A Case Study of Two NRL Pump Prototypes

Myong H. Kang
Ira S. Moskowitz
Bruce E. Montrose
James J. Parsonese
Information Technology Division, Mail Code 5540
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, D.C. 20375

Abstract

As computer systems become more open and interconnected, the need for reliable and secure communication also increases. The NRL Pump was introduced to balance the requirements of reliability, congestion control, fairness, and good performance against those of threats from covert channels and denial of service attacks.

In this paper, we describe two prototype efforts. One implements the Pump at the process (top) layer in terms of a 4-layer network reference model and the other implements the Pump at the transport layer. We then discuss lessons learned and how these lessons will be used in deciding upon the final hardware implementation of the Pump.

1 Introduction

In 1993, Kang and Moskowitz first developed the NRL Pump. Since then the Pump theory has been refined and extended in a series of theoretical papers [4, 5, 6] and applications [8, 2]. This paper discusses the various issues that arose when two implementations of the Pump were developed. These are referred to as the Event Driven Pump (E-Pump), implemented by Montrose [8], and a version of the Pump running on the DOS operating system, implemented by Parsonese, (D-Pump) [11]. The D-Pump and the E-Pump are based on very different philosophies, and both have their pros and cons. We will analyze the design decisions and the performance trade-offs between the two philosophies, and will make use of the lessons learned in our decisions for a final, production level Pump.

The first exposition of the Pump [4, 5] was designed for one, and only one, user/process (Low), sending messages to one, and only one, user/process (High) at a higher security level. This problem grew out of the need for secure and reliable data replication, see [3]. When we wish to be specific, we will refer to this as the "basic Pump." The E-Pump is an implementation of the basic Pump. To ensure security one must not allow any back-traffic from High to Low. Unfortunately, this prevents High from sending back to Low an acknowledgement (ACK) that a message was successfully received. ACKs are necessary to make Low-to-High communication both reliable and recoverable. Communication without ACKs is unreliable because Low does not know if the message arrives at High. Communication without ACKs is unrecoverable because Low may remove the message before High actually receives it. The security problem with ACKs is that the *timing* of the ACKs can be used to covertly send information from High to Low. For example, if High can force the ACKs to arrive at Low at either 1 or 2 ms, with the choice made by High, then this communication is a covert channel which has a capacity [7, 9] of $\log_2 \frac{1+\sqrt{5}}{2} \approx .69$ bits per ms. An obvious solution to this problem would be to remove High's ability to interfere with the timing of ACKs to Low. However, this solution impedes performance because one would have to adopt a worse-case approach and send the ACKs at fixed time intervals.

Kang, Moskowitz, and Lee [6] have extended the Pump to the network environment to handle the situ-

ation of multiple (non-communicating amongst themselves) users/processes at the low level (Low_i) sending a message to one of the multiple (non-communicating amongst themselves) users/processes at a fixed higher level ($High_j$). When we wish to be specific we will refer to this as the "network Pump." The network Pump deals with the added element of fair allocation of resources. The D-Pump is an implementation of the network Pump.

When we discuss something that applies to either the D-Pump or the E-Pump, we will use the term "Pump," and the terms "Low" and "High" will refer to client applications at different security levels.

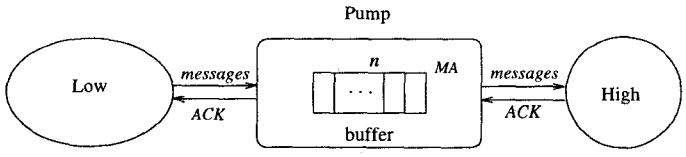

Figure 1. The simplified Pump architecture

The Pump (see figure 1) has several key ideas as follows:

(Intermediate buffer) The Pump, for security reasons mentioned previously, disconnects any direct communication from High to Low by acting as an intermediary between Low and High. Low sends its message to the Pump, *not* to High. The Pump receives the message from Low and retains it in stable buffer until it is accepted by High. The Pump, then sends an ACK to Low, thus replacing direct communication between Low and High by a mediated indirect communication while preserving reliability, and recoverability. Furthermore, the Pump uses historical knowledge of past High behavior to ensure good performance.

(Handshake protocol) The Pump uses a handshake protocol to guarantee reliability. Low waits to send a new message until it has received an ACK of its last message, from the Pump. The Pump can also be used with a sliding window scheme that will allow a certain number of un-ACKed messages to be outstanding.

