The Intel® Management Presence Server (MPS)

In previous generations of Intel® Active Management Technology (Intel® AMT), a client system (computer) could be managed only when it was connected to the corporate network. The new "Fast Call For Help" mechanism for remote access of Intel AMT systems changes this by allowing a management console in the corporate network to connect to an Intel AMT client outside the network, using an intermediate Management Presence Server (MPS) in the corporate demilitarized zone (DMZ).

The Intel MPS is a first generation product for use by manageability independent software vendors (ISVs). It provides a design and implementation which ISVs can use directly, or refer to as a model for their own implementations. MPS is in production use in a pilot project.

In this article, we focus on MPS's overall end-to-end flows, describe its design and implementation, and also its relationships with third-party entities (such as the Apache[1] Web server and **stunnel**, a general-purpose, multi-platform SSL wrapper program). We also discuss the challenges we faced, the resolution of those challenges, and some interesting future challenges related to standardization, scalability, and usability of MPS.

[1] The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

Legal Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined”. Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Table of Contents

1 Introduction

Previous generations of platforms enabled with Intel vPro technology could be managed only when connected to the enterprise network. While this was adequate for desktop personal computers (PCs), it does not work for managing mobile PCs (notebook and laptop computers) in an enterprise. IT shops need a ubiquitous manageability solution that works for both kinds of computers. Since more and more enterprises are increasing their purchases of notebook and laptop computers, starting in 2008, with the aid of Intel vPro technology, IT shops will be able to manage mobile client systems that are outside the enterprise firewall boundaries. These are the main usages supported by this technology:

- **Remediation**: Assisting remotely situated corporate users by diagnosing and repairing malfunctioning systems. This capability extends the outreach of Intel vPro technology (such as the ability to be booted from remote media [IDE-Redirection] and BIOS screen redirection [Serial-Over-LAN (SOL)]), to Internet-based systems.
- **Periodical maintenance operations**: IT administrators can configure systems with Intel vPro technology to connect back to the enterprise network, extending the reach of Intel vPro technology functionality, such as hardware and software discovery, to Internet-based systems.
- **Alerting**: Intel AMT systems can send alert notifications upon the occurrence of pre-configured platform events. One example of this is the Intel vPro technology feature based on Agent Presence, known as alert notification. Once again this extends the alerting capability of Intel vPro technology to outside the corporate firewall.

2 Background: System Manageability and Remote Connectivity

The various manageability standards for client systems, starting with Wired for Management (WfM) [1], continuing to Alerting Standard Format (ASF) [2], Common Model Information_Extensible Markup Language (CIM_XML) [3], and most recently, Desktop and Mobile Architecture for System Hardware (DASH) [4], all utilize a similar Client/Server model. In this model, a management server typically operates as the client, initiating a session to the managed node, which operates as a server and accepts connections from a management server. These protocols assume a typical LAN network. Figure 1 demonstrates a manageability session per the DASH specification:

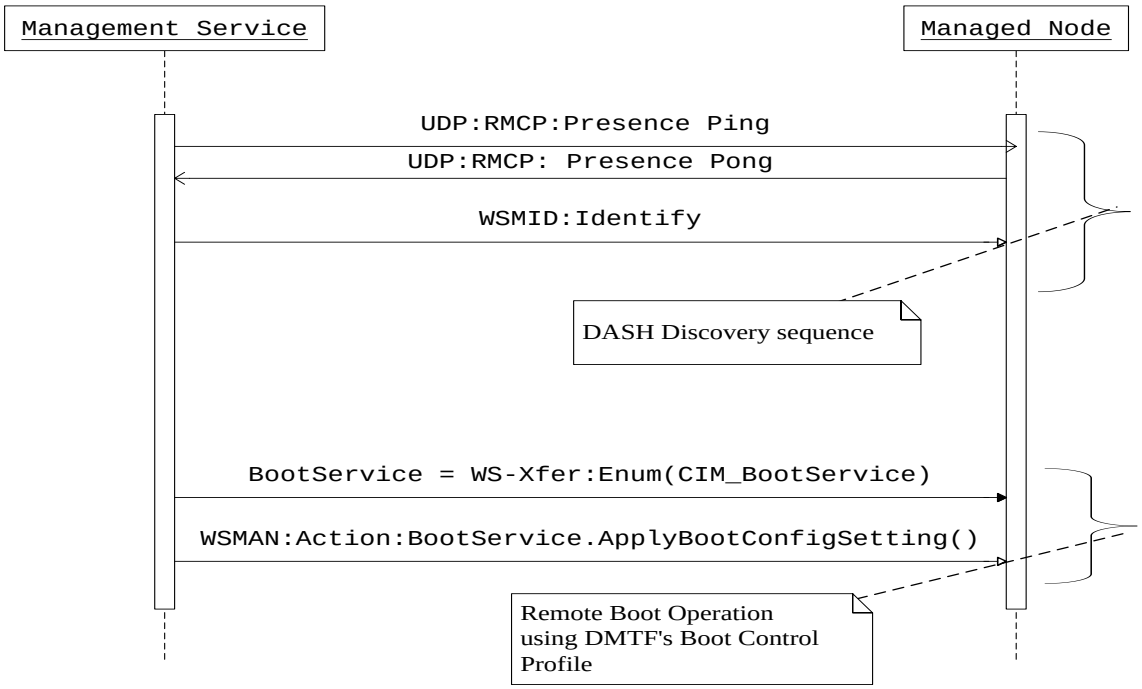

Figure 1: DASH Compliant Session

Per DASH, the management server can discover DASH-compliant managed nodes by using a User Datagram Protocol (UDP)-based presence pong sequence, followed by a Web-Service for Management [5] (WS-MAN) Identity transaction. The subsequent transactions in Figure 1 illustrate how the management server initiates a DASH-compliant remote boot control operation [12], whereby the management server operates as a HyperText Transfer Protocol (HTTP) client. Now suppose that the managed node is a corporate laptop PC situated at the employee's home and connected to the Internet through a home router. The client/server connectivity model described in Figure 1 fails in this environment, due to the following reasons:

- Discovery: The discovery mechanism is not adequate: scanning Internet IP addresses to locate a corporate-managed node is not feasible, due to security and scalability factors. Furthermore, it is quite common for the two entities to reside in different IP address domains: the management console is situated in the corporate Intranet and the laptop is situated behind a home router that provides Network Address Translation [NAT]); therefore, no route exists for the UDP messages. Indeed, most other Intranet discovery methods that are initiated by a management server (such as Domain Name Service [DNS] queries) would equally fail.
- Connectivity: In many circumstances a laptop connects to the Internet through a router providing NAT. NAT allows clients to seamlessly establish outbound sessions, but does not provide the means for a client to seamlessly operate as a server and accept inbound sessions.

The manageability operations provided by Intel vPro technology support DASH and extend the manageability operations by accommodating additional features, such as the aforementioned IDE redirection. When addressing the remote connectivity problem with systems enabled with Intel vPro technology, we had to introduce a remote connectivity technology that would have limited impact on the existing manageability protocols; more specifically, we had to maintain the client/server paradigm for management servers. Hence, we had to allow the manageability ecosystem to benefit from remote connectivity operations with minimal impact on deployment.

The two prominent technologies for remote connectivity are Virtual Private Networks (VPNs) running over either IP Security (IPSEC) [6] or Secure Socket Layer (SSL) [7]; we elected to build the remote connectivity solution based on SSL, due to its superior connectivity and interoperability over IPSEC. One additional key factor was that the firmware already contained an SSL stack, and by reusing SSL we reduced the need for additional flash storage, which would otherwise increase the cost of systems enabled with Intel vPro technology.

The remote connectivity protocol for Intel vPro technology is the Intel AMT Port-Forwarding Protocol (APF), thus named because it uses a port-forwarding technique similar to that defined by the Secure Shell [8](SSH) protocol. APF provides a generic routing capability that allows all existing manageability protocols utilized by Intel vPro technology, such as WS-MAN and IDE-Redirection as well as future protocols, to be supported with systems running Intel vPro technology, operating outside the enterprise firewall. Furthermore, APF allows for emerging manageability solutions, such as Intel Connect Service [14], but a discussion of these is outside the scope of this article.

A remote connectivity solution is not complete without a gateway server or service application. To enable Intel AMT remote connectivity, Intel developed a gateway software product called the Manageability Presence Server (MPS), depicted in Figure 2, which we provide to ISVs as a reference design and implementation of a gateway server.

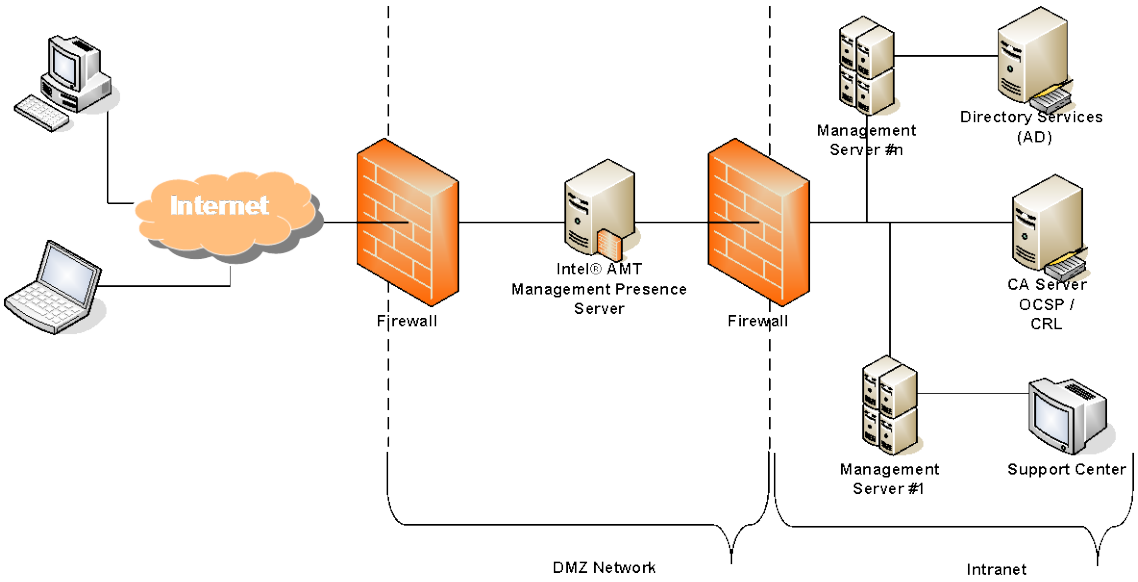

Figure 2: Intel MPS in a Typical Corporate Deployment

MPS has three major tasks:

- It provides an Internet-facing secure service to identify, authorize, and communicate with remotely situated systems running Intel vPro technology.
- It informs enterprise management servers whenever a system enabled with Intel vPro technology connects or disconnects to MPS.
- Moderates the flow of manageability operations to and from the manageability servers.

