

A Network Pump*

Myong H. Kang

Ira S. Moskowitz, Member IEEE

Daniel C. Lee, Member IEEE

Naval Research Laboratory
Washington, D.C. 20375[†]

Abstract

A designer of reliable multi-level secure (MLS) networks must consider covert channels and denial of service attacks in addition to traditional network performance measures such as throughput, fairness, and reliability. In this paper we show how to extend the NRL data Pump to a certain MLS network architecture in order to balance the requirements of congestion control, fairness, good performance, and reliability against those of minimal threats from covert channels and denial of service attacks. We back up our claims with simulation results.

1 Introduction

A multi-level secure (MLS) system stores and processes information of different sensitivity levels in a secure manner. By this we mean that access is controlled so that no high level information is allowed to pass to lower level users/processes (Low) but lower level information is available to higher level users/processes (High) [3].

*This is a revised version of “A Network Version of the Pump” which appeared in the Proc. of the 1995 IEEE Symposium on Security and Privacy, pp. 144-154, Oakland, CA

[†]Respective addresses of the authors are (mail code 5540, mail code 5540, mail code 8140) Naval Research Laboratory, Washington, D.C. 20375. Respective e-mail addresses are *mkang@itd.nrl.navy.mil*, *moskowitz@itd.nrl.navy.mil*, *lee@kingcrab.nrl.navy.mil*

Thus, an MLS system allows Low to send messages to High, but High should not be able to send messages to Low. On the other hand, acknowledgements (ACK) to Low that High has received its messages are necessary for reliability and performance. This is especially true for distributed systems in which the communication channels may not always be reliable. High, however, can manipulate the times that ACKs arrive in order to covertly send unauthorized messages to Low. Therefore we see that MLS security is in conflict with the need for functionality.

The Pump, developed at NRL [6, 7], solves the dilemma of simultaneously assuring reliability, performance and security when there is only one Low and one High. We will refer to this as the basic Pump. The basic Pump allows High to send ACKs to Low indirectly through an intermediary communication buffer. Further, the basic Pump requires that Low receive the ACKs at probabilistic time intervals. This probabilistic ACK is based upon past High activity.

Since computer systems are becoming more open and interconnected, denial of service problems are receiving more attention [21]. Hence, security devices such as the Pump should also provide protection against such attacks.

This paper addresses how to adapt the basic Pump for use in a network environment, where we have multiple Lows and multiple Highs. We will refer to this as the “network Pump.” As we move from a dedicated data network to the Broadband Integrated Service Digital Network (B-ISDN) such as the Asynchronous Transfer Mode (ATM) Network [9], the issues of congestion control, fairness, and reliability become extremely important and extremely complicated. The network community itself has not worked all of these issues out yet. Our problem is even more complex because we are coupling security (i.e., covert channels [10] and denial of service) with the above.

1.1 Assumptions and Terminology

The network environment that is considered is shown in figure 1. Each Low (High) is at the same low (high) level. Possible implementations of the network Pump are a packet switch between workstations, a router connecting local area network segments, etc.

There are many Lows (L_i) and Highs (H_j), and they are untrusted processes, thus they may contain Trojan Horses (malicious software). The network Pump is a trusted process (assured to be free of malicious software) which mediates traffic from Lows to Highs. Each message that will be routed from a Low to a High has a message number

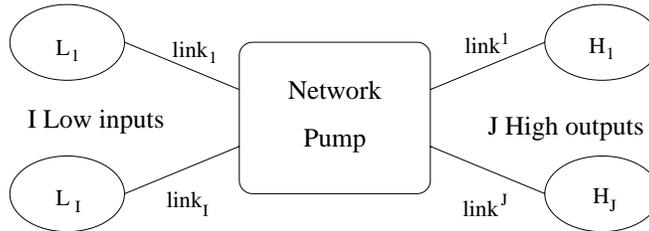


Figure 1: The Pump in a network environment.

(ID), and input and output addresses associated with it. For simplicity, we assume that all messages have the same length. We do not consider multicasting in this paper. We further assume that Lows (Highs) do not communicate among themselves to simplify the covert channel analysis.

A session is a connection between any Low and any High. In figure 1 there are $I \times J$ distinct sessions. During each session $_{ij}$, a message leaves L_i , travels over $link_i$, goes into the network Pump, and after processing leaves the network Pump over $link^j$, and arrives at H_j . We assume that all propagation delays are zero for conceptual simplicity¹. The minimal processing time of the network Pump is a fixed overhead value O_v , which is small enough so that the network Pump itself never becomes a performance bottleneck.

The demand (input) rate λ_{ij} is the rate demanded by L_i destined for H_j . Each H_j behaves as a server with service rate μ_{ij} . This is the inverse of the mean time of service by H_j for messages from L_i .

1.2 Objectives

Most network resources are dynamically shared for efficiency reasons. If this dynamic sharing is not carefully controlled, then inefficiency and delays occur [4]. The main functions of congestion control in a network are:

- To prevent inputs from sending messages faster than the outputs can handle them.
- To prevent throughput degradation and loss of efficiency from overloading the network.
- To prevent unfair allocation of network resources from competing inputs.

¹This assumption is true for the basic Pump also. It can be easily relaxed in both cases.

Since the network Pump is a shared resource among many sessions, it should provide the congestion control mechanism. Let us discuss the specific objectives required of the network Pump.

Reliability / Handshaking

The reliability requirement can be simply stated as *no loss of messages and no duplication of messages*. To satisfy this requirement ACKs and message numbers (ID) are necessary. The network Pump has a reliability protocol that works as follows:

If a Low has not received ACK by `time_out` after sending a message, it will retransmit the same message. If a High receives the same message then it will keep only one copy.

Further, a Low does not send the next message to a specific High until its previous message to that High has been ACKed (handshake protocol).

Performance

We desire good performance. The network Pump's control over the time of sending the ACKs to a Low can be utilized for congestion control. The network Pump is designed to exercise congestion control. The network Pump controls the rates into itself, the realized (input) rates, by slaving Low's message transmission to the average rates out (service rates) of the network Pump. It does so by moderating the ACK rate to a Low, since this Low will not send a new message until it receives an ACK from the previous message from the same session (the handshake protocol).

If the service rates are greater than the demand rates, then the network Pump should not hurt performance. If service rates or output capacities are less than the demand rates, then the outputs cannot handle their inputs. Therefore, the network Pump by slowing the demand rate to the realized rate, alleviates congestion, and at the worst, does not lessen total throughput.

