if the high receiver is sufficiently reliable and sufficiently fast to prevent the gateway's buffer from filling up.

5.1 Downgraders

Goldschlag further proposed to improve the protocol by providing some downward flow of information through *downgraders*. A downgrader is a trusted device that is located between the high and low sides of the gateway and provides the only flow of information from High to Low.(Fig. 9)

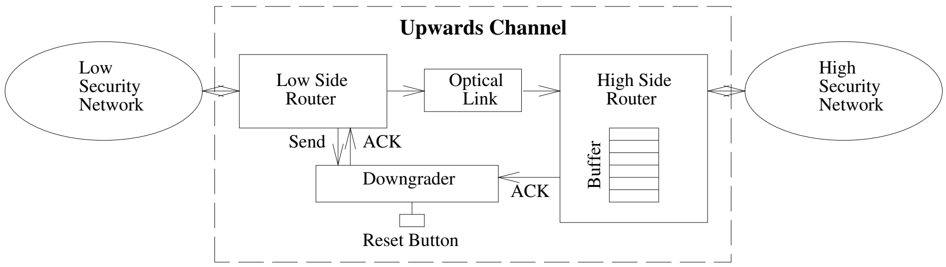

Figure 9: Upwards Channel with a Downgrader

One proposed flavor of a downgrader, called "capacitor," works as follows: every time the low side of the gateway sends a packet to the high side, it signals the capacitor. When the high side receives an acknowledgment for that packet, it also signals the capacitor. At the end of a predefined time period, the capacitor will signal the low side that the packet has been received by High. If the packet has not been acknowledged by the end of the predefined time period, the capacitor will shut down all communication and a manual reset will be required. This allows Low to request information from High at a predefined rate in such a way that it does not carry any timing information (or rather almost no timing information — a lack of a signal certainly does carry a small amount of timing information, although it also causes an immediate termination of operation).

Another flavor of a downgrader proposed by Goldschlag can provide some flow control. This downgrader works just like a capacitor but it also provides flow control by *delaying* the signal to the low side of the gateway by the moving average of the last m acknowledgment times from High instead of signaling at the end of a predefined time period. Again, if the packet has not been acknowledged by the end of the predefined time period, the capacitor will shut down all communication and a manual resetting will be required.

Both of these schemes of downgraders work reasonably well if the communication is maintained within a certain predefined bandwidth. Outside of this bandwidth communication is not possible.

5.2 Implementation

The Upwards Channel was implemented as two protocols: the high gateway protocol and a low gateway protocol, both above TCP. The communication over a one way link was accomplished using the UDP protocol.

A virtual protocol called "RC" or "Rate Control" protocol was inserted between the low gateway Upwards Channel and UDP in order the keep the rate of messages within the capacity of the optical link. Fig. 10 illustrates this configuration.