We describe the specifics of MPS in the following sections, highlighting the most important APF messages throughout.

3 Protocols for Intel AMT Remote Connectivity

In this section we present the various communication flows enabling remote connectivity communication from a management server to a system, enabled with Intel vPro technology, utilizing Intel MPS. As noted previously, the communications between Intel MPS and a system, running Intel vPro technology, are conducted using an SSL session as the underlying transport layer. Within the secured SSL session, APF is used to mediate and multiplex management operations. APF is derived from the SSH Connection [8] protocol.

From a management server perspective, an MPS acts as a standard HTTP proxy and as a SOCKS v5 server [9] [23]. The HTTP proxy is used to mediate the various manageability protocols, such as WS-MAN and Intel AMT HTML Web pages, which are all based on HTTP. SOCKS v5 is used to route the IDE-Redirection and Serial over LAN protocols. SOCKS v5 provides a generic routing mechanism for any protocol based on TCP/IP, and hence it can be used to accommodate future protocols. The following sections explore how these protocols operate in concert to allow remote connectivity operations.

3.1 High-Level Flow

We begin with a high-level view of a remote connectivity session for Intel AMT.
In Figure 3 an Intel AMT device, residing outside the enterprise firewall, establishes an SSL channel to the MPS. Manageability servers route the manageability transactions (such as the remote boot operation we mentioned earlier) to the MPS. These transactions are proxied by the MPS to the Intel AMT system through the SSL channel. The Intel APF protocol is used to bind multiple sessions on top of a single SSL channel.

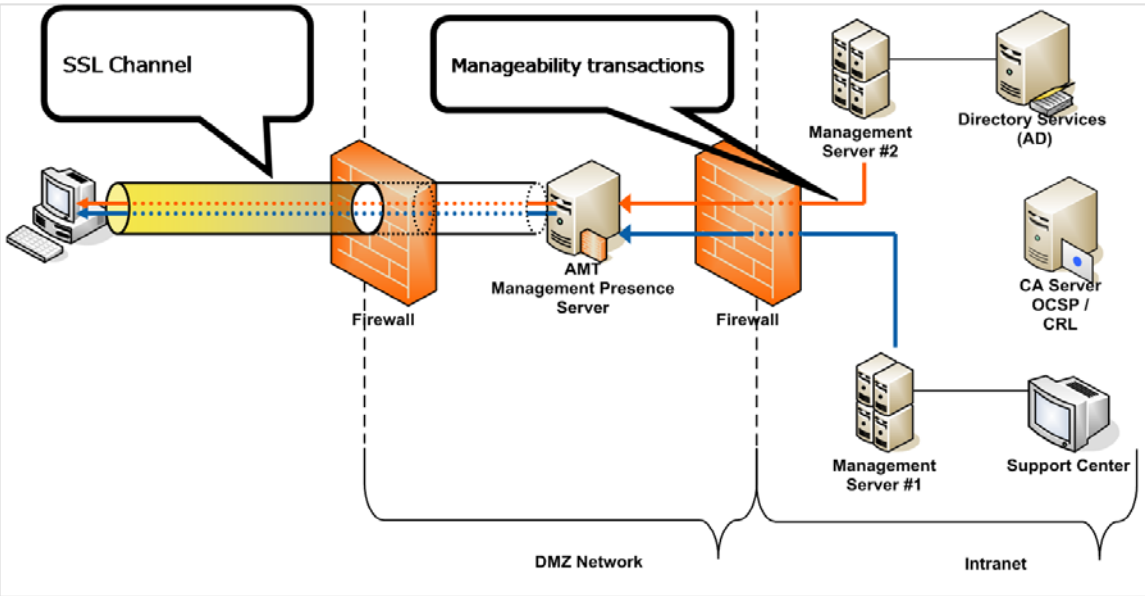

Figure 3: Intel AMT Remote Connectivity Session

In the following sections we explore in detail the various stages of the manageability session.

3.2 Establishing Presence

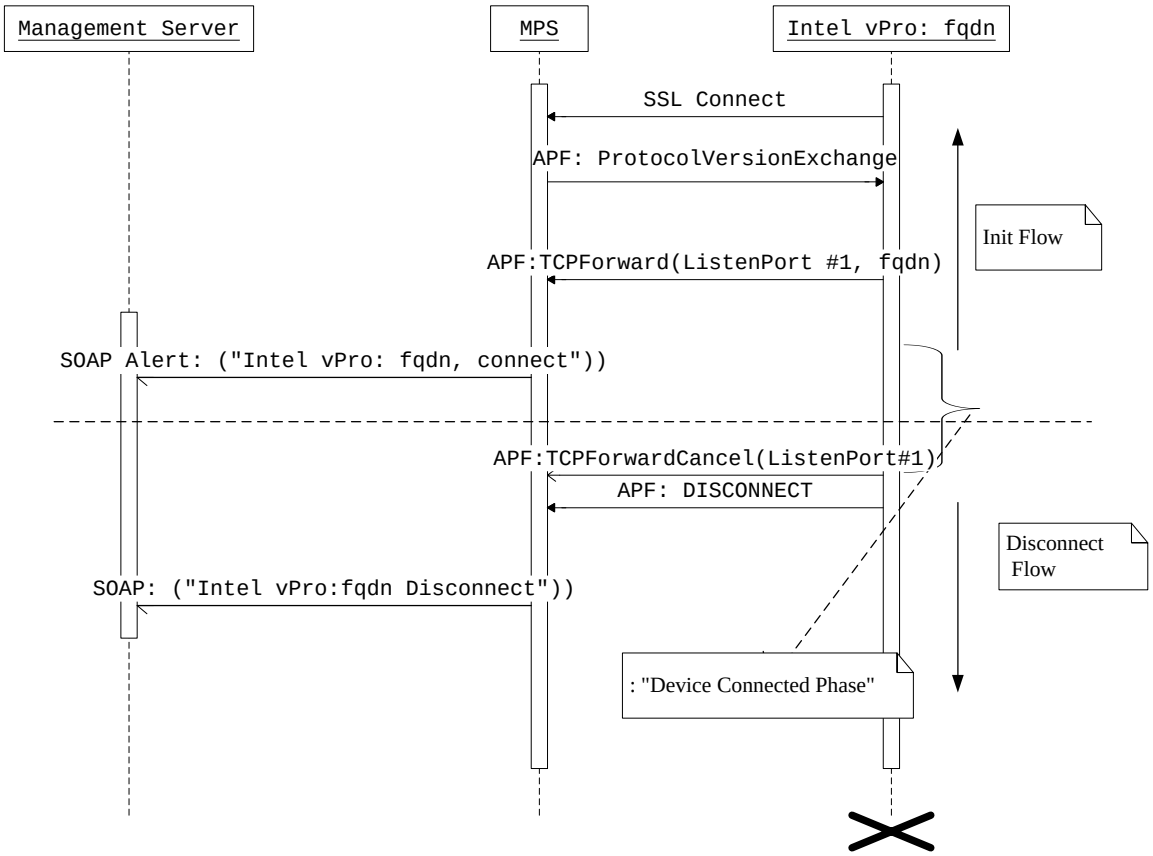

Figure 4: Remote Connectivity Session Begin/End

Figure 4 depicts the steps needed to establish and tear down a manageability session. We go through these steps in the following subsections.

3.2.1 Session Initiation

- In the first step, the system, enabled with Intel vPro technology, establishes an SSL session with an MPS. (The MPS to use is specified when the Intel AMT system is configured.) As part of the SSL session, the system verifies the authenticity of the MPS, based on the SSL certificate provided by the server. Briefly, Intel AMT validates that the certificate is issued by a trusted root certificate for the Fully Qualified Domain Name (FQDN) of the MPS. (The trusted root certificate is embedded in the Intel AMT device's flash storage.) The precise validation process is identical to that performed by HTTP clients in HTTPS [10]. Optionally, MPS can verify the identity of the system, if the system is configured with an SSL client certificate. Since many enterprises today do not deploy a Public Key Infrastructure (PKI) system required to distribute client certificates [11], APF supports an additional proprietary client-side authentication scheme.
- Once a secure SSL session has been established, the two peers initiate APF, and the device provides FQDN identification information to the MPS as part of the APF:TCPForward message.
- Finally, MPS associates the system's FQDN with its currently active session and sends notification to the registered manageability servers indicating that the given device is "connected". (MPS maintains a list of manageability servers to receive these notifications; all registered servers receive all notifications.) Subsequently, MPS uses the FQDN inside HTTP requests from management consoles to find the correct SSL session on which to forward the data.

3.2.2 Session Tear Down

There could be various session tear-down flows. Figure 4 illustrates a "graceful" tear down. However, whenever MPS detects that a given system is not connected, it sends a notification to the registered management servers indicating a device "disconnect" event.

3.2.3 Device Connected Phase

We define the "device connected phase" as the period of time following a successful session initiation and prior to the session tear down. Manageability operations take place during this phase.