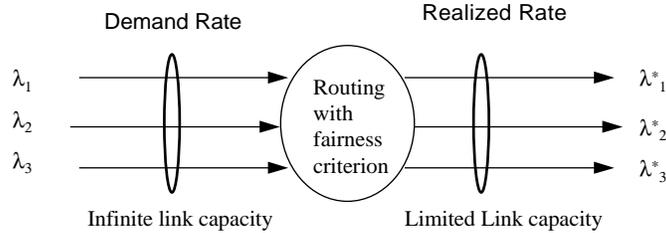


Figure 2: fairness

Fairness

Bandwidth of communication links, transmission speed, and processing speed are all limited. Therefore, if the load of data traffic offered to the network Pump exceeds its capability, some of the load must be cut. The load must be cut fairly for all the sessions that share the network Pump. The idea is shown in figure 2 where the illustrated output limitation is due to limited output link capacity. Note, in denial of service attacks (discussed later), the limitation is usually due to decrease in the High service rate.

Fairness can be defined in different ways. One fairness policy is *max-min fairness*. This policy says all sessions should get bandwidth according to the following criterion — the smallest realized rate is as large as possible and, given this, the second-smallest realized rate is as large as possible, etc.[5]. For example, if there are three sessions whose demand rates (units of messages per unit time) are 0.4, 0.5, 0.6 and the output capacity equals 1, then all three sessions will have realized rates of $1/3$, $1/3$, $1/3$ under max-min fairness. If one session demands less than what it can get, the leftover bandwidth will be equally shared among the rest of the sessions. For example, if there are three sessions whose demand rates are 0.2, 0.5, 0.6 and the output capacity equals 1 then those sessions will have realized rates 0.2, 0.4, 0.4, respectively under max-min fairness.

The advantages of this policy are (1) there is a simple way to implement this policy (i.e., round-robin scheduling [5]) and (2) the scheduling scheme does not need to know the demand rates of sessions which may not always be known. Since this policy gives preference to sessions that have lower demand rate, it does not allow a session to take the entire bandwidth if there is more than one session. One disadvantage of this policy is that a heavily demanded session is penalized more than a lightly demanded session (i.e., not sensitive to demand rates).

There are other fairness policies, such as the proportional policy [20]. This policy allocates bandwidth in proportion to each input demand rate. The network community does not have a “best” fairness policy. The network Pump uses max-min fairness because of the above advantages.

Covert Channels

It is well known that the ACK stream that is required to satisfy the reliability requirement introduces covert communication channels. This was the motivation for developing the basic Pump over the conventional store and forward buffer type of communication. We will show in section 3.2, using results from [6, 7], that the capacity of the covert channels can be made negligible.

Denial of Service

We interpret the denial of service attack in a broad sense in the network environment:

If a session cannot achieve its intended throughput due to the misbehavior of other sessions then the session is under a denial of service attack.

Since the network Pump is a shared resource among several sessions, services for other sessions can be potentially disrupted if too much resource is allocated to one particular session. The design of the network Pump should prevent such a situation.

2 Background — The Basic Pump

In the basic Pump our concern is sending messages from (one) Low to (one) High. In [6, 7, 19], we reviewed why traditional communication protocols (including *read-down* and *blind write-up*) cannot satisfy the needs for reliability, performance, and security simultaneously. As a solution, the basic Pump was introduced as shown in figure 3.

The basic Pump [6] places a buffer (size n) between Low and High, and gives ACKs at probabilistic times to Low based upon a moving average (MA) of the past m High

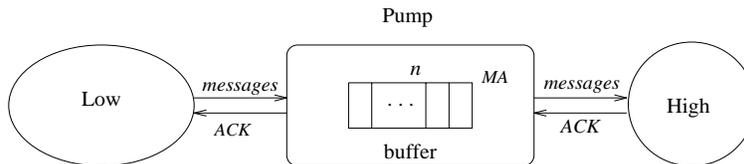


Figure 3: The Basic Pump

ACK times. A High ACK time is the time from when the buffer sends a message to High to the time when the buffer receives High’s ACK. Thus, basing Low ACK times upon past High activity has the double benefit of (1) keeping the buffer from filling up and (2) having a minimal negative impact upon performance. The Low ACK time is the time from when Low sends a message to the basic Pump to the time when Low receives the basic Pump’s ACK. The Low ACK time is a slightly modified exponential random variable [7] with mean equal to MA . Intuitively, the modification is (1) a truncation of the exponential density at a design parameter `time_out` and (2) a shift by an amount of time equal to the minimum processing time (T_r) of a message if there is space on the buffer. When Low must wait for space on the buffer, the above holds, except the shift is equal to $T_r = \max(\text{wait time for space, minimum processing time})$. Thus, the Low ACK time is a random variable that takes values between T_r and `time_out`. (In [7] we have slightly modified this over the first exposition of the basic Pump [6] to introduce extra noise. However, for the network Pump we stay with the simpler formulation due to the extra noise from multiple users and a modified ACK scheme (see section 3.2 covert channel analysis)).

At present, an implementation of the basic Pump is running on a XTS-300 platform [14]. Early results, along with the simulation results of Kang and Moskowitz [7], show a proof of concept for the basic Pump. Based upon this and the need for a secure network congestion mediator we feel the extension to the network environment is proper. Presently, a prototype network Pump is being built in hardware as a guard by NRL’s Center for High Assurance Computer Systems [22].

3 An Architecture of the Network Pump

The architecture of a network Pump is shown in figure 4.

Each component of the network Pump works as follows:

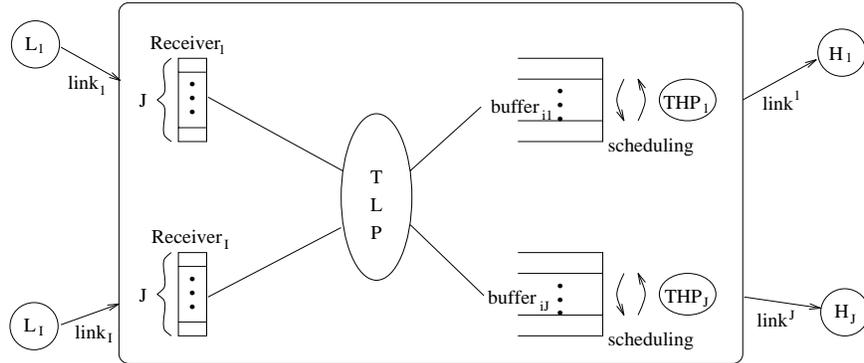


Figure 4: A logical view of the network Pump.