(Stochastic behavior) The Pump uses a secure (indirect) High to Low flow control mechanism. The Pump keeps track of the rate at which High takes messages out of the Pump and sends ACKs to Low based upon a moving average of the High rate. We accomplish this by having the Low ACK time be a draw of a random variable with mean equal to the High moving average (MA). Thus, by using a stochastic approach to dilute the High influence on the Pump's ACK to Low, we minimize the covert channel threat. However, the average behavior still reflects High's long term behavior, which ensures good performance. In fact the basic Pump performs as well as the (non-secure) store and forward protocol (SAFP) [5] and the network Pump performs as well as any other protocol under round-robin scheduling [6].

The timeline of ACKs are shown in figure 2.

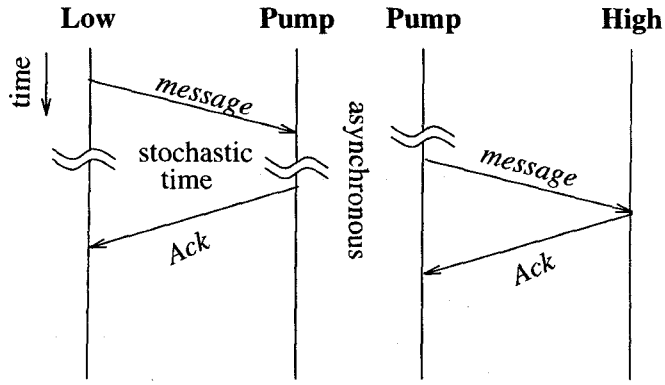

Figure 2. Time diagram of message passing from Low to High

Note that we are not being specific about what a "message" is. This is because it depends on which Pump implementation is used. The E-Pump uses "actual" messages and their ACKs, whereas the D-Pump uses a TCP packet as a message and modulates the TCP ACKs. We will explain this in detail later on in the paper.

2 4-layer model connecting two systems

We restrict our attention to the following (see figure 3) network architecture[1] (similar to [10]) to explain different Pump implementations.

1. **Process layer — P.L.** This is the top-most layer, and the layer that application processes (e.g,. FTP, mail, etc.) use for communication.
2. **TCP layer — T.L.** The communication protocol called transmission control protocol (TCP) is used to provide reliable services to the above P.L. TCP reliably sends packets between the sender and the receiver.
3. **IP layer — I.L.** The internet protocol (IP) provides the basic packet delivery service to TCP.
4. **Ethernet layer — E.L.** This layer handles the protocols needed to manage a specific physical medium, such as Ethernet.

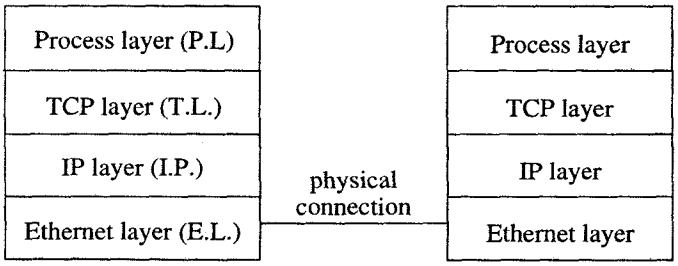

Figure 3. Reference model of network layers

Both the E-Pump and the D-Pump use TCP to send packets. An important distinction between the two Pump implementations is that the E-Pump modulates P.L. ACKs (the true process ACK), whereas the D-Pump modulates T.L. ACKs (TCP ACKs).

2.1 Ethernet

An Ethernet cable can be used for the communication between Ethernet cards on different computers. Ethernet is a "broadcast medium." When a packet[2] is sent out on the Ethernet, every machine on the network sees the packet. Every Ethernet packet has a header that includes the source and destination Ethernet address. Each machine is designed to pay attention only to packets with its own Ethernet address in the destination field. Most networks use Ethernet. Unfortunately, Ethernet has its own addresses. The designers of Ethernet wanted to make sure that no two machines would have the same Ethernet address. Furthermore, they did not want the user to have to worry about assigning Ethernet addresses (in comparison to assigning IP addresses). So each Ethernet card comes with a unique Ethernet address built-in from the factory.

2.2 TCP/IP

Our concern is computers that use internet protocol (IP) to communicate. Each individual computer has a unique IP address (e.g., 192.6.16.23). The IP network software generally needs a 32-bit IP address in order to open a connection or send a datagram[3]. The IP address has two parts: the network address and the host address. If source and destination have the same network addresses, we say they are on the same network; otherwise, they are on a different network.

IP assumes that a system is attached to some local network. It assumes that the system can send datagrams to any other system on the same network. Since we are assuming that Ethernet is handling the physical layer communication, IP simply finds the Ethernet address of the destination system, and puts the datagram (packet) out on the Ethernet. If the destination is on a different network, the datagram is sent to a gateway[4] using a routing protocol. When the gateway receives the datagram, it will forward the datagram to the destination that is on a different network than the originating network.