3.3 Routing Web-based Manageability Operations

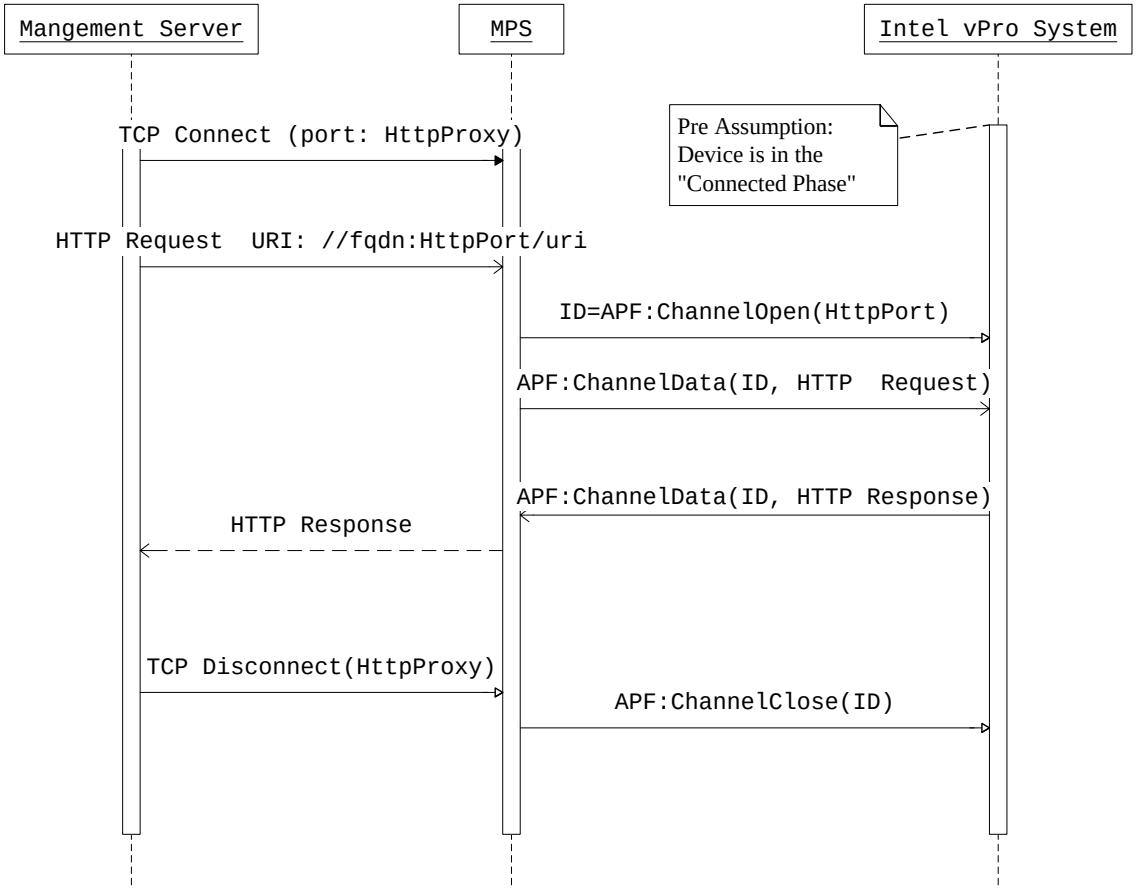

Figure 5: Web-based Routing

Recall that when a device is in the “connected phase,” the relevant management server has already received a notification indicating connectivity.

- When the management server is creating an HTTP transaction (i.e., a WS-MAN call), it sends the request through to the MPS, specifying its address as the HTTP proxy. This is depicted in the “HTTP request” call in Figure 5. Typically, modifying HTTP client software applications to use an HTTP proxy is straightforward. For example, applications built on top of the .NET HTTP client class can specify a session proxy by adding the following line of code:

```
webRequest.Proxy = new WebProxy("http://mps.local:8080”);
```

- Next, MPS uses the provided FQDN information in the HTTP URI field to locate an active session for the given device, assuming one exists.

- MPS then uses the APF:ChannelOpen call to establish a virtual channel with the device. Note that each TCP session between a management server and MPS corresponds to exactly one virtual channel between MPS and an Intel vPro system. This is in essence the port-forwarding technique utilized by Intel vPro technology.
- The newly opened virtual channel is assigned a unique identifier (ID) that is used next to mediate HTTP transactions. MPS uses the APF:ChannelData call to mediate HTTP transactions that are sent to and from the management server.

3.3.1 Routing HTTPS Protocols

Management operations using HTTPS are routed quite similarly: relaying HTTPS traffic through an HTTP proxy is common practice and is achieved by using the HTTP “connect” verb to establish a full-duplex channel through the proxy. Since the HTTPS payload is opaque to the MPS, MPS routes the HTTPS encrypted transactions in a similar fashion to the HTTP flow. It uses the APF:ChannelOpen call to open a virtual channel associated with the HTTP session and the APF:ChannelData call to mediate the encrypted payload between the peers.

3.4 Routing Redirection Protocols

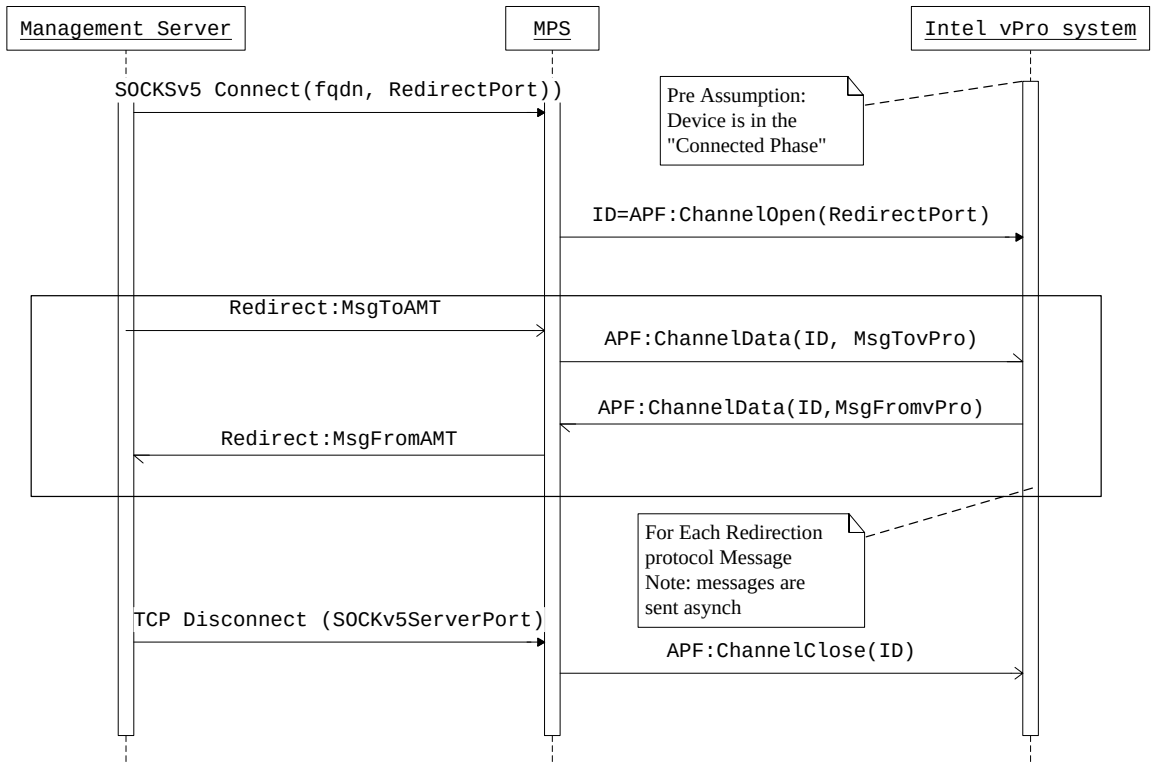

Figure 6: Redirect Protocol Routing

Figure 6 describes how system data and text redirection messages are routed through an MPS.

Since the Intel AMT redirection protocols (SOL and IDE-R) are not HTTP protocols, but rather are implemented directly over TCP or SSL, we utilize the SOCKS v5 protocol that provides generic routing capabilities for TCP-based protocols. One can imagine SOCKS v5 as a generalization of the HTTP proxy construct. In fact it is possible to use SOCKS v5 to replace the HTTP proxy for HTTP transactions in MPS. However, we decided to retain the HTTP proxy due to its wider range of support. To provide ISVs with support for SOCKS v5, Intel provides a software library exposing C language calls that abstracts the SOCKS interface as part of the Intel AMT Software Development Kit (SDK) [13].

Routing of the redirection protocols described in Figure 6 is similar to the routing of HTTP transactions described earlier, so we do not go into it here.

4 MPS Design

4.1 Internal Component Breakdown

MPS is composed of three major components:

- **MPS Core Module**: This is the main MPS component. It acts as a SOCKS proxy for the side of the MPS that faces the management console, and it has a TCP APF tunnel for each Intel AMT machine connected to it on the other side. The MPS core module is implemented in C++, using the ACE framework. We based the design on the ACE thread pool Reactor pattern (for more details on this, see the section on ACE).
- **TLS Server Module**: This is a TLS terminator for the TLS sessions with remote Intel AMT devices. It performs client authentication, based on a standard client certificate validation process, including checking for certificate revocation. In order to allow offloading of SSL connections to any user-provided SSL terminator, we decided to separate the SSL terminator from the MPS core module. We choose stunnel for this purpose [15], because it allows encryption of arbitrary TCP connections inside SSL and it is available for both UNIX* and Windows* operating systems.
- **HTTP/HTTPS Server to SOCKS v5 Client Module**: This module acts as an HTTP/HTTPS proxy server, and it enables management consoles to connect to Intel AMT machines through the MPS, via a standard HTTP proxy interface. This module connects to the SOCK Sv5 server module and proxies requests from management consoles to it. We choose the Apache HTTP server [16] for this component since it is a popular, well-known Web server that is secure, efficient, and extensible. We added to the Apache HTTP server another software module that allows outgoing HTTP and HTTPS requests to be redirected through a SOCKS v5 proxy server.

Figure 7 shows the MPS design and the relationships between the design's various components.