Lows and Highs (Exterior to the network Pump)

Lows (Highs) is the set of inputs (outputs) to the network Pump. Lows (Highs) consists of I (J) processes non-communicating among themselves. Each Low, L_i , can strive to send messages to any High, H_j , with various demand rates as discussed before. Since L_i is outside of the network Pump we assume the L_i has some procedure for sending messages over $link_i$, the only constraint being that the demand rates are λ_{ij} , such that $\sum_j \lambda_{ij} \leq \text{capacity of } link_i$. Consider session $_{ij}$ — After L_i sends a message to H_j , it waits for the ACK to that message from the network Pump. Once this ACK arrives, L_i can send another message to H_j . Therefore, each Low can only send a new message in each session after it has received the ACK of the previous message from the network Pump (handshake protocol). When H_j receives a message from the network Pump it sends an ACK back after the appropriate service time to the network Pump.

Receivers

There is a receiver $_i$ for each L_i . In receiver $_i$, there are J slots; slot $_j$ stores a messages from session $_{ij}$ until it is routed by the TLP.

Trusted Low Process (TLP)

The TLP takes a message from a receiver and routes it to the appropriate output buffer. We denote by T_r the time from when a message is sent from a Low to the time when that message is placed in the appropriate output buffer. (We will also refer to T_r as “routing time” in the network Pump.) If there is available space in the output buffer, T_r is equal to the overhead O_v . If there is no space, the message is not placed until there is a space available. Therefore, T_r includes both O_v and the amount of time the message waits until the output buffer is available.

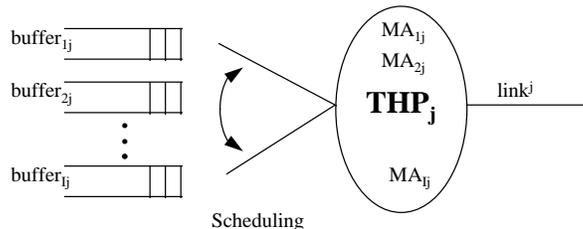


Figure 5: A closer view of a trusted high process.

After the message is routed to the output buffer, the TLP is ready to send an ACK back to the appropriate L_i . The time this ACK arrives at L_i depends on the randomization scheme, but is always at least T_r .

Output Buffers

There are I logical output buffers for H_j , each denoted as buffer_{ij} . A message from session _{ij} will be stored in buffer_{ij} .

Trusted High Processes (THP_j)

THP_j delivers a message from buffer_{ij} to H_j according to a scheduling scheme. THP_j cannot deliver another message from buffer_{ij} until the prior message from buffer_{ij} is ACKed (by H_j).

3.1 A Detailed Design

A detailed design rationale is described in this section.

3.1.1 Trusted High Processes

THP_j plays an important role in scheduling delivery from output buffers to H_j and in computing moving averages. Figure 5 graphically describes the role of a THP_j .

Consider THP_j — it has to deliver messages from each buffer_{ij} to H_j . Since the capacity of link^j is limited by physical considerations and inputs may send more messages than link^j or H_j can handle, THP_j needs some scheduling scheme. This scheduling scheme determines the fairness among different inputs.

The network Pump uses round-robin scheduling because it is simple and achieves

max-min fairness [5]. For example, if THP_j has to serve three output buffers then an opportunity to send a message is given in the order of buffer_{1j} , buffer_{2j} , buffer_{3j} , buffer_{1j} , If buffer_{2j} does not have any message to send then the opportunity is transferred to buffer_{3j} and the next opportunity is given to buffer_{1j} , and so on.

THP_j also maintains and updates moving averages (MA_{1j} , ..., MA_{Ij}). The reason for THP_j to maintain I separate moving averages (i.e., one per session) instead of one combined moving average is that H_j may service messages from different Lows at different rates. Throughout this paper we assume that the message service time is not a performance bottleneck in the benign case. In other words, the bottleneck is output links, not servers, unless the system is under denial of service or covert channel attacks.

Since there is potentially more than one input, the method of computing moving averages is different from when there is only one input. When there is only one input, the moving average is computed based on the interval from the time the message is sent to High, to the time the ACK arrived from High. However, if there is more than one input, the message is ready to be sent by the Pump, but cannot be sent because the output link is not available due to the round-robin scheduling. This additional waiting time must be taken into account or else the input messages will flood the output buffers.

Specifically, MA_{ij} of the network Pump is the moving average of the last m ACK times from H_j to buffer_{ij} . An ACK time from H_j is the difference between when buffer_{ij} receives ACK from H_j and $\max(\text{time that message arrived in } \text{buffer}_{ij}, \text{time that the previous message from } \text{buffer}_{ij} \text{ was ACKed by } H_j)$. In other words, if buffer_{ij} is not empty then the previous ACK time by H_j is used to compute the moving average. However, if buffer_{ij} is empty when a new message arrives then we use the arrival time instead of the previous ACK time.

3.1.2 Output Buffers

The number of messages in buffer_{ij} is important to achieve fairness [5] (the bigger the number of messages in buffer_{ij} the fairer). This is because our round-robin scheduler does not take burstiness into account. The way to handle bursts is to have enough messages queued in buffer_{ij} so that times of abundance and starvation (with respect to message arrivals) are balanced out. In fact, it is desirable to keep the queue length in buffer_{ij} positive so that max-min fairness is preserved. However, if the queue length is too big we have potential covert channel and denial of service problems. Thus, it is desirable to keep the queue length at a certain level. To address this issue, we introduce

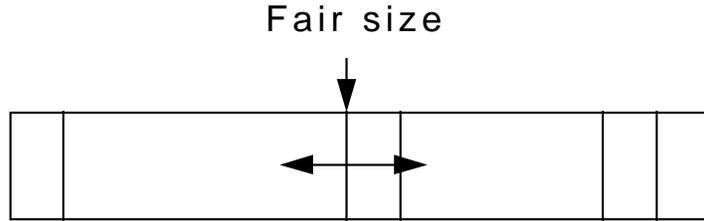


Figure 6: A closer view of buffer_{ij} .

the concept of *Fair size*, which is a design parameter targeted for the desirable queue length. Intuitively, the burstier the input, the larger the *Fair size* must be. Note that *Fair size* has to be intelligently chosen so that one session cannot dominate (fill) the total output buffer and at the same time large enough to accommodate burstiness. Good design requires that the total output buffer has at least *Fair size* surplus spaces in addition to sum of the *Fair size* spaces allocated for all sessions. Intuitively, if all sessions are active and behaving, this design leaves us with at least *Fair size* spaces in the total output buffer. Figure 6 pictorially shows buffer_{ij} where the number of messages in the buffer fluctuates around the *Fair size*.