Note that there is no relationship between the Ethernet address and the IP address. There is a separate protocol for associating an Ethernet address to an IP address, called "address resolution protocol (ARP)." That association is stored in the ARP table. Suppose a process on system 192.6.16.9 (class C IP address) wants to connect to a process on system 192.6.16.7. The originating system will first verify that 192.6.16.7

[1] Our description is not a standard network description. However, it is the description most relevant to understanding the current prototypes of the Pump.

[2] Packet is a unit of message at the Ethernet layer.

[3] Datagram is a unit of message at the IP or TCP layers.

[4] A gateway (or router) is a system that connects a network with one or more other networks. Gateways are often normal computers that happen to have more than one network interface.

is on the same network, so it can talk directly via Ethernet. Then it will look up 192.6.16.7 in its ARP table, to see if it already knows the Ethernet address. If the system is not in the ARP table, it uses the ARP protocol to broadcast an ARP request. When the destination system sees an ARP request for itself, it is required to respond. So 192.6.16.7 will see the request, and will respond with an ARP reply saying in effect "192.6.16.7 (IP address) is 8:0:20:1:55:33 (Ethernet address)." Now the communication can proceed.

TCP, which handles communication at the T.L., is responsible for making sure that messages get through to the other end. It keeps track of what is sent, and retransmits anything that did not get through. If any message is too large for one datagram, e.g. the text of a mail message, TCP will split it up into several datagrams, and make sure that they all arrive correctly. Since these functions are needed for many applications, they are put together into a separate protocol, rather than being part of the specifications for sending mail. One can think of TCP as forming a library of routines that applications can use when they need reliable network communications with another computer. Similarly, TCP calls on the services of IP. Although the services that TCP supplies are needed by many applications, there are still some kinds of applications that do not need them (i.e., some applications may use User Datagram Protocol (UDP) instead of TCP). Note [1] has implemented one-way links at the T.L.

2.3 Processes

There are many P.L. protocols (e.g., mail, FTP). The applications of interest with respect to the Pump are those that pass messages from one host to another (e.g., data replication, mail, FTP). The job of the Pump is to act as an intermediary in the action of Low sending a message to High, and mimicking the ACK of this message from High back down to Low. The E-Pump implements the Pump protocol at the P.L. (i.e., the E-Pump modulates P.L. ACKs). In contrast, the D-Pump modulates the TCP layer ACKs. Of course, we must have high-assurance that our implementations do not allow the high side to affect ACKs at any of the other layers.

3 The E-Pump

The E-Pump is software that resides on Wang's (HFSI's) XTS-300 platform. The XTS-300 was chosen because of (1) its availability, and (2) its B3 rated operating system (STOP 4.1). We use an evaluated operating system in order to avoid evaluating the system services that are used by the E-Pump (e.g., interprocess communication, file-system, etc.). STOP can be described by rings of privileges as shown in figure 4 and discussed below.

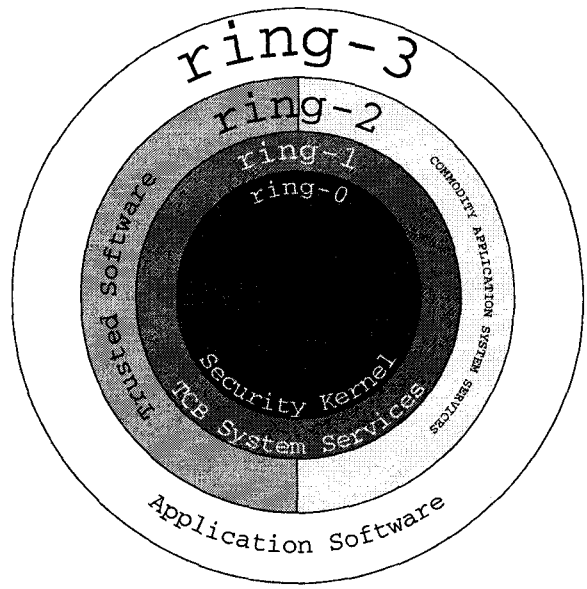

Figure 4. STOP security ring structure

The security kernel provides basic system operating services (e.g., resource management, process scheduling, interrupt, trap handling) and enforcement of system security (e.g., security and integrity rules). Privileged software known as Trusted Software provides additional security services outside the kernel. Commodity Application System Service (CASS) provides untrusted operating system services to application programs on the XTS-300.

The XTS-300 supports trusted processes. A process is trusted if the process has privileges that exempt it from specific access control rules (e.g., no read-up or no write-down rule). Since the basic Pump sends ACKs back to Low, a portion of the basic Pump must have trusted processes.