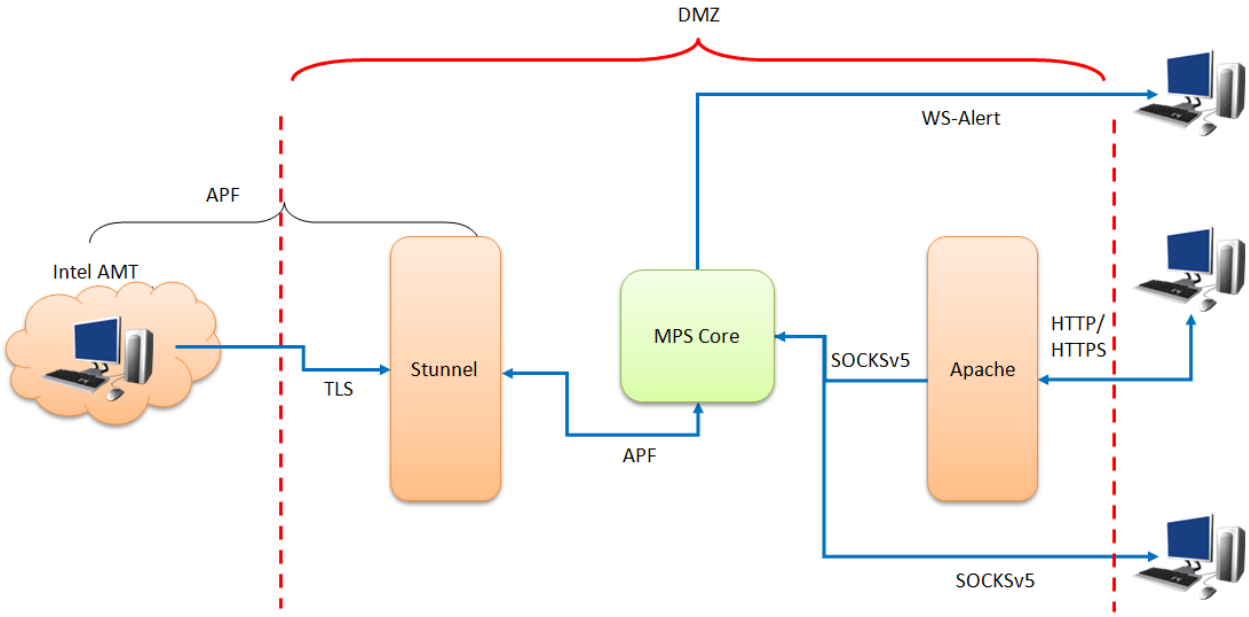

Figure 7: High Level MPS Overview

4.2 The Adaptive Communications Environment

We first describe the Adaptive Communications Environment (ACE), paraphrasing, for the most part, from an important book on ACE [20] and from the ACE tutorial [19].

The Adaptive Communication Environment (ACE) is a highly-portable, widely-used, open-source toolkit for developing host infrastructure middleware. The source code of ACE is freely available and runs portably on several operating systems, including Microsoft Windows (32- and 64-bit versions) and most UNIX versions. ACE is targeted at developers of high-performance and real-time communication services and applications. It simplifies the development of object-oriented network applications and services that utilize inter-process communication, event de-multiplexing, etc.

The ACE frameworks implement a pattern language for programming concurrent object-oriented, networked applications. Using these patterns enhances key qualities of these applications, such as flexibility, extensibility, reusability, and modularity.

We chose ACE as the basis upon which to build MPS because of its portability and because of the high-level abstractions that it provides. We used the patterns described in the following subsections.

4.2.1 Thread-Pool Reactor

The MPS core module uses the ACE thread pool Reactor (“the Reactor”). The thread pool Reactor contains a pool of threads, where one is the leader thread and the rest are followers. The leader thread waits for events on any of the registered handlers. When an event occurs, the leader promotes one of the followers to take its place as leader, and it processes the event itself. This saves unnecessary context switches.

The following options were considered for the threading model of the MPS:

- Single threaded with buffered non-blocking I/O.
- One thread per connection with blocking I/O.
- Multithreaded receiving and sending with buffered non-blocking I/O, utilizing a thread pool.

We chose the third option, multithreaded receiving and sending with a thread pool, since it is the most scalable solution, and because it is not overly complicated to develop for this method. Also, this method scales well on a machine with multiple CPUs (or CPU cores). If we were required to load MPS on an embedded device, we might have chosen the single-threaded model instead.

The ACE Reactor pattern was developed to provide an extensible object-oriented framework for efficient event de-multiplexing and dispatching. Current operating system abstractions for event de-multiplexing are not easy to use, and they are, therefore, error-prone.

The Reactor pattern essentially provides for a set of higher-level programming abstractions that simplifies the design and implementation of event-driven distributed applications [19]. Besides this, the Reactor integrates the de-multiplexing of several different kinds of events into one easy-to-use API. In particular, the Reactor handles timer-based events, signal events, I/O-based port-monitoring events, and user-defined notifications, all in a uniform fashion.

The basic idea is that the Reactor framework determines that an event has occurred (by listening on the operating system’s event de-multiplexing interface, such as select) and issues a “callback” to a method in a pre-registered event-handler object. This object is implemented by the application developer and contains application-specific code to handle the event.

The application developer must then do the following:

1. Create an event handler to handle an event he/she is interested in.
2. Register with the Reactor, informing it that he/she is interested in handling an event and at this time also passing a pointer to the event handler that is to handle the event.

The Reactor framework then automatically does the following:

1. Maintains tables internally, which associate different event types with event-handler objects.
2. When an event occurs that the user had registered for, it issues a callback to the appropriate method in the handler.

In MPS, the service handlers (Tunnel and TCP, described later in this article) execute on a thread from the thread pool upon receipt of input events on their sockets, and upon receipt of output events that occur when messages are added into their queues.

4.2.2 Acceptor-Connector

This framework leverages the Reactor framework and implements the Acceptor-Connector pattern [18]. This design pattern decouples the connection and initialization of cooperating peer services in a networked system from the processing they perform once connected and initialized.

In MPS we used two Acceptors: one for incoming SOCKS requests and the other for incoming Intel AMT requests. Each Acceptor registers itself in the Reactor for input events on the socket. When data come to an Acceptor socket, the Reactor calls the Acceptor's event routine. The Acceptor just accepts the new connection and creates a new service handler (SOCKS or Tunnel) that will handle this new connection from now on.

The ACE connector pattern is very similar to the Acceptor pattern, but it is used to actively connect to a remote host.

MPS also uses the connector pattern to initiate connections from Intel AMT systems to servers inside the enterprise. When a remote Intel AMT system wishes to send an alert (PET or WS-Alert) to a pre-configured server, it sends the APF message to MPS to send that alert. MPS uses the connector to create the connection, and then it passes the data through it.

4.2.3 Message Queue Notification

ACE allows for a message queue to be integrated with its Reactor [20]. The message queue sets a notification strategy method, and after that, each queued message triggers the Reactor to dispatch the notification to the specified event handler.

In MPS, each service handler (SOCKS or Tunnel) registers its queue for notification when a message is added to its queue. When the dispatcher is called, the service handler removes the message from the queue and sends it through its socket.

4.3 MPS Core Module Breakdown

For simplification, in this section we describe only the main components of the MPS. Mixed-case object names (such as TunnelSupplier) represent objects in the code.

4.3.1 Acceptors

The Acceptor handles incoming connection requests. There are two Acceptors, one for handling incoming Intel AMT tunnel connection requests, and one for handling SOCKS connection requests. Upon receiving a new connection, the Acceptor creates either a SOCKS service handler or a Tunnel service handler. The type of handler depends upon whether the Acceptor is listening to incoming SOCKS connection requests or incoming Tunnel connection requests.

4.3.2 Tunnel Service Handler

This component handles all APF input and output with one Intel AMT system. Each established connection endpoint (socket) to the MPS is associated with a single tunnel handler, so its lifetime is for the duration of the connection. It is responsible for translating the data stream to APF and for handling APF flow control messages.

Each handler holds a socket and a queue. When an entity (such as a management console) wishes to send data to the Intel AMT system associated with this tunnel handler, it enqueues the data to this handler's queue, so the handler will send them later on.

Upon creation, the tunnel handler registers itself with the Reactor for two events:

- Input event: There are new input data in the socket.
- Output event: A message was enqueued to the handler's queue.

The Tunnel service handler comprises two major components:

- **Tunnel Supplier**: The Tunnel supplier handler handles incoming data from APF tunnels. Upon receipt of a complete APF packet, it parses the packet and acts according to the received data.
- **Tunnel Consumer**: The Tunnel consumer handler handles outgoing requests for APF tunnels. Upon receipt of a request it creates an appropriate APF request and adds it to its queue. This causes the Reactor to call the output event handler of this tunnel handler (see the ACE section for more details), possibly in some other thread. When this event handler is called, the tunnel consumer then checks if some other thread is currently handling output events for this Intel AMT system. If so, it returns control to the caller (since that thread will handle all the messages in the queue); if not, it sends the messages in the queue.

4.3.3 TCP Service Handler

This component handles all TCP input/output data with one management console. Its design mirrors that of the Tunnel service handler. Each established connection endpoint (socket) to the MPS is associated with a single TCP handler, so its lifetime is for the duration of the connection.

Each handler holds a socket and a queue. When an entity (such as an Intel AMT system) wishes to send data to the management console associated with this TCP handler, it enqueues the data to this handler's queue, so the handler will send them later on.

When the tunnel handler is created, it registers itself with the Reactor for two events:

- Input event: There are new input data in the socket.
- Output event: A message was enqueued to the handler's queue.

The TCP service handler comprises two major components:

- **TCP Supplier**: The TCP supplier handler handles incoming data from the management console. Upon receipt of data, it reads and sends the TCP stream to the attached Intel AMT through its ChannelConsumer module (more on this later).
- **TCP Consumer**: The TCP consumer handler handles outgoing requests from a specific management console. Upon receipt of a request, it adds the request to its queue. This causes the Reactor to call the output event handler of this TCP handler (see the ACE section for more details) possibly in some other thread. When this event handler is called, the TCP consumer then checks if some other thread is currently handling output messages for this management console. If so, it returns control to the caller (since that thread will handle all the messages in the queue); if not, it sends the messages in the queue.

4.3.4 SOCKS Service Handler

This component inherits from the TCP service handler (which handles only data streams) and implements the SOCKS v5 handshake described in the SOCKS v5 RFC [17]. After a connection is established, the SOCKS service handler behaves exactly like the TCP service handler (since it only transfers TCP streams).

4.3.5 Channel Consumer Handler

The channel consumer handler handles outgoing requests on APF channels. Its role is to restrict the flow of data coming from a single management console so that it does not exceed the window size defined by the Intel AMT system. It holds a reference to the matched tunnel consumer handler and is responsible to pass the data through to it.

Upon receipt of a request, the channel consumer handler checks if the available window size is zero. If it is, it adds the request to its queue and returns control to the caller. If not, it calls the associated tunnel consumer handler to send the message. When the Intel AMT system updates the window size for this channel, the channel consumer handler checks to see if there are messages in its queue. If there are, it sends the messages until the new window size limit is reached. Sometimes it is necessary to split one message into two parts—one to send through the tunnel consumer handler, and the second to return back to the queue.

The channel consumer handler's lifetime is for the duration of the channel (that is, for as long as both sides of the connection are still using the channel—the management console until it closes the socket for reading, and the Intel AMT system until it closes the channel).