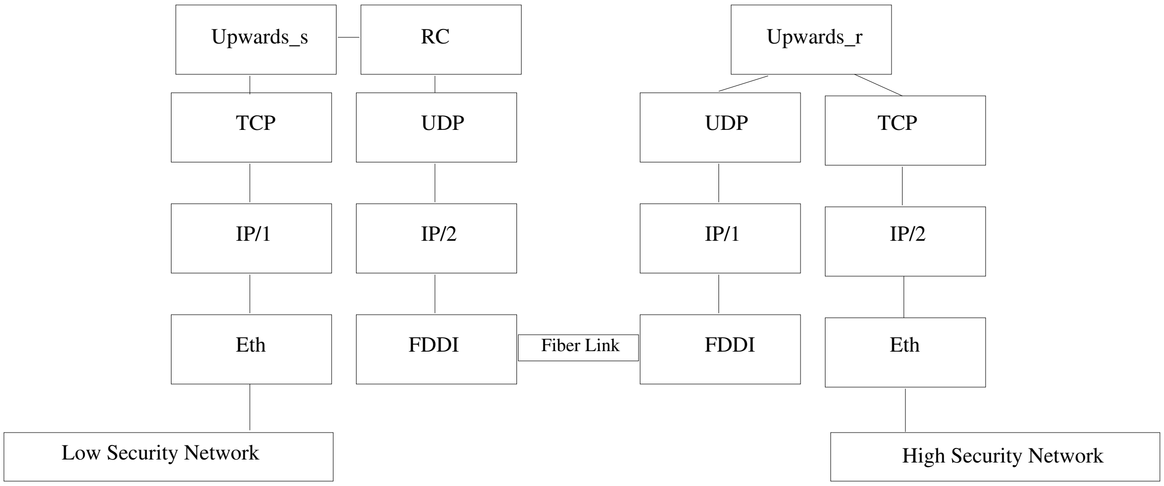

Figure 10: Upwards Channel protocol configuration

The TCP and IP protocols were modified as with SAFP and the Pump in order to provide similar services.

5.3 Experimental Results

Our experiments confirmed that the Upwards Channel is a viable protocol that meets its design goals. We weren't able to derive any concrete performance measurements of a real setup due to the lack of necessary hardware (one-way fiber link, etc.). Our simulations on a Sun SPARC showed sustained data rates comparable to those of SAFP and the Pump: on the order of 10 KB/second.

6 The Quantized Pump

The Quantized Pump proposed here is a hybrid between the Pump and the downgraders. Its design goals are: to make it easily analyzable and controllable without sacrificing any performance. Having a provable bound on the size of a covert channel is a priority in the design of any secure system. Here we present three versions of the Quantized Pump: the straightforward version, the logarithmic, and the linear Quantized Pumps. The latter versions have certain performance advantages/tradeoffs.

The basic idea is simple: if we want to have a covert channel of exactly so many bits per second and no more, then let the trusted high process send exactly that many bits per second to the trusted low process. If there is no other information flowing from the high trusted process to the low trusted process, then there will be no other information flowing from the high security network to the low security network. The crux of the algorithm is then *finding the best use for the bits that are allowed to flow between the high trusted process and the low trusted process.*

On the surface, the Quantized Pump looks just like a regular Pump: it has a low trusted process, a high trusted process, and a buffer in between.(Fig. 11)

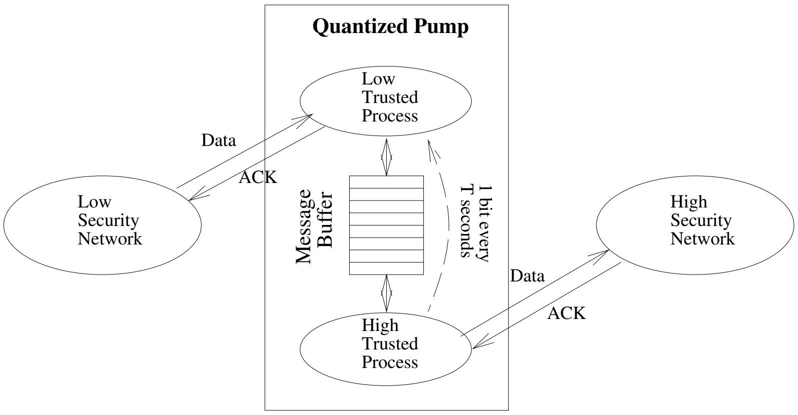

Figure 11: Quantized Pump

There are two important differences from the pump. The buffer does not have a fixed size.[1] This is important because it allows us to completely isolate the low trusted process from the buffer. Such isolation prevents a possible covert channel where both the high and the low trusted process are competing for access to the buffer, which may be measurable from the outside of the gateway. In the regular Pump, the trusted low process has to have a better access to the buffer to make sure that it doesn't overflow; also, one possible implementation of the Pump involves the trusted low process actually measuring how often the trusted high process takes the data out of the buffer in order to figure out the value of the moving average.