Since the network Pump has a built-in mechanism to share output buffers fairly among different sessions (i.e., moving average construction to control input rates which will be discussed in section 3.1.3), all output buffers are dynamically shared among different sessions.

3.1.3 Trusted Low Process

When routing requests arrive from receivers, the TLP routes messages to the proper output buffers and reads the current moving average value. Once the message is delivered, the TLP is ready to send an ACK. However, this ACK will be delayed depending on the moving average of the session and the randomization scheme. The network Pump uses a similar randomization scheme as the basic Pump whose details are presented in [6, 7]. As described in section 2, the TLP of the basic Pump delays ACKs based on a modified exponential random distribution whose mean is the moving average of the session. This ACK rate controls the input rates, through the handshake protocol, if the demand rate is higher than the ACK rate.

As we discussed in section 3.1.2, we wish to make sure that the number of messages

in an output buffer fluctuates around the *Fair size*. To achieve this, we modify the ACK scheme from the basic Pump. The way the basic Pump controls the ACK time to Low can be written as follows:

$$ACK\ time = \begin{cases} T_r & \text{if } MA - T_r \leq 0 \\ \min(T_r + \bar{f}_r[MA - T_r], \text{time_out}) & \text{otherwise} \end{cases}$$

where T_r is the same as in section 2, and $\bar{f}_r[MA - T_r]$ is a draw from a random distribution as also described in section 2. (Note that there is only one session in the basic Pump.) Recall that T_r is the time between when the a message is sent from Low to when the message is placed in the output buffer. Hence, a random delay is included in the ACK time in addition to T_r if $MA - T_r > 0$. A detailed description can be found in [7].

We now describe the way the network Pump controls ACK times to L_i for a message in each session. Let $f_r(x)$ be a draw from the exponential distribution with mean x . Define

$$Q \equiv f_r(MA_{ij} - T_r) + k \cdot (N - Fair\ size)$$

where N is the number of messages in $buffer_{ij}$ at the time the message is placed in $buffer_{ij}$, and $k \cdot (N - Fair\ size)$ is a feedback term. Both k and *Fair size* can be chosen by a system designer. Note that the moving average of the ACK times from H_j to the network Pump is computed separately for each session. For session $_{ij}$, the ACK time to Low for each message is:

$$ACK\ time = \begin{cases} T_r & \text{if } MA_{ij} - T_r \leq 0 \text{ or } Q \leq 0 \\ \min(T_r + Q, \text{time_out}) & \text{otherwise.} \end{cases}$$

We now elaborate the rationale of the extra term $k \cdot (N - Fair\ size)$ in Q . As long as L_i has messages to send to H_j , the network Pump wants to keep $buffer_{ij}$ nonempty. The reason is to prevent missing the round-robin turn, and thus to give each session throughput close to max-min fairness. Therefore, when the number of messages in $buffer_{ij}$ is less than the *Fair size*, the network Pump reduces its ACK time to L_i in order to accelerate the input rate, as seen in the extra term. On the other hand, if $buffer_{ij}$ is often full, we have covert channel problems (see section 3.2). Hence, the network Pump decreases the input rate by increasing the ACK time to L_i when the number of messages in $buffer_{ij}$ is larger than the *Fair size*.

In the simulation that is described in section 4, we choose $k = MA_{ij}/(Fair\ size)$ (see section 4 for the discussion of how the *Fair size* was chosen for the simulation). Thus

$$Q = f_r(MA_{ij} - T_r) + MA_{ij}\left(\frac{N}{Fair\ size} - 1\right).$$

Therefore, we have

$$avg(ACK\ time) \approx T_r + avg(Q) = MA_{ij}\left(\frac{avg(N)}{Fair\ size}\right).$$

Note that $avg(N)$ is close to the *Fair size* due to the second term of Q . Thus we have $avg(ACK\ time) \approx MA_{ij}$.

3.1.4 Receivers

Receivers receive messages from Lows and request routing to the TLP. Each receiver contains J slots (size one buffers) so that the inputs from one session do not interfere with inputs from other sessions. Messages in the slots will either be routed or discarded after `time_out` (if there is no output buffer available).

3.2 Design Review

In this section, we review the design of the network Pump and explain how the objectives in section 1.2 are satisfied. We back our claims on performance, fairness, and denial of service by the simulation results presented in section 4.

Reliability

Due to the reliability protocol requirement that was specified in section 1.2 (i.e., ACK, retransmission of the same message after `time_out`, and message ID), the network Pump provides as much reliability as TCP. Also, the High ACK of the Pump can be realized as an ACK of an application layer protocol to provide reliability at the application layer. There are some secure system protocols which do not use acknowledgements, but they are not reliable [19, 2].

Performance

The network Pump does not hurt performance (throughput). Consider the following two cases where output link capacities are not a bottleneck:

- *Demand rate is faster than the service rate:* The network Pump’s ACK rate, which is tied to the moving average of the server will slow down input to match the servers. However, this will not degrade performance because the throughput will be determined by the service rate which is the performance bottleneck.
- *Demand rate is slower than the service rate:* The network Pump’s ACK rate will not slow down the input rate in this case. Hence, there is no effect on performance.

If the output link capacities are a bottleneck, the network Pump’s max-min fairness criteria and the moving average construction assures us that performance is not penalized (section 3.1.1). Hence, the network Pump does not affect the throughput unless the network Pump itself is the bottleneck.

Fairness

The network Pump uses a round-robin scheduling scheme which achieves max-min fairness at each THP_{*j*} if all inputs can accumulate enough messages at output buffers. The network Pump’s modified moving average construction that was described in section 3.1.3 encourages all inputs to send as many messages as possible up to the *Fair size*. Hence, the network Pump strives to achieve max-min fairness.

Covert Channel Analysis

A major reason that the network Pump uses a randomized ACK stream, aside from fair congestion control is to minimize the threat of covert communication from a High to a Low. Ideally, a secure computer system does not allow a communication channel from a high level user/process to a low level user/process [10]. However, in reality, unless we want a non-responsive or non-reliable system, such covert channels are a fact of life. Given that, one must try to find ways to minimize illicit information leakage from such covert channels. The covert channels that concern us in this paper are timing channels [12, 15, 24] — these are covert channels where the output symbols are distinguished by their different time values.