The relationship between the communication sockets, the service handlers, and the communicating systems (Intel AMT and management console) are shown in Figure 8.

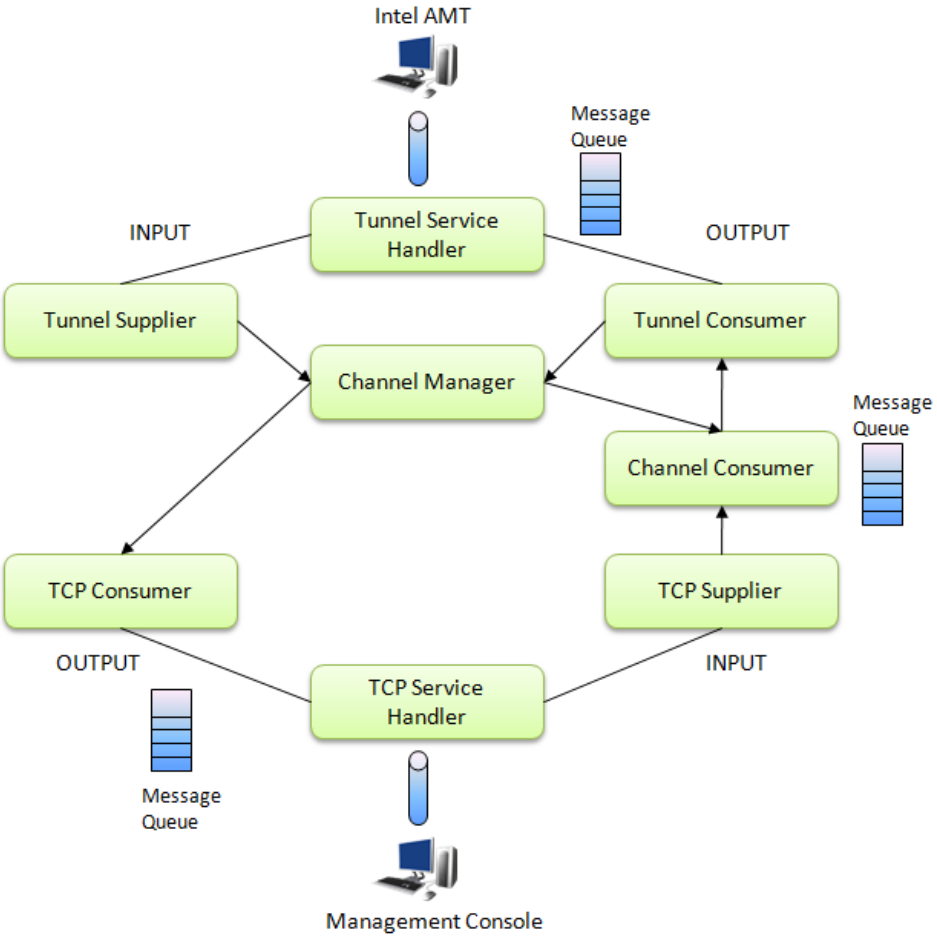

Figure 8: MPS Internal Components

4.3.6 Notification Consumer Handler

The Notification consumer handler handles outgoing notification requests used for notifying subscribers of Intel AMT connection/disconnection events.

The design of this component was changed during the programming and testing phase, and we explain it later in the Challenges section of this article.

4.4 Internal Flows

In this section we briefly explain the main flows of MPS by using the components described in the previous section.

4.4.1 Opening an Intel AMT Tunnel

Figure 9 depicts the flow of information and control upon opening a tunnel to an Intel AMT system.

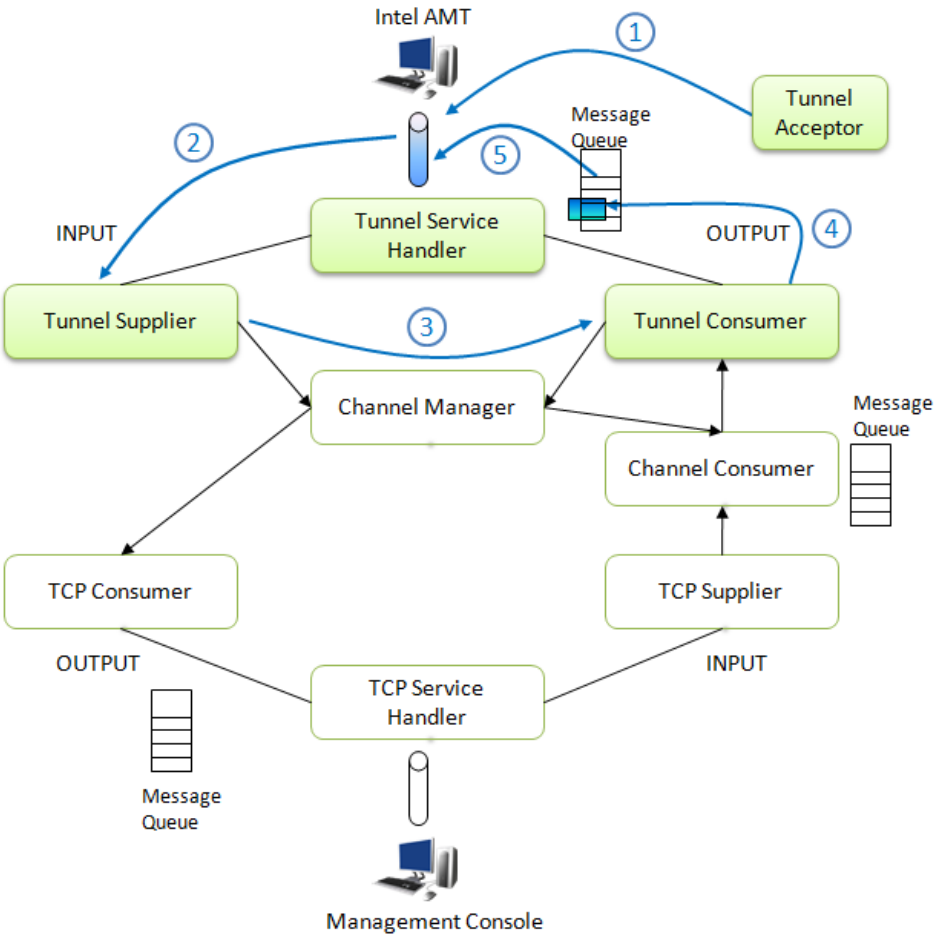

Figure 9: Opening a Tunnel to an Intel AMT System

1. The tunnel acceptor object receives a new request from the Intel AMT system. It accepts the connection and creates a new tunnel handler. When this tunnel handler is created, an "open" dispatcher is called. In this dispatcher the tunnel handler does the following:

   a. Creates its tunnel supplier and tunnel consumer objects.

   b. Registers itself with the ACE Reactor for an input event when there are input data on its socket.

   c. Registers itself with the ACE Reactor for an output event when a message is added to its queue.

2. When APF version data are received, the tunnel handler reads the message from the socket and calls the tunnel supplier to handle it.

3. The tunnel supplier parses the message and checks its legality. It then calls the tunnel consumer to send an APF reply.

4. The tunnel consumer adds the APF message to the queue.

5. The Reactor notices that a message was added to the queue, and it dispatches the "handle_output" event. In this event the message is removed from the queue and sent through the socket to the Intel AMT system.

4.4.2 The Management Console Opens a SOCKS Connection

The flow for a SOCKS connection from a management console is shown in Figure 10. In this scenario, we assume that the tunnel is already connected to MPS. Also, for consistency with the previous figures in this article, the SOCKS service handler is depicted as a TCP service handler (from which it inherits).

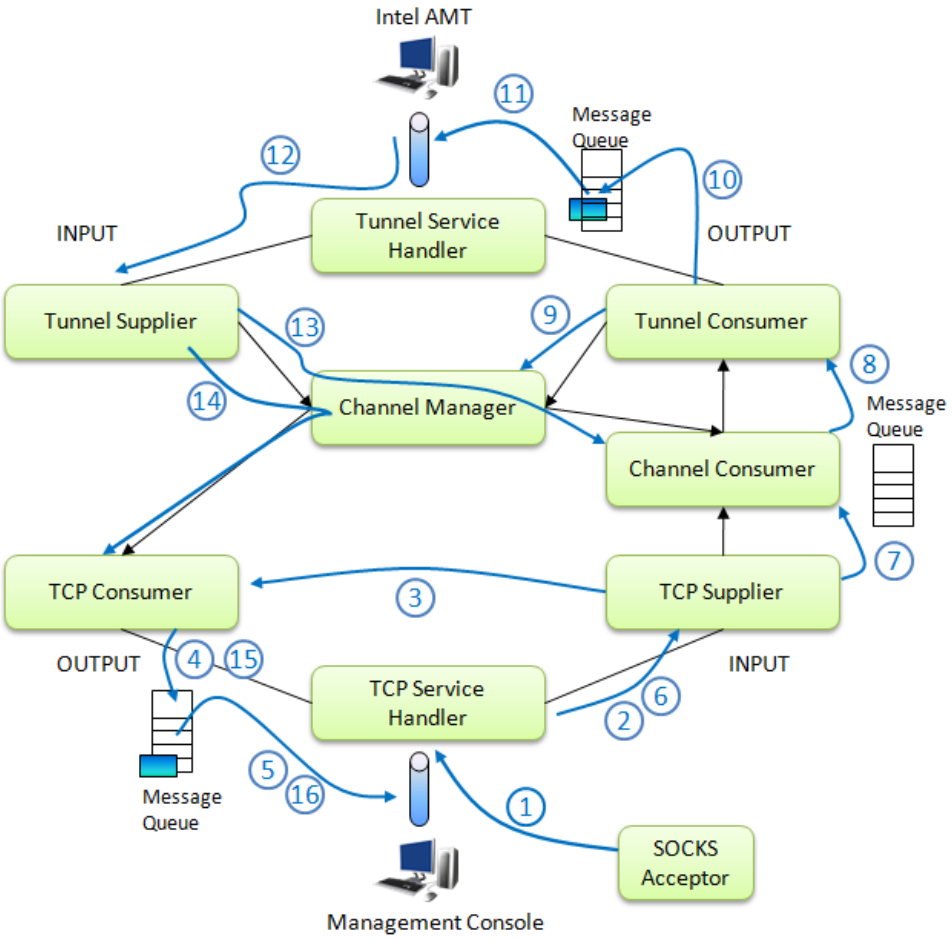

Figure 10: Management Console Opens a SOCKS Connection

1. When the SOCKS Acceptor receives a new request from a management console, it accepts the connection and creates a new SOCKS service handler.