But the main difference is that the trusted high process communicates with the trusted low process in a very restricted way: at the end of every predefined time period T, the trusted high process sends exactly one bit to the low trusted process. That bit carries the following meaning: either raise the rate of acknowledgments to Low by a fixed constant rate R bytes/second or lower the rate of acknowledgments by R bytes/second.

How does the trusted high process decide whether to raise or lower the rate of acknowledgments? The trusted high process keeps track of the number of bytes that High acknowledged during this time period. By dividing this number by T, the trusted high process gets the current average rate in bytes/second. If this average rate is greater than the current rate that the low trusted process uses (remember the high trusted process is allowed to get data from the low trusted process; it is the other direction that is restricted), then the high trusted process will signal the low trusted process to raise the rate of acknowledgments by R. Otherwise, it will signal the low trusted process to lower the rate of acknowledgments by R.

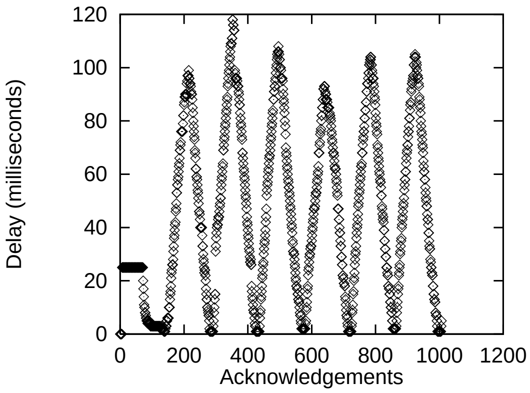

Figure 12: Pump delays

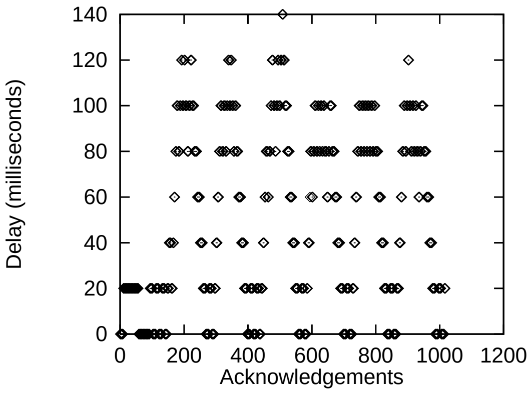

Figure 13: Quantized Pump delays

The above two figures graphically illustrate the difference between the Pump and the Quantized Pump acknowledgment rates. Fig. 12 shows the delays generated by the moving average construction of the pump.

Fig. 13 shows how the Quantized Pump constructs the delays. There are distinct levels present in this

[1] We will show later that despite this characteristic, there is a maximum size that the buffer will never exceed.

illustration. The vertical distance between each two adjacent levels is R and the jump from one level to the next happened every T seconds.

The protocol is called "Quantized Pump" because communication between the trusted high process and the trusted low process happens on a once-per-time-interval basis — quantum, hence the name — Quantized Pump.

6.1 Covert Channel

The number of bytes acknowledged per unit time is entirely within High's control, therefore High can directly affect what the high trusted process will send to the low trusted process inside the Quantized Pump. In other words, we have a covert channel. Fortunately, from the way we setup this protocol we know *exactly* what the bandwidth of that covert channel is, since we have exactly one bit passed from the trusted high process to the trusted low process every T seconds, i.e. the bandwidth of the channel is exactly $(1/T)$ bits/second.[2]

6.2 Maximum Buffer Size

Previously we said that the size of the buffer is unbounded. Sadly, due to current technical limitations, unlimited information storage is not readily available. It turns out however, that despite the absence of any explicit restrictions on the buffer size, the size of the buffer is nonetheless strictly bounded.

Let L_{max} be the maximum rate (in bytes/second) at which Low can send information to High via the Quantized Pump. The largest amount of buffer space is needed when Low starts out at the rate of L_{max} and High refuses to accept anything. Clearly, *at least* this much buffer space might be needed, since this a possible situation.