The specific timing channels that concern us have been discussed in detail in other papers [6, 7, 13, 16], so we present only a brief review here. Let us start off with a simple scenario: a single low (Low) and a single high (High) with a store and forward

buffer (SAFB) between them. Low sends a message to the SAFB. When this message is stored in the SAFB, the SAFB sends an ACK back to Low and then Low can send its next message to the SAFB. The SAFB attempts to send the message to High and when High has successfully received the message from the SAFB, High sends an ACK to the SAFB and the message is removed from the SAFB. For obvious reliability concerns, we do not allow Low to write over messages in the SAFB before they have been read out by High. Therefore, when the buffer is full it stops receiving messages from Low. This opens up what is referred to as the full buffer channel (FBC) between High and Low. High, by intentionally slowing or stopping its receiving of messages from the SAFB, can cause the SAFB to become full. Then High can remove messages to allow space to open up on the SAFB and for Low to again receive an ACK from the SAFB and send new messages. Therefore, High can manipulate the timing of the ACK stream to Low from the SAFB. Let us do a worst-case information theoretic analysis of the FBC to see how much information can actually be leaked from High to Low and hence how insecure our system may be.

The FBC is a timing channel. To exploit the FBC it is assumed that Trojan horses (malicious software in both Low and High) are in the computer system. The Trojan horses control when Low sends a message and when High sends an ACK back to the SAFB.

- A Trojan horse fills the SAFB by (High) not removing messages from the SAFB.
- Now that the SAFB is full, a noiseless timing channel exists between High and Low. Furthermore, this noiseless channel exists as long as the SAFB is full.
- Now Low sends a message to the SAFB. The SAFB cannot send an ACK back to Low until a spot opens up on the SAFB. If High happens to remove a message as soon as (or before) Low sends a message, then Low only waits an overhead time O_v for an ACK. We assume that High, by removing messages from a full SAFB, can affect the ACK time to Low in increments of $i\epsilon, i = 0, 1, 2, \dots$. Assuming Trojan horses know the size of the SAFB (i.e., n) and how fast the Trojan horse can send a message, High knows that Low has filled the SAFB and has just sent a new message to the SAFB. If Low gets an ACK at time $O_v + i\epsilon$, Low interprets the signal as the $(i + 1)$ st symbol. Since every time Low receives an ACK, the SAFB is full again, and Low can then send its new message — High can noiselessly send symbols again.

With this example we are looking at a worst-case scenario. High will try to send symbols as quickly as possible, hence the time values of $O_v + i\epsilon$. Of course this allows unbounded response times. Real systems have timeouts and we assume that the timeouts in question are very large in comparison to $O_v \geq \epsilon$, so that examining a communication channel with i bounded changes the analysis very little from assuming that i is unbounded. The time units of our system are such that ϵ is an integer, i.e., ϵ is an integer number of system clock ticks (either reading from a system real-time clock or instruction counts from a tight loop). The channel capacity [23] of this channel is given by

$$C = \limsup_{k \rightarrow \infty} \frac{\log N(k)}{k} \text{ bits per clock tick}$$

where the logarithms are base two and $N(k)$ is the number of distinct sequences of symbols (ACK times) that take a total of time k . It can be shown [23, 12, 13, 16] that $C = \log \omega$, where ω is the positive root of $1 - (x^{-O_v} + x^{-\epsilon})$. The polynomial arises from the limiting behavior of the recurrence relation

$$N(k) = N(k - O_v) + N(k - (O_v + \epsilon)) + N(k - (O_v + 2\epsilon)) + \dots .$$

Define q by $\frac{O_v}{\epsilon} = q$. Note that q need not be an integer. By changing variables and letting $y = x^\epsilon$ we see that ω^ϵ is the positive root of $1 - (y^{-q} + y^{-1})$. Unfortunately, the only closed form solution for such polynomials [13] involve special functions. For certain special cases such as $q = 1$ we can obtain the trivial closed form solution that $C = 1/\epsilon$. Similarly for $q = 2$, we have $C = \epsilon^{-1} \log \frac{1+\sqrt{5}}{2}$.

The basic Pump, which deals with a single Low and a single High, was designed to minimize the threat of the FBC by modifying the SAFB by using probabilistic ACK times. The probabilistic arrival times of the ACKs introduce noise into the timing channel. Also, the basic Pump prevents the buffer from becoming/staying full. These two effects, the noise and the fact that the buffer is hardly ever full, severely diminish the channel capacity of various exploitations of the FBC. Bounds on the actual covert exploitation of the basic Pump are given by capacity reduction formulas in [6] and more precisely in [7]. In brief, a relationship between the buffer size n and the moving average parameter, m , was given with respect to the desired percent reduction of the covert channel capacity. Hence, either analytically or numerically, we have a way to measure the maximal information leakage from High to Low in the basic Pump. We will show that these reductions also hold for covert channel exploitations of the network Pump.

In the network Pump, our Trojan horse scenario is that one particular H_j and one particular L_i are in cahoots via cooperating Trojan horses (recall that we are looking at the situation where the Lows (Highs) do not communicate among themselves) — thus we again have the potential for the FBC. However, the fact that there are other users of the network Pump means that noise is introduced into the FBC which does not occur in the case of one Low and one High. Thus, the covert channel analysis from the basic Pump still holds for the network Pump as a worst-case scenario (we can very conservatively replace the buffer size of the basic Pump, n , by the *Fair size* in the capacity reduction formulas [7]). This is because in the network Pump, instead of just attempting to keep the total buffer from become full, we attempt to keep the total buffer around (the number of active sessions \times *Fair size*). Hence, for a misbehaving session to fill the total buffer, and thus exploit the FBC, it requires that session to fill the unused total buffer spaces which are at least *Fair size* in number (see section 3.1.2). Therefore, we can use the capacity reduction formulas from [7], where there is one Low and one High as an upper bound. These reduction formulas tell us how to choose the *Fair size* and the moving average parameter, m , to ensure that any covert channel capacity is within a specified bound. This bound becomes even more conservative as I and J increase due to the increasing noise of multiple users being on the system. Therefore, given values of O_v and ϵ , we can choose a *Fair size* and moving average parameter, m , to reduce the potential covert channel capacity to within a specified value.

In the basic Pump there is a statistical channel [17] from High to Low, when the buffer is not full, caused by Low attempting to correlate the ACK times to High's actions. As the number of terms making up the moving average grows, this correlation decreases, and hence the channel capacity decreases. The same holds as well for the network Pump and again the multiple users introduce spurious noise which further serves to confound any meaningful interpretation of the ACK times. Also, the *Fair size* further frustrates correlation attempts by L_i . Therefore, the bounds from the basic Pump again hold.