2. When this SOCKS service handler is created, an "open" dispatcher is called. In this dispatcher the SOCKS service handler does the following:

   a. Creates its SOCKS supplier and SOCKS consumer.

   b. Registers itself with the ACE Reactor for an input event when there are input data on its socket.

   c. Registers itself with the ACE Reactor for an output event when a message is added to its queue.

When a SOCKS message arrives, the SOCKS service handler's "handle_input" event trigger is called. In this event the SOCKS service handler calls the SOCKS supplier to handle the message.

1. The SOCKS supplier parses the message and checks its legality. It then calls the SOCKS consumer to send a proper reply.

2. The SOCKS consumer prepares a matched SOCKS message and adds it to the handler's queue.

3. The Reactor notices that a new message was added to the queue and triggers the "handle_output" event. In this event, the message is removed from the queue and sent to the management console through the socket.

4. When a SOCKS "request" message arrives (the message containing the Intel AMT address that the management console wishes to connect to), the SOCKS supplier parses it and checks its legality.

5. If the message is legal, the SOCKS supplier creates a channel consumer and lets it open the channel with the desired Intel AMT system.

6. The channel consumer checks that the Intel AMT address matches one of the open tunnels in MPS. If it does, the matched tunnel consumer is called to open a new port for this new management console.

7. The tunnel consumer first registers the new management console's data in the channel manager (including a pointer to the channel consumer and SOCKS consumer).

8. The tunnel consumer then prepares an APF:ChannelOpen message and adds it to the queue.

9. The Reactor notices that a new message was added to the queue, so it triggers the "handle_output" event. In this event the message is removed from the queue and is sent to the Intel AMT system through the socket.

10. When an APF:ChannelOpenReply is received, the tunnel consumer reads it from the socket.

11. The tunnel supplier changes the channel consumer's state to open, and updates its initial window size.

12. The tunnel supplier then calls the SOCKS consumer to send the management console a reply message.

13. The SOCKS consumer prepares a suitable SOCKS v5 message and adds it to the queue.

14. The Reactor notices that a new message was added to the queue, so it triggers the "handle_output" event. In this event the message is removed from the queue and sent to the management console through the socket.

4.4.3 Transfer of Data to the Management Console by the Intel AMT System

Figure 11 depicts the transfer of data from an Intel AMT system to a management console. In this scenario, we assume that the connection between the Intel AMT system and the management console is already established (see the previous section).

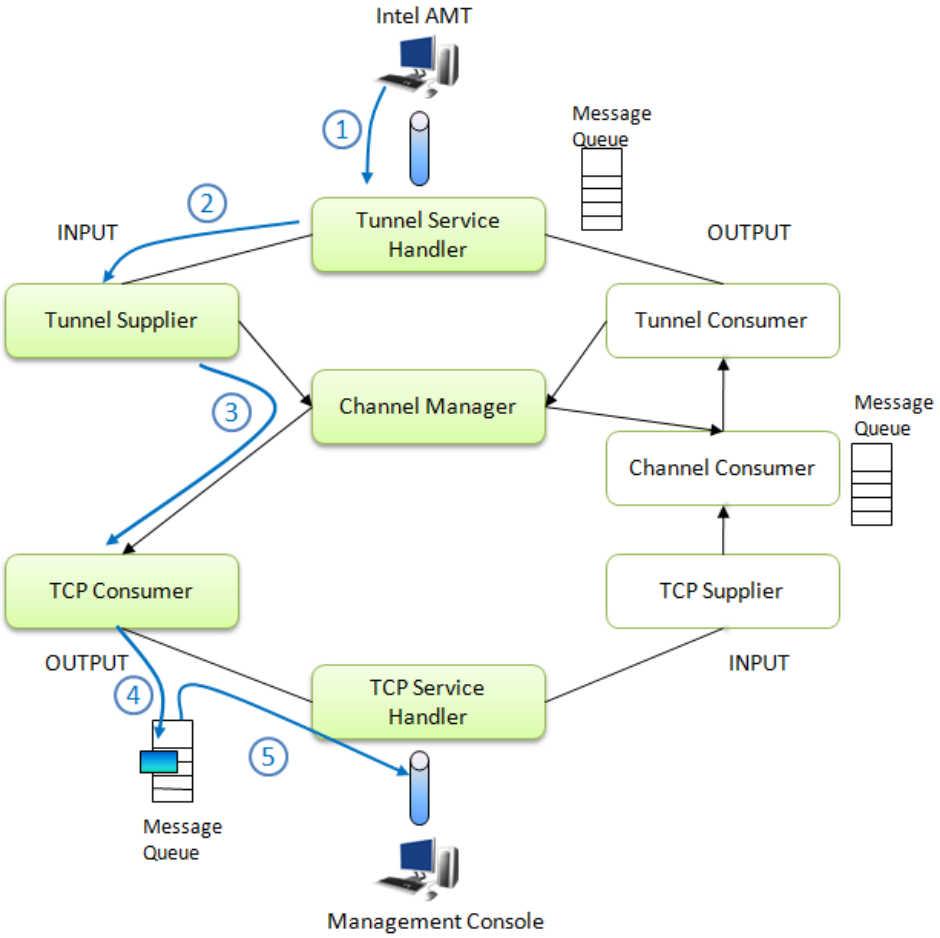

Figure 11: Data Transfer from the Intel AMT to the Management Console

1. When APF data are received, the Reactor dispatches the tunnel service handler's "handle_input" event.
2. The dispatcher calls the tunnel supplier to handle the event and process the data.
3. The tunnel supplier parses the message, strips the APF wrapper, and checks the message's legality. Through the channel manager it finds the corresponding TCP consumer, and calls it to send the data to the management console.
4. The TCP consumer adds the message to the queue.

5. The Reactor notices that a message was added to the queue, so it dispatches the “handle_output” event. In this event the message is removed from the queue and sent through the socket to the management console.

4.4.4 Transfer of Data to the Intel AMT System by the Management Console

Figure 12 depicts the transfer of data from the management console to the Intel AMT system. In this scenario, we assume that the connection between the Intel AMT system and the management console is already established.

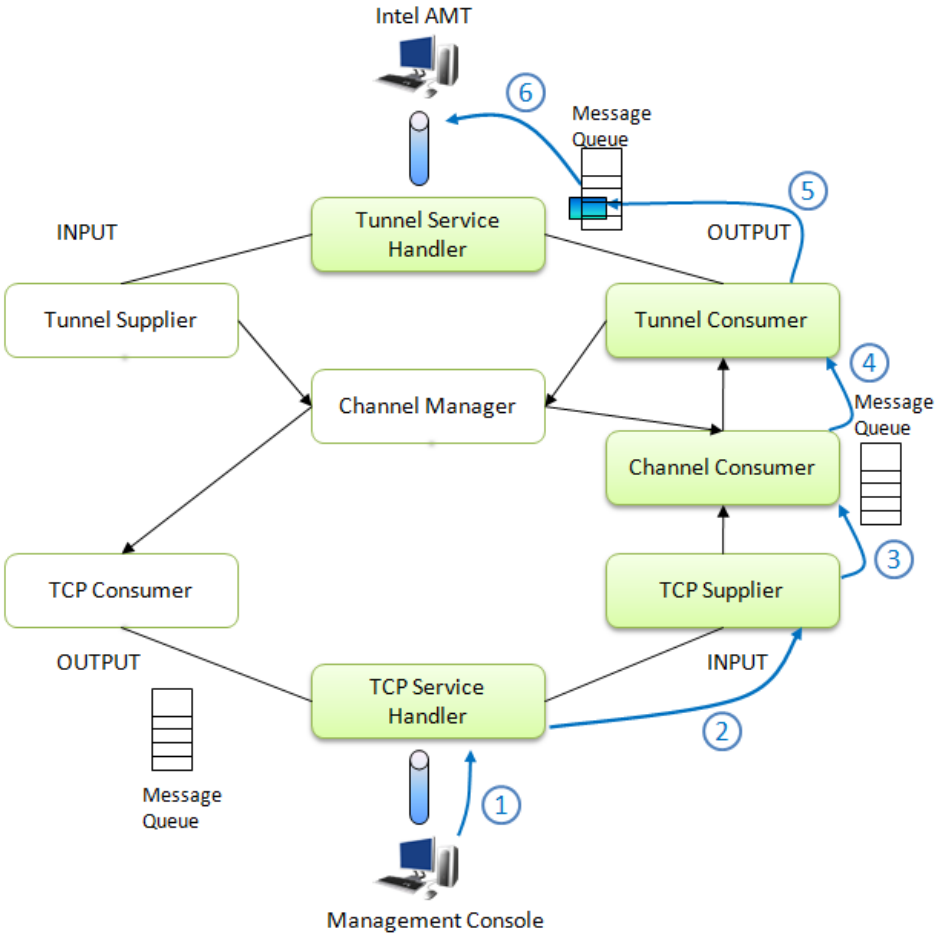

Figure 12: Data transfer from the management console to the Intel AMT

1. When SOCKS data are received, the Reactor dispatches the TCP service handler's "handle_input" event.

2. The dispatcher calls TCP supplier to handle the incoming data.

3. The TCP supplier reads the message and calls the channel consumer to send the data to the Intel AMT system.

4. The channel consumer checks if the remaining window size is sufficient for the size of the current message. If so, it calls the tunnel consumer to send the message; if not, it adds the message to its queue.

5. The tunnel consumer prepares an APF message containing the TCP stream it received and adds it to the queue.

6. The Reactor notices that a message was added to the queue, so it dispatches the "handle_output" event. In this event the message is removed from the queue and sent through the socket to the Intel AMT system.

5 Challenges

In this section we examine the challenges faced in the design, implementation and deployment of MPS.

5.1 Requirements and Testing

Any MPS has to work in a production, enterprise environment, providing 24 hour-a-day, 7 day-a-week service. This always-available requirement dictates that MPS run as a service ("daemon" in Unix/Linux* parlance). The requirement for the reference version of MPS is to run as a service on the Windows Server 2003 operating system only. Supporting only one operating system reduces the amount of development work, and in particular, it also reduces the amount of validation work to manageable proportions. As a service, MPS makes a log entry when the service starts (or fails to start) to the Windows event log. All other information, such as which machines connected or disconnected, is logged to a regular text file. MPS has a configurable logging level, controlling the amount of information that it logs.