To show that this is also the *largest* amount of buffer space that will ever be needed, we make the following observation: if High lowers the rate of acknowledgments by some amount and then raises the rate of acknowledgments by that same amount, then the occupied buffer space will not change. Here is the proof: let L be the initial rate of information that Low is sending and let H be the initial rate of acknowledgments from High. The amount of buffer space needed for the first time interval is: $(L-H)T$. Suppose High lowers the rate of acknowledgments by iR, where i is some integer constant. Then the total amount of buffer space needed before the Low and High rates will be balanced out is:

$$(L-H)T + (L-R-H)T + (L-2R-H)T + ... + (L-iR-H)T$$

This can be rewritten as:

$$T\sum_{j=0}^{i}(L-jR-H) = (i+1)(L-\frac{1}{2}iR-H)T \tag{1}$$

If High now raises the rate of acknowledgments by iR, then the amount of buffer space that will free up in the first time interval is: $(H+iR-L)T$ and the total amount of buffer space that will free up before the Low and High rates will be balanced out is:

$$T\sum_{j=0}^{i}(H+jR-L) = (i+1)(H+\frac{1}{2}iR-L)T \tag{2}$$

Expressions (1) and (2) are equal and opposite in value, which means that if High lowers the rate of acknowledgments by some amount and then raises it by the same amount, the used space in the buffer will not change. The same argument holds true in the other direction, i.e. if High raises the rate of acknowledgments by some amount and then lowers it by the same amount, the used space in the buffer will not change.

Now we are back to our original argument that the largest amount of buffer space ever needed is when Low starts out at the rate of L_{max} and High refuses to accept anything. High cannot lower the rate of

[2] This is only strictly true if we assume that other factors such as time sharing of the same processor, buffer sharing, device sharing, etc. between the trusted high process and the trusted low process do not introduce any further covert channels. Real-time operating systems can significantly help with these issues.

acknowledgments any further, and so the only thing that High can do that will affect the size of the buffer is raise the rate of acknowledgments. But this will free some space in the buffer. If High subsequently lowers the rate of acknowledgments, it will not use up any more buffer space than we had originally by the lemma above. Therefore, we can conclude that the largest amount of buffer space ever needed is when Low starts out at the rate of L_{max} and High refuses to accept anything.

This maximum buffer size now is easy to calculate. It is simply:

$$B_{max} = L_{max}T + (L_{max} - R)T + (L_{max} - 2R)T + ... + RT + 0 = \frac{1}{2}(L_{max}/R + 1)L_{max}T \tag{3}$$

For instance, if the maximum data rate from Low, L_{max}, is 100 Kbytes/second, R is 10 Kbytes/second, and the covert channel size is 1/10 bits/second, i.e. T is 10 seconds, then the buffer space needed is only 5.5 Mbytes. This is the *maximum* buffer space that will possibly be needed. On average, buffer space usage will be much smaller.

6.3 Logarithmic Quantized Pump

The previous example illustrated a buffer size that is acceptable in many applications. However, the maximum size of the buffer grows quadratically with L_{max}, i.e. it is $O(L_{max}^2)$, which may be tolerable but not desirable. We can improve this behavior without introducing any additional covert channel bandwidth by using a cleverer encoding in communication between the high trusted process and the low trusted process.

The Logarithmic Quantized Pump works exactly like the Quantized Pump, except the single bit that is passed from the trusted high process to the trusted low process once every T seconds is interpreted differently. The signal to raise the rate can be treated just as before, since it does not increase the buffer space used.

The signal to lower the rate is now interpreted differently: if the immediately prior signal from the high trusted process was to raise the rate of acknowledgments, then the low trusted process will now lower the rate by R; if the prior signal from the high trusted process was to lower the rate of acknowledgments, then the trusted low process will lower the rate by twice the amount it was lowered previously. For instance, the sequence of six bits: "raise, lower, lower, lower, raise, raise" will result in the following adjustments to the acknowledgment rate by the trusted low process: "raise the rate by R, lower the rate by R, lower the rate by $2R$, lower the rate by $4R$, raise the rate by R, raise the rate by R."