Finally, the network Pump (as well as the basic Pump) is sensitive to the small message criterion [18]. By this we mean even if one has a channel with small, or even zero, capacity it might still be possible to send small, possibly noisy, messages relatively quickly. The network Pump, because of its buffer management and randomization scheme, is designed to thwart an attempt at sending meaningful short messages, see [7, section 4] for details.

Denial of Service

In the network environment denial of service can occur in the following two cases:

1. A server slows down.
2. An input sends messages faster than the rate that the intended server can handle them.

In these cases, the shared resources will be monopolized by this specific session so that other sessions cannot use required resources.

The above cases will not happen if the network Pump mediates between the Lows and Highs because the network Pump monitors the servers' activities and tracks service rates. The service rate will be reflected in the ACK rate to L_i through the moving average construction. Due to the network Pump's handshake protocol and moving average construction, inputs (Lows) cannot send any more than the servers (Highs) can handle.

4 Simulation Results

To substantiate our claims on performance, fairness and denial of service, simulation experiments have been conducted.

4.1 Simulation Set Up

In our simulation scenario, there are three Lows (L_1, L_2, L_3) and three Highs (H_1, H_2, H_3); hence 9 sessions. The capacities of all input and output links are 1.0. All demanded inputs have Poisson arrival distributions. Demand rates from L_1 are $\lambda_{11} = 0.5, \lambda_{12} = 0.3, \lambda_{13} = 0.2$, the demand rates from L_2 are $\lambda_{21} = 0.4, \lambda_{22} = 0.4, \lambda_{23} = 0.2$, and the demand rates from L_3 are $\lambda_{31} = 0.4, \lambda_{32} = 0.5, \lambda_{33} = 0.1$ (see figure 7).

All Highs have 2-Erlang distributed service rates. The 2-Erlang distribution is often used to model the time to complete a task [8]. For the benign case all service rates are set to 2.0 (to demonstrate the output links being the bottlenecks, any value greater than

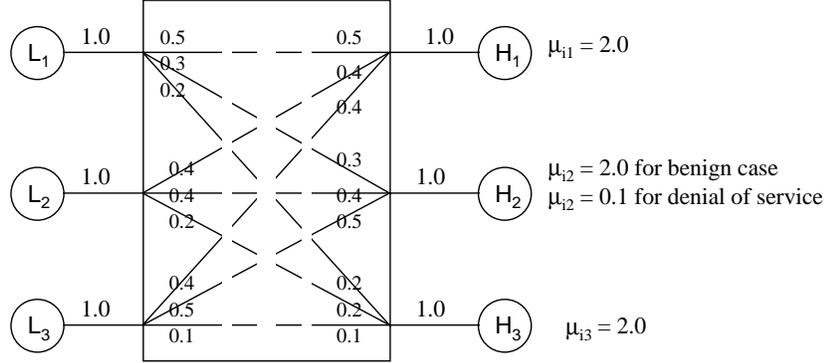


Figure 7: The simulation scenario.

1.0 would suffice). For denial of service simulation, service rates are $\mu_{i1} = 2.0$, $\mu_{i2} = 0.1$, and $\mu_{i3} = 2.0$ for $i = 1, 2, 3$.

The performance and fairness of the following two systems and the “ideal” case are compared under different total output buffer sizes.

- The network Pump as described in section 3. The last 30 High ACK times are used to compute the moving average (i.e., $m = 30$) and the *Fair size* per session was set to $\frac{1}{10}$ of the total output buffer size. Note that there are 9 sessions in our scenario.
- The Nonpump. This is the same as the network Pump (still has the handshake protocol) except that it does not have the moving average or probabilistic construction. (Thus, output buffer space is allocated to each session on a first come first served basis.) In other words, ACKs will be sent to Lows as soon as the message is routed. Hence, demand rates will be forcibly adjusted to the realized rate only when there are no available output buffers. The purpose of this system is to demonstrate the importance of congestion control.
- The ideal case is where the max-min fairness rates are achieved over the output links. The max-min rates for H_1 are $(1/3, 1/3, 1/3)$, for H_2 they are $(0.3, 0.35, 0.35)$, and for H_3 they are $(0.2, 0.2, 0.1)$. Hence, these are the ideal versions to which we compare the network Pump and the Nonpump.

4.2 Simulation Results in the Benign Case

In the benign case (i.e., inputs and outputs behave — no Trojan horses are present), there is not much of a performance difference between the network Pump and the Nonpump even though the network Pump performs slightly better than the Nonpump. This slight performance difference comes from the congestion control mechanism. Since the Nonpump has little congestion control, some inputs still send more messages than the intended server can handle. This causes an unfair sharing of resources and degrades performance. This effect will be magnified under the denial of service attack. Figures 8, 9, and 10 show the performance and fairness among sessions that send messages to H_2 .

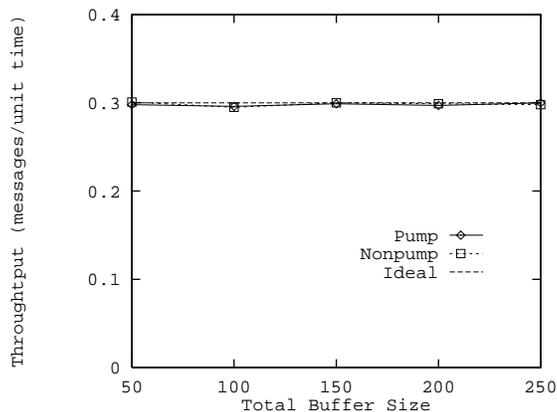


Figure 8: Throughput of session₁₂.

Note session₁₂ achieves its demand rate as realized rate (.30). Hence, the jitter in figure 8 is from the probabilistic nature of the input rather than any effects from routing devices². This probabilistic jitter slightly affects the throughput of other sessions (figures 9 and 10).

Figures 9 and 10 also show the effect of the scheduler, and the size of the output buffer and the *Fair size* to the fairness and throughput of each session, since 0.4 and 0.5 are both greater than 0.35. As the size of output buffer grows the throughput approaches its ideal (fairness) rate.

Even though we do not show the performance of other sessions, the network Pump performs very well (basically the same as in figures 11-16).

²The simulator never exactly generates a rate of 0.3, hence the jitter.

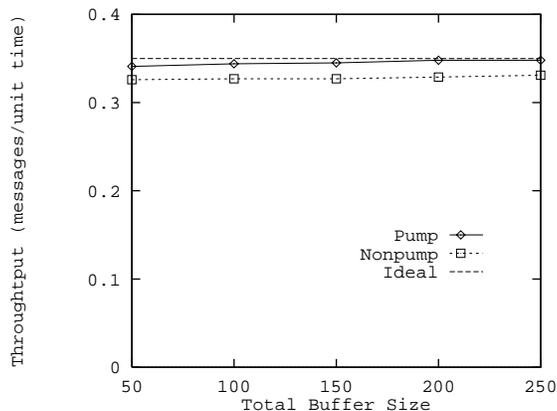


Figure 9: Throughput of session₂₂.