Ideally, the entire E-Pump should be implemented as trusted ring-2 processes. Since ring-2 processes on the XTS-300 currently cannot access TCP/IP, it was

necessary to implement the portions of the Pump requiring access to TCP/IP in ring-3. The file-system is the only mechanism available for communication between ring-3 to ring-2 processes on the XTS-300. We used special files, called FIFOs, with a first-in-first-out protocol, as the communication channel between the ring-2 and ring-3 components of the E-Pump.

We refer to each component of the Event Driven Pump as an Object. Each Object was implemented as a separate process designed to accomplish a specific task when certain events occurred. An event is either a message sent by another Object or the completion of an I/O operation. Inter-Object message passing was accomplished via the Interprocess Communication (IPC) interface provided by the operating system.

Two store and forward buffers (SAFB) were used; one in volatile shared memory and the other in non-volatile disk storage. The two buffers together make up the "Pump buffer" for performance and recoverability reasons. High reads out of volatile memory but ACKs are not sent to Low until the message is written to disk. All ring-2 Objects have access to both SAFBs. The non-volatile SAFB was required for recovery purposes in the event the XTS-300 should halt during operation. The volatile SAFB serves as the I/O buffer for data transferred to the non-volatile SAFB, the Output FIFO, and the Input FIFO.
A brief description of each Object follows.

S2F It delivers data from the low application to the Input Data FIFO and relays the Pump's ACK from the Low ACK FIFO to the low application.

Memory Writer It reads data from the Input Data FIFO, stores the data into Shared Memory, and sends IPC messages to the Disk Writer and Memory Reader Objects, informing them that there is data to be processed. It also sends an IPC message to the ACK Object so that it can keep track of timing (e.g., timeout (NAK), disk writing overhead).

Memory Reader It reads data from Shared Memory and writes the data to the Output Data FIFO. It also sends an IPC message to the Moving Average Object to trigger the moving average computation.

ACK It computes the randomized delay based on the moving average and sends an ACK to the Low ACK FIFO after data is safely stored in the non-volatile SAFB.

Lo Timer It is the timer for the ACK Object.

Moving Average It computes the moving average and sends the updated moving average to the ACK Object.

Hi Timer It is the timer for the Moving Average Object.

Disk Writer It writes data to the disk and sends an IPC message to the ACK Object which indicates that the data is safely stored in the non-volatile SAFB. Unfortunately, this is a costly time operation.

Free Record It reads an ACK from the High ACK FIFO, sends an IPC message to the Moving Average Object so that the new moving average can be computed, and removes the data from the disk (*Zap Hdr*). Then it sends an IPC message to the Memory Writer Object to indicate the space is now available.

F2S It delivers data from the Output Data FIFO to the high application and sends ACKs to the High ACK FIFO upon successful delivery of the data to the high application.