This seemingly trivial change allows us to lower the rate from L_{max} to zero in only $\log_2(L_{max}/R)$ steps instead of (L_{max}/R) steps as before, allowing saving buffer space. The complete expression for buffer space now looks like this:

$$B_{max} = L_{max}T + (L_{max} - R)T + (L_{max} - R - 2R)T + (L_{max} - R - 2R - 4R)T + ... + 0$$

or

$$B_{max} = T(L_{max} + \sum_{i=0}^{\psi}(L_{max} - R\sum_{j=0}^{i} 2^j))$$

where ψ is the number of steps necessary to get down from L_{max} to zero. This expression has a closed form:

$$B_{max} = T(L_{max} + \psi(L_{max} + R) - 2R(2^{\psi} - 1)) \tag{4}$$

To the first approximation, $\psi = \log_2(L_{max}/R)$ and so the expression turns into:[3]

$$B_{max} = T(L_{max} + (\log L_{max} - \log R)(L_{max} + R) - 2(L_{max} - R))$$

The buffer size now grows as $O(L_{max} \log L_{max})$ instead of $O(L_{max}^2)$. Despite this improvement, we did not introduce any additional covert channel bandwidth because we are still sending exactly one bit per time period T from the high trusted process to the low trusted process but now we are simply using this bit a little more cleverly. The drawback of Logarithmic Quantized Pump is that this improvement in buffer space comes from sacrificing some throughput. Fortunately, this sacrifice is quite small: 10% or less. For exact performance numbers see section 6.7.

[3] We say "to the first approximation" because the expression is actually $\psi = \log_2((L_{max}/R) + 1)$ and is exact only if $L = R$ or $L = 3R$ or $L = 7R$ or in general if $L = (2^i - 1)R$; if L does not have such a convenient form, the last term of the summation will not be complete and some approximation will be necessary.

6.4 Linear Quantized Pump

If we consider how the Logarithmic Quantized Pump got its buffer space advantage over the regular Pump, we will realize that we simply converted some percentage of the available bandwidth into less buffer space. The natural question then is: can we save some more buffer space at the expense of sacrificing additional bandwidth, without introducing any extra covert channel capacity? In this section we demonstrate two possible ways to accomplish exactly that.

The Linear Quantized Pump works in exactly the same way as the regular Quantized Pump but the bit passed from the high trusted process to the low trusted process once every T seconds is treated slightly differently. The signal to raise the rate of acknowledgments is treated in exactly the same way, i.e. the low trusted process will raise the rate of acknowledgments by a predefined constant R. The signal to lower the rate of acknowledgments is treated differently. If the high trusted process signals the low trusted process to lower the rate of acknowledgments, then the low trusted process will lower the rate of acknowledgments to zero. The data rate will go down to zero and no data will be accepted from Low for that quantum period T. Subsequent signal from the high trusted process to raise the data rate will raise it from zero to R.

The decision of whether to raise or lower the rate of acknowledgments is made by the high trusted process in exactly the same way as in the regular Quantized Pump. The trusted high process keeps track of the number of bytes that High acknowledged during this time period. Dividing this number by T, the trusted high process gets the current average rate in bytes/second. If this average rate is greater than the current rate that the low trusted process uses, then the high trusted process will signal the low trusted process to raise the rate of acknowledgments by R. Otherwise, it will signal the low trusted process to lower the rate of acknowledgments to zero for the duration of the next quantum.

The bandwidth of the covert channel is again $1/T$ bits/second because the only communication between the high trusted process and the low trusted process is a single bit every T seconds. But the amount of buffer space needed is now substantially less.