Because deployment environments differ, the choice of Transport Layer Security (TLS) terminator in the DMZ is not hard-wired into MPS; rather, MPS can use any TLS terminator. The default terminator for the reference MPS is stunnel, which is a general purpose, multi-platform SSL wrapper program. However, MPS has been tested with other TLS terminators, such as the Juniper SSL VPN appliance [21].

Although the reference implementation was validated only on Windows Server 2003, MPS itself is portable; it compiles and runs on Linux as well. However, the Linux implementation is not quite as capable for the following reasons:

- It does not run as a background daemon.
- It does not use any operating-system logging facilities.
- There is no support for user authentication (no sample authentication library is supplied for Linux).
- There is no automated installer.
- Fewer validation tests were performed with the Linux implementation and Intel does not currently support this version.

Besides the “always available” requirement, the MPS reference implementation has additional performance requirements:

- It must support all forms of TLS that Intel AMT supports. In practice, meeting this requirement is offloaded to the TLS terminator.
- It must support up to 1000 concurrent tunnels.
- It must support up to eight management console subscribers listening for connect/disconnect events from Intel AMT systems.
- It must be able to sustain continuous throughput of 1.5 megabits per second, per system, for 300 systems. This is close to half a gigabit per second total. The use-case scenario here is the need to pass IDE-Redirection data for remote bootstrapping of Intel AMT systems that are behind an Asymmetric Digital Subscriber Line (ADSL) connection.

These requirements provided significant engineering challenges, and in particular, challenges for the validation team. For example, the validation team does not have 1000 Intel AMT systems available for testing MPS! It was necessary to either obtain, or design and implement, several simulators in order to fully test MPS.

Once beta-level software was available, the Functionality and Testing Laboratory (FTL) at the Israel Software Design Center (ISDC) spent two person years of effort involving five engineers to test MPS. (The total effort was greater, but the bulk of the testing occurred once beta software was ready.)

For stress testing, FTL did a number of things. First, to push MPS to its maximum limit, they created a simulator to simulate 1000 Intel AMT systems connecting to MPS to request management services. In addition, FTL created tools to simulate the full data transfer load of 300 connected systems performing IDE-Redirection, each moving data at 1.5 megabits per second. Finally, FTL made sure that all of the Intel AMT functionality testing was conducted through MPS as well; this included up to 50 Intel AMT systems connected to 50 management consoles testing all the different Intel AMT features.

Additional tools developed outside ISDC were used to simulate bidirectional flows of HTTP and SOL data.

FTL used a SmartBits [22] system that injects noise onto a clean network in order to simulate different bandwidths and latencies. Many issues were found and resolved by causing rapid numbers of connect and disconnect requests.

Early on, FTL recognized that despite their best efforts, all of the testing and validation that they could do would still only be "laboratory testing." It was necessary to test MPS in the real world to see if it worked, and to allow Intel to confidently tell its customers that MPS had been properly tested.

Consider this. MPS sits in the corporate DMZ. If there are problems or bugs, it might be possible for a malicious entity to gain access through MPS to systems behind the firewall. Intel needs to be confident that MPS is secure enough that such a scenario will not occur. To that end, FTL began a long negotiation process with Intel Corporate IT to place an MPS system in the Intel DMZ and test it.

Intel IT does not maintain a traditional DMZ; rather, it maintains a "sandbox" network wherein it can test the deployment of systems before moving them to production use. This network has two firewalls: one facing the Internet and the other facing the internal Intel intranet. Thus, this location was close enough to a real DMZ to provide the necessary testing for MPS. Even though the sandbox is not a production network, getting approval to place an MPS system there took six months!

The MPS system was installed in the Intel sandbox in Folsom, California (USA). A management console was set up in Israel, and a test Intel AMT system was also set up in Israel—behind a 1.5 Megabit ADSL link. Once it was set up, access to MPS was restricted to the tightest possible level: only one specific IP address for the Intel AMT system was allowed to connect in through the external firewall, and only two specific management consoles were allowed to connect to the MPS. (There were some technical problems just getting everything going: FTL sent a test system to Folsom that was not a standard build of Windows Server 2003 and thus lacked some needed services. Also, Intel IT had to patch their firewall to allow all the traffic in.)

Once everything was set up, FTL ran tests between the Intel AMT system and the management console through the MPS located almost half a world away. The latency between the Intel AMT system and the MPS was approximately 240 msec; the latency back to the management consoles was only slightly smaller: about 220 msec.

While the slow connection between the Intel AMT system and the MPS is expected, more typically the connection between the MPS and the management console would be much faster. This setup thus represented a pretty severe worst-case scenario. It helped flush out a number of problems (described in the Performance Challenges section).

6 Engineering Challenges

This section describes the challenges encountered during the development of MPS.

6.1 Connection Management

Connection management poses several challenges. First, the relationship between Intel AMT systems and management consoles can be many-to-many: multiple management consoles are communicating with a single Intel AMT, and multiple Intel AMT systems are communicating with a single management console. With the requirement for 1000 simultaneous connections, you have to ensure that all the bookkeeping code is both correct and efficient.

Above and beyond the bookkeeping is the fact that APF is a multiplexing protocol. All connection data for a single Intel AMT system must be sent over a single tunnel.

Much of the work is endemic to the nature of multi-threaded programming combined with non-blocking I/O, since a thread is awakened every time there are incoming data on any socket (this is discussed further later on).

A particular challenge involves correct management of resources in difficult cases, such as a half-closed connection. In such a case, one thread could be closing the output half of a connection while another is still receiving input. MPS must be sure to correctly release memory, channels, tunnels, and sockets in such a case, being careful not to release any resource too early, as well as being careful to finally release resources that are no longer in use.

6.1.1 Authentication Challenges

Initially MPS worked by using only mutual authentication; this is good, since MPS knows for sure that it is talking to a real Intel AMT system.

During development, requirements were changed to also allow server authentication; this means that the Intel AMT system can trust the MPS system, but the MPS system cannot necessarily trust the Intel AMT system. This posed several problems.

First, an authentication mechanism had to be defined. MPS sidesteps this issue by requiring a user name and password, and then defining an API that must be provided by external dynamically loadable libraries (DLLs). This API takes the provided user name and password and returns a simple valid/invalid return code. The API is used to authenticate Intel AMT systems and management consoles, but the configuration file allows specifying different DLLs to provide the API in these two cases. The user name and password from the Intel AMT system are in the clear inside the APF protocol; however, the APF protocol itself is secured inside the SSL tunnel.

Second, authentication cannot be based on the IP address, for several reasons:

- The TLS terminator knows the IP address of the Intel AMT system; MPS doesn't.
- Multiple Intel AMT systems can be connected from behind a single router providing NAT.
- By definition, mobile Intel AMT systems move around; they do not have fixed IP addresses.

Finally, if authentication requests take time (and they might if they have to make a remote database query), then it is possible to cause a Denial of Service (DoS) attack: 1000 machines all connecting simultaneously could tie up the MPS. This issue remains open.

6.1.2 Problems with Other Components

MPS has to interoperate with other components in order to implement a full two-way connection. MPS must be able to handle failure or misbehavior of the other components in the picture.

Because an Intel AMT system can be distant (long latency) from the MPS, the Intel AMT firmware can timeout connections to the MPS; in such cases, MPS receives no notification that a timeout occurred until it receives a new connection request from the same system. As a result, MPS cannot know that the Intel AMT connection timed out and it must keep many sockets open. Application software can exacerbate this problem. Consider an Internet Explorer* session to the Web server on the Intel AMT system. Internet Explorer opens multiple sockets in order to speed up the transfer of data (text and pictures), and then it leaves these sockets open, even after the browser window has been closed!

Other problems were encountered with the Apache Web server that provides "socksification" of HTTP/HTTPS data. The Apache server works in an unusual fashion. While running, it allocates memory as needed when connections are opened and closed. However, it never frees allocated memory. Instead, upon reaching a predetermined threshold of allocated memory, it creates a new instance of itself that inherits the open communication channels, but starts over with only the memory needed for those connections; the old instance then exits. When the memory again reaches the proscribed limit, it again creates a new instance.

The problem was that the open communication channels were not always transferred to the new instance correctly, particularly when the workload was heavy. It took careful trial-and-error work to tune both the maximum number of threads and the memory threshold to cause Apache to work correctly.

An additional issue was that Apache had to be configured to parse the entire HTTP header in order to correctly perform Kerberos authentication. (The MPS documentation describes the configuration parameters.)

6.1.3 The ACE Framework

The ACE framework was used to provide portability and higher-level network programming abstractions (as described earlier). However, ACE was not without its own problems.

ACE is not a small piece of software, and correct use of ACE requires a deep understanding of its design and implementation. For example, the developer must understand the way ACE uses mutexes internally in order to avoid deadlocks.

Additionally, there were bugs in ACE itself. As is not unusual with Open Source software, new versions of ACE were released during the course of MPS development. One of these versions would not compile without modifying a configuration file to define certain macros that affect compilation. The ACE maintainers fixed this bug in a subsequent release.

More challenging was a bug in the Reactor's management of handlers registered with the Reactor. When the MPS developers wished to define a class that inherited from the Reactor and add some additional methods, they were not able to do so, due to the bugs in the Reactor class itself. This bug too was eventually fixed by the ACE maintainers.

6.1.4 Notifying Management Consoles

Whenever an Intel AMT system connects or disconnects, MPS sends a notification message to up to eight management consoles (total) that are subscribed to receive such notifications. The list of consoles to notify is read from a configuration file; the list is re-read every time a message must be sent, so it is easy to update the list by simply revising the file.

The notification messages are sent in SOAP format. Currently, MPS uses a proprietary custom set of SOAP messages. To implement this, MPS uses the gSoap framework [24] for sending and receiving SOAP messages, with a gSoap plug-in.

gSoap has a fundamental limitation: it cannot work in parallel (multi-threaded) environments. Thus, the initial design for sending notifications was very simple:

```
for each system for which there are outstanding notification messages
  for each event which sends a message
    for each subscriber to notification events
          send the message
```

The thread that receives the "open a new channel" event also sends the notification message. Because I/O is blocking, no further input can be received until the notification message is sent. Because notification messages can travel over TLS encryption, they can take a long time to be received and acknowledged by the management console, which means that the pending input from the Intel AMT system can be blocked for a long time as well (several seconds).