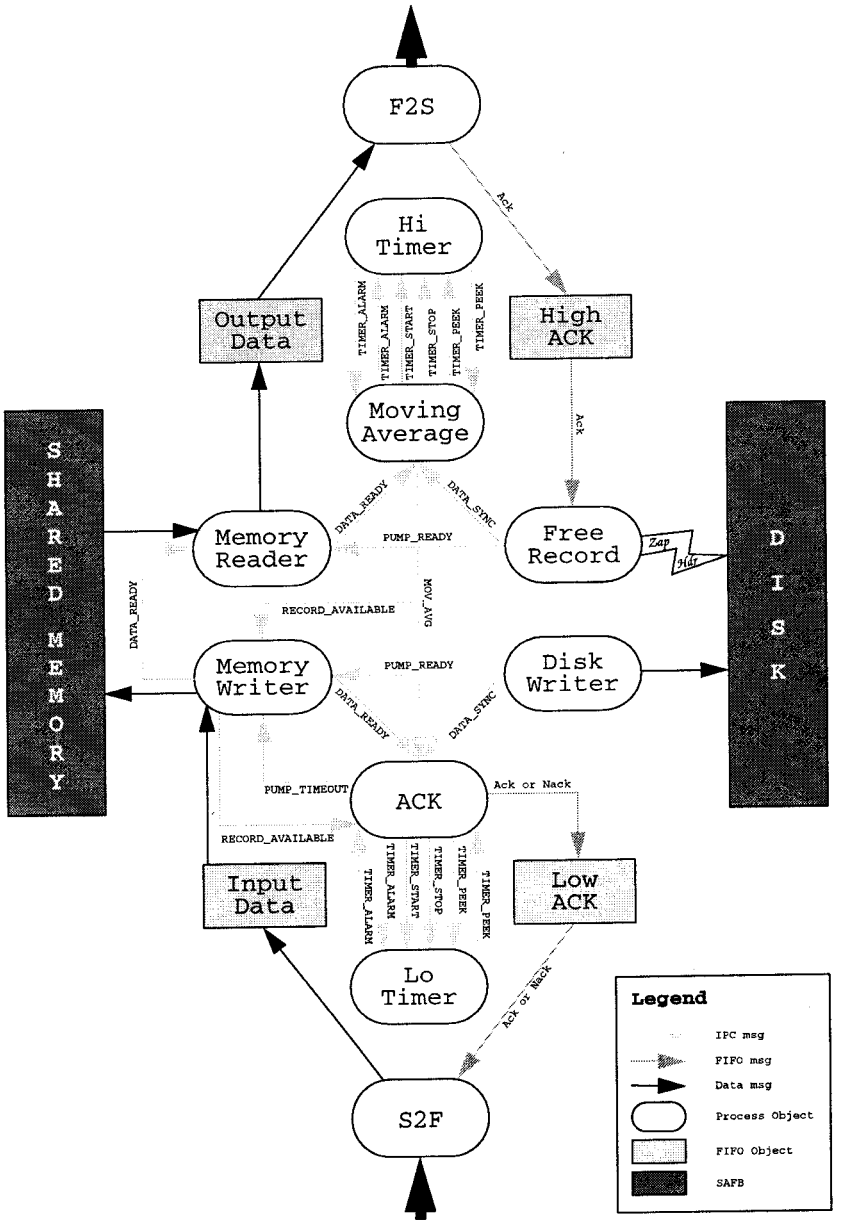

Figure 5: The Event Driven Pump

Not shown in figure 5 are two additional processes called the Recovery Object and the Create Pump Object. The Recovery Object is responsible for the orderly recovery from system failure. The Create Pump Object sets up all data structures and creates all the other Objects described above including the Recovery Object. Unlike the other Objects described above, the Recovery Object and Create Pump Object will terminate after their tasks have been completed.

The E-Pump uses a customized Pump protocol (e.g., [8]) which is a P.L. protocol, in contrast, the D-Pump uses TCP protocol to communicate. The E-Pump modulates P.L. ACKs according to the basic Pump algorithm [8] as shown in figure 6.

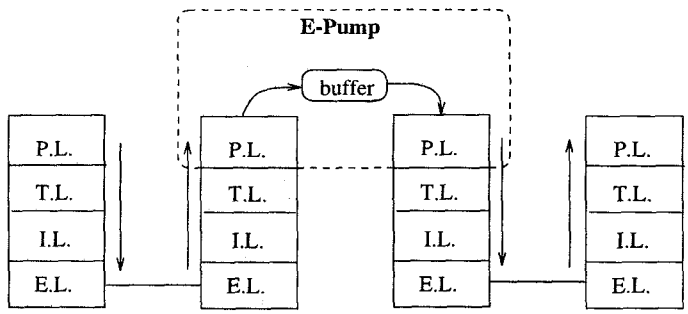

Figure 6: Communication path of E-Pump

4 The D-Pump

TCP is used by many applications (e.g., FTP) to ensure reliability at the T.L. Due to this we thought that the Pump could be implemented at the T.L. by means of adjusting the TCP ACK stream. This implementation of the Pump is called the D-Pump. This modified TCP should be transparent to the application. In section 5 we discuss problems that arise from implementing the Pump at the T.L.

The D-Pump exists as software on a single board 486 class PC with one low Ethernet card and one high Ethernet card. It implements the network Pump algorithm [6]. This algorithm is a modification of the basic Pump algorithm designed to deal with the added issue of multiple low senders at the same security level, and multiple high receivers at the same security level. It is further assumed that the Lows (Highs) do not communicate among themselves. The modification does not affect our comparisons in this paper. The D-Pump uses customized TCP software that modulates the ACKs in the T.L. Recall that the E-Pump modulates ACKs in the P.L.