The largest amount of buffer space is needed when Low starts out at the rate of L_{max} and High refuses to accept anything. But now it takes exactly one quantum time period T to equalize these two rates. This means that the maximum total buffer space needed is only:

$$B_{max} = TL_{max}$$

In other words, the Linear Quantized Pump has the space complexity of $O(L_{max})$ as compared to $O(L_{max} \log L_{max})$ for the Logarithmic Quantized Pump and $O(L_{max}^2)$ for the regular Quantized Pump.

This improvement in buffer size comes at a price: the loss of available bandwidth. If we carefully analyze the behavior of the Linear Quantized Pump, we find that it is quite similar to the behavior of TCP Reno (the standard implementation of TCP) in that from time to time it sharply drops the data rate and then linearly climbs back to its previous value, exceeds it, drops again, and climbs back again. (The difference is that in Reno the data rate drops by half and in Linear Quantized Pump we drop it all the way down to zero.) In other words, Linear Quantized Pump spends most of its time climbing from the data rate of zero bytes/second to the actual best available data rate. The average data rate is approximately half of the available bandwidth. And the experimental results in section 5.7 completely confirm this observation with hard numbers.

As a passing note we should mention that it is possible to have the Quantized Pump behave like TCP Reno (reduce the data rate by half as opposed to reducing it to zero) and it will in fact have a 50% better average throughput than the Linear Quantized Pump while having approximately the same average buffer usage. In the worst case scenario, "TCP Reno-like" Quantized Pump will have only a slightly worse space usage than the Linear Quantized Pump, i.e. $B_{max} = 2TL_{max}$, instead of $B_{max} = TL_{max}$.

6.5 Possible Further Improvements

We can reduce the covert channel bandwidth even further by introducing random noise into the bits (this works much better with the original version of the Quantized Pump, not the Logarithmic or the Linear ones). For instance, even with 20% noise (i.e. one of every five bits is randomly flipped), we get a reduction in the covert channel bandwidth by nearly 87%. However, the usefulness of this measure is doubtful, since the covert channel bandwidth can be precisely set to any value by simply changing the constant T.

Another possible improvement is to have a single gateway incorporate all three versions of the Quantized Pump presented in the previous sections. Since the regular Quantized Pump offers the best data throughput but is the least efficient in terms of buffer space, we start communication using the regular Quantized Pump protocol. If the buffer space starts running short, the gateway would switch to Logarithmic Quantized Pump protocol, which is a lot better in buffer space management but offers slightly worse data rates. And finally, if the buffer space becomes really critical, the gateway might switch into Linear Quantized Pump mode, where the least amount of buffer space is required but the data throughput would also suffer a substantial drop.

These switches between different protocols can be accomplished by the low trusted process without any additional information from the high trusted process, and thus it would not introduce any additional information flowing from High to Low. This implies that such an adaptive gateway would not add any bandwidth to the covert channel from High to Low. Notice that the algorithm for the trusted high process is *identical* for all three versions of the Quantized Pump. The high trusted process does not need to be notified of protocol change — the low trusted process can change its algorithm on its own.

The algorithm for switching between protocols cannot be exact, since the low trusted process does not have direct access to the available buffer space (which might introduce an additional covert channel). The switching between protocols can still be accomplished by using a heuristic: if the high trusted process has been trying to lower the data rate for the last several turns, then the buffer must be starting to fill up and it is time for a more space-efficient protocol. If the high trusted process has been trying to raise the data rate for the last several turns, then the buffer must be emptying out and it is time to switch to a faster protocol.

6.6 Implementation

The Quantized Pump is implemented in a way very similar to the regular Pump. The trusted high process and the trusted low process are threads within the same protocol process. The communication between the trusted high process (thread) and the trusted low process is accomplished by means of a one bit shared variable that is updated once per time interval T. The protocol graph looks exactly like the protocol graph of the Pump; see Fig. 7 for details.