To avoid the blocking I/O issues, we changed the design so that there is now a notification module that holds a queue. Each time an Intel AMT system opens a new port, this information is added to the notification module's queue and the thread continues working (thus this thread is no longer blocking). The notification module registers to the Reactor for an output event when something is added to its queue. Thus, when a message is added, the Reactor dispatches the notification module's event handler. In this handler, which runs in a different thread that doesn't block the handling of the connection with Intel AMT, MPS sends all the messages in the queue to all the subscribers.

This change significantly improved the notification performance, but there remains room for more improvements, which we discuss in the Future Challenges section.

6.1.5 The APF Protocol

The APF protocol is a multiplexing protocol. All connections to a single Intel AMT system are sent over a single APF tunnel. Because of this design decision, MPS could not be designed as a simple gateway; it must manage the multiplexed connections.

An interesting problem occurred with respect to the management of half-duplex connections (where one end of the connection was closed down). For example, if the Intel AMT system closed its side of the channel, the firmware still had to continue to update the window size and send window size messages so that the management console can continue to send data.

Finally, during testing it was found that some home routers will close idle connections. This required the addition of keep-alive messages to the APF protocol as well.

6.1.6 Blocking I/O

Blocking I/O is used throughout MPS. Blocking I/O is much easier to program than non-blocking (asynchronous) I/O, which is an advantage. However, it can lead to other issues. Besides the notification issues described earlier, every entry of data into a queue is blocking. If a management console sends data more quickly than the destination Intel AMT system can read them, the queues get very full and everything slows down.

Furthermore, MPS reads one whole APF message, and not everything that the socket holds. In case a socket holds more data, we process the data one APF message at a time. If the socket doesn't contain a full APF message, the working thread is now blocked until all data are received. A better design would be to just read what the socket has and return the thread to the Reactor until further data arrive. In such a case we would get much better performance.

There is clearly a tradeoff between the number of threads available to perform work (remember that each thread blocks on I/O operations) and the maximum size of a queue (in particular the output queue). These factors require tuning for optimal performance; MPS allows them to be tuned via its configuration file.

6.2 Performance Challenges

The entire end-to-end connectivity between the Intel AMT system and the management console is a long series of systems, leading to long round trip times for management commands and responses. The largest factor in the big round trip time is typically the home router's 1.5 megabit ADSL connection to the Internet.

Once MPS was up and running, it was found that SOL and IDE-Redirection performance over a remote connection was not acceptable; the redirection protocols were unusable.

Two changes were found necessary to improve the performance in the remotely connectd case. First, at the SOL and IDE-Redirection level, the firmware's TCP window size was increased from 16 kilobytes to 64 kilobytes. This allowed TCP to "fill the pipe" on longer latency connections. In addition, the firmware used the TCP_NODELAY flag that disables Nagle's algorithm, to increase response.

The second change, specifically for remote connectivity, was to increase the buffer sizes for IDE-Redirection transfers to 64 kilobytes, along with increasing the APF protocol window size to 64 kilobytes. This was particularly important for IDE-Redirection, which must move large quantities of data. Modern systems boot by using tens or hundreds of megabytes of data from a CD-ROM image instead of from the 1.44 megabyte floppy of yesteryear. This change made the performance of IDE-Redirection over a remote connection acceptable.

In general, the firmware works better with fewer large messages instead of many smaller messages, even if smaller messages are sent more quickly. However, MPS doesn't have a way to manage this; it does not know when SOL or IDE-Redirection traffic is passing through it in order to change the way it buffers messages. If it could know, performance could have been improved even more. Fortunately, the work done by the firmware team was enough and the current SOL and IDE-Redirection performance is acceptable.

7 Limitations and Future Challenges

The initial MPS implementation works, and works well. But there are a number of limitations in the current design and implementation. These represent a subset of the interesting challenges ahead for the MPS developers. The following paragraphs briefly outline the limitations and challenges, and in some cases, possible solutions.

- **More connections**: MPS must be able to support an order of magnitude larger number of simultaneous connections; at least in the tens of thousands. The current implementation works for the initial pilot project, but it is not suitable for very wide-scale deployment. This is a significant problem since the ACE Reactor cannot support so many handlers. It will be necessary to look into some sort of dynamic allocation of incoming connections among multiple MPS systems in the DMZ.
- **A standard notification protocol**: The current notification messages from MPS to the management console use a proprietary SOAP protocol. This area clearly needs standardization, so that MPS can be used with many different vendors' management consoles, without requiring custom code in MPS for each vendor. (The initial pilot project was deployed in an environment that uses a custom-made management system; the lack of standardization was not an issue.)
- **Improving the notification mechanism**: The current serial notification of management consoles is inefficient, and has problems due to blocking I/O. The correct solution is two-fold: first, bundle all the events into a single notification message, so that each console receives one message instead of one per event; second, use a UDP broadcast or multi-cast message instead of a TCP message, so that the notification message need be sent only once, and can be non-blocking. Being able to make these improvements requires the standardization of the notification message format just described.

In the current design, MPS handles both the connection establishment between Intel AMT systems and management consoles, and the transfer and multiplexing of all the data exchanged between them. This leads to the following two challenges:

- **Use in an Intel Connect Service environment**: The Intel Connect Service model has Intel AMT systems connecting to a central Intel site for authentication and then being redirected to a management service provider somewhere in the Internet. Neither MPS, nor the APF protocol, were really designed for this usage model.
- **Avoiding Single Point of Failure issues**: Right now, MPS is a definite single point of failure. If it fails, all the remote connectivity fails too. A mechanism must be defined to allow Intel AMT systems to connect into the enterprise network even if one MPS is down. As a step in this direction the Intel AMT firmware can maintain the names or IP addresses of up to two MPS systems. Beyond that, standard enterprise high availability solutions may be used, such as round-robin DNS assignment of addresses. Another possibility is for the SSL terminator to choose an MPS based on load.

There are some interesting deployment issues as well:

- **Blocking authentication**: Authentication of the user name and password from APF data can take a long time and blocks the thread upon which the data are received. This can end up blocking lots of threads for a long time, which in turn can use up most or all of the threads in the thread pool. It would have been better to have authentication requests defined as events to be handled by a worker thread, so that the input thread could be returned to the thread pool.
- **External management consoles**: The current MPS design assumes a secure corporate intranet. What if the management console is out in the wide Internet? For example, what happens if a company has outsourced its IT management? MPS only does TLS (encryption) on the outward, Internet-facing side, and not on the side facing the management consoles.

Finally, here are a few miscellaneous challenges.

- **Graceful shutdown**: MPS does not currently have a graceful shutdown mechanism; it is necessary to manually kill the program.
- **Thread count versus queue sizes**: It would be worthwhile to analyze the tradeoff between the number of threads and the sizes of the queues for different work loads. This would help us better understand some of the performance and scalability issues that still need work.
- **IPv6 support**: MPS does not currently support IPv6. This will become a problem as IPv6 becomes more widely deployed. There are already parts of the world where IPv4 addresses are simply no longer available and IPv6 is being actively used. Fortunately, support for IPv6 is easy to add, due to the use of ACE.
- **Quality of Service issues**: Currently, MPS functions as a multiplexing gateway, moving data in and out of the enterprise, but without any knowledge of the data being transferred and treating all clients equally. However, exactly because MPS is in this central position, it is in an excellent place to make Quality of Service decisions with respect to different clients and/or different kinds of data. This looks to be a fruitful area for future development.
- **No GUI**: MPS really needs a GUI so that it can be configured remotely, and to provide a visual overview of the connections it is managing. This should include a list of management consoles communicating with each Intel AMT system.
- **More scripts**: MPS currently provides a single Perl script that processes the log file to provide general statistics about the number of open connections, as well as errors and disconnects. It would be useful to have the ability to extract more information for tracking and planning purposes. This would require the ability to log more information as well.

8 MPS Availability

A Windows 2003 binary of MPS is available in the Intel AMT Software Development Kit for use by management ISVs.

9 References

[1] Intel Corporation. “Wired for Management Baseline.” Version 2.0, December 1998.

[2] K. Cline, “Alert Standard Format Specification (ASF),” DSP0136. April 2003.

[3] DMTF. “Specifications for CIM Operations over HTTP.” February 2007.

[4] DMTF. “DASH Implementation Requirements,” Revision 1.0.0b. October 2007.

[5] DMTF. “Web Services for Management (WS-Management) Specification.” February 2008.

[6] S. Kent and K. Seo. “Security Architecture for the Internet Protocol RFC 4301.” December 2005.

[7] Dierks, T. and E. Rescorla. "The Transport Layer Security (TLS) Protocol, Version 1.1." RFC 4346, April 2006.

[8] T. Ylonen and C. Lonvick. “The Secure Shell (SSH) Connection Protocol, RFC 4254.” January 2006.

[9] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D. and L. Jones. "SOCKS Protocol V5," RFC 1928. April 1996.

[10] Rescorla, E. "HTTP Over TLS," RFC 2818. May 2000.

[11] R. Richardson. “CSI Computer Crime and Security Survey,” 2008.

[12] DMTF. “Boot Control Profile,” DSP1012. October 2006.

[13] Intel AMT Software Development Kit At http://software.intel.com/en-us/articles/download-the-latest-intel-amt-software-development-kit-sdk

[14] Intel® Connect Service information available online. At http://www.intel.com/support/services/connect

[15] See http://www.stunnel.org.

[16] See http://www.apache.org.

[17] RFC 1928, SOCKS Protocol Version 5, March 1996. See http://www.rfc-archive.org/getrfc.php?rfc=1928. RFCs are available from many other sites as well.

[18] Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley & Sons, 2000, ISBN 0-471-60695-2.

[19] See http://www.eol.ucar.edu/software/ACE/ACE-tutorial.pdf.

[20] Douglas Schmidt, Stephen D. Huston. *C++ Network Programming Volume 2*, Addison-Wesley, 2003, ISBN 0-201-79525-6.

[21] See http://www.juniper.net.

[22] See http://www.spirent.com/analysis/technology.cfm?media=7&ws=325&ss=110.

[23] From the Hummingbird SOCKS FAQ (http://connectivity.hummingbird.com/products/nc/socks/faq.html?cks=y): “SOCKS is short for “SOCKetS”, which is an NEC internal development name that stuck with the technology.”

[24] The gSOAP Toolkit for SOAP Web Services and XML-Based Applications. See http://www.cs.fsu.edu/~engelen/soap.html.