DOS was chosen as a prototyping environment for the D-Pump in order to host a TCP/IP protocol stack that could easily be ported to any other host or embedded system. Furthermore, the use of DOS, which is one of the simplest operating systems, facilitates the development of custom hardware that would provide a high-level of assurance that the D-Pump code reliably performs the functionality of the (network) Pump. The following steps describe the D-Pump:

1. The Pump listens to ARP requests.
2. If an ARP request contains an IP address of any high systems to which the Pump is configured to deliver messages, then the Pump replies to the ARP request by sending its low Ethernet address.
3. The low sender establishes the TCP connection to the Pump.
4. The Pump establishes the TCP connection to the high receiver on behalf of the low sender (the Pump uses IP addresses and port numbers that are available to the Pump[5]).
5. Activities at the low side of the Pump:
   (a) The low portion of the Pump receives messages and puts them in the Pump's buffer.
   (b) Send TCP ACKs to the low sender according to the Pump algorithm.
   (c) Continue step (a) and (b) until the low sender discontinues the connection.
6. Activities at the high side of the Pump:
   (a) The Pump delivers messages from its buffer to the high receiver.
   (b) The Pump receives ACKs from the high receiver and updates the moving average.
   (c) Continue step (a) and (b) until the low sender discontinues its connection and there are no more messages destined for the high receiver in the buffer.
7. The low sender discontinues the connection to the Pump.

[5] We modified the IP layer so that address information is available to the T.L. We also modified the Ethernet layer to timestamp packets.

8. The Pump discontinues the connection to the high receiver.

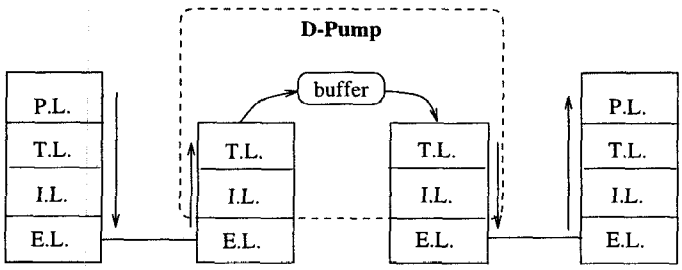

Figure 7: Communication path of D-Pump

5 Comparison of the E-Pump and the D-Pump

In this section, we compare the two approaches of our prototypes. The major difference of interest to us in this paper is that the E-Pump implements the Pump at the P.L., while the D-Pump implements the Pump at the T.L.

5.1 Performance

With respect to overhead, the D-Pump is a much more efficient implementation than the E-Pump. Some of the additional overhead of the E-Pump is as follows:

- Since ring-2 processes on the XTS-300 cannot access TCP/IP, the portions of the E-Pump that are needed to access TCP/IP were implemented in ring-3. Hence, four extra FIFOs (Output data, Input data, High ACK, and Low ACK in figure 5) and two extra processes (F2S and S2F) are used in the E-Pump which are unnecessary in other implementations.
- The E-Pump writes every message from Low to disk, for recovery reasons, before it sends an ACK. However, if the D-Pump wrote every message to disk (which is the most desirable recoverability option) then the same additional overhead would be incurred.
- The message does not have to go up to the P.L. in the D-Pump (i.e., it stops at T.L.), thus it requires less processing time.
- Since the D-Pump's trusted code includes TCP/IP code, it can access IP addresses and port numbers. Hence, the D-Pump does not need any customized address scheme because it uses the TCP protocol. However, since the E-Pump cannot access this address information at the P.L., the address scheme must be added to the Pump protocol, thus increasing overhead.

The above reasons seem to imply that implementing the Pump at the T.L. is a good design decision. However, as we will discuss in the following sections, problems exist with our T.L. implementation.

5.2 The Pump and its wrappers

The Pump is designed to be an application independent device, receiving messages from Low (sender) and delivering them to High (receiver). We emphasize that the Low to High communication is actually a communication between application programs running on Low and High computers. Also, note that application programs that expect some data values coming back as a result of computation cannot be used with the Pump because the Pump does not allow data values to pass through itself. Therefore, the only application programs that can utilize the Pump are those which can function with only simple ACKs (or no ACKs) from the receiver program.

In general, an application program that functions with (simple) ACKs is able to ensure reliability at the application level. Thus, when the sender program sends a message, it expects the receiver program to return an ACK in the specific format determined by the application protocol. If the sender program directly sends a message to the Pump, the Pump, which is an application independent device, cannot return an ACK in the format specified by the application protocol.

Therefore, *wrappers*, which are application-specific, are needed to ensure the correct formatting of application ACKs. In other words, wrappers satisfy both the application-specific protocol and the Pump protocol. Even though the D-Pump can satisfy the ACK requirement at the T.L., it cannot satisfy the application-specific ACK requirements at the application layer. Hence, both Pumps require wrappers to correctly interface with the specific application program. We emphasize that for most applications that utilize TCP, satisfying TCP's requirements are not enough to satisfy the application-specific protocol requirements.