4.3 Simulation Results under Denial of Service Attack

To show the effect of a denial of service attack, we slow down the service rate (i.e., $\mu_{i2} = 0.1$) of one High, namely H_2 . Figures 11 through 16 show the performance and fairness comparison between the network Pump and the Nonpump. The performance of the network Pump is hardly affected by the attack. However, the performance of the Nonpump is greatly affected. The main reason for the degradation of performance is that all output buffer space is occupied by sessions that send messages to H_2 so that the rest of sessions have to wait a long time to obtain them.

Figures 11, 12, and 13 show the throughputs of sessions to H_1 .

These figures (11, 12, and 13) show no jitter of throughput as the total buffer size increases. This shows that the probabilistic nature of inputs are all hidden because all input (demand) rates to H_1 are greater than its realized rate and messages are always waiting for their turn at the output buffer³ (the round-robin scheme takes a message from each buffer in turn and does not pass any buffer because they always have a message ready to send).

Figures 14, 15, and 16 show the throughputs of different inputs to H_3 . Again the jitter of throughput is from the probabilistic nature of inputs rather than the effect of different buffer sizes.

³For example a fluctuation around a rate of 0.5 is not significant when the realized rate is actually much less than 0.5 .

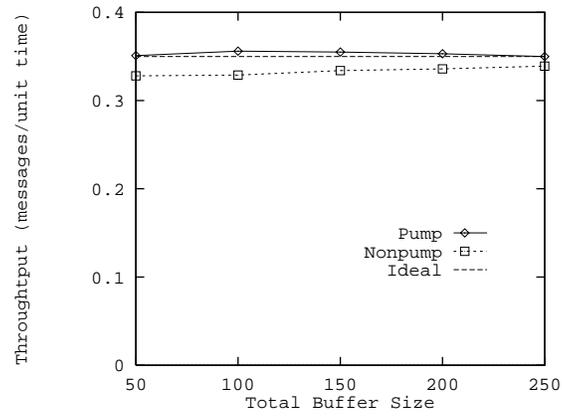


Figure 10: Throughput of session₃₂.

We do not show throughputs of session₁₂, session₂₂, session₃₂ because under the denial of service attack all cases have throughput values around 0.033.

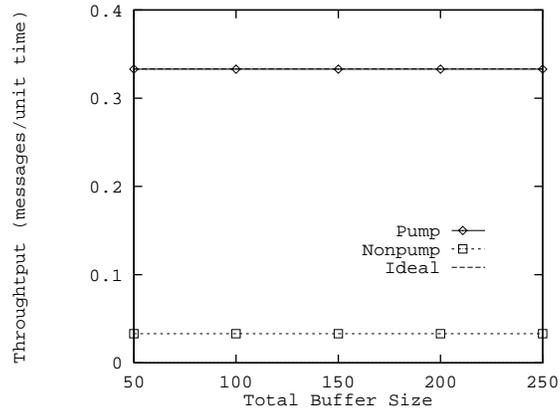


Figure 11: Throughput of session₁₁. Note that the Pump and the Ideal curves are virtually identical.

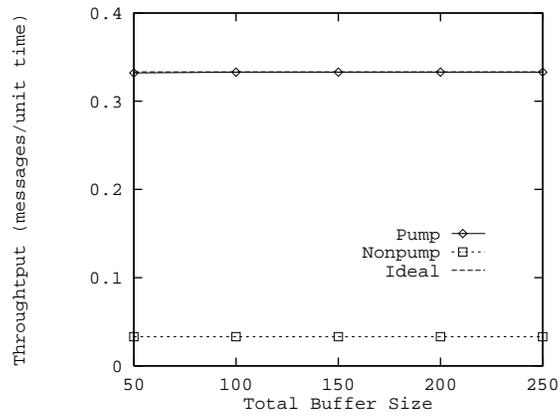


Figure 12: Throughput of session₂₁. Note that the Pump and the Ideal curves are virtually identical.

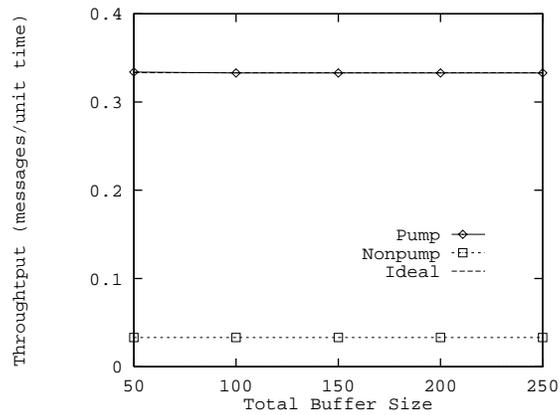


Figure 13: Throughput of session₃₁. Note that the Pump and the Ideal curves are virtually identical.

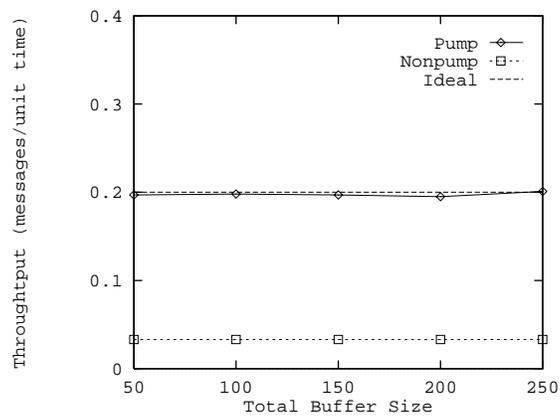


Figure 14: Throughput of session₁₃.

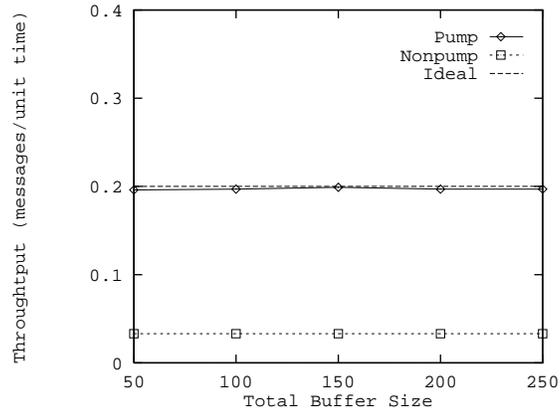


Figure 15: Throughput of session₂₃.