The buffer was isolated from the trusted low and high processes with only the "put" operation available to the low trusted process and only the "get" operation available to the high trusted process. Both TCP and IP protocols were modified as for the regular Pump.

6.7 Experimental Results

Tables 7, 8, and 9 summarize behavior of our implementation of the Quantized Pump with various parameters changing. In all three cases there is a covert channel present. As with SAFP and the Pump, **1** was encoded with a 250 millisecond delay and **0** was no delay.

The parameters for the Pump and the Quantized Pump were chosen to roughly approximate each other: 20ms/byte for $1/R$ parameter for the Quantized Pump and the moving average of 60 for the Pump. These parameters lead to roughly similar times for a data change rate from 0ms delay to 250ms delay of the covert channel and thus allow us a better comparison of the protocols ("apples to apples" comparison).

Data Transferred	360 KB	720 KB	1.4 MB
Transfer Time	37 s	77 s	170 s
Transfer Rate	9.7 KB/s	9.4 KB/s	8.2 KB/s
Quantum (T)	1 s	1 s	1 s
Rate of change (1/R)	20 ms	20 ms	20 ms
Average Packet Size	1380 bytes	1376 bytes	1361 bytes
Average Delay	23 ms	29 ms	29 ms

Table 7: Quantized Pump, covert channel present

Comparing Table 7 to Table 4 and Table 1, we see that the Quantized Pump protocol performed slightly better than the regular Pump and slightly worse than SAFP with the same covert channel. This is not a tremendously surprising result, since SAFP attempts no substantial covert channel elimination and the Pump spends extra time traveling between quantized levels of delays of the Quantized Pump.

Quantum (T)	1 s	2 s	4 s
Transfer Time	37 s	39 s	40 s
Transfer Rate	9.7 KB/s	9.2 KB/s	9.0 KB/s
Data Transferred	360 KB	360 KB	360 KB
Rate of change (1/R)	20 ms	20 ms	20 ms
Average Packet Size	1380 bytes	1398 bytes	1362 bytes
Average Delay	23 ms	29 ms	25 ms

Table 8: Quantized Pump, covert channel present

Table 8 shows what happens to the performance of the Quantized Pump as the bandwidth of the maximum covert channel is decreased from 1 bit/second to 0.25 bits/second. As expected, the data rate went down somewhat because the Quantized Pump now required more time to adjust. Notice that even with covert channel of 0.25 bits/second, the performance of the Quantized Pump is slightly better than that of the regular Pump on Table 4, first column.

Rate of change (1/R)	20 ms	40 ms	80 ms
Transfer Time	37 s	38 s	37 s
Transfer Rate	9.7 KB/s	9.5 KB/s	9.7 KB/s
Data Transferred	360 KB	360 KB	360 KB
Quantum (T)	1 s	1 s	1 s
Average Packet Size	1380 bytes	1426 bytes	1406 bytes
Average Delay	23 ms	23 ms	22 ms

Table 9: Quantized Pump, covert channel present

The effects of changing the parameter R are illustrated in Table 9. Our results force us to conclude that changes in R do not result in any substantial variation of the throughput of the Quantized Pump.

Data Transferred	360 KB	720 KB	1.4 MB
Transfer Time	42 s	80 s	166 s
Transfer Rate	8.6 KB/s	9.0 KB/s	8.4 KB/s
Quantum (T)	1 s	1 s	1 s
Rate of change (1/R)	20 ms	20 ms	20 ms
Average Packet Size	1289 bytes	1343 bytes	1357 bytes
Average Delay	18 ms	23 ms	27 ms

Table 10: Logarithmic Quantized Pump, covert channel present

Table 10 illustrates the performance of Logarithmic Quantized Pump. The throughput is 12% worse than the Quantized Pump for a small data transfer but only 3% worse than the Quantized Pump for a much larger data transfer. For even larger data transfers, the difference becomes even less pronounced. The logarithmic behavior in buffer growth completely justifies this slightly lower performance of the Logarithmic Quantized Pump.