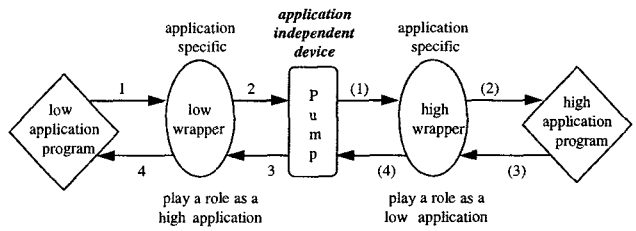

Figure 8. The Pump and wrappers

A brief description of wrappers[6] and their interactions with the Pump and application programs, as illustrated in figure 8, is as follows:

1. The low application program sends (if available) an application message to the low wrapper. The low wrapper reformats the application message into the format that the Pump understands, and produces a Pump message(s). Sometimes one application message may be transformed into several Pump messages. Note that in the E-Pump, messages are created at the P.L. and ACKs are in the P.L., whereas in the D-Pump, messages and ACKs are at the T.L.

2. The low wrapper then sends a Pump message to the Pump.

3. The low wrapper receives an ACK from the Pump.

   - Repeat 2 and 3 until the Pump message(s) corresponding to the original application message have been passed to and ACKed by the Pump.

4. The low wrapper sends an application specific ACK to the sender program. Go to 1.

Hence, the low wrapper is a *proxy* of the high application program that receives a message from the low application program.

Similarly, the high wrapper is a *proxy* of the low application program that receives a message from the Pump and deliver it to the high application program.

(1) The Pump sends (if available) a Pump message to the high wrapper.

- The high wrapper receives a Pump message. If more Pump messages are required to construct a complete application message, then go to (4).

(2) The high wrapper sends an application message to the high application program.

(3) The high wrapper receives an ACK from the high application program.

(4) The high wrapper sends an ACK to the Pump. The Pump message is now removed from the Pump's buffer upon receiving the ACK. Go to (1).

As we discussed earlier, in general, both Pumps need application-specific wrappers at the P.L. The major differences between the two Pump implementations are:

- Since the E-Pump has been implemented at the P.L., the lower layer protocols are transparent to the E-Pump.

- Since the D-Pump has been implemented at the T.L., TCP requirements must also be satisfied.

5.3 Reliability & Recoverability

We wish to have reliable communication between the sender and the receiver. The Pump should guarantee reliability from the sender to the Pump, and from the Pump to the receiver. However, reliability is not composable. For example, after the Pump receives a message and the sender receives an ACK, the sender may perform garbage collection. If, at this time, the communication from the Pump to the receiver breaks and later it is re-established, then orderly recovery from system failure between the Pump and the receiver is needed to guarantee reliable[7] and secure sender to receiver communication.

The D-Pump does not wait for ACKs from the high application (or wrapper if the use of wrappers is required) but rather it uses TCP ACKs for computing the moving average and garbage collection. In general, TCP ACKs from High to the Pump are controllable neither by the application program nor by the wrappers. Hence, there is a possibility the packet will be

[6] A detailed example of wrappers can be found in [2].

[7] If the sender and receiver applications are recoverable (e.g., database replication) then the Pump also preserves the recoverability.

removed from the D-Pump before the high application receives it and stores it in a safe place. Therefore, the D-Pump is not recoverable and cannot guarantee reliable Low to High communication. This is an example of "using a reliable underlying communication protocol at a layer below that of the applications, does not guarantee reliable application to application communication."

On the other hand, the E-Pump waits for ACKs from the high wrapper before it removes messages from its buffer. Therefore, the E-Pump has a recoverable protocol (i.e., the ACK from the high wrapper is a P.L. ACK and guarantees the delivery of the message to the application program or stores it in a recoverable place).

5.4 Routing between two different networks

The Pump resides between low and high systems (enclaves) and relay messages from a low to high system. Generally, low and high systems (enclaves) reside in different networks.

As explained earlier, the D-Pump responds to ARP requests. However, it does not understand routing protocols. Hence, the D-Pump works fine if the high network and the low network share the same network address as shown in figure 9.

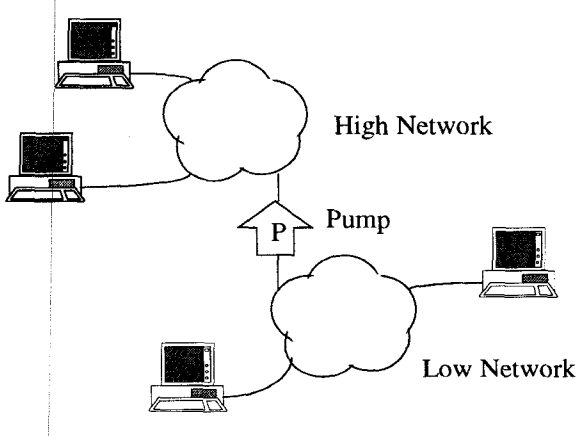

Figure 9. D-Pump between the same logical network

If however, the high system and the low system have different network addresses (referred to as the High and Low networks, respectively) then a router is required to send messages from low to high systems as shown in figure 10.