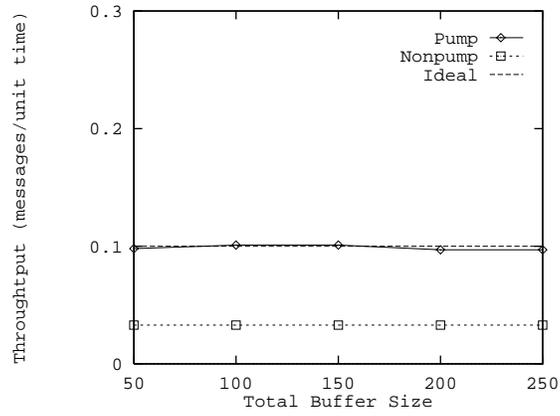


Figure 16: Throughput of session₃₃.

5 Discussion

This paper describes the need for a secure device that can route messages from (multiple) Lows to (multiple) Highs. Even though abstract composition problems have been well studied [11], this paper shows that the actual design of such a device is quite complicated. This secure device should not only meet the requirements of conventional network routers such as performance, reliability, and fairness, but also the requirements of security, such as minimal impact from covert channels and denial of service attacks. The network Pump that we introduced in this paper can balance the above requirements.

It is interesting to note that the network Pump's use of indirect acknowledgements is similar to ideas put forth in the mobile computing community [1]. Further, the network Pump's stable storage is along the lines of "packet caching" used in [25].

One of our future plans includes designing and building the network Pump on top of the ATM layer. We are also investigating extending the network environment described in this paper to deal with a more general lattice-type security structure. We refer to this as the "generalized Pump". Note that if we allow Lows (Highs) to conspire then we must redo our covert channel capacity analysis. We leave this as future work.

Acknowledgement

We wish to thank Ruth Heilizer and the referees for their helpful comments and suggestions.

References

- [1] A. Bakre and B. Badrinath. "I-TCP: Indirect TCP for mobile hosts," 15th Int'l Conf. on Distributed Computing Systems (ICDCS), May 1995.
- [2] J. N. Froscher, D. M. Goldschlag, M. H. Kang, C. E. Landwehr, A. P. Moore, I. S. Moskowitz, and C. N. Payne. "Improving inter-enclave information flow for a secure strike planning application," Proc. 11th Computer Security Applications Conference, pp. 89 - 98, New Orleans, LA, Dec. 1995.
- [3] M. Gasser. Building a Secure Computer System. Van Nostrand Reinhold, 1988.

- [4] M. Gerla and L. Kleinrock. "Flow Control: A comparative survey," *IEEE Trans. Commun.*, vol. 28, no. 4 pp. 553 - 574, Apr. 1980.
- [5] E. L. Hahne. "Round-robin scheduling for max-min fairness in data networks," *IEEE J. Select. Areas Commun.*, vol. 9, no. 7, pp. 1024 - 1039, Sep. 1991.
- [6] M. H. Kang and I. S. Moskowitz. "A Pump for rapid, reliable, secure communication," *Proceedings ACM Conf. Computer & Commun. Security '93*, pp. 119 - 129, Fairfax, VA, 1993.
- [7] M. H. Kang and I. S. Moskowitz. "A data Pump for communication," Submitted for publication, also available as *NRL Memo. Report 5540-95-7771*, 1995 (<http://www.itd.nrl.mil/ITD/5540/publications/CHACS/index1995.html>).
- [8] A. M. Law and W. D. Kelton. *Simulation Modeling & Analysis*. McGraw Hill, 1991.
- [9] J. Lane. "ATM knits voice, data on any net," *IEEE Spectrum*, Feb. 1994.
- [10] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613 - 615, 1973.
- [11] J. D. McLean. "A general theory of composition for a class of "Possibilistic" properties," *IEEE Trans. Software Engineering*, vol. 22, no. 1, pp. 53 - 67, 1996.
- [12] J. K. Millen. "Finite-state noiseless covert channels." *Proceedings of the Computer Security Foundations Workshop II*, pp. 81 - 86, Franconia, NH, 1989.
- [13] A. R. Miller and I. S. Moskowitz. "Reduction of a class of Fox-Wright Psi functions for certain rational parameters," *Computers & Mathematics with Applications*. vol. 30, no. 11, 1995.
- [14] B. E. Montrose and M. H. Kang. "An Implementation of the Pump: Event Driven Pump," *NRL Memo. Report 5540-95-7782*, 1995.
- [15] I. S. Moskowitz and A.R. Miller. "The channel capacity of a certain noisy timing channel," *IEEE Transactions on Information Theory*, Vol. 38, No. 4, pp. 1339-1344, July 1992.
- [16] I. S. Moskowitz and A. R. Miller. "Simple timing channels," *Proceedings 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 56 - 64, Oakland, CA, 1994.

- [17] I. S. Moskowitz and M. H. Kang. "Discussion of a statistical channel," Proceedings IEEE-IMS Workshop on Information Theory and Statistics, p. 95, Alexandria, VA, 1994.
- [18] I. S. Moskowitz and M. H. Kang. "Covert channels — Here to stay?," Proceedings COMPASS '94, pp. 235 - 243, Gaithersburg, MD, 1994.
- [19] I. S. Moskowitz and M. H. Kang. "The Modulated-Input Modulated-Output model," Proceedings IFIP WG 11.3 working conference on database security, Rensselaerville, NY, August 1995.
- [20] B. Mukherjee and S. Banerjee. "Alternative strategies for improving the fairness in and an analytical model of DQDB networks," Proceedings IEEE INFOCOM '91, pp. 879 - 888, 1991.
- [21] R. M. Needham. "Denial of service: An example," Communications of the ACM, vol. 37, no. 11, pp. 42 - 46, 1994.
- [22] J. J. Parsonese. "The basics in networking the data Pump," Working paper.
- [23] C. Shannon and W. Weaver. The mathematical theory of communication. University of Illinois Press, 1949. Also appeared as a series of papers by Shannon in the Bell System Technical Journal, July 1948, October 1948 (A Mathematical Theory of Communication), January 1949 (Communication in the Presence of Noise).
- [24] J. C. Wray. "An analysis of covert timing channels," Proceedings 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 2 - 7, Oakland, CA, 1991.
- [25] H. Xu and B. Bhargava. "Reliable stream transmission in mobile computing environments," Technical Report CSD 95-002, Computer Science, Purdue University, 1995.