Data Transferred	360 KB	720 KB	1.4 MB
Transfer Time	85 s	179 s	386 s
Transfer Rate	4.2 KB/s	4.0 KB/s	3.6 KB/s
Quantum (T)	1 s	1 s	1 s
Rate of change (1/R)	-1000 ms/+20 ms	-1000 ms/+20 ms	-1000 ms/+20 ms
Average Packet Size	1292 bytes	1333 bytes	1299 bytes
Average Delay	13 ms	19 ms	22 ms

Table 11: Linear Quantized Pump, covert channel present

Finally, Table 11 shows the performance numbers for the Linear Quantized Pump. As advertised in the previous sections, the Linear Quantized Pump sacrificed about half the bandwidth to eliminate all but a small size buffer. The table shows rate of change parameter $1/R$ as: -1000ms/+20ms, -1000ms is the duration of the quantum (T), thus for that quantum there was no data transmitted and the data rate is zero.

7 Conclusion

We examined several previously proposed protocols that aim at reducing the bandwidth of covert channels. We discussed our implementations of these protocols and some practical problems that come up when implementing these protocols.

We introduced a new protocol, the Quantized Pump, which is easy to configure, easy to analyze, has a provable lower bound on the covert channel bandwidth, and has better performance characteristics than the Pump. We demonstrated several different versions of the Quantized Pump with different data-rate/buffer-size tradeoffs.

Finally, we supported our claims with comparative performance numbers for all of the protocols discussed.

References

[1] D. Bell and L. LaPadula. “Secure computer systems: Mathematical Foundation,” ESD-TR-73-278, Vol.1, Mitre Corp, 1973.

[2] Ira S. Moskowitz and A.R. Miller. “Simple Timing Channels,” Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, 1994.

[3] Ira S. Moskowitz and M.H. Kang. “Covert channels — Here to stay?,” Proceedings of COMPASS ’94, pp. 235 -243, Gaithersburgs, MD, 1994.

[4] Jonathan K. Millen. “Finite-state noiseless covert channels,” Proceedings of the Compter Security Foundations Workshop II, pages 81-86, Franconia, NH, June 1989.

[5] John C. Wray. “An analysis of covert timing channels,” Proceedings of the 1991 IEEE Compter Society Symposium on Research in Security and Privacy, pages 2-7, Oakland, CA, May 1991.

[6] Myong H. Kang and Ira S. Moskowitz. “A Data Pump for Communication,” NRL Memo Report 5540-95-7771.

[7] M.J. Usher. “Information Theory for Information Technologists,” MACMILLAN PUBLISHERS LTD, 1984.

[8] M. Accatta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. “Mach: A New Kernel Foundation for UNIX,” Proceedings of USENIX, July 1986.

[9] N.C. Hutchinson and L.L. Peterson. “The x-Kernel: An architecture for implementing network protocols.” IEEE Transcations on Software Engineering, 17(1):64-75, January 1991.

[10] Myong H. Kang and Ira S. Moskowitz. "A Pump for rapid, reliable, secure communication," Proceedings ACM Conf. Computer and Commun. Security '93, pp.119-129, Fairfax, VA, 1993.

[11] Myong H. Kang, Ira S. Moskowitz, and Daniel C. Lee. "A Network Version of The Pump," Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1995.

[12] David M. Goldschlag. "Several Secure Store and Forward Devices," Proc. of the Third ACM Conference on Computer and Communcations Security, New Delhi, India, March 1996, pp. 129-137.

[13] J. McDemott, "The b^2/c^3 problem: how big buffers overcome covert channel cynicism in trusted database systems," in *Database Security VIII, Status and Prospects, J.Buskup, M. Morgenstern, C. Landwehr, eds.,* IFIP Transactions A-60, Elsevier Science B.V. Amsterdam, ISBN 0 444 81972 2, pp. 111-122.