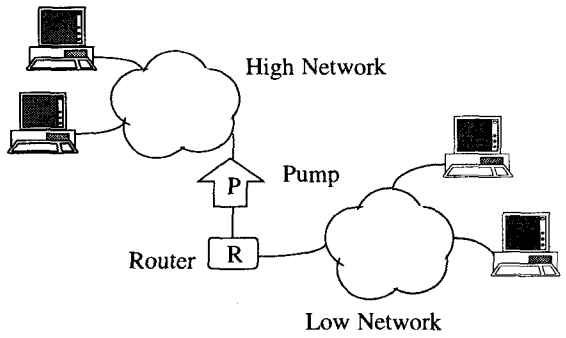

Figure 10. D-Pump between the two different logical networks

Since, the destination address is in a different network, messages are sent to the router. The router sends out ARP requests to the high network to find the correct system to relay the message to. The D-Pump responds to the ARP request (see section 4 steps 1, 2, and 3) and hence the D-Pump receives messages. Note that the router can also be located after the Pump.

On the other hand, the E-Pump does not require a router even if low and high systems are located in different networks. Since the E-Pump is implemented at the P.L., the routing can be done by the E-Pump. The E-Pump looks at the destination address and determines whether the address is a legitimate destination address or not.

6 Lessons Learned & Future Plans

In this section, we describe the major lessons learned from two prototype efforts and discuss our future plans.

6.1 Lessons Learned

- *(Learned from the D-Pump)* The T.L. is the wrong layer at which to implement the Pump. The reasons are:

    1. In general, high side processes do not have control over T.L. ACKs. Thus a T.L. layer

implementation results in reliability and recoverability problems.

2. Usually, high side and low side systems are in separate networks. Hence, the Pump must be able to perform simple routing tasks. Therefore a T.L. implementation would either have to have much additional code to handle the routing, or we would be forced to use an external router.

- *(Learned from the E-Pump)* Using an underlying high assurance secure OS for the Pump causes a large overhead due to the lack of networking support. Furthermore, the development process was too "painful" due to the lack of development tools.

- *(Learned from the E-Pump and D-Pump)* Using a hard disk as stable storage affects performance too much. Writing to disk is just too slow if High is acting as a fast server. An alternative being investigated is static RAM.

6.2 Future Plans

- Build the Pump at the P.L.[8]

- Implement the Pump on customized hardware to reduce the amount of trusted code, instead of using a secure OS for building the Pump.

- Develop a better recovery strategy/strategies. One idea is to use an uninterrupted power supply/backup power source for power failure and write to the hard disk only when a power failure occurs. This is still not as recoverable as writing to disk each time but it is much more efficient.

- Implement the Pump on top of an unreliable but efficient T. L. protocol (e.g., UDP). Reliability problems can be handled by the Pump protocol.

A personal lesson learned by the first two authors is that going from theory to working device is a hard road, full of many potholes.

[8] If high systems are in multiple networks, the Pump would require either additional Ethernet cards or extra routers.

Acknowledgement

We thank Ruth Heilizer, John McLean, and the anonymous referees for their helpful comments and suggestions. The second author acknowledges the hospitality of the Issac Newton Institute for Mathematical Sciences, University of Cambridge.

References

[1] J. Davidson, "Asymmetric Isolation," To appear 12th Annual Computer Security Applications Conference, 1996.

[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," Proceedings of the 11th Annual Computer Security Applications Conference, New Orleans, pp. 89 - 98, December 1995.

[3] M. H. Kang, J. N. Froscher, and O. Costich, "A practical transaction model and untrusted transaction manager for multilevel-secure database systems," The Eighth IFIP Workshop on Database Security (1992).

[4] 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.

[5] 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).

[6] M. H. Kang, I. S. Moskowitz, and D. C. Lee, "A Network Pump," IEEE Transaction on Software Engineering, vol. 22, no. 5, pp. 329 - 338, May, 1996

[7] J. K. Millen. "Finite-state noiseless covert channels." Proceedings of the Computer Security Foundations Workshop II, pp. 81 - 86, Franconia, NH, 1989.

[8] B. E. Montrose and M. H. Kang, "An Implementation of the Pump: Event Driven Pump," NRL Memo. Report 5540-96-7850, 1996.

[9] 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.

[10] W. R. Stevens, "UNIX Network programming," Prentice Hall, 1990.

[11] J. J. Parsonese, "The basics in networking the data Pump," Working paper.